DISSERTATION

A THEORETICAL FOUNDATION, METRICS AND MODELING OF PACKET REORDERING AND METHODOLOGY OF DELAY MODELING USING INTER-PACKET GAPS

Submitted by
Nischal Murthy Piratla
Department of Electrical and Computer Engineering

In partial fulfillment of the requirements
for the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Spring 2006

COLORADO STATE UNIVERSITY

SEPTEMBER 01, 2005

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY NISCHAL MURTHY PIRATLA ENTITLED A THEORETICAL FOUNDATION, METRICS AND MODELING OF PACKET REORDERING AND METHODOLOGY OF DELAY MODELING USING INTER-PACKET GAPS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Committee Member

Committee Member

Committee Member

Advisor

Department Head

ABSTRACT OF DISSERTATION

A THEORETICAL FOUNDATION, METRICS AND MODELING OF PACKET REORDERING AND METHODOLOGY OF DELAY MODELING USING INTER-PACKET GAPS

Increasing complexity, heterogeneity and variety of traffic in the networks makes it essential to understand the variations in delay, jitter, packet reordering, etc., thus leading to characterization and prediction of networks' performance, suitability to applications, scalability, etc. This research proposes a method for representation of packet reordering, metrics to capture reordering in a packet sequence and models relating traffic load-splitting scenarios to reordering. Delay models are investigated with respect to network conditions and properties of probing streams. Tradeoffs in the use of packet delay versus inter-packet gap for end-to-end characterization are considered to overcome the effects of correlations.

Increased parallelism required to handle high link speeds, large routing tables, wireless ad hoc routing, and enhancement features such as QoS and overlay routing, are some of the factors that point to an increase in reordering on Internet. Unchecked, packet reordering will have a significant detrimental effect on the end-to-end performance, while resources required for dealing with reordering will grow significantly. A formal representation of packet reordering is presented based on reorder event, reorder set, reorder segment, and basic patterns. A primary reorder event occurs when a packet is delayed or arrives early, with other affected packets said to have undergone secondary event(s). All the

events in a sequence form a reorder set. A principle of conservation is derived for the displacement, demonstrating the conservation of the overall displacement in a sequence, encompassing the interdependence between packet displacements. Furthermore, three basic reordering patterns are identified: independent reordering (IR) with only one primary event, and embedded reordering (ER), and overlapping reordering (OR) with only two entangled primary events.

"Reorder Density" (RD) metric is defined for measurement and characterization of packet reordering. Properties of RD are derived. RD maintains the interdependency between the packets using principle of conservation of reordering in a sequence, yet it maps to the probability density function for displacement of an arbitrary packet, allowing its use in systems approach, where subnets can be treated as systems and associated with overall responses to packet streams in terms of reordering. Reorder response, defined as the RD of a received sequence when packets are sent in-order at the sender, captures reordering introduced by a network. Under stationary operating conditions, it is shown that the reorder response of the network formed by cascading two subnets is equal to the convolution of the reorder responses of individual subnets.

Recovery from reordering at transport or application levels requires buffering of out-of-order packets. The concept of "Reorder Buffer-occupancy Density" (RBD) is based on the occupancy of a recovery buffer. Measurement approaches that capture such a resource requirement can be considered as useful metrics for packet reordering as well. An ideal metric for packet reordering will capture reordering accurately, provide insight into nature of reordering, and help in analysis and mitigation of its adverse effects. RD, RBD and other proposed metrics for reordering are evaluated using a framework developed for comparison and analysis of reorder metrics. The framework consists of attributes that include capturing

reordering, low sensitivity to lost and duplicated packets in reorder measurements, usefulness, simplicity, evaluation complexity, etc.

Models for packet reordering in terms of RD and RBD are presented for two-path load splitting, limiting reordering to basic pattern formations. Use of basic patterns is motivated by measurements over the Internet, which shows most of the reordering patterns as basic, i.e., IR, ER and/or OR. In the case of RD, general models of packet reordering are presented for both two-path and multi-path load splitting. All these models are verified using emulated topologies.

The end-to-end delay of packets in data streams is characterized with emphasis on effects due to cross traffic, sending rate, and packet size. Measurements indicate that modeling delay of a packet stream with high sending rates, as a fraction of bandwidth, is difficult due to the correlations among the delay values. The correlations among inter-packet gaps (IPG) at these rates, however, are negligible. At lower sending rates, the delay correlations are negligible, and the distribution of delay can be used as a delay model. We exploit the relation between delay and IPG to show how a Markov process can be used to approximate end-to-end delay.

Nischal M. Piratla
Department of Electrical & Computer Engineering,
Colorado State University,
Fort Collins, CO 80523
Spring 2006

ACKNOWLEDGEMENTS

My sincere thanks to Prof. Anura P. Jayasumana for his guidance, in-depth critique of the research, and patience with my ignorance. Thanks are also due to my committee members, Dr. V. Chandra, Dr. W. Bohm and Dr. Y. Malaiya for valuable suggestions, encouragement and criticism that drove me to give my best throughout this research. Thanks to Dr. H. Smith and Dr. J. Kurose for providing nodes at Calpoly and UMass respectively, to perform measurements. I would like to thank Abhijit A. Bare whose timely delivery of scripts and lists have immensely helped in my research work. This research was supported in part by NSF ITR Grant No. 0121546, Ixia University Partners Program and Agilent Technologies.

This research and document would not have been possible without the warm and caring support of my brothers and my wife.

DEDICATION

To my parents Krishna Murthy and Jayalakshmi, and my wife Sangita

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

Increased parallelism required in handling high link speeds and large routing tables, wireless ad hoc routing, and enhancement features such as QoS and overlay routing, are some of the factors that point to an increase in reordering on Internet. Unchecked, packet reordering will have a significant detrimental effect on the end-to-end performance, while resources required for dealing with packet reordering at routers and end-nodes will grow significantly. Lack of characterization and/or models for packet reordering can be attributed to insufficient measurements techniques, mainly percentage reordering. In this research a formal approach to characterize packet reordering and metrics to measure reordering are presented along with an exhaustive study of reordering over Internet and modeling techniques based on the formal representation.

Despite multiple studies of packet delay, the model of latency in a network is still not understood. The problem is further aggravated by variation in outcomes of the previous studies (mainly in terms of distribution, etc). This work also studies network delay extensively, but provides reasons for the variations that previous studies observed. In addition, the packet gap at the receiver is shown to be more significant in understanding packet delays. It is clearly shown that packet delay process is i.i.d for lower data rates and can be modeled as a Markovian process for higher data rates.

This is an introduction, motivation and outline chapter of the dissertation. In this chapter, a brief description is provided on the current state of the Internet in Section 1.1. It is followed by the discussion on the importance of characterization and modeling in Section 1.2, and challenges associated with the current approaches of characterization and modeling in Section 1.3. In Section 1.3, the thought process that motivated this research is also discussed. Finally in Section 1.4, the outline of the dissertation is presented.

1.1 Current State Of the Internet

In mid 1970s, when ARPA (Advanced Research Projects Agency) started its work on Internet technology [Hi94], the usage of networks was limited to two applications, namely file transfer protocol (FTP) and electronic mail (e-mail). The time taken for an e-mail to reach the destination or for files transfer to be complete was not crucial. However, today's user is presented with a variety of applications that run at higher speeds, and if she/he has to wait for even a few extra milliseconds or seconds, then the quality of service can be safely classified as unacceptable. Not only have the requirements/expectations grown, but the applications have become very diverse too.

To a large degree in today's networks, the TV signals are carried over cable networks, communication signals are carried using microwave systems, and home-user's telephone signals are carried over PSTN. However, Internet is a strong contender to carry almost all such real-time application streams [Mo03]. We are not too far from the day when every household will have a LAN (Local Area Network), and IP phones for voice and video communications. Though, we are in the midst of the most exciting era in networking, these new applications pose significant challenges to the current network architectures. Note that these networks have been designed with applications like e-mail and FTP in purview, and the fundamental design philosophy was significantly different for the Internet's architecture

[Cl88]. The end-to-end approach [Sa84] where the complexity is restricted to the endpoints and intermediate elements are just forwarding elements is being challenged constantly, with the changing paradigm among users, applications and service providers [Bl01].

1.2 Characterization and Modeling on the Internet

Internet, as of today, can be associated with increasing number of applications, increasing bandwidth requirements, increasing quality of service requirements, increasing real-time requirements, and increasing number of users. To improve the Internet to meet these growing demands, the dynamics of its behavior should be understood and characterized. Its vast and continuously growing size makes it impossible to analyze the dynamics with respect to memory, processing power and links associated with each and every node comprehensively. However, the statistical behaviors that can be attained by analyzing the end-to-end dynamics, i.e., the dynamics of packets in data streams between source and destination pairs are quite significant in evaluating the performance of data transfer over the Internet. Some of the useful derivations from such an end-to-end study include:

- *Suitability of the underlying network for an application:* If a real-time application like IP telephony is to be launched over a network, the end-to-end dynamics can provide a prior knowledge of delay variations, jitter distributions, etc. For example, an IP telephony application is restricted by a maximum one-way delay of 150 milliseconds. If an end-to-end study shows that this network has only 5% of the packets within this range then the network may not be very suitable to launch the application. Such a decision, a priori, would bring lot of savings as it avoids costly non-functional deployments.
- *Scalability*: When a network is scaled, it is desirable to be able to predict the end-to-end behavior of the new topology by combining the end-to-end behavior of individual

subnets that form this network. For example, if delay dynamics of subnets are known, then the delay dynamics of the network may be computed by combining the individual end-to-end delay dynamics. Thus, eliminating the complexity of characterizing the whole network again. This mechanism is of greater importance while predicting the performance of merged networks before physically doing so. In addition, the end-to-end characteristics may clearly predict the behavior of existing networks with increased bandwidth or increased processing powers. For example, if there are a high percentage of losses due to congestion, a scaling in processing power and/or bandwidth may reduce this problem.

- *Design of simulators and models:* Mappings of end-to-end dynamics to simulate similar conditions create a tremendous opportunity for studying the networks. Once a simulator or a model is built using these dynamics, these prototypes can be subject to rigorous conditions like very high data rates, very large number of flows etc., which may not be advisable on a physical network. The usefulness of models and simulators is very large and [Fl01] lists the immense possibilities of the same.

1.3 Challenges in Approaches to End-to-End Modeling and Characterization

Modeling statistical behaviors of end-to-end packet dynamics over the Internet is not without challenges. Due to the heterogeneity, growing size and time-varying nature of the conditions, which exist in the Internet, it is difficult to capture the end-to-end packet dynamics over the Internet [Pa97, Fl01, Co02]. Queuing theory still serves the purpose of an excellent starting methodology, but an increase in second-order effects over the Internet makes these results impractical. In addition, there is an ongoing debate about traffic dynamics, i.e., traffic models being Poisson, self-similar, etc [Ab02, Ca02, Le94, Pa95, Ho02, Cr97]. Due to these arguments, there is a shift from theoretical models to the combination of the same with measurements over the Internet.

Researchers working in this area use tools such as probe based and passive measurements [Sa03], models based on queuing theory, and simulators. There are case studies where a set of measurements or a set of simulated results or even a set of derived results from models is mapped to the end-to-end dynamics of the Internet [Pa03, Ho04, Ya99, Bo93, Ga03]. But a complete characterization should contain the models of the behavior at different network conditions for various input streams. For example, a study of packet delay should contain its variation with respect to different available bandwidths, packet sizes etc. However, previous studies have not analyzed any such variants. Thus, one cannot predict the behavior of network/subnets under various conditions or even draw common conclusions that would be applicable to any end-to-end path. Absence of such flexibility with models or simulators leads to questions about the accuracy of such studies. Inevitably, we need better models for Internet research [Fl03].

In search of generic models, we approached the problem treating subnets as systems, and characterizing the end-to-end dynamics of subnets under various boundary conditions and then combining the characteristics of different subnets to obtain the end-to-end characterization for the interconnection of subnets. Due to such abstraction, we avoid the details with respect to each and every node in a large topology, as that of Internet. Though our initial concentration was to obtain such characterization for most of the parameters including packet delay, packet loss, packet reordering, etc., it was soon realized that there exists no modeling and characterization work for packet reordering till date, as scant attention has been paid to this phenomenon. Queuing theory approach always concentrated on loss models, queue-occupancy models, queuing delay models, scheduling mechanisms, etc. As we will explain in the following Chapter, packet reordering is more inevitable in future and more proactive approach to this problem is warranted.

This research aims at the development of a theoretical foundation, metrics, techniques and measurements to characterize packet reordering. During this process, the development of methodologies was such that the combination of the characteristics of individual subnets to derive characteristics for their cascade is feasible. As it is a first attempt in defining comprehensive metrics for packet reordering, this work is being actively pursued at IETF [Ja04]. Thus, the majority of this research contains a comprehensive evaluation of packet reordering.

The work is also supplement by a detailed study of existing packet delay models and development of a methodology to validate these models and also combine the delay models of subnets to derive a delay model for their cascade. Using abundant measurements of packet delay over the Internet under various conditions, we identified that inter-packet gap is a very useful parameter in generic modeling of packet delay. This work concludes with identification of delay being Markov process, high correlations between packet delays, impact of inter-packet gap correlations, spectrum of distributions with respect to load, etc.

1.4 Dissertation Outline

As mentioned above, packet reordering and packet delay are two parameters that are characterized end-to-end in this research. In Chapter II, we introduce packet reordering phenomenon and packet delay and their importance. We present a case for the inevitability of reordering in future, its nature and its impact.. As literature on packet reordering characterization is almost non-existent, different aspects of packet reordering are reviewed such as the impact of packet reordering on the transport and application protocols, and multiple studies on the measurement of reordering over the Internet. In this Chapter, we also review the several proposed models for packet delay and various techniques used to reduce the impact of clock skew and synchronization issues in measurements. However, as we will

show, these models are presented considering different network conditions and there exists no consolidation of works in this area.

Chapter III presents the first formal and mathematical representation of packet reordering, followed by a derivation of the reorder metric, Reorder Density (RD). Properties of RD are further derived in this Chapter along with the theorem of convolution that will allow the characterization of reordering of a cascade from the characteristic of individual subnets. Measurements of reordering over the Internet are presented along with the identification of most common reordering occurrences.

Preliminary modeling based on common reordering occurrences over the Internet (as observed in measurements) is presented in Chapter IV. We then present the generic packet reordering models for traffic splitting scenarios with two, followed by multiple parallel links. These models are also verified using emulated topologies.

In Chapter V, Reorder Buffer-occupancy Density metric for packet reordering is further extended to include properties, models for reordering for traffic splitting scenarios using common occurrences over the Internet. A variation of RBD, motivated by IPPM charter of IETF, and known as Loss Orthogonal – RBD is presented.

In Chapter VI, a framework is developed for the evaluation and analysis of packet reordering metrics. Various proposed metrics for packet reordering that were developed in parallel to this work are compared with metrics RD and RBD.

In Chapter VII, we present packet delay characterization based on the analysis of exhaustive Internet measurements. These measurements contain delay data for various sending rates, packet sizes, etc., from two points on Internet to an end-point at Colorado State University. Curve fitting of these data is shown to indicate a spectrum of distributions of delay based on sending rate and packet size. The correlation for each case is studied along

with the correlations among inter-packet gap data. Delay Process is shown to be equivalent to Random-Walk problem with inter-packet gaps depicted as steps, and thus shown to be a Markovian process. The variation of correlation in packet delays with lag is derived and analyzed exploiting the relation with packets gaps. This study includes the identification of inter-packet gap as a crucial parameter in delay models. Lastly, this discussion provides a complete methodology to model packet delay, consolidating previous studies. Chapter VIII concludes the dissertation.

Chapter II

REVIEW OF REORDERING AND DELAY

This research focuses on packet reordering phenomenon that includes theoretical foundation, metrics and modeling, and is also supplemented by the methodology on packet delay modeling. In this chapter, current understanding and available studies on these topics are reviewed.

In Section 2.1, packet-reordering phenomenon is described along with the studies on its prevalence in the current Internet, followed by the identification of the reasons for its occurrence, in subsection 2.1.1, and existing methods to counter its effects on the performance of protocols, namely TCP, in subsection 2.1.2. In subsection 2.1.3, the strategies used to reduce and/or recover from packet reordering are presented. In this subsection, based on the published results on the growing trends of networks speeds and routing architectures, we also show that packet reordering is inevitable in future and reordering reduction techniques may not be useful. In subsection 2.1.4, the need for a proper metric to better understand the reordering phenomenon is discussed, using examples from current measurement techniques.

In Section 2.2, an introduction is provided to packet delay. The significance of delay and jitter and mitigation of its impacts are also discussed in this section. Packet delay components are described in subsection 2.2.1. As argued in Chapter I, the current modeling

techniques include the measurements over the Internet. Therefore in subsection 2.2.2, the challenges associated with measuring packet delay parameter, mainly clock synchronization and clock skew issues are presented, along with the proposed methods to counter these effects. In subsection 2.2.3, various proposed models for packet delays are presented and critiqued. In conclusion, a lack of consolidation of these results is discussed.

2.1 Packet Reordering

Packet reordering in the context of networking refers to the phenomenon in which packets arrive out-of-order or out-of-sequence at a receiver. Usually, applications using these packets need them in order to run successfully, at the receiver. In [Aa04], this phenomenon is also called mis-ordering, considering one of the general meanings of the term reordering is putting things back in order. There exist various types of packet reordering [Ja03]. For example, TCP sender retransmits the packets that are classified as lost, based on duplicate acknowledgements. Hence, these packets are sent out-of-sequence with a purpose. Even though these packets may arrive at the receiver in the order the transmitter sent them, the receiver considers them as out-of-sequence. In this work, we use the terms out-of-sequence and out-of-order interchangeably with packet reordering. Thus, the retransmitted packets that arrive in order based on sender's side may be deemed as reordered. This notion is more useful as we are more often than not interested in the order of packets at a monitoring point, away from the sender and without the knowledge of sender's order. However, in Chapter 3 we will discuss a case where the order of packets at the sender's side can be exploited to diagnose a network.

Studies based on Internet measurements have shown that packet reordering is not a rare phenomenon [Pax97, Be99, La02, Gh04, Pr05]. In [Pr05], authors list few practical observations that are cited often by vendors of routers and customers, for example,

reordering in Juniper M160's OC-192 interfaces. Independent test results presented by LightReading [Li01] show that reordering in routers will not happen, until the line card has 73% load, or 56% of load in worst case, when customer traffic is composed of 40-byte IP packets only. The charts show that reordering occurs when packets are of different sizes too. Also, when different weights were assigned to LBE (Less than Best Effort) and BE (Best Effort) on a platform architecture of M160, reordering appeared at the destination. A second example exhibiting reordering was 10GE cards of Black Diamond switches (BD 6808 and 6816).

Measurements of packet reordering at four monitoring points that are peering points for three DARPA SuperNet partners, which connect to ISPs through OC-48 links, are presented in [Gh04]. Out of the selected 155 flows, 73 flows showed packet reordering. It is concluded that packet reordering should be paid as much attention as packet loss. In addition, it is shown that packet reordering increases with increasing data rates, mainly in terms of packet gaps. Though the tabulated results displayed percentage reordering, the authors attempt to capture reordering in terms of degree of reordering depicting the difference between the place a packet is expected to arrive and its actual arrival. Packets were observed to be as late/early as 55 to 60 places. This work in 2004 was published after our proposed metric "RD", introduced in 2003 at IETF, and authors could have exploited this metric in the place of degree as it conveys similar information with additional feature set.

[La02] observes reordering as a phenomenon that may result in significant application throughput degradation, while leaving little or no trace. They measured and quantified the effect of reordered packets in a backbone link that multiplexes multiple TCP flows on application throughput. Their results indicated that a small percentage of packet reordering, by at least three packet locations in a backbone link causes a significant

degradation in application. The performance of long packet flows are affected the most due to reordering. With 10% reordering, the 400MB file transfer's throughput dropped by 40% percentage whereas the 100KB file transfer throughput dropped by 20%.

The notation of reordering being an uncommon phenomenon is questioned in [Be99], which argues that packet reordering is not caused purely by route pauses or fluttering but also by parallelism with the components of Internet.. This study considers 140 Internet hosts, which were believed to be close to MAE-East, such as server machines of the ISP's peering with MAE-East. The study contained measurements spaced a month apart. As the load increases, they observed that reordering increases. The load on MAE-East in 1997 was such that the amount of reordering for first ten packets was observed only in 33 hosts out of 150. However, in 1998 when the load increased this number increased to 90. However, here MAE-East made no efforts to reduce reordering, using the methods that we list in subsection 2.1.3. Nevertheless, this study clearly indicates that with increasing loads on the Internet, reordering problem becomes more prevalent and also there exist numerous causes for packet reordering.

The above studies show that packet reordering is more prominent in the Internet than it is commonly believed. In the next subsection, the various reasons that cause packet reordering over the Internet are described along with the impact of packet reordering on both TCP and UDP based applications.

2.1.1 Causes of Reordering and its Impacts on Applications

Packet reordering is a common phenomenon over the Internet. The causes for out-of-sequence delivery of packets include, but not limited to the following:

- *Packet striping at layer 2 and 3 links*: Packets arriving to switches and routers are queued if a processor is busy switching the earlier packets. These nodes usually have

multiple processors and multiple queues. When the load increases, packets may follow a scheduling mechanism where packets from same stream could be in different queues or processed by different processors. When an earlier packet of a stream is placed in a longer queue and later packet in a shorter queue, the packets in longer queue may arrive after the packets in shorter queue leading to out-of-order packet arrivals [Be99, Ja03].

- *Retransmissions*: Some transport protocols, e.g., TCP have reliable transfer mechanisms where all packets are reliably transferred to the receiver. When a packet loss occurs during the packet transfer, these protocols assume congestion and send the packet stream at a lower rate. However, the lost packets are retransmitted. Due to retransmissions the sender may purposely send a packet out-of-sequence. Layer 2 retransmissions on wireless links [Be02] is another example.
- *Diffserv Scheduling*: Internet Service Providers (ISPs) offer class of the service to the customers who are ready to pay more. Differentiated Services or Diffserv is one such service implemented in current day routers. In an example scenario, selected streams of privileged customers are allowed at higher rates till a threshold rate beyond which all the customers are treated equally. If a privileged customer flow exceeds these constraints, the non-conformant packets are either dropped or given a lower priority. In the latter case, the lower priority packets will be placed in different queues resulting in out-of-order delivery as in packet striping scenario [Bo03, Ia01].
- *Ad hoc Routing*: Unlike wired networks, ad hoc routing is more common in wireless links. In ad hoc routing, the routes to the destination are changed dynamically. These changes could even occur when a stream transfer has already started between two endpoints. Due to ad hoc paths taken by the packets of such streams, the earlier packets may take longer route compared to the later packets, leading out-of-order delivery of packets at the receiver [Fu02].

- *Load Splitting*: Routers use different schemes of placing packets on outgoing links. For example, all the packets of a stream may be placed on same outgoing link (per stream scheduling) or packets are placed on different outgoing links depending on the traffic on these links or in a round-robin manner (per packet scheduling). In the latter case, the traffic is split among different links to balance the load on links. Sending packets of same stream along different outgoing links from a router, could lead to out-of order arrival at the destination.
- *Route Fluttering*: Rapidly oscillating routes, for example due to incorrect configuration, cause different packets of the same stream to take different routes leading to different delays thereby a possible out-of-order delivery at the receiver [Pax97]. As stated in [Be99], this was considered as one of the main reasons for packet reordering in the past, and since the configuration errors can be minimized, reordering was treated as insignificant.

Irrespective of its cause, packet reordering is a significant impediment to the performance of applications that are based on either TCP or UDP. In TCP, as discussed earlier, packets are retransmitted and as the protocol itself uses sequence numbers, reordering can be clearly observed. UDP does not use sequence numbers, but the protocols that run with underlying UDP, such as RTP/RTCP use sequence numbers. There are no retransmissions in UDP, yet packets reordered due to other reasons still impact the performance of applications. In delay sensitive applications based on UDP, e.g., IP telephony, an out-of-order packet that arrives after the elapse of playback time, theoretically 150 milliseconds, is treated as lost thereby decreasing the perceived quality of voice. In addition, to recover from reordering within the playback time, the out-of-sequence packets are buffered until they can be played back in sequence to the application. Thus reordering can also increase buffer requirements at the receiver as well as processing related latency.

When out-of-order packets are received, TCP perceives it as loss of packets, resulting in following [Be99, Bl02, La02]:

- As the frequency and amount of reordering increase, the number of unnecessary retransmissions increases, and that results in drop in throughput.
- The congestion window becomes very small due to multiple fast retransmissions. Thus, TCP has a problem raising the window size, resulting in decreased bandwidth utilization.
- Due to multiple retransmissions, TCP does not sample the round trip time (RTT) frequently, thus degrading the estimate of RTT.
- Performance of receiver also suffers, because whenever the receiver receives out-of-order packets, it has to buffer all the out-of-order packets and they need to be sorted as well.
- Detection of loss of packets is delayed because of out-of-order delivery, due to which retransmission request for a lost packet is sent only when TCP times out.

Reverse-path reordering, which is reordering of acknowledgements, results in the loss of TCP's self-clocking property, i.e., property of TCP that it only sends packets when another packet is acknowledged doesn't remain valid, leading to bursts of transmissions and a possibility of increased congestion [Be99].

2.1.2 Mitigation of Impact of Reordering

In [La02], it is observed that a long flow of 400MB when ftp'ed has more impact on the degradation of throughput (40%) compared to 100KB file where the throughput decreased by only 20%. As ftp uses TCP as underlying mechanism, it is clear that reordering could have cumulative effect on the performance as the sender falsely reacts to reordering as a pure congestion event. Approaches for mitigating the impact of out-of-order packet

delivery on TCP performance include adjusting '*tcp_reordering*' parameter in Linux implementation also known as '*dupthresh*', i.e., the number of duplicate acknowledgements to be allowed before classifying a following non-acknowledged packet as lost. Such an adjustment will reduce the amount of reordering due to retransmissions. However, delayed reaction to true congestion of network, i.e, having a longer *tcp_reordering*, will not resolve the issue as the sender may add to the congestion on the network.

[Bl02] evaluates different approaches to better tune this parameter. Adjustment of the duplicate acknowledgement threshold used to trigger fast retransmit and the insertion of a small delay before retransmitting the "lost" packet were found to be effective means of reducing the frequency of spurious retransmissions under the variety of reordering conditions. This work evaluates different methods to adjust the *tcp_reordering* (or *dupthresh*), such as: (i) exponentially weighted moving average (EWMA) of the length of the perceived reordering length. Length 'N' of the perceived reordering event is defined as the detection of N duplicate Acknowledgements. The average length in EWMA is update based on most recent length and the average length over time (ii) constant increase of dupthresh by some value K every time a spurious retransmit is detected (iii) increasing threshold based on length of reorder event. This scheme is similar to (i), except history and current values have equal weightage (iv) using a combination of *dupthresh* value and timer to wait before retransmission. In all these studies the authors used a two-path load balancing with random swaps in queues and acknowledged that there are no empirical models for packet reordering. Also, the usage of length of reorder event cannot be justified, as there are no criteria for entangled reordering.

In addition, there are ongoing efforts to improve the robustness of TCP to such non-congestion events, reordering being an example [Bh05]. As in previous work, the receipt of three duplicate acknowledgements as an implicit indication of congestion in the network,

more specifically loss of packets, is identified as incorrect. [Bh05] proposed Non-congestion Robustness (NCR) for TCP. This proposal presents two important changes. First, the trigger for retransmitting a segment is changed from three duplicate acknowledgements to a congestion window's worth of duplicate acknowledgements. This allows recovery from reordering within the segment (segment reordering). Second, TCP-NCR decouples initial congestion control decisions from retransmission decisions. They provide two alternatives- (i) *careful limited transmit*: reducing the sending rate at approximately the same time implementation reduce the congestion window, while at the same time withholding a retransmission for approximately one round-trip-time (RTT) (ii) *aggressive limited transmit*: maintaining the sending rate in the face of duplicate acknowledgements until the TCP concludes a segment is lost and needs to be retransmitted.

In {Bo03], a new version of TCP to make TCP more robust to reordering is suggested. The proposed TCP variant, TCP for Persistent Packet Reordering or TCP-PR, does not rely on duplicate acknowledgements to detect a packet loss. Instead, timers are maintained to keep track of how long ago a packet was transmitted. In case the corresponding acknowledgement has not yet arrived and the elapsed time since the packet was sent is larger than a given threshold, the packet is assumed lost. Because TCP-PR does not rely on duplicate acknowledgements, packet reordering (including reverse path reordering) has no effect on its performance. Simulations verify this claim and when there was no reordering the authors' claim that TCP-PR is as good as TCP-SACK and shares resources fairly.

The above methods reduce the impact of reordering on TCP performance, in other words they make TCP robust with respect to reordering but cannot reduce or eliminate segment reordering. In addition, the non-robust nature of TCP to reordering, i.e., adjustment of *tcp_reordering* parameter makes the traffic vulnerable to security attacks. In [Aa04],

authors evaluate damage that difficult-to-detect attackers can cause. One of these attacks is on closed-loop flows such as TCP, which is called Jellyfish attack and is protocol complaint. The goal of a Jellyfish node is to reduce goodput of all traversing flows to near zero while dropping zero or a small fraction of packets. One of the three mechanisms that a Jellyfish node employs is that is packet-reordering attack. In this attack, Jellyfish nodes deliver all packets, yet after placing them in a reordering buffer rather than a FIFO buffer, thus creating a loss in clocking mechanism and reduction in sending rates.

2.1.3 Reduction of Reordering in Routers and Recovery at Endpoints

Traditionally, recovery from effects of reordering has been left to the application (in case of UDP) and the transport layer (in case of TCP). To recover from reordering, the out-of-sequence packets are buffered until they can be provided in sequence to the application. Thus an increase in out-of-order delivery by the network consumes more resources at the end-host, and also affects the end-to-end performance of the applications. Consequently, there is an effort to address this issue at intermediate nodes, at IP level. Many existing routers attempt to eliminate the reordering caused by the scheduling schemes within these nodes. This involves either a) input sorting, i.e., identifying the individual streams and forwarding the packets of same stream to the same queue thus preventing reordering, or b) output re-sequencing, i.e., buffering packets at the output of the router to ensure that the packets belonging to the same stream are released in sequence [Li02]. For example, the network processors from vendors such as IBM, Motorola, Vitesse, TI and Intel, have built-in hardware to track flows [e204]. However, there exist intermediate routers that fail to achieve the same. While input/output reordering approaches reduce the reordering that occurs inside a router, they cannot eliminate reordering due to multiple paths. Furthermore, the complexity of these approaches will increase significantly as the number of parallel flows in a pipe increases (due to the need to keep information on a large number of parallel flows), and as

the ratio of packet time to routing latency decreases. In [Xi03], it is shown that the delay in sequencing the packets back in order (as in output buffering) is proportional to log (C_s/U) where C_s is the capacity of the link and U the packet size. The buffer size required to put the packets back in order is also shown to increase dramatically with link speed. With the increase in number of flows, keeping track of them becomes difficult too. Moreover, the number of table entries in routers is growing exponentially [Ru01]. The net result is a higher end-to-end delay to packet time ratio, leading to more load splits and higher packet reordering [Ba04]. Increase in latency required at the intermediate switches and routers to reorder the packets add to the end-to-end latency and the round-trip delay, thereby affecting performance as well.

Next generations of networks therefore have to proactively deal with the problem of out-of-sequence packets. However, scant attention has been paid so far towards understanding the nature of reordering that occurs in networks. Information on the nature of reordering that occurs in a given end-to-end path may be used to enhance the ways that the protocols deal with this problem. There exist no modeling and characterization of reordering in packet sequences, which can provide insight into this problem, can lead to the development of scalable techniques to deal with reordering, and perhaps even to development of tools that diagnose network problems.

2.1.4 Metrics for Reordering

The percentage of out-of-order packets is often used as a metric for characterizing reordering. However, this metric is vague and lacks in detail. Further, there is no uniform definition of the degree of reordering of an arrived packet. For example, consider the two packet sequences (1, 3, 4, 2, 5) and (1, 4, 3, 2, 5). It is possible to interpret the reordering of packets in these sequences differently. For example,

- Packets 2, 3 and 4 are out-of-order in both cases.
- Only packet 2 is out-of-order in the first sequence, while packets 2 and 3 are out-of-order in the second.
- Packets 3 and 4 are out-of-order in both the sequences.
- Packets 2, 3 and 4 are out-of-order in the first sequence, while packets 4 and 2 are out-of-order in the second sequence.

In essence, the percentage of out-of-order packets as a metric of reordering is subject to interpretation and cannot capture the reordering unambiguously and hence, accurately.

Many other metrics are proposed in parallel to this research to achieve a representation for packet reordering [Mo05]. They include *Reordered Packet Ratio*, *Reordering Extent, Reordering Late-Time Offset, Reordering Byte Offset, Reordering Free Runs and n-Reordering*. Except n-reordering, these metrics are based on concepts of lateness and reordering discontinuity. During the arrival of packets at the point of measurement, a packet with successive sequence number is expected for successive arrival. If a packet greater than this sequence number arrives, then the new expected is a unit increment of the sequence number of such arrival. Moreover, this condition indicates a sequence discontinuity, either because of packet loss or reordering. Reordered packets must arrive for the sequence discontinuity to be defined as a reordering discontinuity. If a packet with a sequence number less than the expected sequence number arrives, then the arrived packet is treated as late and reordered. *Reordered Packet Ratio* corresponds to the fraction of packets that are late compared to the received number of packets. *Reordering Extent* is the distribution of the late packets with respect to their extents, which correspond to the number of packet from the reorder discontinuity to the late packet. *Reordering Late-Time Offset* is similar to Reordering extent except that the extents are in the units of time for which other packets are buffered.

However, this buffer estimation is not always correct with extent computation provided. For example, in the sequence (1, 2, 3, 4, 7, 5, 6, 8), packet with sequence number 6 has an extent of 2 but it does not require buffer for the arrival of packet 5 leading to overestimation of buffer offset. *Reordering Byte Offset* is the offset of buffer requirements in bytes. *Reordering Free Runs* is the count of packets between consecutive late packets in a sequence. *n-Reordering* on the other hand is a subset of late packets, which include single late packets (i.e., preceded and followed by in-order packets) or first packet of the bunch of late packets that have no in-order packets within them. For example, in the sequence (1, 2, 3, 4, 7, 5, 6, 8), only packet 5 is considered as late and not packet 6. All these metrics account for only lateness in the reordering involved in a packet sequence. In a sequence such as (1, 200, 2, 3, 4, ...,199), lateness is in all 198 packets, but one can obviously conclude that packet 200 could have come earlier and others as expected. Thus both lateness and earliness are important in understanding the reordering phenomenon.

The algorithms associated with proposed metrics have multiple additional challenges that include spatial and time complexities, real-time evaluation, robustness, etc. It is not feasible to classify a packet arriving early as reordered unless the preceding packets are not lost and similarly a late arrival is not reordered unless there are no earlier copies of the same packet. Thus, reorder measurements consist of buffering and latencies due to the detection of lost and duplicate packets, thereby affecting the complexity of the measurement algorithms. In addition, a TCP sequence rollover may falsely indicate certain packets as reordered in a TCP stream, unless the algorithms are robust to such peculiarities. The definitions of reordering, usages and complexities of proposed metrics vary widely. Thus, a framework of attributes should be considered to evaluate a reorder metric. The essential attributes such as not treating a duplicate packet as a reordered packet and not treating the in-order packets following a lost packet as reordered, are vital for the validity of metrics. A

framework for metrics for the Internet [Pax98] states: "The metrics must aid users and providers in understanding the performance they experience or provide." A metric for packet reordering has to go beyond reporting a mere measure that varies with reordering, to a measure that is useful to characterize reordering. Desirable attributes would make a metric more relevant to the Internet applications and protocols. These include but are not limited to simplicity, computation complexity, memory requirement for computation, and robustness of the measures. A comprehensive comparison of existing and proposed reorder metrics with respect to desirable and essential attributes is presented in Chapter VI, where all the metrics are compared.

Observing the lack of detailed study of reordering phenomenon but its much known prevalence through Internet measurements, as discussed in Section 2.1, we propose a mathematical representation for packet reordering, followed by a metric based on this representation, namely Reorder Density (RD), in Chapter III. RD has many useful properties that allow us to interpret the possible causes and affects of reordering, and better represent the phenomenon itself. It has an in-built system-level representation, which is useful for the study of reordering in cascade of networks. Since it is a derived metric from a robust mathematical representation of the order of sequence numbers, it allows us to derive models of packet reordering for any of the causes as discussed earlier. These models are presented in Chapter IV.

Reorder Buffer-occupancy Density (RBD) is another packet reordering metric that emulates the buffer-occupancy at the receiver to recover from reordering [Ba02]. This research work further extends the metric to derive its properties and models for reordering over the Internet, in Chapter V. As stated earlier, other proposed metrics are compared to RD and RBD in Chapter VI.

2.2 Packet Delay

End-to-end delay of a packet refers to the time taken by a packet to transit through the network in reaching the destination. Precisely, it is the difference in the time when the last bit of a packet exits the transmitter and last bit of the packet reaches the receiver. It is an important characteristic that not only affects the application performance, but also provides information about the state of a network. Increase in packet delay affects the TCP protocol significantly. TCP relies on synchronized acknowledgements to detect congestion in network and a very high delay could lead to an action (such as sending data at reduced rate to further avoid congestion) ineffective. In fact researchers have questioned the validity of the usage of TCP for network such as Inter-Planetary where delays are in minutes. Real time applications based on UDP are also concerned with the average one-way delay and jitter [Zh01]. For example, G.114 standard of ITU (International Telecommunication Union) recommends 150 milliseconds as the maximum one-way delay for voice-traffic, as anything above this value degrades the perception of the voice quality. ISPs monitor delay characteristics to make routing decisions, e.g., higher delay in an end-to-end path is attributed to congestion and the routes are altered. Second order parameters of delay, such as jitter that is a difference between delay of two consecutive packets, have gained significant importance. A higher jitter indicates more buffering leading to a decreased perception of quality of voice or video. Large deviations in delay values may also lead to other undesirable effects such as packet reordering. Packet loss is also speculated as correlated to delay values [Mo98].

Further more, in [Mu94], the low frequency components of delay are shown as indicators of congestion. Examples include the presence of slow oscillation components in smoothed network delay, increase in conditional expected loss and conditional our-of-sequence acknowledgements as a function of various statistics of delay, and change in delay

distribution parameters as a function of load, while the distribution itself remains the same, etc.

Issues with delay are handled at different levels. For example, vendors of routers provide lad-balancing options where packets are sent over various outgoing links to decrease the impact of congestion happening in one single network. In addition, there exist multiple queues and multiple processors within the router to reduce the time spent by a packet within the router. However, as discussed earlier, such solutions will lead to more and more reordering among the packets. On the other hand delay is also reduced within the endpoints by providing the IP stack on the network cards (that are still not cost effective for single users but highly used in enterprise environment). By moving the stack to the card, number of memory copy operations is significantly decreased.

Jitter buffer management is a highly researched topic due to its impact on performance. Jitter buffer is the buffer used mainly by voice/video applications where packets are played out by the application at a constant rate. Since the jitter value is known only after the packets arrive, it is important to provide a better estimate of this buffer. RFC 3550 presents the usage of jitter buffer and its estimation for RTP based applications.

2.2.1 Components of Delay

A packet could be delayed due to multiple reasons. Packet delay consists of deterministic and stochastic components. Deterministic components of packet delay include (i) *transmission delay*: the time to transmit an entire packet, from first bit to last bit over a communication link. (It is also referred to as serialization delay), and (ii) *propagation delay*: the time to propagate a bit through the communication link, which is determined by travel time of electromagnetic wave through a physical channel of communication path. These deterministic components depend on parameters such as packet lengths, capacities of the

links and physical distances. For example, if the packet length is large then the amount of time to serialize the packet on to a link of given capacity is higher compared to that of a smaller packet length. With increasing capacity of the links, given the hardware matches this capacity, the amount of serialization delay decreases and also the propagation delay. Unless the link is Trans-Atlantic, satellite or inter-planetary, the propagation delay is negligible.

Stochastic components of delay include (i) *queuing delay*: the waiting time of packets in a buffer in layer 2 switches or layer 3 routers before transmission, and (ii) *processing delays*: the time needed to process a packet at each node. In intermediate nodes, queuing delays could vary due to the presence of packets (cross-traffic) from the other streams. With increasing cross-traffic there will be more packets in the intermediate queues waiting to be processed. Queuing delay also varies with the arrival process. Processing delay, the time duration to process a packet before forwarding, is affected by factors such as the heterogeneity among the hardware of the nodes, variable lookup times in the routers, etc. If a router cache does not contain the prefix to which a packet is destined, its processing time will effectively be larger compared to the packets with prefixes in cache. Similar delays can be observed in level 2 switches when destination MAC/IP mapping is not present in cache of a switch. However, such effects are usually limited to the first packet and first frame respectively.

Delays of packets in the same stream differ not only due to stochastic nature described above but also due to scheduling strategies, e.g., load balancing.

2.2.2 Delay Measurement

As discussed in Section 2.2, delay is primarily monitored by ISPs and it is also useful for researchers to characterize and better understand the behavior of networks, making its measurement technique, an integral part of its study. Packet delay is measured in time units,

and it is mostly in milliseconds range for packets over the Internet. However, it could fall to microseconds or even lower, in near future as Giga and Tera networks technology reaches most of the Internet core and edges. However, there are a couple of challenges associated with measuring packet delay [Pa98]:

- *Clock Synchronization between sender and receiver*: Sender and receiver could have different clock readings even when they are physically located in the same time zone. NTP (Network Time Protocol) [Mi03] has been used for a long time to synchronize clocks of systems to high precision atomic clocks located in various parts of the world. However, this operation occurs over network, and due to varied network delays the clocks could still remain unsynchronized. Other method that is becoming more popular is to use GPS receivers at the nodes to get accurate time stamping [Pa03]. However, this method is not cost effective as the number of nodes involved in the measurement process increases.
- *Clock Skew*: Assuming that the sender and receiver clocks are synchronized, we face another problem, i.e., every clock has its own rate at which it operates. Though, the difference in the rates is very minute, after few seconds this difference tends to create a wide variation among clocks that were synchronized earlier. With current technology, this variation is in microseconds for tens of seconds. Thus, if we update and synchronize the clocks after every 10 seconds, the delay measurements over the Internet will have errors in microseconds. These errors are acceptable when actual delays over networks are in milliseconds. In addition, there are numerous methods proposed to correct clock-skew problem [Zh02, Ho00, Mo99, Pa98]. The usage of these tools could be inevitable as we move from milliseconds to microseconds or lower ranges.

It is agreed by researchers that the clock skew has a growing linear trend with time. However, if the clock is synchronized to timeserver, the slope of this linear trend is reset. In

[Pa98], the skew correction algorithm uses median line fitting technique. The median of the delay observations is used to estimate the slope of the skew trend. When the data is highly variable then this technique provides a poor estimate of the slope. The linear regression algorithm is discussed in [Pa98] and [Mo99]. The algorithm performs poorly as the delay measurements could be highly variable, whereas linear regression is for data that are normal.

[Zh02] propose convex-hull approach to detect the clock skew trend. They provide algorithms, which are linear in the number of measurements points for the case with no clock resets. For more challenging case with clock reset, they provide algorithms to identify these points and then apply the earlier techniques. Technique in [Ho00] is proposed for audio applications based on RTP. They exploit the addition of offset in each arriving RTP packet and use the trend to detect skew. However, if the delay variations are low, these offsets could falsely show the variations as skew instead.

[Gu05] suggests another method to estimate delay to overcome synchronization problem. They propose estimation of deterministic and stochastic parts of the delay independently. To determine the deterministic delay, the authors use the sum of single hop one-way delay measurements along a cyclic path that effectively eliminates synchronization problem. Higher the cyclic paths the better is the estimation. However, this work assumes that there are possible cyclic paths in the Internet. Due to policies such as no-valley constraint, such topologies are not feasible. Moreover the clock-drift is assumed to be negligible to attain the stochastic part of the delay value.

2.2.3 Delay Models

There are multiple studies that have been performed using the measurements over the Internet, to explain the end-to-end delay dynamics. We have selected few research works that have drawn varied conclusions from such measurements. [Bo93] studies the delay

variations over the Internet by studying round-trip times. For lower data rates, the delay value has an increment of exponential random value with zero mean and small variance. Thus there is an approximate linear relation between delay values. At higher rates, the delay values have high correlation. However, there is no concrete analysis with respect to the packet size, and 20ms gap used for inter-packet gap is too large to attain any significant lower data rates. The limitation could have come from the link speeds that were used. In addition, the correlations between the delays are not analyzed either.

Delay distributions are characterized as Gamma-like distributions with heavy tails of sub- exponential in [Bo02]. In this work, with 84% of the measurements, which are parts of the RIPE NCC measurements between two points over a period of one day, the results seem to be generalized from a set of measurements. However, the network conditions for the measurement period are missing and any such study may not be applied to complete characterization of end-to-end packet dynamics.

End-to-end delay dynamics of the Internet are also modeled using system identification [Oh01], i.e., it models end-to-end as black-box as a SISO (Single-Input and Single-Output) system. In this work, the ARX (Auto-Regressive eXogenous) model is used and its coefficients are determined using system identification. Due to the varying network conditions, we actually may have SIMO (Single-Input and Multiple-Output).

There are also studies where delay is modeled as time varying exponential [Hu00]. This study states that the one-way transit time is best modeled by a composition of states, where each state is modeled as a shifted exponential distribution with varying paramters. To better explain how long a network stays in a given state a new parameter known as 'sojourn' time is introduced. Based on this approach, a synthetic delay generation method is proposed. However, the generation scheme uses the parameters that are computed for the network

during modeling. Thus, it is specific to the network and also its conditions. Again, the network conditions are not evaluated comprehensively.

[Pa03] makes a very interesting observation based on measurements over a Sprint IP backbone links. They conclude that most of the delay on single routers for OC-3 and OC-12 link speeds is due to processing time and transmission of packet across the switch fabric. However, the queuing delay has heavy tail Weibull distribution. These observations, as authors claim are more specific to Sprint backbone networks.

Study in [Ho01] models the stochastic part of the delay using end-to-end delay measurements over a fixed path obtained from RIPE NCC, using IP probe-packets. Along with the delay due to interference of the Internet traffic, identified as ON/OFF renewal process, these packets experience a random processing delay due to scheduling and conversing of IP packets to various lower layer technologies in the routers on the fixed path. Though the distributions were heavy tailed, the authors state that neither a Pareto nor Weibull law provided sufficient accurate fits.

In time-series approach, the model for delays is independent of any recent activity in the network [Ya04], and the present delay value is considered to be independent of the past. However, this assumption is not always true. [Mo98], using measurements over the Internet, shows that delay values are highly correlated and a given delay value of a packet can impact the delays of almost 40 successive packets. Furthermore, delay distributions have been observed to be gamma-like with heavy tail or with Gaussian or triangular lobe [Ho01].

Thus, a distribution for delay may not always result in a complete model, as there exist correlations between the values and they should be an integral part of delay characteristics. Network emulators like NIST Net [Ca03] do use such correlation coefficients to generate delay values for data streams but there is no acceptable criterion.

With such variation in observations, it is vital to understand and develop a generic model that fits all the above results yet can be used for any subnet and or their cascades.

Emulators have an additional problem of delay assignment For example; dummynet emulator [Ri97] transmits packets back to back in 10 milliseconds bursts and remains silent afterwards even if arrival time of packets were equidistant. This artificial delay jitter is due to the 10-millisecond granularity in operating systems inherited from FreeBSD. This is also true for NIST Net on Linux boxes. Under FreeBSD device drivers or other kernel modules such as dummynet can attach themselves to the system timer interrupt. Under ideal conditions, this should provide a timer callback to the kernel module for every timer tick. In reality, however, there exists a locking mechanism called privilege levels. These levels have a precedence structure. Thus, it is possible that the higher privilege level can block the NIC access, leading to more jitter in delay generation. There are hardware based delay emulations such as the ones based on real-time operating systems that are less cost-effective. Recently, authors in [Ha05] suggested a hybrid approach, where in hardware and software is used together to attain better delay jitter control and delays are introduced at frame-level on NIC. In addition, this approach includes Real Time Application Interface with high priority to avoid blocking of NIC.

In the last decade, delay Jitter, a difference in delay values of consecutive packets has gained significant importance due to real-time application such as IP Telephony. Delay jitter as mentioned in Section 2.1 has a detrimental effect on the perceived quality of information. Applications such as telephony use jitter buffer (measured in time units) to recover from delay jitter before playback. Lack of proper characterization and modeling for delay indirectly impacts the jitter buffer estimations. The estimation of delay jitter suffers significantly. There are several suggested models to characterize delay jitter [Gu91][Ma94][Fu95][Ok05].

In Chapter VII, we develop a generic characterization based on comprehensive measurements for various packet sizes at various rates with respect to available bandwidth. In addition, we identify that the study of inter-packet gaps, which can safely be used to model Internet delay, is more promising. The correlations among the inter-packet gaps are negligible when there are correlations between delay values. Due to the relation between these parameters, we successfully model packet delay.

Chapter III

REPRESENTATION OF REORDERING AND REORDER DENSITY

The definition of commonly used metric for packet reordering, i.e., percentage of out-of-order packets, is ambiguous, does not represent the extents and does not differentiate earliness and lateness of reordering, in a sequence. On contrary, variation in extent values and directions of displacements of packets affect the performance of an application significantly. Consider two sequences: (1, 3, 4... 99, 100, 2) and (1, 3, 2, 4, 5.... 99, 100). Assuming late packets as reordered, with percentage metric, both the sequences contain 1% reordering. Now, the arrival of packet '2' in first sequence may be rendered useless in a real-time application, whereas the same packet in second sequence could be recovered with a buffer size of one packet. Hence, percentage reordering cannot capture packet reordering in a useful manner. With reordering becoming more inevitable there is a need for understanding of packet reordering phenomenon in a better manner.

In this chapter we develop a methodology to accurately represent packet reordering, which in turn help capture and understand reordering phenomenon. In Section 3.1, representation of packet reordering is presented along with the development of the concepts reorder event, reorder set, principle of conservation of displacement, reorder segment,

minimal-reorder segment, primary event, secondary event, and basic patterns. In Section 3.2, a packet reorder metric Reorder Density is derived. Reorder Response of a network is discussed in Section 3.3, along with theorem of convolution that allows the computation of reorder response of a cascade of subnets from the reorder responses of individual subnets, under stationary operating conditions.

In Section 3.4, we discuss the challenges of computing reordering at the receiver and how RD metric, in two different approaches, overcomes these problems by using a threshold on packet displacements. Section 3.5 presents the reorder measurements over the global Internet and we also study the patterns of reordering that commonly occur over the Internet. Concluding remarks for this chapter are provided in Section 3.6.

3.1 Representation of Reordering

Without loss of generality, consider a sequence of packets (1,2...N) transmitted over a network. A *receive_index* (1,2...) is assigned to each packet as it arrives at the destination. Lost and duplicate packets are not assigned a *receive_index*. First consider the case in which no losses or duplications of packets occur in the network. If the *receive_index* assigned to packet m is $(m + d_m)$, with $d_m \neq 0$, we say that a reorder event has occurred, and this event is denoted by $r(m, d_m)$. In the absence of reordering, the sequence number of the packet and the *receive_index* are equal, i.e., $d_m = 0$ for each packet. A packet is late if $d_m > 0$, and early if $d_m < 0$. Thus, packet reordering of a sequence of packets is completely represented by the union of reorder events, R, referred to as the reorder set:

$$R = \bigcup_m \{r(m, d_m) \mid d_m \neq 0\} \tag{3.1}$$

If there is no reordering in a packet sequence then R = {ϕ}. Conventionally, we represent elements of R with increasing order of m.

Table 3.1 (a) – (c) show a few examples of arrived sequences (sequence number), assigned *receive_index* and displacement, as well as the corresponding reorder sets. The three cases presented include: a) a sequence without packet losses or duplications, b) a sequence with packet loss, and c) a sequence with packet duplication. In case (a), packets 4, 5, 6, 7, and 8 are out of their places and have displacements equal to 2, -1, -1, 1, -1 respectively. In case (b), packet 3, 5, 6, 8 and 9 are out of their original places with respective displacements of 2, -1, 2, 1 and –3. However, note that packet 4 is lost and therefore *receive_index* value is never equal to 4. In addition, even if the lost packet goes out of order, due to lack of such information, the amount of reordering is purely based on the received sequence. In case (c), packet 3 is duplicated and the first arrival of packet 3 is considered for the computation of R, as any application would use this packet instead of the second arrival. The second arrival of packet 3 is not assigned any *receive_index* value. In

Table 3.1. Examples of reordered sequences with corresponding Reorder Sets 'R' (a) No packet losses (b) Packet with 4 is lost (c) Packet 3 is duplicated

Arrived Sequence	1	2	3	5	6	4	8	9	10	7
receive_index	1	2	3	4	5	6	7	8	9	10
Displacement	0	0	0	-1	-1	2	-1	-1	-1	3

(a) R = {(4, 2), (5, -1), (6, -1), (7,1), (8, -1)}

Arrived Sequence	1	2	5	3	9	7	6	8	10	11
receive_index	1	2	3	5	6	7	8	9	10	11
Displacement	0	0	-2	2	-3	0	2	1	0	0

(b) R = {(3,2), (5, -2), (6,2), (8,1), (9,-3)}

Arrived Sequence	1	2	5	3	4	6	3	8	7	9	10
receive_index	1	2	3	4	5	6	-	7	8	9	10
Displacement	0	0	-2	1	1	0	-	-1	1	0	0

(c) R = {(3,1), (4,1), (5, -2), (7,1), (8, -1)}

fact the packet does not have any displacement either.

Consider the reorder set 'R' for case (a), the sum of all displacements is equal to (2) + (-1) + (-1) + (1) + (-1) = 0. Thus, the mathematical representation of reordering has a principle of conservation of displacement. Below, we prove this important observation:

Lemma 1: In the absence of losses and duplicates, the sum of displacements of all packets in a sequence is equal to zero, i.e. $\sum d_i = 0$.

Proof: Consider a packet sequence of size N: (1,2 ...N). If d_i denotes the displacement of i^{th} packet due to reordering, then the new positions of the packets 1,.. N are:

$(1+d_1),\ (2+d_2),\ ..,\ (i+d_i)..,\ (N+d_N)$

Since the number of positions at the sender and receiver are equal, the sum of *receive_index* values is the sum of integers from 1 to N. Therefore we have,

$$\sum_{i=1}^{N} i = \sum_{i=1}^{N} (i+d_i) \Rightarrow \sum_{i=1}^{N} d_i$$ (Q.E.D.)

Now, consider the sum of displacements for cases where packets may be lost or duplicated in transit, e.g., case (b) and case (c) for Table 3.1. The displacements add up to zero in both cases. This is an extension to *Lemma 1*, stated and proved below:

Lemma 2: When lost and duplicate packets are detected and *receive_index* is assigned as specified, Lemma 1 holds true, i.e., $\sum d_i = 0$.

Proof: Consider the case where packet 'e' is lost and 'j' is duplicated. Since we ignore the duplicate copy of j, no index is assigned to the duplicate of that packet. As the index 'e' is

reserved for the lost packet, it is not used in the (N-1) receive indices. Thus, the sum of (N-1) *receive_index* values assigned at the receiver is

$$\sum_{i=1, i\neq e}^{N} (i + d_i) = \sum_{i=1}^{N} (i) - e \Rightarrow \sum d_i = 0 \quad (Q.E.D)$$

Consider the packet sequence (1, 3, 4, 2, 5, 8, 6, 7), where packet 2 is late by two positions and packet 8 is early by 2 positions. Assume a load-balancing scenario where packet 2 might have taken an alternate path with almost three times the delay of primary path. Due to lateness of 2, both 3 and 4 are effectively early by one position. Assuming additional alternative faster path with twice the speed of primary path that packet 8 takes, both 6 and 7 are effectively late by one position. Thus, for $d_m > 0$ (< 0), the next d_m (previous d_m) packets are effectively displaced by −1 (+1) positions respectively. In this case, the packet with sequence number 'm' is said to be associated with a primary event or p-event, while the affected d_m packets have undergone secondary events (s-events). The p-event r (m, d_m) results in following set of s-event(s):

$$\varsigma = \{r(m+j, -1),\ 1 \leq j \leq d_m\} \qquad \text{for } d_m > 0 \qquad (3.2)$$

$$\varsigma = \{r(m+j, 1),\ d_m \leq j \leq -1\} \qquad \text{for } d_m < 0 \qquad (3.3)$$

In general, the effects of reordering of packets in a sequence are localized. Therefore, one can envisage partitioning the sequence of packets arriving at the receiver into reordered-segments (RS), such that the reordering is localized within reordered-segments, i.e., reordering event(s) due to packets within RS are not propagated beyond it. If a p-event belongs to a particular RS, then all its associated s-events also have to belong to the same RS. The definition does not preclude in-order packets before or after the event(s) from

belonging to RS. RS also satisfies a principle of conservation of displacements of packets in a segment. For example, in the sequence (1, 3, 4, 5, 2, 6, 7, 9, 10, 8, 11, 12, 13), the possible RS' are (1, 3, 4, 5, 2), and (6, 7, 9, 10, 8, 11, 12, 13), where p-events associated with packet 2 and 10 have their s-events in their respective reorder-segments respectively and sum of displacements of packets is zero. Note the sequence could be a single RS in itself, without partitioning. Note, a RS could also be partitioned further to obtain more RS'. For example, the above RS: (1, 3, 4, 5, 2) can be partitioned into (1) and (3, 4, 5, 2) that also satisfy the requirements for a RS.

A reordered-segment that cannot be partitioned further into two or more reordered segments is defined as a minimal-reordered-segment (MRS). If there is a single p-event in a MRS then it is called an independent reordering (IR) event. Measurements over the Internet, provided in Section 3.5, show that majority of the MRS' associated with the Internet contain two p-events, and MRS' with more than two p-events are relatively rare. In the case of two p-events within an MRS, one event may be embedded in the other, or the two events may overlap with each other. The former is known as an embedded reordering (ER), and the latter as an overlapped reordering (OR). The basic patterns are illustrates in Fig. 3.1. The straight line corresponds to the sequence of numbers and the bi-directional arrows correspond to the p-events. The arrows are bi-directional as a p-event could be either lateness based on earliness based.

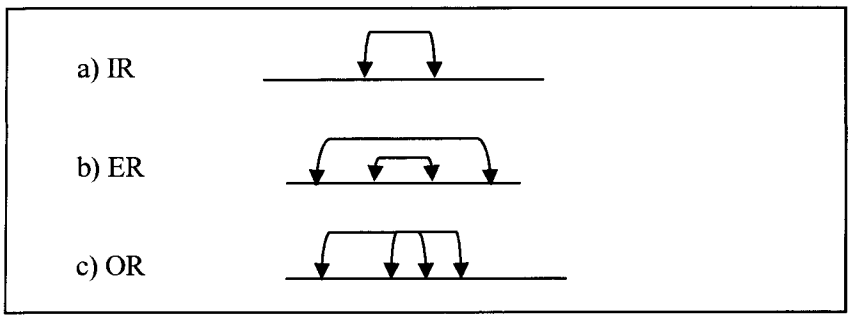


Fig. 3.1 Illustrations of basic patterns, (a) IR (b) ER and (c) OR, arrows represent the movements of p-event packets. Bi-directional represents that p-events could be a late event or an early event.

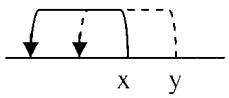
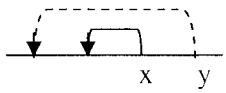
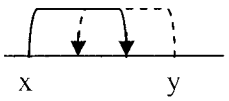
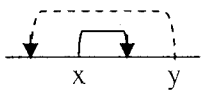
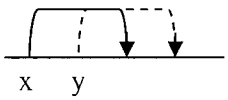
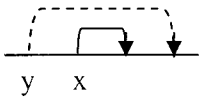
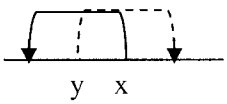
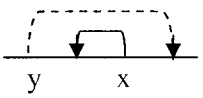
IR, OR and ER are together called basic patterns of reordering. In an IR, p-event is associated with earliness or lateness, but for ER and OR patterns, the p-events that combine may have different signs of displacements. Consider two p-events given by r(x, d_x) and r(y, d_y) in MRS, x < y in overlap cases expect for the case where lateness overlaps with earliness. The possible combinations in OR and ER patterns are listed in Table 3.2. Conventionally, for ER, r(y, d_y) indicates the enclosing p-event, whereas for OR it is the second p-event in MRS. To differentiate between these events, a dashed line is used for r(y, d_y).

Though the concept of basic patterns is introduced to discuss reordering over current Internet, we feel that this concept will also be useful in approximating complex reordering scenarios as we show in Chapter IV.

In this Section, we represented reordering completely, and have shown that the total displacement is conserved with or without losses, in any reordered sequence. We also introduced the concepts of primary event (p-event), secondary event (s-event), reorder segments (RS), minimal-reorder segments (MRS), and basic patterns (IR, OR and ER). In

the next Section, the mathematical notation of reordering is exploited to derive a comprehensive and very useful packet reorder metric, Reorder Density (RD).

Table 3.2. Different combinations of p-events in basic overlapped and embedded reordering patterns

Overlapped r(x, d_x) and r(y, d_y)	Embedded r(x, d_x) and r(y, d_y)
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: $d_x < 0$, $d_y > 0$ 

3.2 Reorder Density (RD)

Reorder Histogram (RH) is defined as the discrete distribution of the frequency of packets with respect to their displacements, i.e., the lateness and earliness from the original position. Let $S[k]$ denote the set of reorder events in R with displacement equal to k, i.e.,

$$S[k] = \{r(m, d_m) \in R \mid d_m = k\} \tag{3.4}$$

Thus, RH[k] is defined as size of $S[k]$

RD is normalized form of RH with respect to the total number of received packets (N'). Note that N' does not include duplicates or lost packets. In fact, N' is also the summation of RH components over all k. corresponds to the packets for which receive index is the same as the sequence number. RD is given as:

$$\left\{\begin{array}{l} RD[k] = |S[k]|/N' \text{ for } k \neq 0 \\ \text{and } RD[0] = 1 - \sum_{k\neq 0}|S[k]|/N' = 1 - \sum_{k\neq 0} RD[k] \end{array}\right\} \quad (3.5)$$

RH, where $S[k]$ is used directly without normalization, may prove useful for certain applications, where the actual number of packets that is reordered is given.

For an example of RD representation, consider a sequence (1, 2, 3, 5, 6, 4, 8, 9, 10, 7) of size N' = 10. The corresponding reorder set R = {{(4, 2), (5, -1), (6, -1), (7,1), (8, -1)}. The subsets of interest are $S[-1]$ = {(5, -1), (6, -1), (8, -1)} and $|S[-1]|$ = 3, $|S[1]|$ = 1 and $|S[2]|$ = 1. From Eq. (3.5), RD can be computed as RD[-1] = $|S[-1]|/N'$ = 3/10, RD[1] = 1/10, RD[2] = 1/10 and RD[0] = 5/10.

RD, is the most comprehensive metric of packet reordering to date as it measures both earliness and lateness with their extents. In addition, it is based on the mathematical representation of reordering that is independent of lost and duplicate packets. In Chapter VI, we will present other attributes of packet reordering metrics and grade the current metric, proposed metrics and RD, according to their characteristics.

As in the conservation of displacement principle of reordering representation, RD also conserves its weighted components with respect to displacement. This *Lemma* is stated and proved below:

Lemma 3: Reorder density function (RD) satisfies $\sum_{k}(k * \mathrm{RD}[k]) = 0$

Proof: Since RD[k] implies that there are $N * RD[k]$ packets with displacement k, the sum of displacements of packets is given by

$$\sum_{k}(k*\mathrm{N}*\mathrm{RD}[k]) = \mathrm{N}*\sum_{k}(k*\mathrm{RD}[k])$$

$$= \sum_{m} d_m = 0 \ (\textit{from Lemma 2}) \ \text{(Q.E.D)}$$

This is a unique property of RD that shows that the effective amount of earliness is compensated by the effective amount of lateness. Other useful properties of RD that can provide more insight into the causes and effects of packet reordering are presented below.

Property 1: If packet reordering introduced by a network consists only of IR associated with early arrival of the packets, in the absence of losses, then RD satisfies the following conditions:

$$\left\{\begin{array}{l}\mathrm{RD}[k] = 0 \text{ for } k > 1 \\ \mathrm{RD}[1] = \sum_{k<0}(|k|*\mathrm{RD}[k])\end{array}\right\} \tag{3.6}$$

For every early p-event in IR there are late s-events whose number can be obtained from the magnitude of the displacement of this p-event, as shown in Eq. (3.3). Consider a sequence at sender's side: (1, 2, 3, 4, 5, 6, 7, 8). If there exist two alternate paths to destination such that packet 4 arrives early by three positions and packet 8 comes early by two positions then the arrival sequence is given as (4, 1, 2, 3, 5, 8, 6, 7). The p-events are (4, -3) and (8, -2). The corresponding s-events sets 'ς' are {(1, 1), (2, 1), (3, 1)} and {(6, 1), (7, 1)} respectively. The sizes of these sets are '3' and '2' respectively, which are equal to magnitude of displacements of corresponding p-events. Thus, RD[1] = (3+2)/8 = 5/8 and

$\sum_{k<0}(|k|*\mathrm{RD}[k])$ is equal to 3*1/8 + 2*1/8 = 5/8. Thus with early p-event of displacement k, we have $|k|$ s-events of displacement equal to one. Hence equation (3.6).

However, if packets are lost and these packets were to suffer an s-event, then the effective displacement of other packets also suffering s-events may not be one. For example, consider a packet sequence (4, 1, 3, 5, 6) and let us say that packet 4 took and alternate path that had significantly lower delay, causing it to arrive early. As per packets with s-events, they are 1 and 3. Due to the loss of packet 2, packet 1 is now assigned a *receive_index* value of three causing it to be early by two positions. Therefore, the property may not hold, if there are lost packets.

Property 2: If packet reordering introduced by a network consists only of IR with late arrival of the packets, in the absence of losses, then RD satisfies the following conditions:

$$\left\{\begin{array}{l}\mathrm{RD}[k] = 0 \text{ for } k < -1 \\ \mathrm{RD}[-1] = \sum_{k>0}(k*\mathrm{RD}[k])\end{array}\right\} \tag{3.7}$$

As in *property 1*, for every late p-event in IR there are early s-events whose number can be obtained from the magnitude of the displacement of the p-event, as shown in Eq. (3.2).. Consider a sequence at sender's side: (1, 2, 3, 4, 5, 6, 7, 8). If there exist two alternate paths to destination such that packet 1 arrives late by two positions and packet 5 comes late by three positions then the arrival sequence is given as (2, 3, 1, 4, 6, 7, 8, 5). The p-events are (1, 2) and (5, 3). The corresponding s-events sets 'ς' are {(2, -1), (3, -1)} and {(6, -1), (7, -1), (8, -1)} respectively. The sizes of these sets are '2' and '3' respectively, which are equal to magnitude of displacements of corresponding p-events. Thus, RD[-1] = (2+3)/8 = 5/8 and

$\sum_{k>0}(k * \mathrm{RD}[k])$ is equal to 2*1/8 + 3*1/8 = 5/8. Thus with late p-event of displacement k, we have k early s-events of displacement equal to -1. Hence equation (3.7). As explained for *property 1*, lost packets may impact the displacements of packets involved in s-events, justifying the condition of absence of losses.

Property 3: If packet reordering introduced by a network consists only of IR with late and/or early arrival of the packets, in the absence of losses, then RD satisfies the following condition:

$$\mathrm{RD}[-1] - \sum_{k>1}(k*\mathrm{RD}[k]) = \mathrm{RD}[1] - \sum_{k<-1}(|k|*\mathrm{RD}[k]) = c, \text{ where } c \geq 0 \qquad (3.8)$$

This property is an intuitive result derived by combining of *Lemma 3, Property 1* and *Property 2*. From *Lemma 3*, $\sum_{k}(k * \mathrm{RD}[k]) = 0$, i.e.,

$$\sum_{k<-1}(k*\mathrm{RD}[k]) + (-1)*\mathrm{RD}[-1] + 0*\mathrm{RD}[0] + \mathrm{RD}[1] + \sum_{k>1}(k*\mathrm{RD}[k]) = 0$$

Taking sign of k out of summation from the first term, we have

$$-\sum_{k<-1}(|k|*\mathrm{RD}[k]) - \mathrm{RD}[-1] + \mathrm{RD}[1] + \sum_{k>1}(k*\mathrm{RD}[k]) = 0$$

$$\mathrm{RD}[-1] - \sum_{k>1}(k*\mathrm{RD}[k]) = \mathrm{RD}[1] - \sum_{k<-1}(|k|*\mathrm{RD}[k])$$

If there were only late IRs with p-event displacements greater than one, then the LHS of the above equation is zero. Similarly for only early IRs with p-event displacements greater than minus one, the RHS of the above equation is equal to zero. In both the cases RD[1] and RD[-1] are due their respective s-events. The only IRs that can make the above LHS and RHS not equal to zero are those pairs of p- and s-events with magnitude displacements each equal to one. As the number of such events can only be non-negative, we have

$$RD[-1] - \sum_{k>1}(k*RD[k]) = RD[1] - \sum_{k<-1}(|k|*RD[k]) = c, \text{ where } c \geq 0$$

Property 4: Consider a sequence of 'N' packets partitioned into 'm' RS', R_1, R_2, ..., R_m with lengths n_1, n_2, ..., n_m respectively. RD of the sequence is the weighted average of RDs of individual RS', i.e.,

$$RD[k] = \sum_{i=1}^{m}(\frac{n_i}{N} * RD_i[k]) \tag{3.9}$$

where $RD_i[k]$ is the RD for i^{th} RS.

From the definition of RS, reordering is not propagated beyond an RS. Applying this condition to sequence (1, 4, 2, 3, 5, 6, 9, 10, 8), one of the many ways of partitioning this sequence is into three RS': (1, 4, 2, 3), (5, 6) and (9, 10, 8). Thus, for a p-event in a RS, all its corresponding s-events are also in RS. Also, considering the lowest sequence number in each RS as the first expected arrival, i.e., first *receive_index* value, the corresponding reorder sets of above RS' are: {(2, 1) (3, 1), (4, -2)}, {} and {(8, 2), (9, -1), (10, -1)). Therefore, the principle of conservation of displacement and amount of reordering are also applicable to RS, i.e., the reorder sets of RS' are independent leading to independent subsets of same displacement, i.e., if $S_i[k]$ is the subset of reorder set for i^{th} RS, with reorder events associated with displacement, then for any two RS' *i* and *j*,

$$S_i[k] \cap S_j[k] = \varnothing \quad for\ i \neq j$$

Thus, the cardinality of subset of reorder set for the whole sequence with elements of displacement *k* is given by:

$$|S[k]| = |S_1[k]| + |S_2[k] + ... + |S_m[k]| = N * RD[k] \tag{3.10}$$

where $|S_i[k]|$ is the cardinality of subset $S_i[k]$. However, $|S_i[k]| = n_i * RD_i[k]$

Therefore, $RD[k] = \sum_{i=1}^{m}(\frac{n_i}{N} * RD_i[k])$.

In this Section, we presented Reorder Density (RD), a comprehensive metric to measure packet reordering. In addition to the *Lemma* of conservation of amount of reordering, we listed several very important properties of RD, relating the shapes of RD to certain probable causes. The determination of RD for a sequence from its RS' is very important in characterizing packet reordering, as shown in Chapter IV. In the following Section, we will discuss the evaluation of RD with systems approach applied to subnets.

3.3 Reorder Response

The characterization of reordering in a sequence of packets in the form of RD, in addition to capturing the out-of-orderliness of a packet sequence accurately, also allows us to model networks with respect to reordering introduced by sub-networks forming a network. We consider this to be an extremely important attribute of RD concept. None of the other metrics can achieve this mainly due to their lack of ability to capture reordering completely.

First, note that RD corresponding to a sequence of in-order packets is a unit impulse (at $k = 0$). Consider sending this sequence of packets through a network. RD of the sequence observed at the output corresponds to the reordering introduced by the network, and hence we call it the reorder response ($\boldsymbol{J}[k]$) of the network. Note that ($\boldsymbol{J}[k]$) is defined with respect to an input and an output of a network. Thus each input/output port pair of the network has a corresponding reorder response. $\boldsymbol{J}[k]$ of a network in which there is no reordering corresponds to a unit impulse at k=0. $\boldsymbol{J}[k]$ can also be interpreted as the probability that a certain packet gets displaced by k.

The reorder response of a network captures the effect of the network on a sequence of packets flowing from the source port to the destination port of interest. Since a network is a time-varying and highly dynamic environment, it may not be possible to characterize the network completely using a single reorder response function. In fact, the reorder response will be a function of many factors including the background or cross traffic in the network, and statistical characteristics of the inter-packet gap. We hypothesize however that a reorder response exists for a network operating under stationary conditions and a given inter-packet gap distribution, provided the reordering introduced by the network is not dependent on the sequence number of the packet. This excludes, for example, a node that looks at the packet sequence number and puts the packets in a certain order. Thus the following discussion applies only to networks operating under stationary conditions, i.e., statistical characteristics of the network and packet sequences do not change with time. We expect that a theory developed for reordering in networks under such stationary conditions will lead the way towards a more general solution in future.

Next we present a theorem that allows us to combine reorder responses of two subnets to obtain the reorder response of the cascade formed by these subnets.

Theorem: The reorder response $\mathbf{J}[k]$ of a network formed by cascading 'N' multiple subnets, with reorder responses $\mathbf{J}_1[k]$, $\mathbf{J}_2[k]$, .., $\mathbf{J}_N[k]$ respectively, is given by the convolution of $\mathbf{J}_1[k]$, $\mathbf{J}_2[k]$, .. , $\mathbf{J}_N[k]$ i.e., $\mathbf{J}[k] = \mathbf{J}_1[k] * \mathbf{J}_2[k] * \ldots * \mathbf{J}_N[k]$.

Proof: Consider an arbitrary packet with sequence number 'm' that enters the cascade of two networks. It undergoes a displacement $d_m^{(1)}$ in Net-1, corresponding to the

event r(m, $d_m^{(1)}$). Thus, packet m occupies the position of m' as it enters Net-2, with $m' = m + d_m^{(1)}$. When the packet with sequence number m' enters Net-2, it undergoes a displacement $d_m^{(2)}$ in Net-2. The overall displacement of the packet is:

$$d_m = d_m^{(1)} + d_m^{(2)} \tag{3.11}$$

Similarly, if this packets continues to go through N networks in all then,

$$d_m = d_m^{(1)} + d_m^{(2)} + \ldots + d_m^{(N)} \tag{3.12}$$

The corresponding reorder event is

$$\mathrm{r}(m, d_m) = \mathrm{r}(m, d_m^{(1)} + d_m^{(2)} + \ldots + d_m^{(N)}) \tag{3.13}$$

The probability distributions of $d_m^{(1)}$, $d_m^{(2)}$ .... $d_m^{(\ )}$ are given by reorder responses of Net-1 $\mathbf{J}_1[k]$, Net-2 $\mathbf{J}_2[k]$ ... Net-N $\mathbf{J}_N[k]$, respectively. Using (3.12), and the independence of packet reordering with respect to the sequence numbers, the probability density distribution of d_m is given by the convolution of the distributions of $d_m^{(1)}$ and $d_m^{(2)}$ i.e.,

$$\mathbf{J}[k] = \mathbf{J}_1[k] * \mathbf{J}_2[k] \ldots * \mathbf{J}_N[k]. \text{ (Q.E.D)} \tag{3.14}$$

Packet reordering in a network is dependent on the rate at which the data is sent. While cascading two networks, if the first network slow down the data rate significantly then the packet gaps will also change considerably. With this change in packet gaps, the reorder response may also change for the input-stream. Thus, equation (3.14) may not remain valid. However, a slight change in packet gaps due to earlier networks in a cascade may not have significant impact on the reorder response itself.

Another advantage of having reorder responses is to evaluate TCP reordering independent of reordering due to networks. In fact, packet reordering in TCP can be

characterized better with such approaches as the sender's side RD may not be an impulse due to retransmissions. However, we can identify the reordering caused purely due to the network by using systems approach, where we derive the system response, given the input and output of the system. In our case, input corresponds to the RD at TCP sender's side, and output to the RD obtained at the receiver.

With these features of RD, in the next Section we explain the evaluation of RD at a receiver. As the packets arrive at the monitoring point or the receiver, the *receive_index* values should be assigned and RD should be computed in real-time. However, to classify a packet as reordered, it is vital to differentiate such packets with duplicates and also keep track of lost packets to assign proper *receive_index* values. Such issues are listed and discussed extensively.

3.4 Measurement of RD

Representation of reordering presented in Section 3.1 takes lost packets and duplicates into account by skipping the *receive_index* values corresponding to lost packets, and not assigning a duplicate packet a *receive_index*. However, this process needs detection of losses and duplicates, both of which provide implementation challenges. As RD is derived from this mathematical representation, this is an integral part of RD evaluation algorithm. Nevertheless the detection of duplicates and lost packets is a requirement of any reorder metric. If a sequence has packet loses and no reordered packets then a metric should not report packets arrive after lost packets as early, in anticipation of the arrival of lost packets. Similarly, a duplicate arrival of a packet after its successive packets should not be considered reordered.

When can a packet be considered lost? It is very difficult to answer this question. One possibility is to consider it as lost if it does not arrive when it is expected, but if it arrives later, go back and treat all the packets that have arrived from the point where it is expected as early. Moreover, if a lost packet from the beginning of a sequence arrives at the very end of the sequence, then we will have re-classified a large number of packets as early that we assumed in order. As an alternative, which is the other extreme, one can wait till the end of the received sequence to declare a packet as lost or reordered and then begin the classification process. However this requires keeping track of all received packets. Both these approaches are memory consuming, and also preclude real-time evaluation of the metrics.

In RD we address this issue by maintaining a threshold, D_T, and an early arrival buffer. If a packet is not received in next D_T packets from where it is expected, it is considered lost. A packet is classified as a duplicate on its arrival, if it already exists in early arrival buffer or current D_T window. The motivation for the scheme comes from the way the Internet applications and Transport protocols operate. Applications, protocols, and the network itself operate within finite resource constraints, which introduce practical limits beyond which the choice of certain values become irrelevant. In the case of D_T, it is common to find a value above which D_T does not have an impact on the reorder metrics. For example, in case of a VoIP application with a bit-rate of 128kbps and packet size of 200 bytes, a practical D_T value can be determined as follows. Assume that the application can wait a maximum of 50 ms for an expected packet and that the packets arrive at constant rate. Within 50 ms, the application can receive (128*1000*0.05)/(200*8), i.e., 4 packets. Since packets arriving with latency greater than this duration are effectively lost, the D_T value

could be set at 4. If the operational nature of an application is such that a D_T can be defined, then using D_T in the computation of reorder metrics will neither invalidate nor limit the effectiveness of the metrics, i.e., increasing D_T does not provide any benefit. In the case of TCP, the transmit- and receive-window sizes impose a natural limit on the useful value of D_T. If there are no operational constraints imposed by factors as described above or if one is purely interested in a more complete picture of reordering, then D_T can be made as large as required.

Since it is not known whether a packet is lost until D_T packets are received, real-time RD evaluation using this threshold may be done in one of two ways:

Go-back D_T: In this method, the rules are applied at each arrival. If a packet that was supposed to arrive D_T places ago does not arrive, then this sequence number is removed from the *receive_index*, and RD is recomputed for the previous D_T steps. Consider a received sequence (1, 3, 4, 5, 6, 7, 2, 8, 10, 9) and $D_T = 3$. Computation of RD using Go-back is illustrated in Table 3.3, where second run corresponds to going-back procedure. In the sequence, when packet 3 arrives, we assume that packet 2 will arrive later. Sequence number 2 is stored in an expected sequence numbers' buffer. The waiting for packet 2 continues after the arrivals of packets 4 and 5. However, if packet 2 were to arrive after

Table 3.3. Example of RD computation using Go-back method

Arrived Sequence	1	3	4	5	6	7	2	8	10	9
receive_index (first run)	1	2	3	4		7	-	8	9	10
Displacement (first run)	0	0	0	-1		0	-	0	-1	1
receive_index (second run)		3	4	5	6					
Displacement (second run)		0	0	0	0					

packet 5 then its displacement would be '+3', which is equal to D_T. If packet 2 fails to arrive at this point but arrives later, then its displacements would be greater than D_T. Thus in the sequence, as soon as packet 6 arrives, 2 is classified as lost and we go back and correct the previous D_T *receive_index* values from 2, 3 and 4 to read as 3, 4, and 5 respectively. Also, the displacements are recomputed as shown in second run of Table 3.3. After this rerun through the segment of the sequence, when packet 2 actually arrives later, a *receive_index* value is not assigned for that arrival and it is discarded completely. The spatial and temporal overheads from this method are recording the previous D_T packet numbers, and additional processing as displacements are computed again, given a packet is lost. However, if the amount of loss is low, the overall computation overhead of this method is low. Pseudo code is listed in [Ba04].

Stay-back D_T: With this method the computation of RD lags by D_T packets, i.e., the packet with *receive_index i* is not used in the evaluation until D_T more packets have arrived after that. Thus, we do not correct or adjust any displacements. Consider the previous example sequence (1, 3, 4, 5, 6, 7, 2, 8, 10, 9). For this sequence the *receive_index* values are assigned for each arrival after waiting for D_T more arrivals, to avoid a rerun as in Go-back method. For example, the arrival of packet 1 does not receive its *receive_index* until packet 5 arrives. After the arrival of 5, we assign *receive_index* value equal to the least among packet 1 and D_T arrived packets, i.e., *receive_index* = 1 for packet 1. Similarly for the arrival 3, we wait until next D_T arrivals, which correspond to packets 4, 5 and 6. The least among these arrivals is 3, so we assign *receive_index* = 3 for packet 3's arrival. Thus, we have eliminated the assignment of *receive_index* value of 2 as it did not arrive within threshold.

However, one can argue that the assignment of *receive_index* values need not be delayed by D_T every time, for example, as in for packet 1. This packet came in its position and we can exploit its arrival right away for faster computation. Thus, this method requires buffering of the next D_T arrivals only if the packets are reordered or lost. But waiting for D_T number of packets is not cumulative, i.e., the delay is only D_T for any size of the sequence. Therefore, the gain from early assignment of *receive_index* may not exist, given at least one packet is lost or at least one packet is displaced by D_T units.

Use of the threshold D_T also improves the robustness of RD. For example, if a packet was late by a large number say 1000, and then the next 1000 packets would be shown as early in the absence of a threshold. By using a threshold, we can eliminate such large impacts on RD due to a single reordering event. Consider a sequence (1, 25990, 2, 3, 4, 5, 6, 7, 8, ..., 100) . Here packet 25990 could have arrived due to TCP rollover or other unknown reason. Without a threshold, all the packets from 2 to 100 are classified as late by one position. Thus a single (unwanted) arrival has huge and completely disproportionate impact on reorder measurement. However, with a use of threshold, say 10 or 20, we recover from this peculiar behavior and rather drop the packet 25990 from the RD computations. Similarly RD also allows recovery from large number of missing packets. For example, consider a sequence (1, 2, 3, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20). Using threshold of say 7, we classify the burst of packets as lost as soon as packet 11 arrives (which is D_T positions from previous arrival namely packet 3). Thus, RD is robust against both peculiar arrivals and bursts of losses. Perl scripts and algorithms for RD computation are available in Appendix A.

3.5 Internet Measurements

Lately, there have been studies as discussed in Chapter 2, where packet reordering is measured over the Internet. However, these studies were hindered due to lack of proper metric to measure reordering. In this section, we present the reordering measured over the Internet using RD metric. In addition, due to a theoretical foundation for reordering, we also analyze the different type of reordering patterns that exist over the Internet. Such an analysis, as we will see in the next chapter allows us to model packet reordering, consider common patterns of reordering occurrences.

3.5.1 RD Measurements

Multiple files were downloaded from websites around the world, of sizes greater than 2MB (to obtain greater than 1000 packets) at an endpoint in subnet 129.82.x.x (located at Colorado State University, USA). The subnets of the sites chosen for downloads are listed in Table 3.4. Ping program and IP2Location [IPLo] software were used to obtain average one-way delay (D), standard deviation of delay (SD) and geographical location of the source node. Though ping program uses ICMP and it is not representative of the actual delays suffered by the packets of collected streams, it provides a close estimate of these values.

After identification of a link to the file on a website, a tcpdump [Tcpd] session with source host filter was initiated. Then the file was downloaded. The collected dump data, followed by its conversion to text file, was parsed and the sequence numbers from the packets were mapped to 1, 2, 3, ..., etc. Examples of commands of tcpdump, used in order are shown below:

tcpdump –i eth0 src host www.foo.com -w country.dmp

tcpdump –I eth0 src host www.foo.com -r country.dmp > country.txt

The script file used to obtain sequence numbers from .txt file is listed in Appendix B. These sequence files were then subjected to RD script as shown in Appendix A, to obtain RD of the received stream.

Table 3. 4. List of subnets of servers with their mean latency and standard deviations, used for reorder measurements over the Internet

Net	Subnet	Location	Avg. Delay (D) (in ms)	Std.Dev (SD) (in ms)
1	192.35.x.x	Armenia	75.5	1.5
2	209.211.x.x	MA, USA	45.5	0.81
3	62.94.x.x	Lazio, Italy	81.9	0.19
4	130.195.x.x	Wellington, NZ	95.4	0.17
5	202.54.x.x	India	145	3.01
6	202.14.x.x	ACT, Australia	121.4	0.37
7	203.223.x.x	Japan	77.1	0.14
8	66.187.x.x	MI, USA	20.5	1.79
9	61.8.x.x	Sydney, Australia	115.1	0.43

Tables 3.5 (a)-(b) illustrate the RDs obtained for the packets downloaded from the listed servers. The first column indicates displacements and the other columns show the normalized frequencies of packets with such displacements. The dates of the data collection are also listed in the last row of the tables.

Table 3.5. (a) Reorder Density for packets captured from Net-1, 2, 3, 4, 5, and 6 (b) Net-7, 8 and 9, with collection times (MM/DD/YY), $D_T = 25$

Displacement	Net-1	Net-2	Net-3	Net-4	Net-5	Net-6
-4	0.00199	0.0051	0	0	0	0
-3	0.00398	0.0358	0.01472	0.0015	0	0
-2	0.0557	0.0818	0.0225	0.0048	0	0
-1	0.18501	0.1048	0.01753	0.0399	0.01351	9.770E-4
0	0.55172	0.6751	0.88054	0.9179	0.98403	0.9985
1	0.1187	0.0076	0.02829	0.0327	0	9.770E-4
2	0.05902	0.0204	0.03077	0	0	0
3	0.01658	0.0127	0.00562	0	0	0
4	0.00663	0.0179	0	0.03	0.00123	0
5	0	0	0	0	0.00123	0
Collection	04/11/04	10/06/04	11/27/04	04/11/04	02/24/04	02/29/04

(a)

Displacement	Net-7	Net-8	Net-9
-1	0.02354	0.0096	0.01261
0	0.96778	0.98947	0.98517
1	0.00743	--	0
4	0	2.82382E-5	4.94315E-4
5	0	--	7.41473E-4
6	0	1.12953E-4	2.47158E-4
7	0	1.41191E-4	4.94315E-4
8	0	8.47147E-5	2.47158E-4
9	0	5.64764E-5	0
10	0	5.64764E-5	0
11	0	1.12953E-4	0
12	0	5.64764E-5	0
13	0.00124	8.47147E-5	0
Collection	04/11/04	04/10/04	04/10/04

(b)

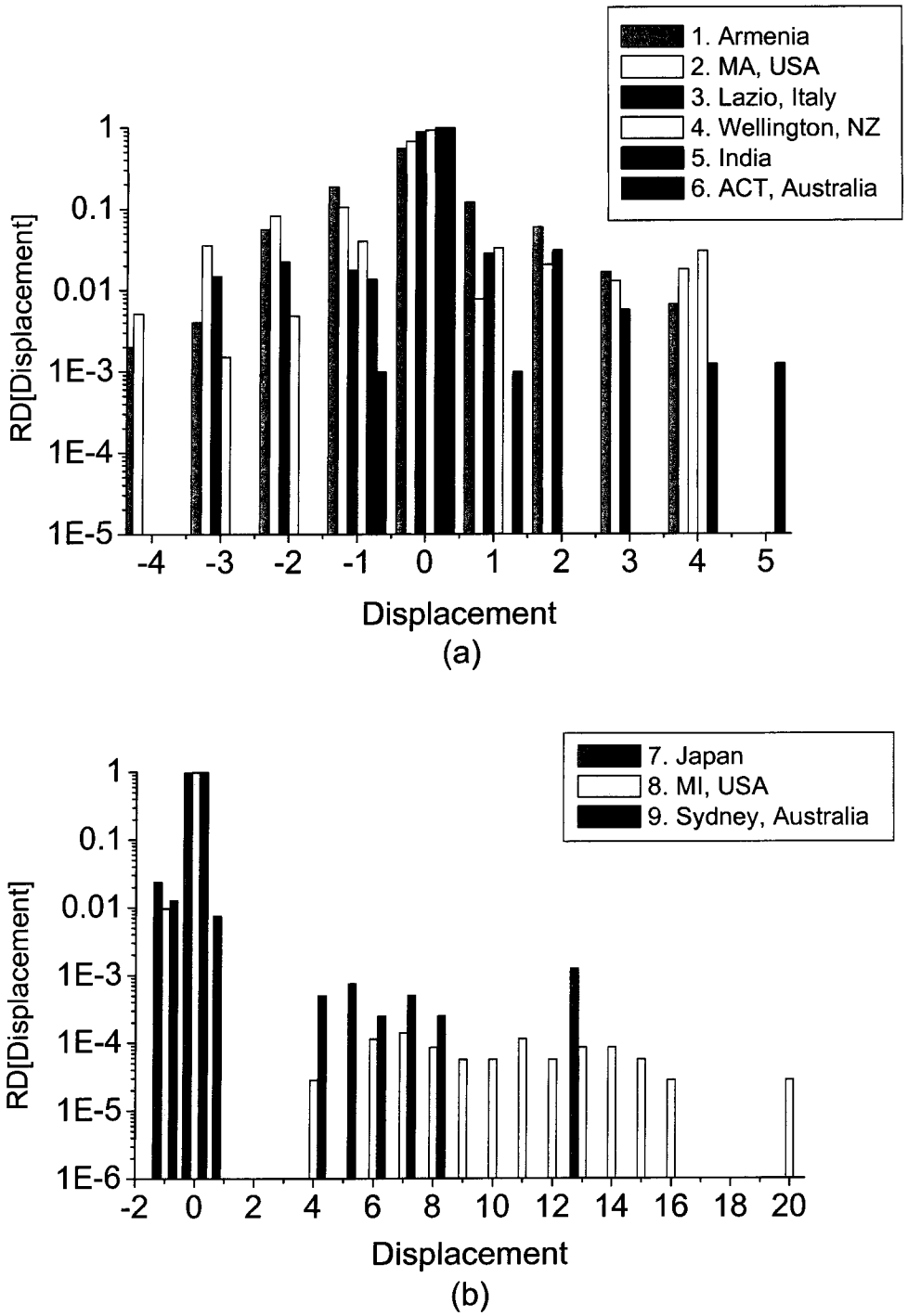


Fig. 3.2. RD based on Internet measurements (a) from Net-1, 2, 3, 4, 5 and 6 (b) from Net-7, 8, 9 to 129.82.x.x, D_T = 25 for all these measurements

Fig. 3.2 (a) and (b) depict RDs obtained from the data downloaded from Net-1, 2, 3, 4, 5, 6, 7,8 and 9 to a workstation in 129.82.x.x subnet with displacement threshold set to 25. The characteristics that can be derived from RDs (Table 3.5 and Fig. 3.2) include the following:

(i) Net-1 and Net-2 have approximately one in every three packets displaced.

(ii) Net-1 server was farther (geographically) from receiver when compared to Net-2 server yet, packets from Net-2 had more reordering than Net-1. Net-1's SD is less than Net-2's SD indicating that SD value affects the amount of reordering in a network. In other words, though it is believed that the amount of reordering will increase with distance, it is not always true. However, increasing distance could increase the number of hops, i.e., increasing intermediate nodes. But these nodes need not reorder packets.

(iii) For Net-1, the component RD[-1] is significantly higher than the other components, i.e., there is around 20% of packets, which are early by one position. It reflects a higher occupancy of buffer, which is used to recover from reordering.

(iv) Reorderings are more spread out in Net-1 to Net-4. On the contrary, as percentage reordering may account for only lateness or earliness, the percentage of reordering (considering only lateness) will falsely indicate less reordering in Net-4 compared to Net-5.

(v) Net-5 has minimum number of packets arriving out of their position, as more than 90% of the packets had zero displacement. Net-5 has only three components, i.e., RD[-1] and RD[4] and RD[5]. From *property 3*, as explained earlier, Net-5 may have only IRs

with displacements 4 and 5. Recovery from reordering is feasible with a buffer size of five packets, at the receiver. Also, the percentage of retransmissions triggered by 3 duplicate ACKs mechanism in TCP is equal to sum of RD[4] and RD[5] components.

(vi) Net-6 has packets that are displaced by only −1, 0 and +1. The reason of reordering may be due to one in every thousand packets taking an alternate route or more likely suffering additional delay in processing. Also this Net has the least amount of reordering both with respect to positions, and buffer required to recover (equal to one) from reordering.

(vii) Net-7, 8 and 9 have greater displacements than the previous Nets. However, the displacements are spread to the right of the origin. Interestingly, all the reorderings could be attributed to lateness based IRs.

(viii) Net-8 has wider spread than Net-7 and Net-9 indicating a higher variance in delay values, also evident from Table 3.4. However, the delay variance need not be the only reason for more spread out reordering. For example in TCP streams, if packets are lost then they are retransmitted. These retransmitted packets may have same delay as transmitted packets that are not lost. However, the retransmitted packets are out of order by definition. In Net-9, reordering is spread out, but the amount of variance is not significant, therefore for this case, retransmissions could be pointed as the reason for out-of-orderliness.

In the following section, we will analyze the packet reordering patterns that commonly occur over the Internet and quantify them with respect to basic patterns.

3.5.2 Basic Patterns over the Internet

The analysis of reordering using the Internet measurements was extended to identify the types of reordering patterns that are common in current Internet. The number of MRS' along with the type of events associated for Net-1 to Net-9, as discussed in Section 3.5.1 is listed in Table 3.6. The patterns that did not belong to the set of basic patterns, i.e., IR, OR and ER patterns are classified into 'Other' category. To classify between a primary late or a primary early event, the arrival times are used, e.g., sequence (1, 3, 4, 2, 5), may be due to one primary late event involving packet 2, or two primary overlapping early events due to packets 3 and 4. However, if the gap between packets 1 and 3 is high compared to gap between packets 2 and 5 then it is highly probable that packet 2 is late. It was observed that a majority of p-events are lateness-based events, as evident from Net-5 to Net-9. These Nets have contained only lateness based IRs.

For reorder measurements corresponding to servers on Net-1,2, 3 and 4, there were patterns that contained more than two p-events. However, majority of the patterns were still basic patterns, as seen in Table 3.6. Therefore, it can be safely concluded from these readings that the basic patterns form significant MRS' over the Internet. Thus, RD for basic pattern MRS' can be used to characterize most of the reordering over the current Internet.

In the next chapter we develop models for packet reordering, by limiting the consideration of patterns to basic patterns. However as explained in Chapter II, packet reordering will be more prevalent in near future high speed networks, and in such networks the fraction of other types of reordering events is likely to be significantly higher. These

advanced models will consider networks in which packets traverse through multiple parallel paths.

Table 3.6. Percentage of Basic Patterns (two p-events) in the reordering data collected over the Internet

Net	*IR%*	*OR%*	*ER%*	*Other*
1	64	24	0	12
2	45	38	17	0
3	86	12	1	1
4	84	2	4	10
5	100	0	0	0
6	100	0	0	0
7	100	0	0	0
8	100	0	0	0
9	100	0	0	0

3.5.3 Emulations of Reorder Responses over Cascades

To verify the convolution theorem for reordering, we emulated two networks using NISTNet [Ca03] on Linux boxes. The connectivity used is shown in Fig. 3.3. Net-X was emulated using mean (D) and standard deviation (SD) of delays as 150 ms and 7 ms respectively and Net-Y is emulated using D = 100 ms and SD = 4 milliseconds. Packet streams of sizes 10,000 packets were sent from A to B and B to C separately. These emulators running on the workstations were then connected using a cross-cable. Another packet stream of 10,000 packets was sent from A to C on this cascade. Constant inter-packet gap (of 500μs) was maintained in each of the input streams.

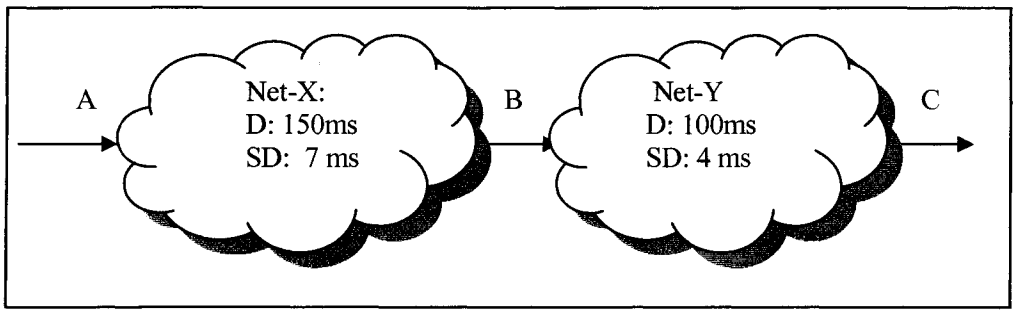


Fig. 3.3. Emulated network topology to verify convolution theorem for RD

To obtain reorder responses of individual networks sequence numbers from 1 to ten thousand were inserted in the data field of consecutive UDP packets. The process was repeated and RD was measured between points A to B and B to C. As the input sequence numbers were in order, RD at the output points, i.e., B and C respectively, are the reorder responses of these networks. The RD computed in connected case was compared to the result based on convolution of the RDs for A to B and B to C cases. RDs for A to B, B to C, A to C and convolution of A to B with B to C are shown in Table 3.7. The root mean square error (RMSE) between the two RDs was equal to 2.9e-03, strongly validating the theorem (3.14). The plots of these RDs are illustrated n Fig. 3.4.

It is interesting to note that for the packet stream from A to C, even though the inter-packet gap at A was 500μs, by the time packets transit to B, the inter-packet gap is no longer a constant. On the other hand, the reorder response for Net-Y used for convolution, shown in Fig. 3.4(a) corresponds to a constant inter-packet gap. This indicates that the reorder response of a network, while dependent on the inter-packet gap distribution in general, may not be highly sensitive to it.

Table 3.7. Reorder Response from A to B, B to C, A to C and convolution of responses from A to B and B to C

Displacement	RD from A to B	RD from B to C	RD from A to C	Convolution
-4	0	0.00185	0.0062	0.00702
-3	0.006	0.01785	0.04365	0.03818
-2	0.193	0.10665	0.13085	0.12587
-1	0.2093	0.26655	0.23375	0.23782
0	0.5736	0.31005	0.24935	0.25526
1	0.15515	0.1458	0.16015	0.16158
2	0.034	0.0925	0.09265	0.09652
3	0.00565	0.0398	0.04695	0.04684
4	0.0024	0.01065	0.02125	0.01795
5	0	0.00435	0.00915	0.00685
6	0	0.0022	0.0025	0.00296
7	0	9.5E-4	0.00145	0.00136
8	0	8E-4	5.5E-4	7.3121E-4
9	0	0	4E-4	1.7929E-4
10	0	0	4.5E-4	3.7848E-5

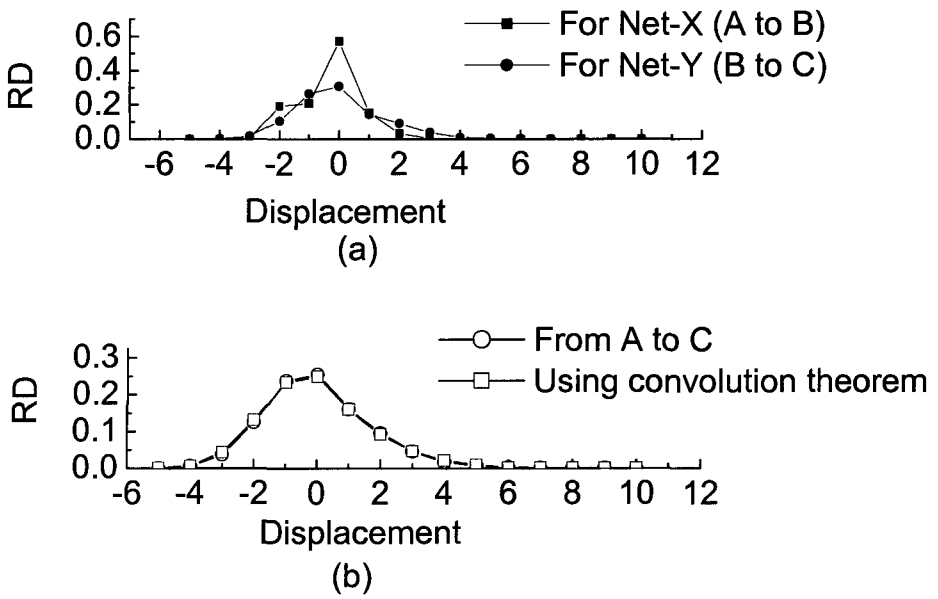


Fig. 3.4. (a) RD from A to B and B to C with separate streams (b) Verification of the theorem of convolution using network emulators for RDs as reorder responses

3.6 Remarks

This chapter covered the representation of packet reordering comprehensively with lemmas and theorems, and metric Reorder Density along with its properties. As this work is first to the best of our knowledge, in this section, we will provide remarks on any debatable or practical issues within these topics.

Let us revisit the assignment of *receive_index* values in the presence of lost packets. Recall that a *receive_index* value equal to the sequence number of a lost packet is never assigned. Now, consider the packet sequence (1, 2, 3, 4 , 5, 6). During the transit, let packet 4 be lost and packets 3 and 5 be reordered. Thus, the received sequence be (1, 2, 5, 3, 6).

Here a packet 4's original position lies within two reordered packets. Based on the proposed notation, the corresponding values of *receive_index* are (1, 2, 3, 5, 6). Computing displacements, we have (0, 0, -2, 2, 0). However, one can argue and misinterpret the displacements to be (0, 0, -1, 1, 0) by just looking at the sequence (1, 2, 5, 3, 6). As mentioned earlier, we assume that lost packets are not reordered unless we there is an evidence of such an occurrence. Since packet 4 is lost and cannot provide any such evidence, we assume packet 4 being in its position. Thus, the only possible displacement value for packet 3 to arrive after 5 is by being displaced by two positions. In addition, if we assume that lost packet is reordered then we have numerous possible sets of displacements for packets 3 and 5. Thus, RD metric provides reordering measure based on received sequence and we could insert more intelligence, if there is a need for it.

The selection of D_T is important to the computation of RD. Therefore, it is vital to understand the impacts of values chosen for reorder measurements. D_T may be set much higher for practical measurements. A larger threshold value results in higher memory requirements but provides a more comprehensive picture of reordering. However, if measurements of interest consist of only two positions or three positions of reordering, then the threshold could be smaller. We can draw the similarities between choosing a D_T with choosing an instrument to measure lengths. Vernier calipers may be required to get more precise measurement, equivalent to using a higher value of D_T for more comprehensive measurement, but for measuring the lengths on highways, use of caliper is too complicated and does not provide any benefit.

Chapter IV

MODELING OF PACKET REORDERING IN NETWORKS

Apart from scant attention paid to packet reordering and its measurement techniques, the absence of models for packet reordering has been an obstacle to the understanding of this phenomenon. Lack of packet reordering models for networks has also led to difficulty in understanding of the impact of reordering on TCP [Bl02]. Whatever is the specific cause of reordering, it can primarily be attributed to parallelism, i.e., parallelism within the node (multiple queues and/or multiple processors for a single packet stream) or parallelism among the links (due to load-splitting at the intermediate routers). A secondary cause for reordering is retransmissions. Therefore, a model derived using parallelism as underlying cause of reordering phenomenon, is applicable to many of its causes. In other words, if reorder models are derived using load-splitting among links as a primary cause for change in the order of sequence numbers, then the same model can be used for reordering due to load-splitting for processing.

In this chapter, we provide two approaches to the modeling of packet reordering. Both the approaches consider load-splitting among links as the underlying cause for packet reordering. In the first approach, the kinds of reordering patterns are limited to the basic patterns, thus accounting only for the amount/extent of reordering over the current Internet

(as shown in previous chapter). In the second approach, all possible reordering patterns are considered with two-path and multi-path parallelism scenarios.

4.1 Reorder Models when Reordering Limited to Basic Patterns

Basic patterns form a significant part of different reordering patterns observed over the Internet. Given RD expressions for basic patterns, if one can estimate the number of different basic pattern RS' in a sequence then using *property* 4 of RD in Section 3.2, the reordering in a sequence can be computed. In subsection 4.1.1, the expressions of RD for basic patterns are derived, and in subsection 4.1.2, using two-path load-splitting as a cause for reordering, the number of each basic pattern in a sequence is estimated. The results from these two subsections are combined to derive RD model for reordering with occurrence of reorder patterns limited to basic patterns, in subsection 4.1.3. Later in subsection 4.1.4, the model is verified against two-path emulated topologies.

4.1.1 RD for Basic Patterns

As presented in previous chapter, basic patterns consist of Independent Reordering (IR), Overlapping Reordering (OR) and Embedded Reordering (ER). Different kinds of OR and ER are listed in Table 3.2.

If the single p-event of IR pattern is $r(m, d_m)$ and the length of a RS with this single p-event is n_i, then using Eqs. (3.2) and (3.3), RD for IR is given as:

$$\left.\begin{array}{l} \{RD[d_m] = 1/n_i, RD[-sgn(d_m)] = |d_m|/n_i \\ \text{where } sgn(d_m) = +1 \text{ if } d_m > 0 \text{ and } = -1 \text{ if } d_m < 0 \end{array}\right\} \quad (4.1)$$

Any value of RD[k], $k \neq 0$, that is not explicitly specified is zero. RD[0] is always given by expression in Eq.. (3.5).

Following the convention of two p-events r(x, d_x) and r(y, d_y) for OR, where r(y, d_y) is the second p-event in the sequence of packets, we define a related quantity called overlap length V_{xy}. Overlap length indicates the number of sequence numbers shared by the s-event sets, if these two p-events were to occur independently. For example, consider the sender's side sequence (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). If only a reordering event r(4, 3) occurs, then the s-event set is {(5, -1), (6, -1), (7, -1)}. Similarly, if only a reordering event r(6, 4) occurs then the s-event set is {(7, -1), (8, -1), (9, -1), (10, -1)}. The common sequence number in s-event sets is just '7', for this case, as illustrated in Fig.4.1. Therefore, $V_{xy} = 1$.

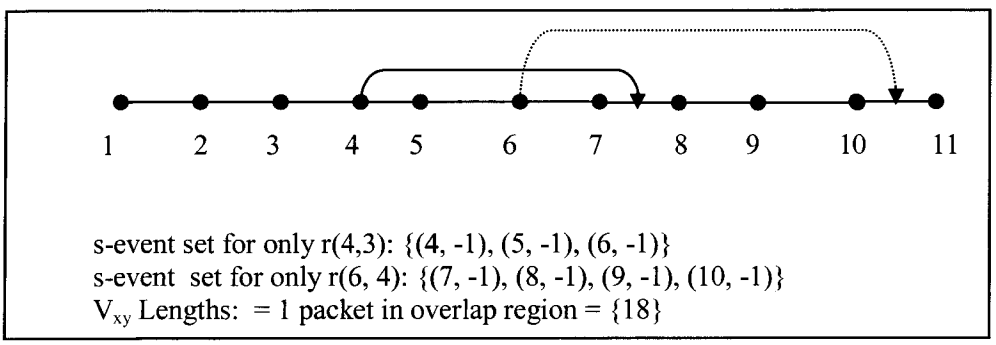


Fig. 4.1. Reordering of packets illustrating the computation of overlap length V_{xy}

Let us consider the derivation of RD for each possible kind of OR patterns. Initially, for all kinds of patterns, RD[k] for all *k* is initialized to zero. We use notation of increment similar to C/C++ language during the assignment in derivations. The representation of the events is as explained in Section 3.1 and illustrated in Table 3.2.

(i) *Earliness overlaps with earliness:* $d_x < 0$, $d_y < 0$

Displacements of overlap region packets are affected by both p-events. Since both of these events are early packets, V_{xy} packets are late by two positions. All the remaining packets associated with s-events of both p-events, exclusive of themselves are late by one

position. Therefore, RD for a RS of size n_i with only single OR of type earliness overlaps with earliness and no other patterns is given as:

$$\left\{\begin{array}{l} RD[2] \mathrel{+}= V_{xy}/n_i \\ RD[1] \mathrel{+}= (-d_x - d_y - 2*V_{xy})/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \quad (4.2)$$

Consider an example sequence (1, 2, 7, 3, 4, 9, 5, 6, 8, 10). Let us assume that packets 7 and 9 are associated with early p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(7, -4) and r(9, -3). Overlap region contains packets 5 and 6. Therefore, V_{xy} = 2. From Eq. (4.2), (recall RD[k] for all k is initialized to zero), RD[2] += V_{xy}/n_i = 2/10, RD[1] += (- d_x -d_y -2*V_{xy})/n_i = (- (-4) – (-3) – 2*2)/10 = 3/10, RD[-4] += 1/10 and RD[-3] = 1/10. From Eq. (3.5), RD[0] = 1 – RD[2] – RD[1] – RD[-4] – RD[-3] = 3/10.

(ii) *Earliness overlaps with lateness:* $d_x > 0$, $d_y < 0$

Displacements of overlap region packets are affected by both p-events. Since one of them is a lateness p-event and other an earliness p-event, all V_{xy} packets effectively have zero displacement. All the remaining packets associated with s-events of first p-event undergo a displacement of -1, and s-events of second p-event undergo a displacement of +1. However, this basic pattern is different from the previous basic pattern, as the displacements of packets 'x' and 'y' contain contributions from each other, i.e., if the events were to occur independently then displacements of these events would be one less than that of the overlap case. Therefore the number of s-events with minus one and plus one displacements are not (d_x - V_{xy}) and ($-d_y$ - V_{xy}), instead they are (d_x -1 -V_{xy}) and ($-d_y$ -1 -V_{xy}) respectively.

Therefore, RD for a RS of size n_i, with only single OR of type earliness overlaps with lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} RD[-1] \mathrel{+}= (d_x - 1 - V_{xy})/n_i \\ RD[1] \mathrel{+}= (-d_y - 1 - V_{xy})/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \quad (4.3)$$

Consider an example sequence, (1, 2, 3, 5, 10, 6, 7, 8, 4, 9). Let us assume that packets 4 and 10 are associated with late and early p-events respectively. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 5) and r(10, -5). Overlap region contains packets 6, 7 and 8. Therefore, V_{xy} = 3. From Eq.(4.3), RD[-1] += (d_x -1 -V_{xy})/n_i = 1/10; packet 5 arrived early by one position, RD[1] += (-d_y -1 -V_{xy})/n_i = (- (-5)-1-3)/10 = 1/10, packet 9 is late by one position, RD[5] += 1/10, RD[-5] = 1/10. From Eq. (3.5), RD[0] = 1 – RD[-1] – RD[1] – RD[-5] – RD[5] = 6/10.

(iii) *Lateness overlaps with lateness:* $d_x > 0$, $d_y > 0$

Displacements of overlap region packets are affected by both p-events. Since both the p-events are late, overlap region V_{xy} packets effectively undergo a displacement of -2 each. All the remaining packets associated with s-events of first and second p-events undergo a displacement of -1. Therefore, RD for a RS of size n_i, with only single OR of type lateness overlaps with lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} RD[-2] \mathrel{+}= V_{xy}/n_i \\ RD[-1] \mathrel{+}= (d_x + d_y - 2*V_{xy})/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \quad (4.4)$$

Consider an example sequence, (1, 2, 3, 5, 7, 8, 4, 9, 6, 10). Let us assume that packets 4 and 6 are associated with late p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 3) and r(6, 3). Overlap region contains packets 7 and 8. Therefore, V_{xy} = 2. From Eq.(4.4), RD[-2] += V_{xy}/n_i = 2/10, packets 7 and 8 are early by two positions, RD[-1] += ($d_x + d_y - 2*V_{xy}$)/n_i = 3/10, packets 5 and 9 arrived early by one position, RD[3] += 1/10 and RD[3] += 1/10 resulting in effective RD[3] = 2/10; Note, the usage of increment operator. From Eq. (3.5), RD[0] = 1 – RD[-2] – RD[-1] – RD[3] = 2/10

(iv) *Lateness overlaps with earliness:* $d_x < 0,\ d_y > 0$

Displacements of overlap region packets are affected by both p-events. Since first p-event is early and second p-events is late, overlap region V_{xy} packets effectively undergo zero displacement. All the remaining packets associated with s-events of first and second p-events undergo a displacement of +1 and -1 respectively. However, as in (ii), the packets 'x' and 'y' contribute to each others displacements. The number of s-events due to the p-events associated with them are not ($-d_x - V_{xy}$) and ($d_y - V_{xy}$), instead they are ($-d_x - 1 - V_{xy}$) and ($d_y - 1 - V_{xy}$) respectively Therefore, RD for a RS of size n_i, with only single OR of type lateness overlaps with earliness, and no other patterns is given as:

$$\begin{Bmatrix} RD[-1] \mathrel{+}= (d_y - 1 - V_{xy})/n_i \\ RD[1] \mathrel{+}= (-d_x - 1 - V_{xy})/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{Bmatrix} \quad (4.5)$$

Consider for example, the sequence (1, 6, 2, 4, 5, 7, 3, 8, 9, 10). Assume that packets 3 and 6 are associated with p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(6, -4) and r(3, 4). Overlap region contains packets 4 and 5. Therefore, V_{xy} = 2. From

Eq.(4.5), RD[-1] += (d_y -V_{xy})/n_i = 1/10, packet 7 is early by one position, RD[1] += (-d_x-1 -V_{xy})/n_i = 1/10, packet 2 is late by one position, RD[-4] += 1/10, and RD[4] += 1/10. From Eq. (3.5), RD[0] = 1 – RD[-1] – RD[1] – RD[-4] –RD[4] = 6/10.

RD expressions for ER basic patterns are simpler than OR patterns due to the absence of overlap length. The events involved are denoted by r(x, d_x) and r(y, d_y) events with packet of sequence number 'y' enclosing event with packet number 'x'. As above, we will consider one kind of ER at a time (again, notations are as in Table 3.2 of Section 3.1):

(i) Earliness encloses earliness: $d_x < 0$, $d_y < 0$

Due to the displacement of packet 'y', there are d_y packets that are late by one position each. Since packet 'x' is one of them, for it to be effectively displaced by d_x packets, it has to actually arrive early by (-d_x+1) positions. Thus the inner event involves

(-d_x+2)packets. There are (-d_x+1) packets late by two positions and (-d_y + d_x-2) packets late by one position. Therefore, RD for a RS of size n_i, with only single ER of type earliness enclosing earliness, and no other patterns is given as:

$$\left\{\begin{array}{l} \text{RD}[1] \mathrel{+}= (-d_y + d_x - 2)/n_i \\ \text{RD}[2] \mathrel{+}= (-d_x + 1)/n_i \\ \text{RD}[d_x] \mathrel{+}= 1/n_i \\ \text{RD}[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \qquad (4.6)$$

Let the packet numbers at the receiver be (1, 7, 2, 5, 3, 4, 6, 8, 9, 10). Let us assume that packets 7 and 5 are associated with early p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(5, -1) and r(7, -5) respectively. From Eq. (4.6), RD[1] += (-d_y + d_x-2)/n_i = 2/10, packets 2 and 6 are late by one position, RD[2] += (-d_x + 1)/n_i =

2/10, packets 3 and 4 are late by two positions, RD[-1] += 1/10 and RD[-5] += 1/10. From Eq. (3.5), RD[0] = 1 – RD[-1] – RD[1] – RD[2] –RD[-5] = 4/10.

(ii) Earliness encloses lateness: $d_x > 0,\ d_y < 0$

Due to the displacement of packet 'y', there are d_y packets are late by one position. Since packet 'x' is one of them, for it to be effectively displaced by d_x packets, it has to be displaced only by (d_x-1) positions. Thus the inner event involves d_x packets. There are (d_x-1) packets that undergo zero displacement, and $(-d_y - d_x)$ packets late by one position. Therefore, RD for a RS of size n_i, with only single ER of type earliness enclosing lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} \mathrm{RD}[1] \mathrel{+}= (-d_y - d_x)/\mathrm{n_i} \\ \mathrm{RD}[d_x] \mathrel{+}= 1/\mathrm{n_i} \\ \mathrm{RD}[d_y] \mathrel{+}= 1/\mathrm{n_i} \end{array}\right\} \tag{4.7}$$

Using an example sequence (1, 7, 2, 4, 5, 3, 6, 8, 9, 10), where packets 7 and 3 are associated with p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(3, 3) and r(7, -5) respectively. From Eq.(4.7), RD[1] += $(-d_y - d_x)/\mathrm{n_i}$ = 2/10, packets 2 and 6 are late by one, RD[3] += 1/10 and RD[-5] += 1/10. From Eq. (3.5), RD[0] = 1 – RD[1] – RD[3] – RD[-5] = 6/10.

(iii) Lateness encloses lateness: $d_x > 0,\ d_y > 0$

Due to the displacement of packet 'y', there are d_y packets are early by one position. Since packet 'x' is one of them, for it to be effectively be displaced by d_x packets, it has to be actually displaced by (d_x+1) positions. Thus the inner event involves (d_x+2)

packets. There are (d_x+1) packets effectively undergo a displacement of -1 each, and $(d_y - d_x-2)$ packets are early by one position. Therefore, RD for a RS of size n_i, with only single ER of type lateness enclosing lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} RD[-1] \mathrel{+}= (d_y - d_x-2)/n_i \\ RD[-2] \mathrel{+}= (d_x+1)/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \quad (4.8)$$

Consider an example sequence (1, 3, 5, 6, 7, 4, 8, 2, 9, 10). Let us assume that packets 4 and 2 are associated with late p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 2) and r(2, 6) respectively. From Eq.(4.8), RD[-1] += $(d_y - d_x-2)/n_i$ = 2/10, packets 3 and 8 are early by one, RD[-2] += $(d_x+1)/n_i$ = 3/10, packets 5, 6 and 7 are early by two, RD[2] += 1/10 and RD[6] += 1/10. From Eq. (3.5), RD[0] = 1 – RD[1] – RD[-2] – RD[2] – RD[6] = 3/10.

(iv) Lateness encloses earliness: $d_x < 0,\ d_y > 0$

Due to the displacement of packet 'y', there are d_y packets are early by one position. Since packet 'x' is one of them, for it to be displaced by $-d_x$ packets, it has to be displaced only by $(-d_x-1)$ positions. Thus the inner event involves $(-d_x)$ packets. There are $(-d_x-1)$ packets that undergo zero displacement, and $(d_y + d_x)$ packets that are early by one position. Therefore, RD for a RS of size n_i with only single ER of type lateness enclosing lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} RD[-1] \mathrel{+}= (d_y + d_x)/n_i \\ RD[d_x] \mathrel{+}= 1/n_i \\ RD[d_y] \mathrel{+}= 1/n_i \end{array}\right\} \quad (4.9)$$

Let us assume that packets 7 and 2 are associated with p-events in a received sequence (1, 3, 4, 7, 5, 6, 8, 2, 9, 10). Then r(x, d_x) and r(y, d_y) in the notation correspond to r(7, -3) and r(2, 6) respectively. From Eq.(4.9), RD[-1] += $(d_y + d_x)/n_i$ = 3/10, packets 3, 4 and 8 are early by one, RD[-3] += 1/10 and RD[6] += 1/10. From Eq. (3.5), RD[0] = 1 – RD[1] – RD[-3] – RD[6] = 5/10.

RDs for the basic templates derived in this section can be treated as fundamental elements to build an RD of a large sequence. As majority of the reordering observed on the Internet (as shown in Section 3.5.2), preliminary models of reordering can be modeled using these RDs. In the following section, an estimate of such occurrences is provided to further derive the model of reordering due to two-path load-splitting.

4.1.2 Basic Reorder Pattern Count for Two-Path Load-Splitting

We consider a simple load-splitting scenario as an example that can use the expressions derived in Section 4.1.1 to analyze reordering due to parallelism. First we will evaluate the number of possible basic patterns in this load-splitting scenario and then apply the expressions in Section 4.1.1 to each of the basic patterns. Finally, using *property 4* of RD as explained in Section 3.4, the RD expression for two-path load splitting example is derived.

Consider the case where a certain number of packets in a flow undergo significantly higher delay due to packets being routed via a longer path Y instead of path of choice X. Such a condition could occur, e.g., as an outcome of load balancing at a router. Let the difference between the average delays of the paths 'X' and 'Y' be such that the packets traversing via path Y arrive at the destination after 'Δ' of its subsequent packets that traverse

via path X. Fig. 4. 2 illustrates the displacement of packet with sequence number 11, where $\Delta = 3$. Packet with sequence number 11 arrives after packets with sequence numbers 12, 13 and 14. In networks, the value of 'Δ' may also vary depending on the variance of delays in paths X and Y (due to internal parallelism). But for the development of a reordering model with load balancing scenario, 'Δ' is considered to be constant.

To derive a RD for a packet stream received over the network with the above-described behavior, the number of IR, OR and/or ER patterns in the received sequence is statistically estimated, and then the RD expressions provided in Section 4.1.1 for the basic patterns are used to compute the effective RD. However, with a two-path scenario containing a unique Δ value, the ER basic pattern is not feasible. Also, if the delayed path has lower number of packets than the faster path then all p-events will be with respect to lateness. Thus, we have lateness based IR patterns and lateness overlapping lateness OR patterns.

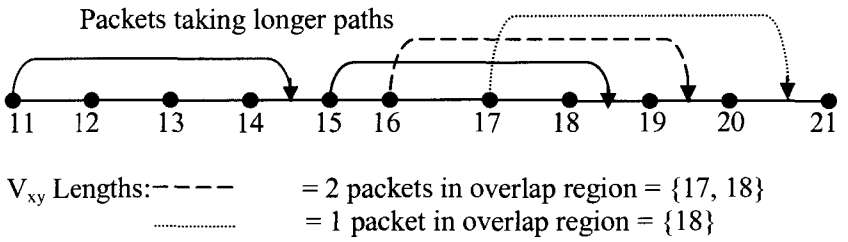


Fig. 4.2. Displacement of packets illustrating independent reorder event with packet 11 and feasible overlapping reorder events, given the displacement of packet 15

Let the probability that a packet takes path Y be '*P*' and path X be (1-*P*). Consider an arbitrary packet '*i*', then

$$\text{Probability}\{i^{th} \text{ 'pkt. displaced by } \Delta\} = P \tag{4.10}$$

Consider a sequence of size 'N,' where N >> Δ. For a constant 'Δ', the resultant multiple reordering types are restricted to IR and OR. Given that packet '*i*' takes path Y, it is involved in an IR event if none of its previous previous 'Δ-1' packets and 'Δ' packets within the reorder event get displaced. Therefore,

$$\text{Probability}\{\text{IR with } i^{th} \text{ pkt.}/\ i^{th} \text{ pkt. takes Y}\} = (1-P)^{2\Delta-1} \tag{4.11}$$

Note, even if packet '*i*-Δ' is displaced, it is still an IR, due to the displacement of the i^{th} packet. The probability of such an occurrence given packet '*i*' and '*i*-Δ' are displaced is given by:

$$\text{Probability}\{\text{IR with } (i-\Delta)^{th}\ pkt./\ i^{th}\ pkt. \text{ takes Y}\} = P(1-P)^{2\Delta-2} \tag{4.12}$$

However, these IR events have 'Δ-1' displacement associated with primary events. To summarize, the number of IR events with 'Δ' displacement in the sequence using Eq. (4.10) and (4.11), is approximated as:

$$\text{IR\_Count\_with\_}\Delta \cong\ NP(1-P)^{2\Delta-1} \tag{4.13}$$

and the number of IR events with 'Δ-1' displacement in the sequence, using Eq. (4.10) and (4.12), is approximated as:

$$\text{IR\_Count\_with\_}\Delta\text{-}1 \cong\ NP^2(1-P)^{2\Delta-2} \tag{4.14}$$

As all primary events correspond to late packets passing through Y, in the case of OR patterns, only "lateness overlapping lateness" type interference is feasible. However for a given 'Δ', the overlap length 'V_{xy}' between two p-events has a range of [1, Δ-1]. For example, as illustrated in Fig. 4.2, given the sequence (11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21), if 15 is displaced by 3 units, then 17 displaced by 4 units or 16 displaced by 4 units (considering packet 15 in displacement both cases) results in an OR event with 'V_{xy}' equal to 1 and 2 respectively. Note, the effective displacement of packet 15 is two whereas the displacement of the following primary event packet (16 or 17) is three.

Let us consider the case where only 15 and 17 are displaced forming an OR event. In this scenario, if any of the packets 13, 14 (previous Δ-1 packets) or 16, 18 (inner Δ-1 packets) or 19, 20 (Δ-V_{xy} packets) get displaced, and then a cluster of more than two overlapping events is formed. Note, packet 20 may be displaced as in the case discussed for IR events. However, the probability of such combined occurrences is small, and we neglect it. Therefore, for a basic OR pattern with overlap V_{xy}, there is a set of '3Δ-2-v_{xy}' packets that should not be associated with primary events. The probability of the occurrence of an OR event, given packet 'i' is displaced, is:

$$\begin{aligned} &\text{Probability \{OR with } V_{xy} \text{ associated with } i^{th} \text{ pkt./ } i^{th} \text{ pkt. displaced\}} \\ &\quad = (1-P)^{3\Delta-2-V_{xy}} P \end{aligned} \tag{4.15}$$

Thus, using Eq.(4.15), the average number of OR patterns for a large received sequence of N, is approximated as:

$$\text{OR\_Count} \cong \sum_{V_{xy}=1}^{\Delta-1} N(1-P)^{3\Delta-2-V_{xy}} P^2 \tag{4.16}$$

As P increases, the number of complex reorder patterns involving more than two primary events (both overlapped and embedded) increases. As observed in Section 3.5, the overlap patterns with more than two p-events are not common over the Internet.

4.1.3 RD Model for Two-Path Load Splitting

Given a sequence of size N, the counts of reordering are given by Eq.(4.13), (4.14), and (4.16). Applying expressions for RD to each IR and OR patterns, derived in Section 4.1.3, the resulting RD for the load-balancing scenario can be evaluated using the following assignments:

For IR of size 'Δ', using Eq. (4.13),

$$\begin{Bmatrix} \text{Initialize RD}[k] = 0 \text{ for all } k, \text{ then} \\ \text{RD}[\Delta] \mathrel{+}= N P(1-P)^{2\Delta-1} \\ \text{RD}[-1] \mathrel{+}= N\Delta P(1-P)^{2\Delta-1} \end{Bmatrix} \tag{4.17}$$

and for IR of size 'Δ-1', using Eq. (4.14),

$$\begin{Bmatrix} \text{Initialize RD}[k] = 0 \text{ for all } k, \text{ then} \\ \text{RD}[\Delta-1] \mathrel{+}= N P^2(1-P)^{2\Delta-2} \\ \text{RD}[-1] \mathrel{+}= (\Delta-1) N P^2(1-P)^{2\Delta-2} \end{Bmatrix} \tag{4.18}$$

In the case of OR, using Eq. (4.16) and (4.4), initialize RD[k] = 0 for all k, then do the following assignments:

$$\begin{Bmatrix} \text{for } V_{xy} = 1 \text{ to } '\Delta - 1' \{ \\ \quad \text{RD}[\Delta] \mathrel{+}= N(1-P)^{3\Delta-2-V_{xy}} P^2 \\ \quad \text{RD}[\Delta-1] \mathrel{+}= N(1-P)^{3\Delta-2-V_{xy}} P^2 \\ \quad \text{RD}[-1] \mathrel{+}= N(2\Delta - 2V_{xy} - 1)(1-P)^{3\Delta-2-V_{xy}} P^2 \\ \quad \text{RD}[-2] \mathrel{+}= N V_{xy} (1-P)^{3\Delta-2-V_{xy}} P^2 \quad \} \end{Bmatrix} \tag{4.19}$$

4.1.4 Emulation and Verification

To verify the model for RD derived in previous section, the network topology shown in Fig. 4.3 was emulated using NISTNet network emulator. The average one-way delays of paths X and Y were D_X = 28 ms and D_Y = 80 ms respectively. The inter-packet gap at the source was set to 10 ms. From these values, it is apparent that the packet in path Y may get displaced by approximately five positions with respect to those taking path X, i.e., Δ = 5 was used for the expressions in the model. The standard deviation (SD) values for delays on these paths were set to 5% of the average delays of the respective paths, i.e., 1.4 ms and 4.0 ms that are maintained less than the inter-packet gap. As uniform distributions are used

for delay, there were displacements of lengths that were greater and less than $\Delta = 5$. However, the model was able to approximate the measurements even with such deviations.

A stream of 50,000 UDP packets was sent through the network, in which a packet takes an alternate route Y with probability of P, (= 0.01, 0.03, 0.05 and 0.07), to emulate varied scenarios.

From these assigned delay values, it is apparent that the packet in path Y may get displaced by approximately 5 positions with respect to those taking path X, i.e., $\Delta = (D_Y - D_X)$/packet gap = 5, was used for the expressions of the model. A two millisecond additional delay is path Y is used to compensate for any discrepancies due to dispersions of packets on path X. The standard deviation (SD) values for delays on these paths were set to 5% of the average delays of the respective paths, i.e., 1.4 ms and 4.0 ms. Due to these delay deviations, there were displacements of lengths that were greater and less than $\Delta = 5$. However, the model was able to approximate the measurements even with such deviations.

A stream of 50,000 UDP packets was sent through the network, in which a packet

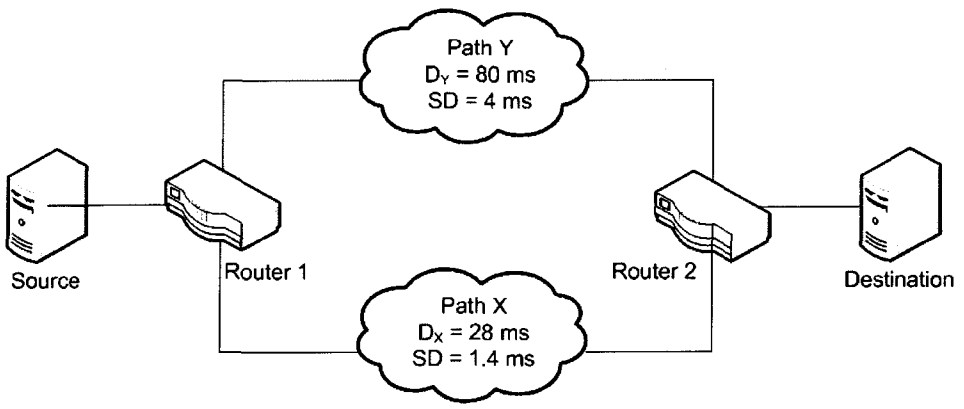


Fig. 4.3. Emulated network topology using NIST Net, to attain two data paths, X and Y

takes an alternate route Y with probability of P. Measurements were repeated for 'P' = 0.01, 0.03, 0.05, 0.07 and 0.1, to emulate varied scenarios.

Fig. 4.4 illustrates RDs from the derived model using Eqs. (4.17)-(4.19), and from measurements using emulation that were computed on-the-fly. The vertical axes are in log10 scale to clearly show all components of the RD. It can be observed that the RD from model

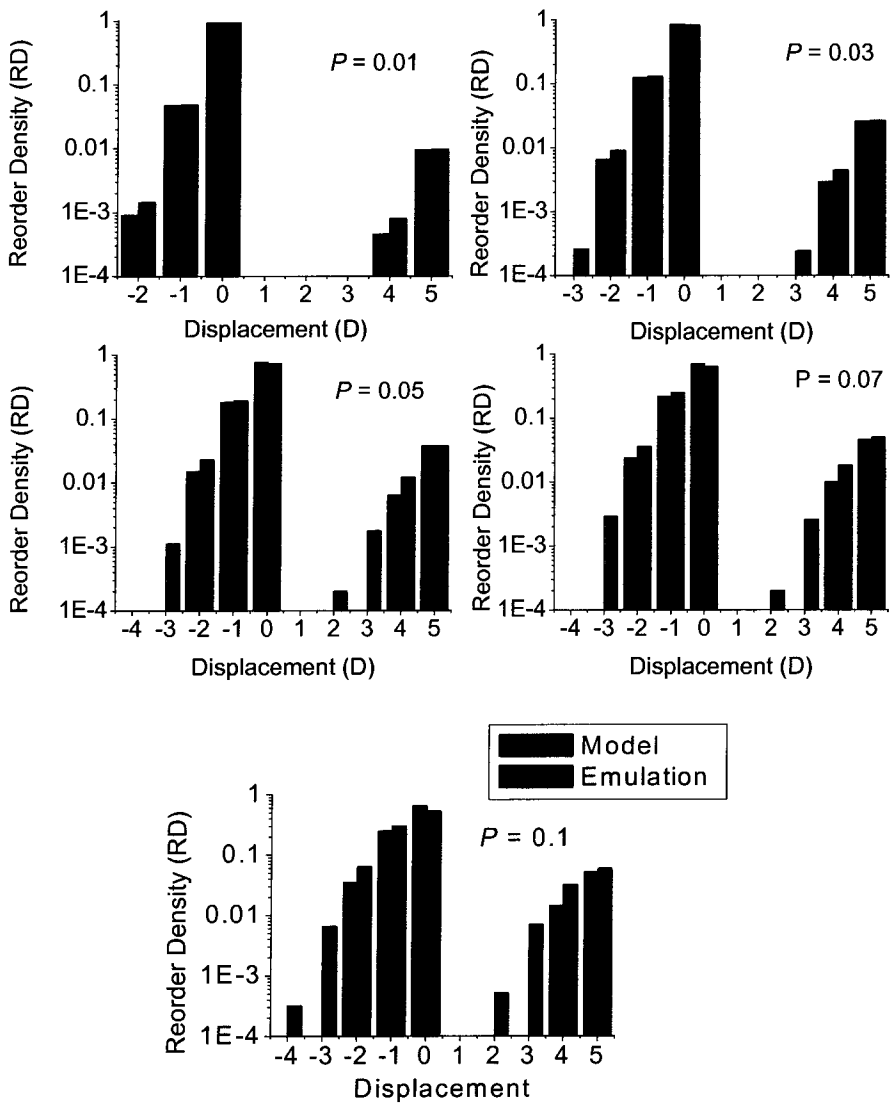


Fig. 4.4. Comparisons of RDs derived from model and emulations for P = 0.01, 0.03, 0.05, 0.07 and 0.1

follows the RD obtained from emulation for low values of P, validating the assumption during the derivation of the model about the magnitude of P. However, as P increases beyond 0.05, the number of multiple overlap events with more than two p-events becomes significant. As we approach 0.1, the spread of RD is significant with increasing contributions from different displacements. Also, the RMSE values for P = 0.01, 0.03, 0.05, 0.07 and 0.1 between model and emulation RDs are 8.83E-4, 0.0040, 0.0096, 0.022 and 0.0421 respectively, clearly indicating that the model has limitations with respect to P value. By considering three p-event-overlap cases during the analysis, the methodology for the model can be extended to follow the RD from emulated measurements, even for higher values of P. Moreover, if only current common occurrences of reordering patterns in the Internet have to be modeled then this limited modeling technique is very efficient.

In the next section, we will develop a general model for packet reordering that obeys all the P values and considers all the possible reordering patterns. In addition, we remove the limitation of two-path load-splitting by deriving and verifying multi-path models.

4.2 Reorder Model for Load-Splitting without Reorder Pattern Limitations

The comprehensive nature of RD lets reordering to be modeled for any parallel scenarios. The models for packet reordering are developed for packets taking multiple links. Such scenarios of traffic splitting and bandwidth aggregation are common in data networking [Kh99]. Any parallelism within the nodes can be modeled similarly, as waiting times in different queues within a node can be substituted for path characteristics on links. Unlike limited models where we estimated the number of basic patterns, we will concentrate

on the distribution of displacement of any arbitrary packet in the sequence. Its displacement distribution is nothing but RD itself.

4.2.1 RD Model

To explain the terminology used in modeling, consider the two-path load splitting example, illustrated in Fig. 4.3. The packets of stream generated at source enter Router 1, and go through either path X or path Y to reach Router 2, which is linked to the destination. In the example, the average delay of path Y is 80 ms and path X is 28 ms. Therefore, if one of the packets from the stream takes path Y, then it is delayed on average by additional 52 ms compared to all other packets that take path X. If the inter-packet gap in the packet stream at source is equal to 10 ms, then the packet going through path Y arrives on average after five of its successive packets in source stream, thus causing packet reordering. Therefore, for each path we can associate a displacement Δ that a packet suffers going through it. For primary preferred path (namely, X), Δ is set to zero, and in the above example $\Delta = 5$ for alternate path Y.

Now, consider the case with 'n' alternate paths in addition to the preferred primary path from Router 1 to the destination. Let P_n, P_{n-1},..., P_2, P_1 correspond to the path probabilities with which packets leaving Router 1 choose the respective paths. These paths have associated Δ values of $n, n-1, ..., 2, 1$ respectively. Note that some of the path probabilities may be zero. Thus P_0, the probability with which a packet takes preferred primary path, is given by $P_0 = 1 - P_1 - P_2 - ... - P_n$. The effective displacement d_m of each packet m at the destination depends of combination of displacements of its neighboring packets.

Due to its simplicity, again consider the example of two-path load splitting to understand the impact of neighboring packets on the effective displacement of packets at the

destination. Fig. 4.5 shows the displacement of an arbitrary packet '*i*' that takes the alternate path (as path Y in Fig. 4.3). In this example, a non-zero Δ value (say 4 as in the example) is associated with alternate path. Thus, packet '*i*' is four places away from the original position.

The packets that affect the displacement of packet '*i*' are '*i+1*', '*i+2*'... till '$i+\Delta$'. If one of these packets also takes the alternate path then the effective displacement of '*i*' is 'Δ-*1*', and if two of them take path Y then the displacement is 'Δ-*2*' and so on. The packets subsequent to '$i+\Delta$' have no effect, as their displacements do not change the displacement of '*i*'. Similarly, packets before '*i*' do not affect the displacement of '*i*' either. On the other hand, if packet '*i*' is one that has taken the primary path, the packets that affect the displacement of packet '*i*' are '*i-1*', '*i-2*'... till '$i-\Delta$' Thus, the displacement of any packet '*i*' is affected by its neighboring packets in range '$i-\Delta$' to '$i+\Delta$'. The distribution of displacements of this arbitrary packet '*i*' provides RD of the packet stream at the destination.

Extending the above understanding to multiple-path load splitting, each alternate path is associated with a different Δ value. Let us use the notation described above and consider the displacement of an arbitrary packet '*i*'. If this packet '*i*' is one that has taken the primary path, then packets before '$i-n$' will not affect its displacement, as n is the maximum displacement a packet suffers due to alternate paths. Similarly, packets after '*i*' will not affect displacement of '*i*' as the least displacements that they can have is $\Delta = 0$. This case is illustrated in Fig. 4.6(a).

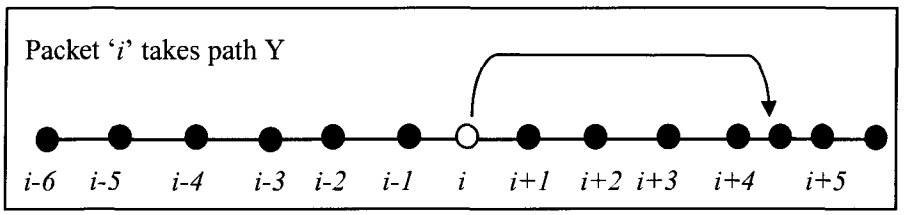
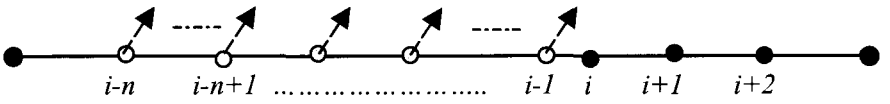
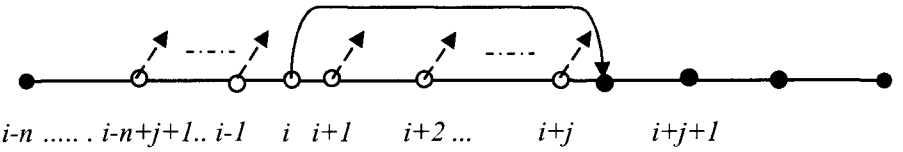


Fig. 4. 5. Displacement of packet '*i*' due to additional delay in path Y, traversed by it



(a) Packet '*i*' is not displaced. The packets that may affect the effective displacement of '*i*' are shown using arrows



(b) Packet '*i*' takes j^{th} path. The packets that may affect the effective displacement of 'i' are shown using arrows

Fig. 4.6. Effective displacement of packet '*i*', given (a) '*i*' takes preferred path, (b) '*i*' takes j^{th} path; packet numbers that may affect the displacement of '*i*' are in gray color

If packet '*i*' gets displaced by '*j*', i.e., it takes the path with path probability P_j, then the packets with sequence numbers less than '*i-n+j+1*' and greater than '*i+j*' will not affect its effective displacement as depicted in Fig. 4.6(b). Packets with sequence numbers less than '*i-n+j+1*', i.e., '*i-n+j*', '*i-n+j-1*', may get displaced by a maximum of n, but fail to enclose packet '*i*'s new position. Similarly, packets with sequence numbers greater than '*i+j*', i.e., '*i+j*', '*i+j+2*', etc., may get displaced by a minimum of zero units, yet cannot enclose the new position of packet '*i*'. Now to model the packet reordering for multi-path scenario, we will derive the probabilities associated with the effective displacement 'L' of this arbitrary packet '*i*'. We begin modeling multi-path by first considering the case where '*i*' takes the preferred path, followed by the case in which '*i*' takes a alternate path. Recall, we use the term 'effective displacement' to refer to the difference between the packet sequence number and its *receive_index*. An interference probability is associated with packets of the arrival sequence. A packet '*i-m*' that does not affect the effective displacement of packet '*i*', has an interference probability, $P_{mi} = 0$.

Consider the case where packet '*i*' takes the preferred path. The probability of packet '*i-m*' interfering with the effective displacement of '*i*', i.e., packet '*i*-m' moving by more than 'm', is:

$$P_{mi} = (P_m + P_{m+1} + \ldots + P_n) = \left\{ \begin{array}{l} \sum_{k=m}^{n} P_k \quad \text{for } 0 < m \le n \\ 0 \text{ otherwise} \end{array} \right\} \tag{4.20}$$

Therefore, the probability that none of the packets interfere with effective displacement of '*i*', given '*i*' is not displaced, i.e., $d_i = 0$, is:

$$\text{Probability}\{L = 0 / i \text{ takes preferred path}\} = \prod_{l=1}^{n} (1 - P_{li}) = C_0 \tag{4.21}$$

Consider the case where only packet '*i-m*' belonging to the range [*i-n, i-1*] interferes with the displacement of packet '*i*', then

$$\begin{aligned} &\text{Probability}\{L = -1/(i \text{ takes preferred path, only pkt.'} i-m \text{'interferes})\} \\ &= P_{mi} \prod_{l=1, l \neq m}^{n} (1 - P_{li}) = \left(\prod_{l=1}^{n} (1 - P_{li}) \right) \frac{P_{mi}}{(1 - P_{mi})} = C_0 \frac{P_{mi}}{(1 - P_{mi})} \end{aligned} \tag{4.22}$$

$$\text{Probability}\{L = -1 / i \text{ takes preferred path}\} = C_0 \sum_{m=1}^{n} \frac{P_{mi}}{(1 - P_{mi})} \tag{4.23}$$

Similarly, let packet $i\text{-}m_1$ and $i\text{-}m_2$ be the only two packets belonging to the range [*i-n, i-1*] interfering with packet '*i*'. These could be first and second packets preceding '*i*', first and third packets, first and fourth so on, or second and third preceding '*i*', second and fourth, etc. Considering all the possible combinations of m_1 and m_2 values and transforming the index variables, following expression is obtained for the probability that packet '*i*' gets displaced by '–2' units:

$$\text{Probability}\{L = -2 / i \text{ takes preferred path}\} = C_0 \sum_{m_1=1}^{n-1} \sum_{m_2=m_1+1}^{n} \frac{P_{m_1 i} P_{m_2 i}}{(1 - P_{m_1 i})(1 - P_{m_2 i})} \tag{4.24}$$

Generalizing this expression for displacement of packet 'i' equal to $-k$ with '$k > 0$' packets interfering, we have

$$\text{Probability}\{L = -k/i \text{ takes preferred path}\}$$
$$= C_0 \sum_{m_1=1}^{n-k+1} \sum_{m_2=m_1+1}^{n-k+2} \cdots \sum_{m_k=m_{k-1}+1}^{n} \frac{P_{m_1 i} P_{m_2 i} \cdots P_{m_k i}}{(1-P_{m_1 i})(1-P_{m_2 i})....(1-P_{m_k i})} \tag{4.25}$$

Now, lets consider the case where packet 'i' takes j^{th} alternate path, where $j > 0$, and obviously less than or equal 'n' as there are only 'n' alternate paths. For this case, as illustrated in Fig. 4.6(b), packets from 'i-n+j+1' to 'i+j' may affect the displacement of packet 'i'. The interference probabilities for these packets are given as:

$$P_{mi} = (P_{m+j+1} + P_{m+j+2} + ... + P_n) = \left\{ \begin{array}{l} \sum_{k=j+m+1}^{n} P_k \quad \text{for } -j \le m \le n-j, m \ne 0 \\ 0 \text{ otherwise} \end{array} \right\} \tag{4.26}$$

If none of the packets interfere with 'i', then packet 'i' will have a displacement of 'j' units:

$$\text{Probability}\{L = j/i \text{ takes } j^{th} \text{ path}\} = \prod_{l=-j}^{n-j-1} (1-P_{li}) = C_j \tag{4.27}$$

Note $P_{ii} = 0$, by definition. As in (4.23), we can show that

$$\text{Probability}\{L = j-1/(i \text{ takes } j^{th} \text{ path, pkt.'}m\text{'interferes})\} = C_j \frac{P_{mi}}{(1-P_{mi})}$$

$$\text{Probability}\{L = j-1/i \text{ takes } j^{th} \text{ path}\} = C_j \sum_{m=-j}^{n-j-1} \frac{P_{mi}}{(1-P_{mi})} \tag{4.28}$$

$$\text{Probability}\{L = j-2/i \text{ takes } j^{th} \text{ path}\} = C_j \sum_{m_1=-j}^{n-j-2} \sum_{m_2=m_1+1}^{n-j-1} \frac{P_{m_1 i} P_{m_2 i}}{(1-P_{m_1 i})(1-P_{m_2 i})} \tag{4.29}$$

Extending it for only '$k > 0$' packets interfering with packet i',

$$\text{Probability}\{L = j-k/i \text{ takes } j^{th} \text{ path}\}$$
$$= C_j \sum_{m_1=1}^{n-j-k} \sum_{m_2=m_1+1}^{n-j-k+1} \cdots \sum_{m_k=m_{k-1}+1}^{n} \frac{P_{m_1 i} P_{m_2 i} \cdots P_{m_k i}}{(1-P_{m_1 i})(1-P_{m_2 i})....(1-P_{m_k i})} \tag{4.30}$$

Note, a same effective displacement of '*i*' could result from various paths taken by it. For example, when packet '*i*' takes the preferred path then the interference of one of its preceding packets from the range [*i*-n, *i*-1] leads to displacement of '-1' for packet '*i*'. But if '*i*' takes the alternate path with $\Delta = 2$, and three of the packets from range [*i-n+3,* i+*2*] interfere with its displacement, then packet '*i*' can still have an effective displacement of '-1'. Accounting for all such occurrences and using (4.21) to (4.30), RD[*k*] for any 'k' can be computed as follows:

$$\mathrm{RD}[k] = \begin{Bmatrix} (\text{Probability}\{L = k/i \text{ takes preferred path}\}) * P_0 \\ +\text{Probability}\{L = k/i \text{ takes } 1^{\text{st}} \text{ alternate path}\} * P_1 \\ +\text{Probability}\{L = k/i \text{ takes } 2^{\text{nd}} \text{ alternate path}\} * P_2 \\ \vdots \\ +\text{Probability}\{L = k/i \text{ takes } \mathrm{n}^{\text{th}} \text{ alternate path}\} * P_n \end{Bmatrix} \tag{4.31}$$

resulting in

$$\mathrm{RD}[k] = \sum_{j=0}^{j=n} \left\{ (\text{Probability}\{L = k/i \text{ takes } \mathrm{j}^{\text{th}} \text{ path}\}) * P_j \right\} \tag{4.32}$$

The case where the packets are routed through two paths, depicted in Fig. 4.3 is of significant interest. Therefore we simplify the model for this case as follows. Let P be the probability of a packet taking the alternate path, and let the offset of packets taking this path be Δ. In this case, $P_j = 0$ for '*j*' not equal to Δ, P_Δ = P, and P_0 = 1-P. Thus equation (4.32) reduces to

$$\mathrm{RD}[k] = \begin{Bmatrix} (1-\mathrm{P})\binom{\Delta}{|k|}\mathrm{P}^{|k|}(1-\mathrm{P})^{\Delta-|k|} & for\ -\Delta \le k < 0 \\ \mathrm{P}^{\Delta+1} + (1\text{-}\mathrm{P})^{\Delta+1} & for\ k = 0 \\ \mathrm{P}\binom{\Delta}{k}\mathrm{P}^{\Delta-k}(1-\mathrm{P})^{k} & for\ 0 < k \le \Delta_k \end{Bmatrix} \tag{4.33}$$

It can be observed that RDs for both positive and negative displacements are binomial distributions. Thus, these components are symmetrical around P = 0.5, and also

RDs for P = x and P = 1-x, where $0<x<1$, are mirror images about origin for two-path load splitting case.

4.2.2 Model Verification and Reordering Analysis

The model for RD developed in previous subsection is verified using the same emulated network topology, as shown in Fig. 4.3 for two-path as an example, followed by a case with five-path load splitting. Nature of packet reordering is further analyzed for two-path load splitting to derive more intuitive understanding. NISTNet emulator was used for routing packets probabilistically to networks X and Y, as well as to introduce path delays. A stream of 50,000 packets, with a constant gap of 10 ms between the packets, was sent through this emulated network and reordering was measured at the destination.

The average one-way delays of the paths were D_X and D_Y. The value of D_X was equal to 28 ms, and D_Y was varied to obtain various Δ values for alternate path. The value of Δ for path Y depends on the difference between D_X and D_Y values and the inter-packet gap value for the stream of packets. Fig. 2 shows an example value of D_Y = 80 ms. As shown earlier, Δ =5 for path Y, in this topology. The standard deviations (SD) of these delay values were also varied for different sets of measurements.

Fig. 4.7 depicts the RDs obtained from Eq. (4.33) and from the emulation testbed. Loss of packets was not introduced in the emulations, as RD evaluation is insensitive to the losses. SD value for both paths was set to 0.5% of the mean. Fig. 4.7 indicates very close agreement between measured RD and the prediction based on the model. The distributions of RDs for P = 0.1 and 0.9, and P=0.3 and 0.7, are mirror images around displacement equal to zero. As P increases from 0.1 to 0.5, the number of packets taking alternate path increases leading to more non-zero displacements. However, after P=0.5, the number of packets in

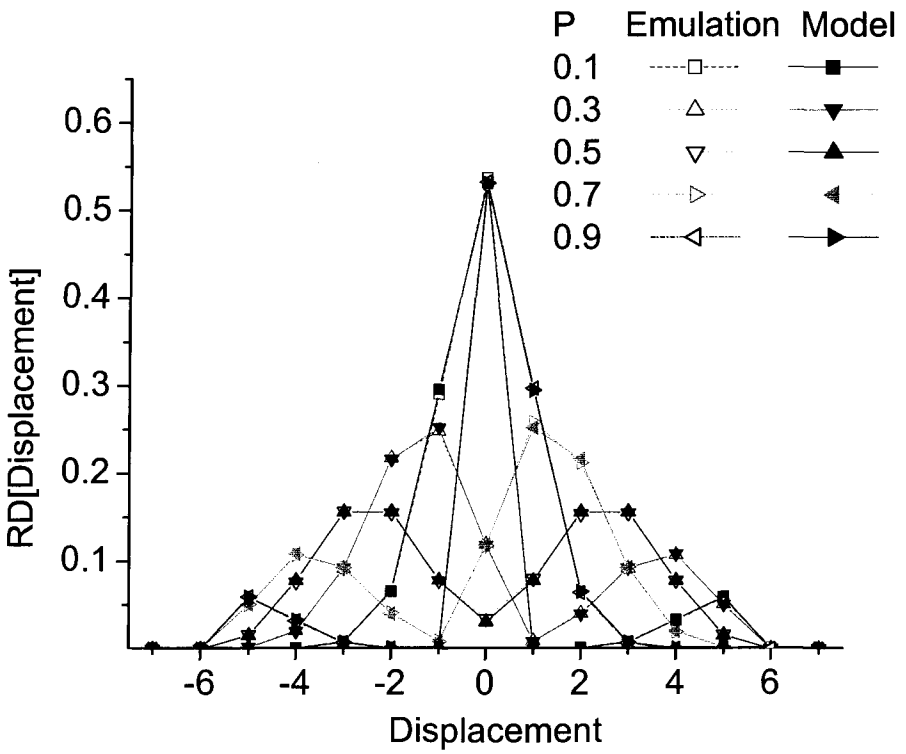


Fig. 4. 7. RD from emulation Vs. RD from model for various *P* Values

alternate path are more, making it a preferred path. Thus, path X has a $\Delta = -5$ and path Y has a $\Delta = 0$.

In Fig. 4.8, we illustrate the impact of the delay of alternate path on reorder profile by varying the delay of the alternate path, yet keeping the probability of taking the alternate path constant (P = 0.3). As the displacement that a packet suffers in an alternate path, Δ increases, the effective spread in RD distribution also increases. There is good agreement of the model with the measured value. Increasing delay of alternate path leads to more packets with non-zero displacements. The values of RD[k] for $k < 0$ correspond to early packets. To recover from reordering, these packets need to be buffered till the arrival of the next in-sequence packet. This observation also confirms that to counter reordering, it is advisable to choose a next path that has closer latencies to those on preferred path.

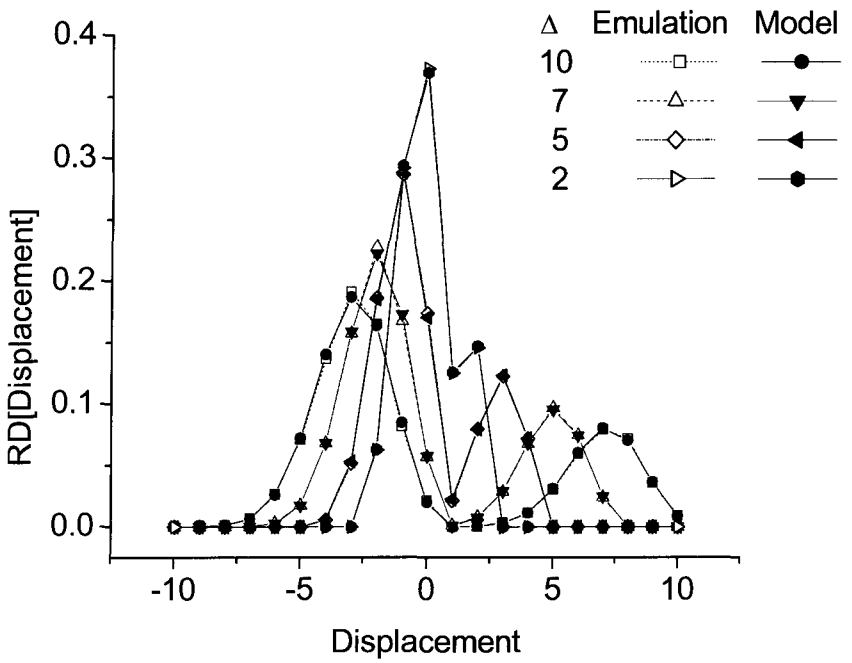


Fig. 4.8. RD variation with Δ, for a fixed *P* value = 0.3 for alternate path

In the above analyses, the distribution of delay for path Y had an SD of 0.5% of mean delay. The two-path model assumes constant Δ. To evaluate the effectiveness of the model when this assumption does not hold, we varied the value of SD to 0.5, 1, 2, 5, 10 and 15% of mean delays on both paths. Recall that the average delay values were chosen such that the effective Δ was computed to be five. As SD was increased beyond 5%, i.e., 1.4 ms and 4.0 ms on D_X and D_Y respectively, RD had components beyond displacement 5. This change can be attributed to the delay deviations becoming significant with respect to the inter-packet gap value, which was set to 10 ms. The RMS error between RD from model and RD from emulation shown in Fig. 4.9 for SD values equal to 0.5%, 1%, 2%, 5%, 10% and 15% are 0.0028, 0.0038, 0.0054, 0.0152, 0.015, and 0.015 respectively. Even though the analysis did not take the delay variations into account, these results indicate the validity of the model for such cases too.

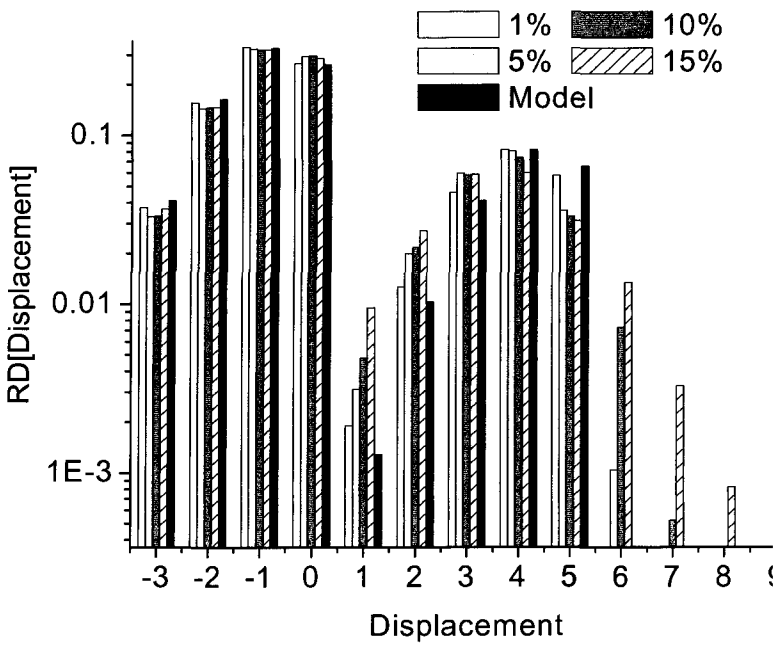


Fig. 4.9 Impact of Standard deviation (SD) in delay D_Y on RD

Though the components of RD are very small for values beyond 'Δ', as illustrated in Fig. 4.9 (note Log10 scale for Y axis), further increase in SD value led to greater spread in the RD distribution. However, such a variation can be accommodated in the model by taking multiple Δ values with respect to path Y, which can be computed by sampling the distribution of delay values.

Next we evaluate the reordering caused by a node that routes packets to the destination via five different paths. The delay value of primary path is kept at 28 ms, and the delays of alternate paths were set to 50, 80, 100 and 130 ms respectively. These delays correspond to 2, 5, 7 and 10 positions of displacement for the four alternative paths. An SD value of 0.5% of its mean delay was used for each of the links. The packets take the corresponding paths with probabilities P_0= 0.5, P_2= 0.25, P_5= 0.125, P_7= 0.0625 and P_{10}= 0.0625 respectively. The values of 'P' were chosen with decreasing magnitude with

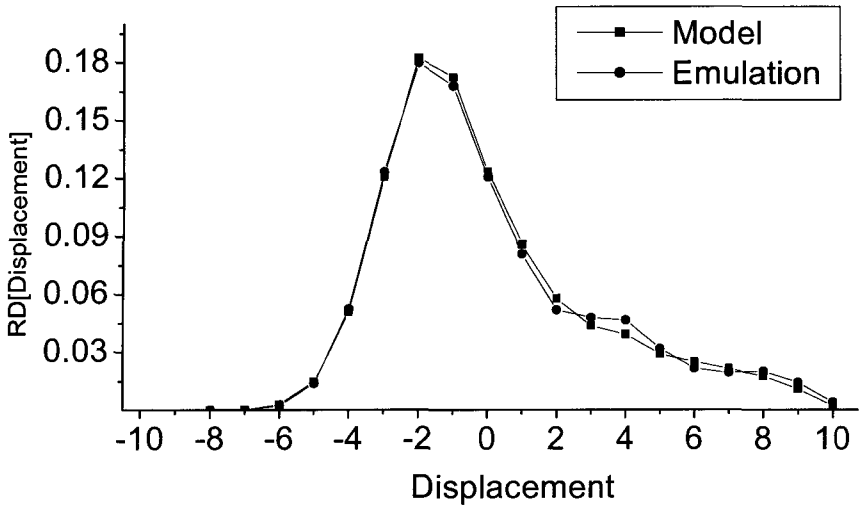


Fig. 4.10. Comparison of the RD from the model and RD obtained by emulation of multi-path load splitting topology. $P_0 = 0.5$, $P_1 = 0$, $P_2 = 0.25$, $P_3 = 0$, $P_4 = 0$, $P_5 = 0.125$, $P_6 = 0$, $P_7 = 0.0625$, $P_8 = 0$, $P_9 = 0$, $P_{10} = 0.5$

increasing delay, as we feel that it is logical to choose longer path only if shorter path is not available during load-splitting.

Fig. 4.10 illustrates the comparison between the RD derived from the model in Eq. (4.32) and RD computed at the destination of the emulated topology. As observed, the model fits with RD from emulation results. The RMS error between the two curves is only 0.0034. In the case of five-path load splitting, the earliness and lateness are not binomial distributions as they result from different Δ values associated with five different paths. As the longer paths had lower probabilities the RD path is monotonically decreasing towards earliness and lateness

4.3 Remarks

This chapter introduced the first models of packet reordering to the best of our knowledge. There are assumptions with respect to packet gaps at source. However, a TCP stream cannot be assumed to contain constant gaps and also the deviation of packet delays

following a single path could vary largely. For these cases, the model does not fail and in fact the same model with Δ of each path being a random variable is computed. We have also seen the impact of higher SD values on the agreement between models and RDs from emulated. This variance can be accounted in the expression for Δ value as suggested for packet gaps above.

Both researchers and engineers will be able to make use of the models presented in this chapter for rigorous analysis of packet reordering. The approach presented here can be generalized to consider more complex scenarios. When the study is restricted to reordering in current or past Internet, we recommend limited modeling that is less computation intensive.

Chapter V

REORDER BUFFER-OCCUPANCY DENSITY

Reorder Buffer-occupancy Density (RBD) is a metric to measure packet reordering that was proposed in [Ba02] in a simpler form. This research extended and investigated the use of Reorder Buffer-occupancy Density (RBD) as a metric for measurement of out-of-order packets, as well as for evaluation of resources required to recover from reordering, using limited modeling technique. RBD is the normalized histogram of the occupancy of a buffer that would allow the recovery from out-of-order delivery of packets [Ba02, Ja05]. This can be considered as a hypothetical buffer for measurement purposes, while an actual physical buffer is needed for recovery from reordering. A threshold on buffer size is used to declare a packet as lost. The threshold, selected based on the application requirement or the underlying transport protocol, keeps the complexity of measurement, in both time and space requirement within bounds, without affecting the accuracy of the measurement. The threshold also makes the metric robust against cases such as a burst of losses and/or duplicates. At times the early arrivals are buffered in anticipation of a packet that probably is not reordered but lost. To accommodate such cases, a variant of RBD is also presented in that makes it completely orthogonal to packet losses.

In Section 5.1 the proposed metric RBD and its computation are discussed. The choice of buffer threshold in RBD is discussed, along with its impacts on the reordering measurement, in Section 5.2. Section 5.3 presents the properties of RBD metric. The basic patterns of reordering are revisited with derivations of RBDs for these patterns. As in Section 4.1 for RD, a RBD model is presented by using the derived expression of basic patterns taking a two-path load splitting as an example parallelism scenario that causes reordering, in Section 5.4. Measurements of reordering over the Internet using RBD metric are presented in Section 5.5. In the same section, a loss-orthogonal version of RBD is presented for cases where reordering and packet losses need to be measured independently.

5.1 Reorder Buffer-Occupancy Density (RBD)

The reorder buffer-occupancy density (RBD) is defined and explained next. Without loss of generality we consider a sequence of packets (1, 2, ... N) transmitted from a source node to a destination. RBD is also applicable for cases where the sequence number is incremented in a different manner, such as with TCP, or where the sequence of packets may be identified based on timestamps.

5.1.1 Definitions

Expected Packet (E): Expected packet '*E*' is the largest sequence number such that all the packets with sequence number less than *E* have already arrived or have been identified as lost.

Buffer-occupancy (B): An arrived packet with a sequence number greater than that of expected packet is considered to be stored in a hypothetical buffer long enough to recover from reordering. At any instance of packet arrival, the buffer occupancy is equal to the number of such out-of-order packets in the buffer including the arrived packet (assuming

one buffer for each packet). For example, for sequence of arrivals (1, 2, 4, 5, 3), the corresponding expected packet values are (1, 2, 3, 3, 3). The buffer occupancy value immediately following the arrival of packet with sequence number 4 is 1; it has to be buffered as an earlier packet is still being expected. Similarly, following the arrival of packet 5, the buffer occupancy becomes 2. Finally, when the packet with sequence number 3 arrives, packets 4 and 5 can be recovered from reordering and the buffer-occupancy changes to zero.

Buffer Occupancy Threshold (B_T): Buffer occupancy threshold B_T, is a parameter that places a threshold on the maximum size of the hypothetical buffer that is used for recovery from reordering. It is also used for classification of a packet as lost during RBD computation. When the expected packet has not arrived and buffer-occupancy reaches B_T, the expected packet is classified as lost. B_T also provides robustness in addition to limiting the computation complexity of RBD.

Buffer-occupancy Frequency (FB): The buffer occupancy 'B' is recorded following each arrival. The value of 'B' ranges from 0 to B_T. The buffer occupancy frequency FB[B] is the number of times the occupancy takes the value of 'B' during the arrival of the packet sequence.

Reorder Buffer-occupancy Density (RBD): Buffer occupancy frequencies are normalized against the total number of non-duplicate packets received to compute reorder buffer-occupancy density, i.e., RBD[B] = FB[B]/N' where N' is the length of the received sequence ignoring lost and duplicate packets. N' is equal to ΣFB[B] where $B \in [0, B_T]$ and $\sum_{B} RBD[B] = 1$.

5.1.2 Evaluation of Reorder Buffer-Occupancy Density (RBD)

Let 'N' be the total number of packets in a sequence, and S be the sequence number of the received packet under consideration for computing RBD.

Example 1 - Case with no packet loss or duplication: Consider the sequence of packets (1, 2, 3, 5, 6, 7, 4, 9, 8, 11, 10). Let B_T be equal to 4. Table 5.1 shows the steps in computing buffer occupancies for RBD. Table 5.2 illustrates the computed frequency values and RBD after normalization. Frequency component for $B > 4$ is always 0, as B_T is set to 4. Fig. 5.1 shows the reorder buffer-occupancy density for the above sequence.

Table 5.1. FB computation for sequence with no packet loss

E	1	2	3	4	4	4	4	8	8
S	1	2	3	5	6	7	4	9	8
B	0	0	0	1	2	3	0	1	0
Frequency (FB[B])	*FB[0] =1*	*FB[0] =2*	*FB[0] =3*	*FB[1] =1*	*FB[2] =1*	*FB[3] =1*	*FB[0] =4*	*FB[1] =2*	*FB[0] =5*

Table 5.2. RBD corresponding to sequence (1, 2, 3, 5, 6, 7, 4, 9, 8)

Buffer-occupancy (B)	*Frequency FB[B]*	*Normalized Frequency*
0	5	0.555
1	2	0.222
2	1	0.111
3	1	0.111
4	0	0

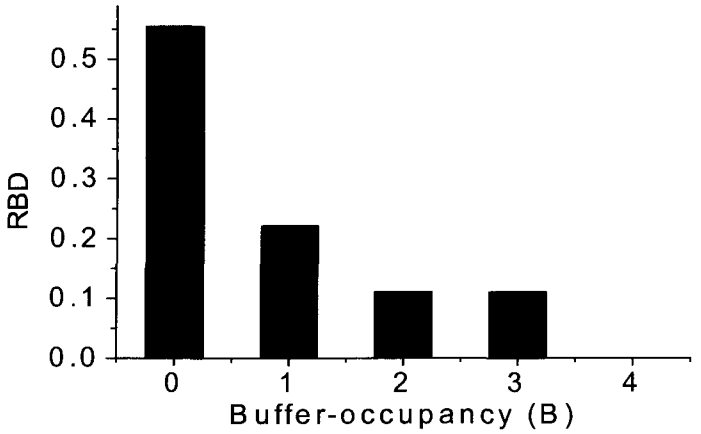


Fig. 5.1. RBD for packet sequence (1, 2, 3, 5, 6, 7, 4, 9, 8) and $B_T = 4$

Example 2 - Case of loss and duplication of packets: A certain sequence number may not be present in a given packet arrival sequence at the time it is expected, because the packet was either lost in the network or it was delayed. RBD expects such a sequence number until buffer-occupancy $B \leq B_T$, but when the buffer requirement exceeds B_T, RBD assumes the packet to be lost and proceeds. The buffer occupancy thus never exceeds B_T. Consider the arrival sequence (1, 2, 3, 5, 6, 7, 9, 10, 12, 13, 13, 11). Here, packet 4 is either lost or delayed. This packet is declared lost when the buffer becomes full while still waiting for this packet, i.e., when B_T subsequent packets have been received. Table 5.3 shows steps in computing buffer-occupancies for RBD for this example assuming B_T to be 2. At the end of the sequence, FB[0] = 5, FB[1] = 3, and FB[2] = 3 resulting in RBD, as shown in Fig. 5.2.

Table 5.3. FB computation for sequence with loss of packets

E	1	2	3	4	4	7	8	8	11	11	11	11
S	1	2	3	5	6	7	9	10	12	13	13	11
B	0	0	0	1	2	0	1	2	1	2	-	0
FB[B]	1	2	3	1	1	4	2	2	3	3	-	5

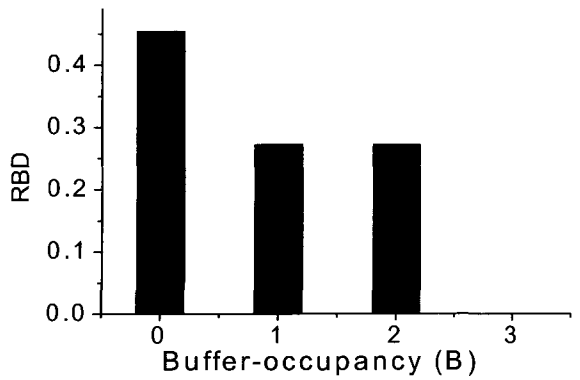


Fig. 5.2. RBD for packet sequence (1, 2, 3, 5, 6, 7, 9, 10, 12, 13, 13, 11) and $B_T = 2$

When packets are delayed beyond the buffer-occupancy threshold, the effect is the same as with lost packets. In addition, the duplicate of sequence number 13 is discarded, i.e., there is no buffer-occupancy assignment for it.

5.2 Selection of B_T

Although assigning a threshold for determining lost packets might appear to introduce an error into the reorder metrics, in practice this need not be the case. Buffer threshold for RBD has similar importance as displacement threshold for RD Applications, protocols, and the network itself operate within finite resource constraints, which introduce practical limits beyond which the choice of certain values become irrelevant. In the case of B_T, one can find a value above which B_T does not have an impact on the reorder metrics. For example, in case of a VoIP application with a bit-rate of 128kbps and packet size of 200 bytes, a practical value for B_T can be determined as follows. Assume that the application can wait a maximum of 50ms for an expected packet and that the packets arrive at constant rate. Within 50 ms, the application can receive (128*1000*0.05)/(200*8), i.e., 4 packets. Since packets arriving after this duration are effectively lost, the B_T value could be

set at 3. If the operational nature of an application is such that a value for B_T can be defined, then using that value in the computation of reordering will neither invalidate nor does limit the effectiveness of the metrics, i.e., increasing B_T further does not provide any benefit. In the case of TCP, the transmit and receive window sizes impose a natural limit on the useful value of B_T. If there are no operational constraints imposed by factors as described above, or if one is purely interested in a more complete picture of reordering, then B_T can be made as large as required. If B_T is equal to the length of the packet sequence (worst case scenario), a complete picture of reordering is seen. This requires that either the length of the packet sequence is known beforehand, or that B_T is allowed to grow without bound.

5.3 Properties of RBD

As in the case of RD metric, RBD contains an important property with respect to RS' in a sequence. In addition, there are useful relations between shapes of RBD and type of reordering in the sequence.

Property 1: Consider a sequence of 'N' packets containing's' RS', R_1, R_2, ..., R_s with lengths n_1, n_2, ..., n_s respectively. RBD of a sequence is the weighted average of RBDs of individual RS', i.e.,

$$RBD[k] = \sum_{i=1}^{s} \left(\frac{n_i}{N} * RBD_i[k]\right) \tag{5.1}$$

where $RBD_i[k]$ is the RBD for i^{th} RS.

By definition, if a RS has a p-event then all its corresponding s-events are in the same RS, and if a s-event is in a RS then all its associated s-events including the p-event that led to it exist in the same RS. Thus, the hypothetical buffer is always empty by the end of an

RS. Number of packets with buffer-occupancy 'k' in a sequence is the sum of packets with buffer-occupancy 'k' in individual RS'. Therefore,

$$FB[k] = \sum_{i=1}^{S} (n_i * RBD_i[k]) \rightarrow RBD[k] = \sum_{i=1}^{S} (\frac{n_i}{N} * RBD_i[k])$$

Property 2: If packet reordering introduced by a network consists only of late IR events, then RBD is monotonically decreasing, i.e., for non-negative integers m and n

$$\text{if } m > n \text{ then } RBD[m] \leq RBD[n] \tag{5.2}$$

Each late p-event in associated with late delivery of packets increases the buffer size incrementally till the packet of the p-event arrives, i.e., if packet 'i' gets displaced by 'd' then the buffer size increases as 1, 2, 3, ..., d, until 'i' arrives when it becomes '0'. Thus, every event corresponds to occupancy zero, 1, 2, 3 ... until its displacement making RBD monotonically decreasing.

In the following section, we will derive limited models for RBD. As in RD, we will first derive RBD expressions for basic templates. Later, the estimate for the number of basic patterns in a sequence as derived in Chapter IV are used to model RBD for this scenario.

5.4 RBD Models when Reordering Limited to Basic Patterns

As shown in Chapter III, using measurements over the Internet, basic patterns form a significant part of different reordering patterns observed over the Internet. Given RBD expressions for basic patterns, if one can estimate the number of different basic pattern RS' in a sequence then using *property* 1 of RBD, the reordering in a sequence can be computed. In the first subsection of limited modeling, the expressions of RBD for basic patterns are derived, and in the second subsection, using two-path load-splitting as a cause for reordering, the number of each basic pattern in a sequence are estimated. Later, the model based on the results from these subsections is verified against two-path emulated topologies.

5.4.1 RBD for Basic Patterns

Consider an IR of single p-event r(m, d_m) in an RS of size n_i. If p-event is lateness based then all the packets starting with the packet that came when the expected packet was 'm' till packet m's arrival, have to be buffered. The occupancy increases by one for each of such arrival. If the p-event is earliness based then packet 'm' is kept in the buffer until it is expected. The buffer size for this case remains one till it is cleared. Therefore, the RBD for IR with p-event r(m, d_m), can be summarized as:

$$\left\{ \begin{array}{l} \text{if } d_m > 0 \quad \text{RBD}[k] = 1/n_i \quad \text{for } k \in [1, d_m] \\ \quad else \quad \text{RBD}[1] = |d_m|/n_i \\ \quad \text{RBD}[0] = 1 - \sum_{k>0} \text{RBD}[k] \end{array} \right\} \tag{5.3}$$

Unless stated otherwise, in this dissertation, RBD[0] expression is same as in Eq. (5.3).

Recall from Chapter IV, following the convention of two p-events r(x, d_x) and r(y, d_y) for OR, where r(y, d_y) is the second p-event in the sequence of packets, we define a related quantity called overlap length V_{xy}. Overlap length indicates the number of sequence numbers shared by the s-event sets, if these two p-events were to occur independently.

Let us consider the derivation of RBD for each possible kind of OR patterns. Initially, for all kinds of patterns, RBD[k] for all k are initialized to zero. We use notation of increment similar to C/C++ language during the assignment in derivations. Dashed lines in figures below correspond to r(y, d_y) p-event. (Refer to Table 3.2 for notations)

(i) *Earliness overlaps with earliness:* $d_x < 0,\ d_y < 0$

Due to early arrival of packet x, it has to be buffered until it is expected. However, even before it is expected packet 'y' comes early adding to the occupancy of the buffer.

Thus, the occupancy of buffer is one from the arrival of packet 'x' to the arrival just before packet 'y' (d_x-V_{xy} +1), and arrivals after the expected position of packet 'x' till the expected position of packet 'y' (d_y -V_{xy}). The occupancy of buffer is two when packet 'y' arrives and stays two for all overlap region packets except the last one. This last overlap packet belongs to the expected place for 'x'. Therefore, RBD for a RS of size n_i with only single OR of type earliness overlaps with earliness and no other patterns is given as:

$$\left\{\begin{array}{l} \text{RBD}[1] \mathrel{+}= (-d_x - d_y + 1 - 2*V_{xy})/n_i \\ \text{RBD}[2] \mathrel{+}= -V_{xy}/n_i \end{array}\right\} \tag{5.4}$$

Consider an example sequence (1, 2, 7, 3, 4, 9, 5, 6, 8, 10) at the receiver. Let us assume that packets 7 and 9 are associated with early p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(7, -4) and r(9, -3). Overlap region contains packets 5 and 6. Therefore, V_{xy} = 2. From Eq.(5.4), (recall RBD[k] for all k is initialized to zero), RBD[1] += $(-d_x - d_y + 1 - 2*V_{xy})/n_i$ = 4/10, corresponding to the arrivals of packets 7, 3, 4 and 6 and RBD[2] += V_{xy}/n_i = 2/10; corresponding to the arrivals of packets 9 and 5. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] = 4/10.

(ii) *Earliness overlaps with lateness:* $d_x > 0,\ d_y < 0$

Packet 'x' is late by d_x positions, i.e., all the associated s-event packets will be buffered till the arrival of 'x'. After the arrival of 'x', the buffer still contains one packet, which is 'y', as this packet arrived early ($-d_y$ -1- V_{xy}). This packet is used when the last packet of its associated s-event arrives. Therefore, RBD for a RS of size n_i, with only single OR of type earliness overlaps with lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} \text{RBD}[1] \mathrel{+}= (-d_y - 1 - V_{xy})/n_i \\ \text{RBD}[k] \mathrel{+}= 1/n_i \quad \textit{for } k \in [1, d_x] \end{array}\right\} \tag{5.5}$$

Note, the second line of Eq (5.5) is a loop assignment.

Let us assume that packets 4 and 10 are associated with late and early p-events respectively in a received sequence (1, 2, 3, 5, 10, 6, 7, 8, 4, 9). Then r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 5) and r(10, -5). Overlap region contains packets 6, 7 and 8. Therefore, $V_{xy} = 3$. From Eq.(5.5), RBD[1] += $(-d_y - 1 - V_{xy})/n_i = 1/10$ corresponding to the arrival of packet 4, RBD[1] += 1/10 resulting in RBD[1] = 2/10, corresponding to the arrival of packet 5, RBD[2] += 1/10, corresponding to the arrival of packet 10, RBD[3] += 1/10 corresponding to the arrival of packet 6, RBD[4] += 1/10, corresponding to the arrival of packet 7 and RBD[5] += 1/10, corresponding to the arrival of packet 8. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] – RBD[3] – RBD[4] – RBD[5] = 1-2/10-1/10-1/10-1/10-1/10 = 4/10.

(iii) *Lateness overlaps with lateness:* $d_x > 0$, $d_y > 0$

As packet 'x' is late, all the packet arrivals till the arrival of 'x' are buffered, i.e., buffer-occupancy goes from one to d_x. However, when packet 'x' arrives, the buffer cannot be emptied as 'y' has left its place. All the packets in overlap region have to be in the buffer, thus the occupancy is V_{xy}. This occupancy keeps increasing till packet 'y' arrives, at which point the buffer is emptied. Therefore RBD for a RS of size n_i, with only single OR of type lateness overlaps with lateness, and no other patterns is given as:

$$\left.\begin{array}{ll} \text{RBD}[k] \mathrel{+}= 1/n_i & \text{for } k \in [1, d_x] \\ \text{RBD}[k] \mathrel{+}= 1/n_i & \text{for } k \in [V_{xy}, d_y] \end{array}\right\} \qquad (5.6)$$

Consider the arrival of packet with their sequence numbers in order (1, 2, 3, 5, 7, 8, 4, 9, 6, 10) at the receiver. Given packets 4 and 6 are associated with late p-events, r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 3) and r(6, 3). Overlap region contains packets

7 and 8. Therefore, V_{xy} = 2. From Eq. (5.6), RBD[1] += 1/10, corresponding to the arrival of packet 5, RBD[2] += 1/10, corresponding to the arrival of packet 7, RBD[3] += 1/10, corresponding to the arrival of packet 8, RBD[2] += 1/10 => RBD[2] = 2/10, corresponding to the arrival of packet 4, and RBD[3] += 1/10 resulting in final RBD[3] = 2/10, corresponding to the arrival of packet 9. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] – RBD[3] = 1-1/10-2/10-2/10 = 5/10.

(iv) *Lateness overlaps with earliness:* $d_x < 0,\ d_y > 0$

Packet 'x' is buffered until the position where it is expected, as it is associated with early p-event. However due to lateness of packet 'y', the buffer is also occupied by packets in the overlap and beyond. Thus, buffer-occupancy remains one till the overlap region, and increases one packet at a time in the overlap region and beyond. Therefore, RBD for a RS of size n_i, with only single OR of type lateness overlaps with earliness, and no other patterns is given as:

$$\left\{\begin{array}{l} \text{RBD}[1] \mathrel{+}= (-d_x - V_{xy})/n_i \\ \text{RBD}[k] \mathrel{+}= 1/n_i \quad \textit{for } k \in [2, d_y] \end{array}\right\} \tag{5.7}$$

A sequence of ten packets is sent at the receiver and they arrive in the order (1, 6, 2, 4, 5, 7, 3, 8, 9, 10). If packet numbers 3 and 6 are associated with p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(6, -4) and r(3, 4). Overlap region contains packets 4 and 5. Therefore, V_{xy} = 2. From Eq. (5.7), RBD[1] += $(-d_x - V_{xy})/n_i$ =2/10, corresponding to the arrivals of packets 6 and 2, RBD[2] += 1/10, corresponding to the arrival of packet 4, RBD[3] += 1/10, corresponding to the arrival of packet 5, and RBD[4] += 1/10, corresponding to the arrival of packet 7. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] – RBD[3] – RBD[4] = 1 – 2/10 – 1/10 –1/10 – 1/10 = 5/10

RBD expressions for ER patterns can be classified into two categories. One with the enclosing p-event being late and the other with enclosing p-event being early. The events involved are denoted by r(x, d_x) and r(y, d_y) events with packet of sequence number 'y' enclosing event with packet number 'x'. In the former category above, the inner event has no impact on buffer-occupancy as all the packets have to be buffered irrespective of any internal reordering, as packet 'y' is late and is still expected. In the latter category, the buffer size is one till the point of enclosed event. At the enclosed event, the occupancy will change to two, if it is earliness enclosing earliness, otherwise, it will increase step wise till the end of the enclosing event and becomes one again at the expected place of 'x'. The expressions for RBD for ER patterns are given below (using the notations in Table 3.2):

(i) *Earliness encloses earliness:* $d_x < 0,\ d < 0$

This pattern belongs to the second category as discussed above. The buffer is filled by packet 'y' and when packet 'x' arrives, it is added to the buffer too. At the end of enclosed event, packet 'x' is used by the application bringing down the buffer-occupancy to one again. The buffer is emptied at the end of the pattern. Therefore, RBD for a RS of size n_i, with only single ER of type earliness enclosing earliness, and no other patterns is given as:

$$\left.\begin{array}{l} \text{RBD}[1] \mathrel{+}= (d_x - d_y - 1)/n_i \\ \text{RBD}[k] \mathrel{+}= (-d_x + 1)/n_i \end{array}\right\} \tag{5.8}$$

With r(x, d_x) and r(y, d_y) in the notation corresponding to r(5, -1) and r(7, -5) respectively in a received sequence (1, 7, 2, 5, 3, 4, 6, 8, 9, 10), Eq. (5.8), RBD[1] += $(d_x - d_y - 1)/n_i = 3/10$, corresponding to the arrivals of packets 7, 2 and 4, and RBD[2] +=

$(-d_y+1)/n_i$ = 2/10, corresponding to the arrivals of packets 5 and 3. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] = 1-3/10-2/10 = 5/10.

(ii) *Earliness encloses lateness:* $d_x > 0,\ d_y < 0$

This pattern also belongs to the second category as classified earlier. The buffer size is one till the enclosed event. At the point of enclosed event, the buffer size increments one packet at a time expecting 'x'. At the arrival of 'x' buffer-occupancy returns to one carrying packet 'y' again. Buffer is emptied at the end of the pattern. Therefore, RBD for a RS of size n_i, with only single ER of type earliness enclosing lateness, and no other patterns is given as:

$$\left\{\begin{array}{l} \text{RBD}[1] \mathrel{+}= (-d_x - d_y + 1)/n_i \\ \text{RBD}[k] \mathrel{+}= 1/n_i \quad for\ k \in [2, d_x] \end{array}\right\} \tag{5.9}$$

Consider an example sequence (1, 7, 2, 4, 5, 3, 6, 8, 9, 10). Let us assume that packets 7 and 3 are associated with p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(3, 3) and r(7, -5) respectively. From Eq. (5.9), RBD[1] += $(-d_x - d_y + 1)/n_i$ = 3/10, corresponding to the arrivals of packets 7, 2 and 3, RBD[2] += 1/10, corresponding to the arrival of packet 4 and RBD[3] += 1/10, corresponding to the arrival of packet 5. From Eq. (5.3), RBD[0] = 1- RBD[1] – RBD[2] –RBD[3] = 1-3/10-1/10 –1/10 = 5/10.

(iii) *Lateness encloses lateness:* $d_x > 0,\ d_y > 0$

This pattern belongs to the first category of ER where packet 'y' is late and irrespective of any reordering that occurs before its arrival, all the packets have to be buffered. Therefore, RBD for a RS of size n_i, with only single ER of type lateness enclosing lateness, and no other patterns is given as: :

$$\text{RBD}[k] \mathrel{+}= 1/n_i \quad for\ k \in [1, d_y] \tag{5.10}$$

In sequence (1, 3, 5, 6, 7, 4, 2, 8, 9, 10), let packets 4 and 2 are associated with late p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(4, 2) and r(2, 5) respectively. From Eq.(5.10), RBD[1] += 1/10, corresponding to the arrival of packet 3, RBD[2] += 1/10, corresponding to the arrival of packet 5, RBD[3] += 1/10, corresponding to the arrival of packet 6, RBD[4] += 1/10, corresponding to the arrival of packet 7 and RBD[5] += 1/10, corresponding to the arrival of packet 4. From Eq. (5.3), RBD[0] = 1-5/10 = 5/10.

(iv) *Lateness encloses earliness:* $d_x < 0,\ d_y > 0$

This ER basic pattern also belongs to first category, i.e., all the packets have to be buffered till packet 'y' arrives. Thus, the RBD for a RS of size n_i, with only single ER of type lateness enclosing earliness, and no other patterns is given as: :

$$\text{RBD}[k] \mathrel{+}= 1/n_i \quad \textit{for } k \in [1, d_y] \tag{5.11}$$

Consider an example sequence (1, 3, 4, 7, 5, 6, 2, 8, 9, 10). Let us assume that packets 7 and 2 are associated with p-events. Then r(x, d_x) and r(y, d_y) in the notation correspond to r(7, -3) and r(2, 5) respectively. From Eq. (5.11), RBD[1] += 1/10, , corresponding to the arrival of packet 3, RBD[2] += 1/10, corresponding to the arrival of packet 4, RBD[3] += 1/10, corresponding to the arrival of packet 7, RBD[4] += 1/10, corresponding to the arrival of packet 5 and RBD[5] += 1/10, corresponding to the arrival of packet 6. From Eq. (5.3), RBD[0] = 1-5/10 = 5/10.

Recall the two-path load splitting example from Sections 4.1.1 and 4.1.2, where path were considered for a packet to traverse and the probability that a packet takes (alternate) longer path is *P* and primary path *1-P*. The offset suffered by a longer path with respect to the packets is primary path is Δ. We restate the count of basic patterns here, given a packet stream of size 'N', for quick reference:

The number of IR patterns with displacement 'Δ' is:

$$\text{IR\_Count\_with\_}\Delta \cong NP(1-P)^{2\Delta-1} \tag{5.12}$$

The number of IR patterns with displacement 'Δ-1' is:

$$\text{IR\_Count\_with\_}\Delta\text{-}1 \cong NP^2(1-P)^{2\Delta-2} \tag{5.13}$$

The number of OR patterns for a large received sequence of N, is approximated as:

$$\text{OR\_Count} \cong \sum_{V_{xy}=1}^{\Delta-1} N(1-P)^{3\Delta-2-V_{xy}} P^2 \tag{5.14}$$

In the following subsection, we will use these counts and apply the expressions for RBD derived in this subsection, to obtain the model of reordering for two-path load splitting scenario.

5.4.2 RBD Model for Two-Path Load Splitting

Given a sequence of size N, the counts of RS' corresponding to basic patterns are given by Eqs., (5.12), (5.13) and (5.14) in terms of P and Δ. The RBD assignment for this sequence, $RBD_{(P,\Delta)}[k]$, using the RBD assignments for IRs and ORs represented are given as:

$$\begin{aligned} RBD_{(P,\Delta)}[k] = (&\text{IR\_Count\_with\_}\Delta\text{*RBD}[k] \text{ for IR with displacement of } \Delta \\ &+ \text{IR\_Count\_with\_}\Delta\text{*RBD}[k] \text{ for IR with displacement of } \Delta-1 \\ &+ \sum_{Vxy=1}^{\Delta-1} \text{OR\_Count*RBD}[k] \text{ for OR lateness overlaps lateness}) / N \end{aligned} \tag{5.15}$$

Using the RBD expressions from subsection 5.4.1, the above equation for positive values of 'k' can be written as:

$$\mathrm{RBD}[k] = \begin{cases} (1-P)^{2\Delta-1}P + \sum_{V_{xy}=1}^{\Delta-1} (1-P)^{3\Delta-2-V_{xy}} P^2 & \text{for } k = \Delta \\ (1-P)^{2\Delta-2}P + \sum_{V_{xy}=1}^{\Delta-1} (1-P)^{3\Delta-2-V_{xy}} P^2 + \sum_{V_{xy}=1}^{k} (1-P)^{3\Delta-2-V_{xy}} P^2 & \text{for } 0 < k < \Delta \\ 0 \text{ for } k > \Delta \end{cases} \quad (5.16)$$

5.4.3 Emulation and Verification

The emulated topology in Section 4.1.4, in particular Fig. 4.3, was used to verify the accuracy of the model. Once again, P values were varied from 0.01 to 0.07. Fig. 5.3 illustrates RBD from the derived model using Eq.(5.16), and from measurements using emulation that computed RBD on the fly. The vertical axes are in log10 scale to clearly show all components of the RBD. It was observed that the RBD from model follows the RBD obtained from emulation for low values of P. The RMSE values for the RBDs from models and emulations for P value of 0.01, 0.03, 0.05 and 0.07 are 7.03E-4, 0.003, 0.0087 and 0.02029 respectively. Thus, as P increases beyond 0.07, the number of multiple overlap events with more than two p-events becomes significant leading to disagreement of RBD from emulation with RBDs from models. By considering three p-event-overlap cases during the analysis, the model could be extended to follow the RBD from emulated measurements even for higher values of I. Moreover, this process could be continued by including four p-event-overlap cases etc., depending on the precision that one requires from the model.

Models of reordering not only provide an estimation of buffer size requirement to recover from reordering but also illustrate the occupancies of such buffers. For example in a shared memory scenario, if recovery from reordering is attempted then such occupancies emulate the resource scheduling for recovery. Such estimations will help evaluate jitter buffer variation and network suitability for real-time applications, e.g., IP telephony. The buffer occupancy distribution can also be used for efficient allocation of buffers at nodes

that have to deal with resequencing of multiple streams. Modeling of reordering will help in understanding of packet loss variation too, given limited buffer resources.

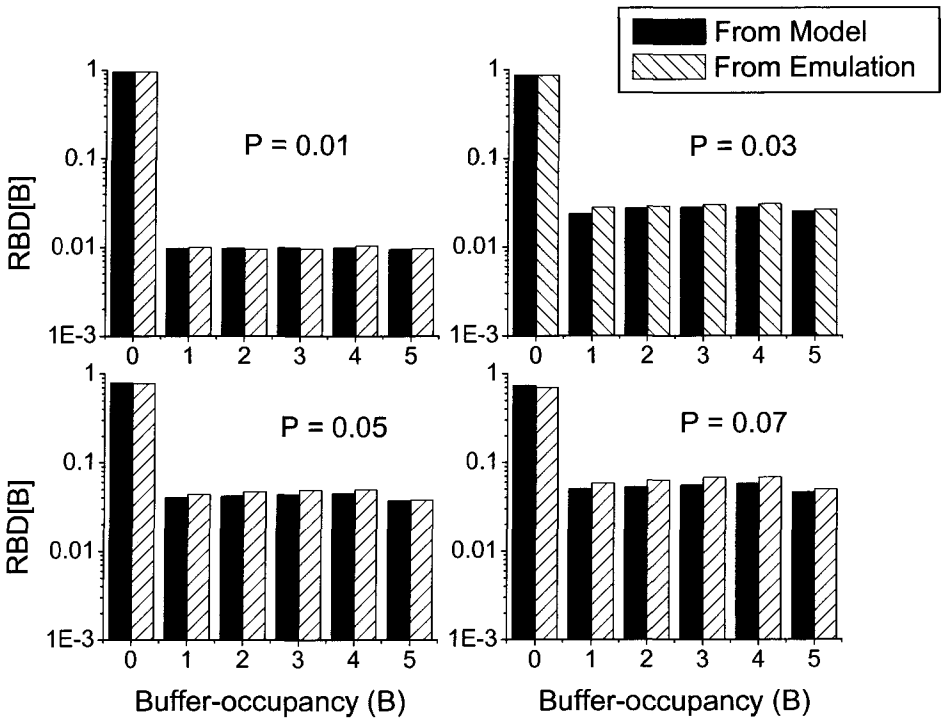


Fig. 5.3. Comparison of RBD derived from model and measured using emulations for *P* = 0.01, 0.03, 0.05 and 0.07

5.5 Measurement of Internet Reordering using RBD

Consider a sequence without reordering, e.g., (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). Since there is no reordering, RBD has a single component of value 1 for buffer-occupancy equal to zero. However, if there is no reordering in a sequence but a single packet is lost, e.g., (1, 2, 3, 4, 5, 6, 7, 9, 10, 11), then the RBD will have components for B =1, 2, 3 corresponding to the arrivals of packets 9, 10 and 11 as packet 8 is expected (assuming B_T = 10) until the end of the sequence. This occurs due to RBD's inability to detect lost packets until later. Thus,

even without packet reordering, the measure of reordering changes with lost packets. It is arguable as to whether a reorder measure should be independent of lost packets or not. A metric based on reorder recovery mechanism, that is sensitive to both reordering and losses, is more useful for applications than the one that could provide a pure measure of reordering that is independent of lost packets. However, it is possible to measure RBD using threshold for loss detection (as in RD), to make it orthogonal to loss by compensating for the losses. In a streaming environment, one of the following methods can be used for the evaluation of RBD:

Go-back Method: In this method, RBD is computed as packets arrive, and if the expected packet is not received after threshold positions, it is classified as lost. At this point, RBD is recomputed (adjusted) for the already arrived packets with sequence numbers greater than the lost packet. Note that the repeated computation of RBD occurs only when a packet is lost. Consider the sequence (1, 2, 4, 5, 6, 7, 8, 9, 10) and $B_T = 3$. This sequence has no reordering. However, we assign buffer-occupancies 1, 2 and 3 for packet arrivals 4, 5 and 6 respectively. Due to B_T value, the packet 3 is now declared lost. Therefore, we go-back B_T positions and reassign buffer-occupancies. The packet arrivals 4, 5 and 6 have expected sequence numbers 4, 5 and 6 respectively, leading to no buffering.

Stay-back Method: Alternatively, the computation of RBD can be allowed to lag up to B_T positions when a packet is missing. When the expected packet does not arrive in next threshold steps, E is updated to the following sequence number. Once again, let us consider the sequence (1, 2, 4, 5, 6, 7, 8, 9, 10) and $B_T = 3$. For each step, we wait till next B_T arrivals. Therefore, when packet 4 arrives, we wait for the arrival of two more packets, i.e., arrivals

of 5 and 6. Since packet 3 has not arrived, it is considered lost and E is assigned to 4 instead of 3.

These methods are similar to the methods we use for RD. In both the approaches, buffering of the size of threshold is required. B_T is the threshold placed on the buffer-occupancy of hypothetical buffer used to recover from reordering. If a packet arrives after this buffer is full and it is not the expected one, then the expected one is as good as lost. Thus, B_T is used as a threshold to detect lost packets in both Go-back and Stay-back methods. Finally, to discard duplicate packets, any one of the copies of a packet could be treated as a duplicate and discarded. However, an application normally uses the first copy. Thus, we may safely assume that subsequent copies that arrive after the first one are duplicate packets. To detect a duplicate packet, we check an arriving packet against the expected sequence number and any early arrivals in the hypothetical buffer. If an arriving packet is less than the expected packet's sequence number then it is discarded, as it may be a duplicate or a packet that is deemed as lost earlier. The algorithm for computing the loss-orthogonal RBD with Stay-back method is shown in Appendix C.

In Chapter III, we showed results corresponding to download of large files from a set of nine servers available on the Internet. The sequence of the packets was depicted using RD. One the same data, RBD was also used as a metric to measure packet reordering. Fig. 5.4 illustrates the amount of packet reordering using RBD metric, using Stay-back method.

RBD provides a comprehensive measure of reordering and we can draw a number of inferences from these measures:

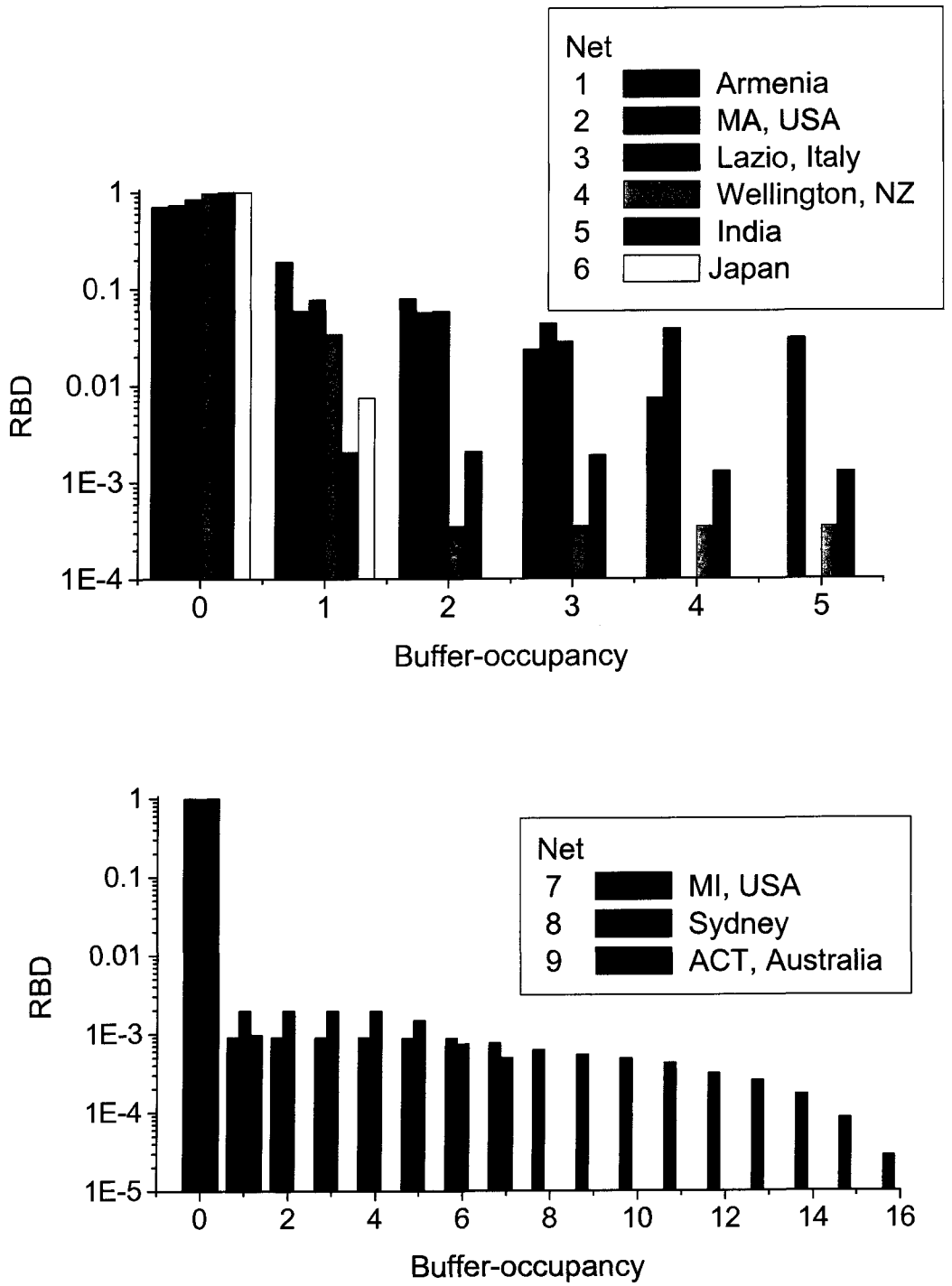


Fig. 5.4. RBDs for reordering over the Internet for Net-1 to 9 (B_T = 20)

1. Net-1 requires a buffer size of four for recovery. However, the usage of buffer is high. Similarly for Net-2, the buffer size required is five and the occupancies are high. However, Net-3 and Net-4 also need a buffer-size of five but the occupancies are very low. Thus, a shared-memory type scenario is more useful for Net-3 and 4 than Net 1 and 2
2. Net-2 has the smallest average delay, but due to larger deviations, the amount of reordering is comparatively high. Conversely, looking at the shape of RBD, we can comment on the delay deviations in the network. The wider the RBD spread, the higher the variance in delay.
3. For Net-2, the application can recover from reordering by having a buffer size equal to three packets whereas Net-1 and Net-3 require a buffer size of five packets and one packet respectively.
4. Net-6 and Net-9 have a buffer size requirement of one packet. As observed in the case of RD, the packets are late by a maximum position of one.

5.6 Remarks

There has been lot of discuss at IPPM forum of IETF on RBD. It is now recognized as a very useful metric. Though there were issue with loss sensitivity of RBD, introduction of threshold based loss detection solved this problem. However, there are practical reasons why RBD is more useful with loss-orthogonal property. It has a one-to-one relation with required buffering at the receiver. RBD is very useful to study queuing at the intermediate and endpoints that could occur while attempting to recover from reordering.

Chapter VI

COMPARISON OF PACKET REORDERING METRICS

In this chapter we will discuss packet reordering metrics that are proposed by us and other research teams working in parallel with this research. Section 6.1 introduces the metrics Reordering Extent and n-Reordering. In Section 6.2, a framework is developed to compare all these metrics. The basis of the framework is a set of essential and desirable attributes, in subsections 6.2.1 and 6.2.2 respectively, which contain simplicity, usability and other practical issues. Later in Section 6.3, these metrics are used to measure packet reordering over the Internet, and inferences are drawn from the observations that depict the merits and demerits of the metrics.

6.1 Packet Reordering Metrics

Percentage of reordered packets has often been used as a measure to evaluate reordering [Be99, Li02, Ia01]. However, as discussed in Chapter II, its definition is vague, and no uniform definition exists. One main disadvantage of this metric is its inability to quantify the displacement of packets in a sequence.

The amount of displacement of a packet directly influences the complexity of hardware/software to recover from reordering. In the sequence (1, 3, 4, 2, 5), if buffers are

available to store packets 3 and 4 while waiting for packet 2, it is possible to recover from the reordering. However, there may be cases where the application requirement is such that the arrival of packet 2 after this delay renders it useless. While one can argue that a good packet reordering measurement scheme should capture such effects, a counter argument can also be made that packet reordering should be measured strictly with respect to the order of delivery and not application dependent.

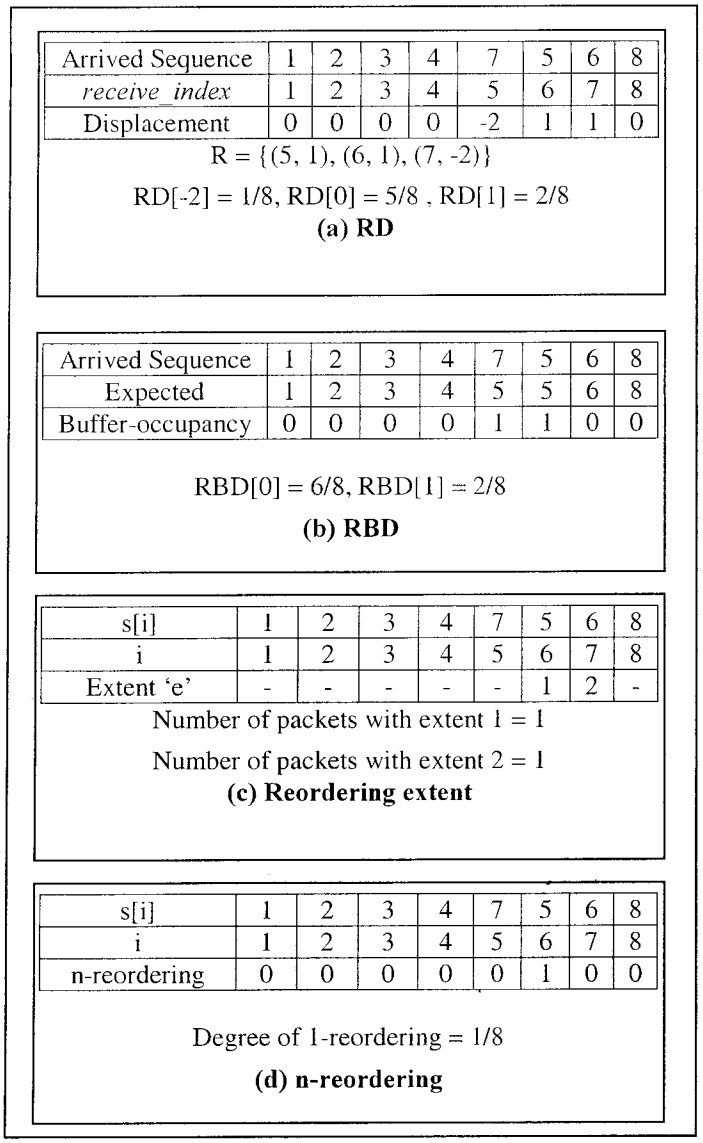
Next we present two other metrics that have been proposed for measurement of reordering, namely Reordering extent and n-Reordering.

6.1.1 Reordering Extent [Mo05]

Reordering Extent, Reordering ByteOffset and n-Reordering [Mo05] are lateness-based metrics, i.e., a packet is not considered reordered unless it is late. Reorder ByteOffset metric in [Mo05] is similar to the RBD [Ba02, Ja05], and it can be derived from RBD.

The reordering extent is the maximum distance, in packets, from a late packet to the earliest packet received that has a larger sequence number. If a packet is in-order, its reordering extent is undefined. The first packet to arrive is in-order by definition, and has undefined reordering extent. Consider a stream of packets (1, 2,.., N), and let N' be the total number of packets received out of the N packets sent at source. Reordering extent assumes that duplicates are removed, so N' <= N, the difference corresponding to losses. Let s[1], s[2],.. s[N'] represent the original sequence numbers associated with the packets in order arrival. Consider a reordered packet with arrival index i and source sequence number s[i]. Note that values of i are same as the *receive_index* values defined in RD. There exists a set

of indices j ($1 <= j < i$) such that s[j] > s[i]. The reordering extent, e, of packet s[i] is defined to be (i-j) for the smallest value of j such that s[j] > s[i].



Arrived Sequence	1	2	3	4	7	5	6	8
receive_index	1	2	3	4	5	6	7	8
Displacement	0	0	0	0	-2	1	1	0

R = {(5, 1), (6, 1), (7, -2)}

RD[-2] = 1/8, RD[0] = 5/8 , RD[1] = 2/8

(a) RD

Arrived Sequence	1	2	3	4	7	5	6	8
Expected	1	2	3	4	5	5	6	8
Buffer-occupancy	0	0	0	0	1	1	0	0

RBD[0] = 6/8, RBD[1] = 2/8

(b) RBD

s[i]	1	2	3	4	7	5	6	8
i	1	2	3	4	5	6	7	8
Extent 'e'	-	-	-	-	-	1	2	-

Number of packets with extent 1 = 1

Number of packets with extent 2 = 1

(c) Reordering extent

s[i]	1	2	3	4	7	5	6	8
i	1	2	3	4	5	6	7	8
n-reordering	0	0	0	0	0	1	0	0

Degree of 1-reordering = 1/8

(d) n-reordering

Fig. 6.1. Example of reordered sequence (1, 2, 3, 4, 7, 5, 6, 8) with corresponding (a) RD, (b) RBD, (c) Reordering extent and (d) n-Reordering

Fig. 6.1 (a), (b), (c) and (d) depict the reorder measure from RD, RBD, Reordering extent and n-Reordering respectively, for packet sequence (1, 2, 3, 4, 7, 5, 6, 8). Consider the arrival of the packet with sequence number 5, i.e., s[i] = 5 and i = 6. The value of j for which

s[j] > s[i] is 5 as (s[5] = 7) and (s[6] = 5). Therefore, the extent e of reordering for this arrival = i-j = 6-5 = 1. Considering the arrival of the packet with sequence number 6, i.e., s[i] = 6 and i = 7. The value of j for which s[j] > s[i] is 5 as (s[5] = 7) and (s[7] = 6). Therefore, the extent e of reordering for this arrival = i-j = 7-5 = 2. For other packet, the extent is undefined.

For both the arrivals considered above there are no values of j *less* than 5, such that s[j] > s[i]. The sequence number s[j] for the lowest value of j is termed as the point of discontinuity. Thus, for both the arrivals the point of discontinuity is the same and is equal to 7. Note that no reordering extent is defined for packet s[i] =7, as the metric implicitly assumes the early packets not to be reordered. Moreover, the frequencies of the extents in a received sequence could be used to obtain a histogram of reordering extent (e), though such a representation is not a part of the metric's definition, we observe that it is useful to interpret reordering.

6.1.2 n-Reordering [Mo05]

n-Reordering defines the extent as the maximum distance, in packets, from reordered packet to the earliest packet received that has a larger sequence number. However, unlike reordering extent, the definition of reordering is skewed. For example, if packets are late in sets, i.e., two or more consecutive packets are late but maintain their positions with respect to each other then only the first packet is considered as reordered.

Using the same conventions as in reordering extent, a packet number i and with source sequence number s[i] is n-reordered (n < i <= N), *iff for all* j such that i-n <= j < i, s[j] > s[i]. The degree of n-reordering of the sample is m/N', where m is the number of n-

reordered packets in the sample and N' is the size of the arrival sequence after discarding duplicate packets. A sample is said to have no reordering if its degree of n-reordering is 0 for all n (1<= n < N'). Fig. 6.1(d) illustrates n-reordering for sequence (1, 2, 3, 4, 7, 5, 6, 8). Consider the received packet number i = 7, and its source sequence number s[7] = 6. This packet does not have any consecutive values of j corresponding to the condition, *iff for all j*, in n-reordering metric's definition. Thus, it has no 'n' parameter associated with it.

There are other reorder metrics as discussed in Section 2.1.4, whose usefulness is minimal. All the metrics discussed so far can use time stamps to derive the order instead of sequence numbers, and provide a reorder measure with very little modification.

In the following section, a frame work is developed for packet reordering metrics that will allow to compare the above two metrics along with RD and RBD.

6.2 Attributes of Reorder Metrics

Previous section presented different metrics for measurement of packet reordering. We know that RD takes into account the earliness and lateness of packets in a sequence. RBD evaluate reordering from the point of view of resources required to recover from reordering, while reordering extent and n-reordering identify only the late packets as reordered packets. In this section, we discuss three essential attributes of a reorder metric, and four desirable attributes for reorder metrics.

6.2.1 Essential Attributes

Packet reordering metrics must meet a certain set of features without which the metric can be deemed incomplete. Obviously, this set includes the ability to capture reordering, which must account for both earliness and lateness. The other attributes includes the independence of reorder measurements with respect to packet loss and duplication, and usefulness as stated in [Pax98]. In this section, we will discuss these attributes in detail.

1. Capture Reordering

The first and foremost requirement of a packet reorder metric is its ability to capture the amount and extent of reordering in a sequence of packets. Fulfillment of other attributes is of no use if this property is not satisfied. Consider the sequence (1, 15, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14). If a metric is based only on early arrivals, then packet 15 is identified as early, i.e., out of order. However, a lateness-based metric would indicate all the packets from 2 to 14 to be out of order, even though it is more likely that a single packet arrived early. A metric based only on earliness or only on lateness captures only a part of information associated with reordering. A metric capturing both early and late arrivals in contrast provides a complete picture of reordering in a sequence. n-Reordering captures only a part of late arrival based reordering, i.e., it treats a subset of late arrivals as out of order. For example, in the sequence illustrated in Fig. 6.1(d), packet 6 is considered in-order though it came after packet 7 in the sequence.

RD captures both late and early packets along with their extent. As observed in Fig. 6.1(a), the early packet 7 and late packets 5 and 6 underwent non-zero displacements associated with the arrivals. Thus, RD satisfies this fundamental requirement to full extent.

In the case of RBD, the early packets are buffered, until they can be used, which happens when late packets arrive. Non-zero occupancy components of these metrics represent early arrivals. However, the zero-occupancy component may contain both in-order and late arrivals. Thus, RBD fulfills the reorder capture requirement partially. Reordering Extent is based only on late packets, by definition.

2. Low Sensitivity to Packet Loss and Duplication

A reorder metric must treat only an out-of-order packet as reordered, i.e., if a packet is lost during transit then this must not result in its following packets, which arrive in order, to be classified as out of order. Consider the sequence (1, 3, 4, 5, 6). If packet 2 has been lost, the sequence should not be considered to contain any out-of-order packets. Similarly, if multiple copies of a packet (duplicates) are delivered, this must not result in a packet being classified as out of order, as long as one copy arrives in the proper position. For example, sequence (1, 2, 3, 2, 4, 5) has no reordering. The lost and duplicate packet counts may be tracked using metrics specifically to measure those, e.g., percentage of lost packets, and percentage of duplicate packets. In RD, a packet is declared lost if it does not arrive within a threshold. In RBD, a packet is considered lost if the number of buffered packets exceeds the buffer threshold. The use of a threshold allows these metrics to tradeoff the accuracy and complexity. Other metrics that do not rely on a threshold assume either that a packet is lost (by default) until it arrives, or that the packet is not lost until the sequence is complete, and these metrics expect a sequence with duplicates removed for computation of reorder measure.

3. Usefulness

Rather than being a mere value or a set of values that changes with the reordering of packets in a stream, a reorder metric should serve some purpose. An application or a transport protocol implementation, for example, may be able to use the reordering information to allocate resources to recover from reordering. A network diagnosis tool may use the metric to rule in or rule out certain causes of reordering. In this paper, usefulness is defined below with respect to three features, namely a) flow control, b) resource allocation, and c) network diagnosis

Usefulness in TCP Flow Control: For example, lateness displacements from RD may be used to adjust the "dupthresh" parameter in TCP. Conversely, if "dupthresh" value is fixed then RD can provide an upper and a lower bound to the number of unnecessary TCP retransmits. An example to scenario is explained below:

Consider Fast Retransmit definition from RFC 2581 and delayed ACK criterion from RFC 1122. Assume there is no packet duplication within the network. In addition, if a packet is reordered then the following packets in the sequence are not reordered with same or higher displacement. For such cases, a TCP packet is retransmitted after its ACK is not found in four attempts. Table 6.1 depicts two such cases. In Table 6.1 (a) the packet 3 arrives after packet 6 where as in Table 6.1 (b) packet 2 arrives after packet 5. The sender sends two packets at a time, given acknowledgements are in order, otherwise it sends only one packet.

Arrivals	1	2		4		5		6		3
ACKs			3		3		3		3	
receive_index	1	2		3		4		5		6
Displacement	0	0		-1		-1		-1		3

(a)

Arrivals	1		3		4		5		2
ACKs		3		3		3		3	
receive_index	1		2		3		4		5
Displacement	0		-1		-1		-1		3

(b)

Table 6.1. TCP packet sequence numbers and ACKs when (a) Packet 3 arrives out of order (b) Packet 2 arrives out of order at the receiver

Hence, the displacement (according to RD definitions) is greater than 2 for retransmitted packets. Therefore, "Sum of (all RD[k] for k >= 2) corresponds to fraction of packets that is unnecessarily retransmitted."

From [Mo05], knowledge of n-reordering is particularly useful for determining the portion of reordered packets that can or cannot be restored to order in a typical TCP receiver buffer based on their arrival order alone (and without the aid of retransmissions). For example, if n equal to three component exists in an n-reordering measure, a New Reno TCP sender would retransmit 1 packet in response to an instance of 3-reordering and therefore consider this packet lost for the purposes of congestion control (the sender will halve its congestion window). Detecting instances of 3-reordering is useful for determining the portion of reordered packets that are in fact as good as lost.

Usefulness in Buffer/Resource Allocation in Reorder Recovery: RBD emulates sequencing packets back in order without any exceptions, and thus, may be used to determine the buffer requirements to recover from reordering. Reordering extent 'e' value is suggested in estimations of buffer requirement to recover from reordering, but it fails to do it

when there is packet reordering that involves consecutive packets arriving late. For example, in the sequence (1, 2, 3, 4, 7, 5, 6, 8) (also shown in Fig. 6.1(c)), the buffer requirement before the arrival of 6 is one, but using the extent value the buffer requirement is estimated as two packets. However, from Fig. 6.1(a), the estimate of buffer size requirement is still one from RD, for this arrival.

Usefulness in Network Diagnosis: RD can relate the causes and effects of reordering and in turn can point to problems in the networks, as discussed in Chapter III. As an example, if an intermediate node suffers from route fluttering then RD will have dominant non-zero components for displacements corresponding to the delays of multiple paths taken by packets during fluttering. The periodicity of fluttering may also be related to the RD component values at these displacements.

6.2.2 Desirable Attributes

Desirable attributes refers to those features of packet reordering metrics that are useful to have but are not necessary. However, with such additional attributes a metric may be preferable over others. This set as explained in the remaining of the subsection includes simplicity, low evaluation and memory complexity, robustness, usefulness to evaluate reordering in cascades of subnets using individual subnets, and modeling of reordering in networks.

1. Simplicity/Informativeness

An ideal metric is the one that is simple to understand and evaluate, yet able to provide a complete picture of reordering. Percentage packets reordered is the simplest

metric. But the ambiguity in its definition as discussed earlier and its failure to carry the extent of reordering make it less informative. On the other hand, keeping track of the displacements of each and every packet without compressing the data will contain all the information about reordering, but it is not simple to evaluate or use.

RD, being able to capture earliness and lateness, and both amount and extent of reordering, is the most comprehensive metric. It is also simple to understand and easy to evaluate. RBD, the occupancy of a buffer used to recover from reordering, is extremely useful for resource allocation. Both reordering extent and n-reordering are simple to evaluate, given a sequence without duplicates. However, the information carried in reordering extent is just lateness among the packets and is not useful, as it fails to estimate any buffer requirements for de-reordering. One the other hand, n-reordering captures only some of the late packets, as explained earlier.

2. Low Evaluation Complexity

Memory and time complexities associated with evaluating a metric play a vital role in implementation and real-time measurements. Spatial/Memory complexity corresponds to the amount of buffer required for the overall measurement process, whereas time/computation complexity refers to the number of computation steps involved to effectively compute the amount of reordering in a sequence. On-the-fly evaluation of the metric for large streams of packets require the computational complexity to be O(N), where N is the number of packets. This allows the metric to be updated in constant-time as each packet arrives. In the absence of a threshold defining losses or the number of sequence numbers to buffer for detection of duplicates, the worst-case complexity of loss and

duplication detection will increase with N. The rate of increase will depend among other things on the value of N and the implementation of duplicate detection scheme.

RD and RBD have a memory requirement of sizes D_T and B_T respectively. In addition, RD relies on an early-arrival buffer that is also of size D_T. Application and transport requirements place an upper bound on D_T and B_T values. Thus, the buffer requirements for these metrics are always a constant. Apart from buffering packets, RD and RBD compare the current packet's sequence number with *receive_index* and expected sequence number respectively, leading to a time complexity of O(N) for a sequence, in both cases.

Both n-reordering and reordering-extent do not have thresholds or in-built schemes to detect duplicates. Thus the buffer requirements have an upper bound of N, as the previously received sequence numbers for each current sequence number have to be tracked. The tracking is required for the detection of point of discontinuity in reordering extent and n-parameter computation for n-reordering. The worst case for N^{th} packet could result in looking up all the previous N-1 arrivals for this tracking. Thus, n-Reordering and reordering extent metrics have a spatial complexity of O(N) and time complexity of $O(N^2)$. Note this complexity is true irrespective of detection of duplicate packets.

3. Robustness

Reorder measurement should be robust against different network phenomena and peculiarities in measurement or sequence such as a very late arrival of a duplicate packet, or even a rogue packet due to undetected error or sequence number wrap-around. The impact due to an event associated with a single or a small number of packets should have a sense of

proportionality on the reorder measure. Consider the arrival sequence: (1, 5430, 2, 3, 4, 5, 6, ..) where packet 5430 appears to be very early; it may be either due to sequence roll over in test streams [Ja05] or some unknown reason. For this case, reordering extent observes e = 1, 2, 3, etc. for all packets until one with sequence 5429, and the measure of reordering changes drastically. One packet out of a large number of packets can have a disproportional effect on the measurement. Thus, reordering is not robust to such peculiarities. For n-reordering, there is a change in degree of 1-reordering but due to partial measure of lateness, it can counter such peculiarities.

RBD is also affected by a very early arrival, as it is stored in buffer, until it can be used in order and thus, shows occupancy of at least 1 for all the intervening arrivals. Use of thresholds (on both early and late arrivals) in RD allows it to recover quickly (within D_T in all cases) from these anomalies, thus making RD robust against such peculiarities.

A reorder metric should be robust against phenomena such as bursty losses, and very early/ very late arrival of a packet(s). For example, the measure should continue to meet all the desirable requirements even if there are bursts of losses creating a significant increment in sequence numbers. The assignment of *receive_index* and thresholds provide such a mechanism for RD and RBD.

4. Extension to Cascaded Networks

Given the reorder measures for two individual networks, the ability to predict the reorder measure for end-to-end connection formed by cascading them is an extremely useful property. In the absence of this property, no amount of individual network measurements, short of measuring the reordering for each pair of endpoints of interest, would be useful in

predicting the end-to-end properties. In addition, such a property will significantly reduce the number of measurements required to characterize complex networks. Using RD metric, the reorder density RD[n] of a cascade of two networks with reorder densities RD1[n] and RD2[n] respectively, is shown to be equal to RD1[n]*RD2[n] where * is the convolution operator. This result is valid under stationary network conditions, and the result has been verified using measurements. A metric that does not capture both earliness and lateness cannot satisfy this property, as it is unable to account for reordering caused by earliness in one subnet and lateness in the other. Since other metrics fail to capture earliness of packets, such a relationship cannot be expected of them.

5. Reorder Models

Packet reordering models using RD metric are derived and verified in Chapter IV. In addition, the modeling of reordering limits the patterns to basic patterns is also presented and verified with RBD metric. With the other metrics, it is not possible to derive complete reordering as they measure only lateness, i.e., one side of the reordering. Moreover, if two consecutive packets have reordering, n-reordering shows only one packet as reordered. A metric with such a disadvantage will completely fail to identify patterns of overlapping.

Table 6.2 summarizes the capabilities of each metric with respect to essential and desirable attributes. If a packet arrives after one or more of the packets that have higher sequence numbers than it, then it is classified as late in lateness based percentage reordering. Using the in-built loss detection scheme as in reordering extent, this metric has low sensitivity to losses. As n-Reordering and reordering-extent [Mo05] do not specify a mechanism for duplicate detection, we categorize 'sensitivity to loss and duplicates' as

Table 6.2. Summary of essential and desirable attributes with respect to corresponding metrics

X – the attribute is absent, ◖ - the attribute is partially present, √ - the attribute is present.

Requirement/Metric	*(RD)*	*RBD*	*Lateness based Percentage Reordering*	*Reordering Extent*	*N - reordering*
Capture reordering	√	◖	◖	◖	X
Low sensitivity to loss and duplication	√	◖	◖	◖	◖
Usefulness – TCP Flow Control	√	X	X	X	√
Usefulness – Buffer/Resource Allocation	√	√	X	X	X
Usefulness – Causes and Effects	√	X	X	X	X
Simplicity/ Informativeness	√	√	X	◖	◖
Evaluation Complexity- Spatial	**Constant**	**Constant**	**O(N)**	**O(N)**	**O(N)**
Evaluation Complexity- Computation	**O(N)**	**O(N)**	**O(N)/O(N^2)***	**O(N^2)**	**O(N^2)**
Robustness	√	X	X	X	√
Extension to cascaded networks	√	X	X	X	X

* For detection of duplicates

partially present, which is also true for lateness based percentage reordering. It is observed that RD fulfills all the essential and desirable requirements, whereas other metrics fail to satisfy these attributes or meet them only partially. Analysis and simulation based modeling of networks often rely on the ability to reproduce network phenomenon based on models. Regeneration of packet sequences satisfying a given reorder metric thus would play a useful role. Regeneration of sequences satisfying a given RD or RBD measure has been

demonstrated in [Ba04]. Cascading metrics for individual subnets to predict reordering in cascades of subnets, and generation of sequences satisfying a given measure, have yet to be demonstrated with other metrics.

6.3 Internet Measurements

Multiple files were downloaded from three web servers around the world to 129.82.x.x subnet. The reordering in the packets downloaded from these sites was measured using various metrics. The procedure of download and parsing from sequence numbers is explained in detail in Chapter III.

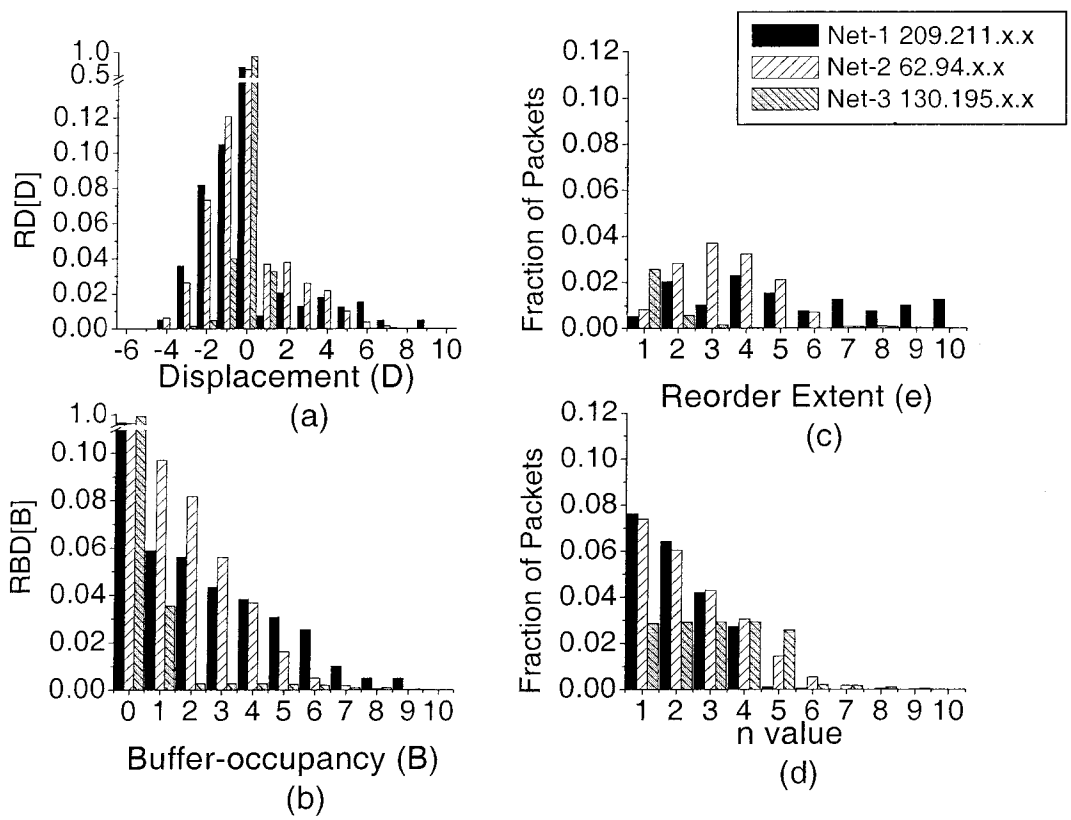


Fig. 6.2. Reorder measurements using different metrics for Net-1, Net-2 and Net-3 (a) RD, (b) RBD, (c) Reordering extent and (d) n-Reordering

Fig. 6.2 illustrates the reordering measures using RD, RBD, Reordering Extent and n-Reordering. Reordering extent does not follow the normalization with respect to total number of non-duplicate packets received as other metrics. However, the normalization is applied for comparing it with other metrics to illustrate the fraction of packets with corresponding extents. In the case of RD and RBD plots, i.e., 6.2(a) and 6.2(b) the vertical axes are broken to show the reordering components clearly. The threshold values for these metrics are $D_T = 15$ and $B_T = 15$ respectively.

As an example, considering measures corresponding to Net-1, it can be observed that reordering extent has values until 10 whereas the buffer required to recover, from RBD or maximum lateness in RD, is nine. This clearly shows the overestimate of buffer requirement using *e* value. RBD also illustrates how often the buffer is filled. Using n-reordering measure for Net-1, there is no substantial information that one can derive as the maximum value of 'n' in n-reordering is limited to 4, whereas packets are as late as 9 (from RD).

RD with its component RD[0] = 0.63, shows that for Net-2, the number of packets that arrived in their actual positions is lower than that for other Nets. The higher reordering can also be derived from the RBD plot where it shows more buffer usage for Net-2 compared to other Nets. Net-3 had the least number of packets out of their actual positions, as shown by RD. Though, there are few packets with displacements as far as 8, the fraction of such packets is less.

6.4 Remarks

There have been numerous discussions in IPPM, IETF meetings and ippm mailing forum about the comparison and usefulness of the metrics. Unfortunately, IPPM has agreed to accept multiple definitions for packet reordering, if the measure is useful in some way. This will lead to metrics with huge demerits, especially with respect to modeling and characterization of packet reordering. Thus, researchers must try to derive all the existing and proposed metrics from RD, whose packet reordering representation is very robust. Moreover, RD provides both earliness and lateness with rich set of attributes, and its usefulness is modeling is proven in Chapter IV.

Chapter VII

DELAY MODELING

In Chapter II, we discussed various studies on delay modeling and characterization, and lack of unique distributions in various studies. In this chapter, using measurements performed over the Internet, we show that the previous studies are all valid. However, the varied observations in prior studies can be attributed to variations in network conditions, Goodness-of-Fit (GoF) criteria used for fitting measurements and other properties of the networks under consideration. Section 7.1 presents our experimental setup used to conduct the measurements over the Internet. The discussion in Section 7.1.1 includes system setup and traffic generation, and in Section 7.1.2, the clock related issues as explained in Chapter II are revisited, and finally in Section 7.1.3, the chosen set of GoF criteria and its significance to distribution fitting is explained.

Section 7.2 presents the analysis of measurement data. The impact of GoF selection on the kind of delay distribution obtained from measurements is shown in Section 7.2.1. In Section 7.2.2, we discuss the effect of packet size and sender side data rates on delay modeling. This discussion leads to a conclusion that delay distributions form a spectrum, where the region of variations depends on packet size and sending data rate. Analysis of

delay values obtained from measurements is extended to the study of correlations among delay values in Section 7.2. 3.

In Section 7.3, as the correlation observed among delay values is high at high data rates (with respect to available bandwidth), we study the inter-packet gap (IPG) that does not show such dependencies. Exploiting the relation between inter-packet gap and delay, in Section 7.4, it is shown that delay can be modeled as a Markovian process and IPG is easier to measure and model latencies.

7.1 Measurement of Delay and Inter-Packet Gap

Delay measurements are usually performed either by sending probe packets at random intervals over a time interval, or by sending a continuous stream of packets with a constant inter-packet gap. In the former case, the resulting distribution is used as a model for the delay distribution of packets traversing between the same endpoints; this approach helps understand a network's behavior for individual packets. However, we are interested in the latter approach, in which the effect of network on a data stream is captured.

A comprehensive study of delay and inter-packet gap (IPG) distributions of packet streams from end to end over two points in the Internet requires the following capabilities: (a) generation of data streams at a source, (b) control and variation of sending data rates and packet sizes, (c) technique to measure the delay of each packet at the destination, (d) measurement technique for IPG at the destination, (e) techniques to fit the measured delay and IPG values into known distributions, and (f) tools to measure the dependence/independence among the values. In the next subsection, we will discuss the experimental setup used to perform measurements of delay and IPG over the Internet.

7.1.1 Experimental Setup

End-to-end measurements over the Internet are posing many challenges as they demand access and control of resources at the source and the receiver. For a comprehensive set of measurements there is a need to monitor the network at various conditions. All such challenges including a reliable time stamping technique led us to pursue the nodes that have adequate resources and also packet generation schemes built into the systems. As Ixia [Ixia] partners with educational institutes to provide these chassis, mainly for research purposes, it was feasible to obtain such equipment at both ends of the measurement.

The study of delay was performed over two end-to-end paths that span across the North American continent. Data was generated at nodes in California Polytechnic (129.65.x.x) and Colorado State University (129.82.x.x) that were destined to Colorado State University (129.82.x.x) and University of Massachusetts, Amherst (128.119.x.x) respectively. The former end-to-end network is labeled as Net-1 and latter as Net –2. In Net-1, both source and destination are Ixia 1600T chasses [Ixia], whereas the destination for Net-2 is a Linux box while the source is an Ixia 1600T chassis. The capacity on both the networks is 100Mbps.

For traffic generation, we used an Ixia 1600T chassis as it provides packet transmission and capture through dedicated hardware, along with a timestamp resolution of 20 nanoseconds with extremely low clock-drift. It also allows GPS based clock synchronization with remote Ixia boxes. The time stamping occurs as soon as the packet leaves/reaches the customized NIC. It has enough processing power to stamp the departure packet's first bit and add the value to its data contents at sender's side.

Packet delay is measured as the difference between the time stamps taken at the sender and the receiver when the first bit of the packet leaves the sender and when it enters the receiver respectively. As discussed, the time stamping of Ixia chassis while leaving the system is inserted in the latter part of the packet to obtain sender side timestamp. IPG is defined as the difference between the arrival times of the packets with consecutive sequence numbers. To obtain the order at the receiver, sequence numbers are inserted as a part of data in the packet structure.

As the destination of Net-2 is a Linux box, to collect timestamps on Net-2 , running of tcpdump tool was necessary. Root privilege to run this command was obtained at the destination node for these purposes. A data-monitoring tool such as 'tcpdump' can provide the receiver timestamp, and the sending timestamp can be inserted as a part of the data content in the packet. However, using 'tcpdump' for delay measurements has several known problems, such as a receiver time stamp occurring only when OS kernel notes the packet. More than one packet could be queued by the time the kernel notes the arrival, and all the packets in the queue may get approximately the same arrival time stamp.

Six UDP test streams of 20,000 packets each, corresponding to packet sizes 64, 80, 96, 128, 256 and 1024 bytes respectively, were sent one at a time over these networks. Delay and IPG measurements were carried out at sending rates of 0.5, 1, 10, 20, 30, 40, 50, 60, 70, 80 and 90 Mbps, which are achieved using a constant inter-packet gap at the sender. At the receiver end, in Net-1, Ixia capture-hardware adds the receive-timestamp along with the corresponding packet contents to a buffer. In Net-2, 'tcpdump' provides the arrival time stamping. At the same time, 'Network Estimator' (also known as 'netest') [Ji03] was used to

monitor the available bandwidth on these end-to-end paths. The scripts used to parse the dumps collected using Ixia capture and tcpdump to measure delay and IPG are listed in Appendix D.

7.1.2 Techniques to Measure Delay and IPG

Clock synchronization and clock skew at source and at destination can cause errors in measurement of delay of packets. Numerous heuristic based techniques are used to reduce the effects due to clock issues, as discussed in Chapter II, but they cannot eliminate the error completely. Clock skew is usually in the range of microseconds in an interval of few seconds. Thus, if the measurements are performed over networks within few seconds and the delays are in milliseconds range, most of these errors can be neglected. However, the problem will persist over a long-run measurement, for example few minutes to few hours.

Clock skew is not eliminated completely in the set of measurements provided in this work. However, in one of the two sets of data used, an Ixia 1600T chassis [Ixia] captures data at the receiver and in both the sets of data a similar Ixia chassis is used. Ixia 1600T has a custom-built high-performance data capture card with nanosecond accuracy. The clock drifts are very low making the measurements more reliable. Finally, as we are interested in delay distributions, an offset to all measured delay values is used to counter the clock synchronization problem. For measurements, this offset is estimated by setting the first packet's delay to the average one-way delay (measured by ping application) between the end points.

Since the IPG measurement refers to the time of arrival of two consecutive packets, the clock skew is negligible for these two readings. Clock synchronization is not an issue for

IPG as there is only one clock (i.e., at receiver) that is used for measurements. Thus IPG measurement has a clear advantage over delay measurement.

7.1.3 Goodness-of-Fit Criteria and their Significance

The measured delay and IPG values from the dumps are fitted with a set of standard distributions that includes beta, chi-square, Erlang, exponential, gamma, inverse Gauss, log-logistic, log-normal, Pareto, Pearson, Rayleigh, triangle, uniform and Weibull distributions. The three most commonly used goodness-of-fit (GoF) tests [Gr93] to fit the data into a known distribution are Chi-Square test (χ^2), Anderson-Darling statistic (A-D) and Kolmogorov-Smirnov statistic (K-S). K-S test statistic does not require binning thereby eliminating the error due to binning as in the χ^2-test. Though A-D statistic is independent of bin-size, its emphasis is on tail distributions. Bestfit 5.0 software [Best] is used to fit distributions, and K-S statistic is used as GoF criteria during analysis.

The correlations between the delay/IPG values and the independence among the IPG values are tested using correlograms and scatter diagrams [Vi98] respectively. On Net-1 and Net-2, we measured and modeled delay and IPG values for different rates and packet sizes using each GoF test. Also, the correlograms and scatter plots were taken for each of the cases.

7.2 Delay Measurements

The measurements were taken at 11PM (EST) between Mar. 16, 2004 and Apr. 18, 2004. The process was repeated 20 times over this period to increase the confidence in the observations. During these hours, the cross-traffic (monitored by 'netest') remained within a

narrow range; the available bandwidth varied between 65-70 Mbps in Net-1, and 60-65 Mbps in Net-2.

For Net-1 and Net-2, the measured delay values were offset to make the first packet delay equal to 22 and 28.5 milliseconds respectively, (i.e., one-way delays using ping application) thus compensating for the difference in clocks at sources and destinations. Since the delays are in the range of milliseconds, as explained earlier, it can safely be assumed that the effect due to clock skew is negligible. These data values are then fit into a known distribution set as listed in Section 7.1.3.

7.2.1 Effect of GoF Criteria

The choice of GoF criteria dictates the kind of distribution that fits the measurements. Table 7.1 summarizes the fits for Net-2 for different data rates and 64-byte packet size using different GoF tests, with parameters in microseconds. The listed test values correspond to the GoF test that qualifies the fit. The test value for χ^2-test requires the number of bins for qualification to be specified. Taking an example, the entry for 0.5 Mbps data rate using K-S test led to G (9.43,122.50) Shift=26818, which indicates the standard gamma function with shape parameter 9.43 and scale parameter 122.5, but all the values are shifted by 26818 microseconds. The distributions in the table cover the set of all distributions that other researchers have observed through measurements, as discussed in Chapter II.

7.2.2 Effect of Data Rate and Packet Size

Fig. 7.1 shows the delay distributions that fit the delay values computed for different sending rates in Net-1, for packet size of 64 bytes. Column graphs represent the input data while the line graphs show the fitted curve. Delay in milliseconds is plotted against the frequency of its occurrence. These results indicate that the delay distribution approaches gamma/Erlang distribution when the data rate is low. For higher data rates, it approaches beta distribution.

Table 7.1. Delay Distributions at Various Sending Rates for Net-2, using different GoF tests

Data Rate (Mbps)	*Chi-Square Test*	*A-D Test*	*K-S Tests*
0.5	Ex[1] (1155.9) Shift=26818	G[2] (33.313, 62.648) Shift=25886.85	G (9.43, 122.50) Shift=26818
10	LL[3] (27483, 1134.5, 41938)	LL (27483, 1134.5, 41938)	G (4.99, 245.66) Shift=27510
30	InG[4] (2737.6, 6077) Shift=26981.3	InG (2737.6, 6077) Shift=26981.3	G (2.02, 1178.4) Shift=27332
50	B[5] (70.699, 5.9617, -43.28, 41848)	B (70.699, 5.9617, -43.28, 41848)	B (70.699, 5.9617, -43.28, 41848)
70	B (1.12959, 0.66084, 28393, 60398)	R[6] (16901) Shift=27964	B (1.12959, 0.66084, 28393, 60398)
90	0.83685, 28287, 75255)	R (22911) Shift=26776	B (1.1820, 0.83685, 28287, 75255)

[1] Exponential (b): b> 0 continuous scale parameter
[2] Erlang (m, b): m > 0 integral shape parameter, b > 0 continuous scale parameter; if m is real, Erlang (E) is Gamma (G)
[3] Log-logistic (s, b, a): s location parameter, b > 0 scale parameter, a > 0 shape parameter
[4] Inverse Gauss (x, y): x, y >0
[5] Beta (w, x, y, z): w, x > 0 continuous shape parameters, y <z, y -minimum boundary and z- maximum boundary
[6] Raleigh (b), b >0 continuous scale parameter

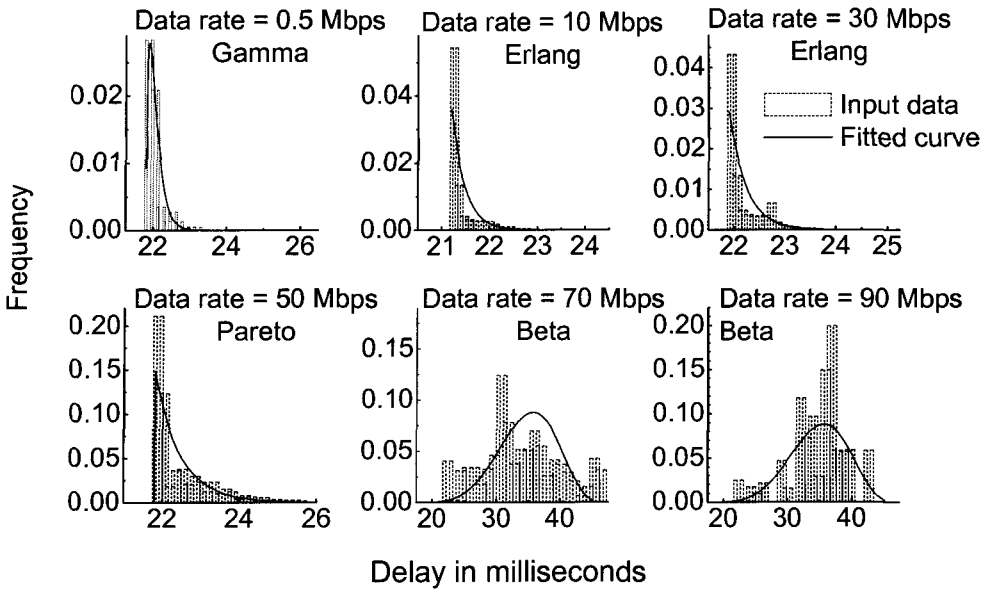


Fig. 7.1. Delay data and distribution fits for 64-byte packets in Net-1 for different data rates

Fig. 7.2(a) and (b) depict the delay distributions for Net-1 and Net-2, with varying packet sizes for different data rates. The plots show the type of distributions in vertical bar that fit the delay values when sent at a rate given on vertical axis, for data stream with packet size in bytes indicated on horizontal axis. For a packet size of 64 bytes in Net-1, the distribution that fits the delays is gamma for data rates 0-30 Mbps, Pareto until 60 Mbps, followed by beta for higher rates. The same distributions fit the delay values for 80 bytes, but the ranges are 0-50 Mbps, 50-80 Mbps, and 80 Mbps or higher for gamma, Pareto and beta respectively. Thus given a network, the type of distributions that fit the delay values remains the same. However, the type of distribution that fits for a given data rate changes with packet size.

Though the delay distributions in Fig. 7.2 show a clear demarcation of the distribution regions, we believe that the transition from one distribution to the other is gradual. In case of Net-2, the same set of distributions fits the delay values for varied data rates with changing packet sizes. Nevertheless, the change in distribution regions with packet size for Net-2 is similar to that of Net-1.

Moreover, an interesting observation was made when we computed the gap between the last-bit of the previous packet to the first-bit of the current packet at sender's side. This is termed as packet separation. For the same packet separation, the transition occurs in the type of distribution, even for different packet sizes.

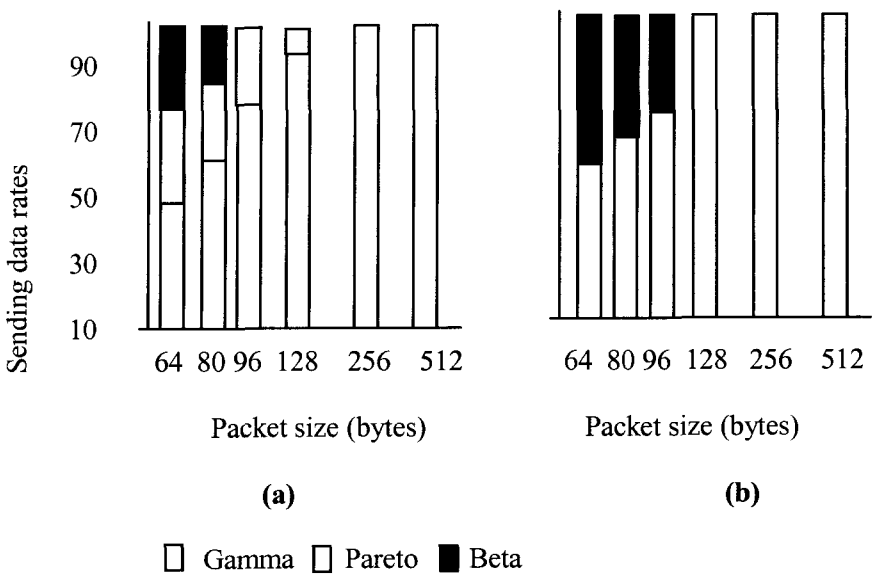


Fig. 7.2. Delay distributions spectrums with respect to packet size and data rate (Not to scale) (a) for Net-1 (b) for Net-2

Table 7.2 shows the packet separations, sending rates and corresponding delay distributions for packet sizes 64, 80 and 96 bytes of Net-2. Here the transition from gamma distribution to beta (shaded) occurs around 12.8 microseconds of packet separation. It is also observed that for the same sending rate, 64-byte packets have more losses than for the 80-byte or higher packet sizes. This phenomenon is intuitive. To send at the same rate, the 64-byte frame test stream needs a larger number of packets with smaller gaps compared to 96-byte frame, making the stream more vulnerable to cross-traffic streams. This leads to more congestion while waiting on processing and queuing resources, leading to higher drop rates.

Table 7.2. Variation of Delay Distribution with packet separation (Gap in microseconds) in Net-2, for different packet sizes

Size/Rate (Mbps)	64 bytes		80 bytes		96 bytes	
	Fit	Gap	Fit	Gap	Fit	Gap
40	E (1,6278.3) Shift=27883	16.2	G (5.644,149.3) Shift=28200	19.3	G (5.06,177.04) Shift=27835	22.37
50	*B (70.699,5.961, -43.28,41848)*	*12.8*	G (3.2414) Shift=27532	15.3	G (2.04,4769.4) Shift=27970	17.73
60	*B (1.0885,0.697, 28454,70554)*	*10.6*	*B (1.307,0.5689, 28254,80963)*	*12.6*	G (2.03,14589) Shift=27897	14.64
70	*B (1.12959,0.66, 28393,60398)*	*9.0*	*B (2.6301,1.152, 26843,45024)*	*10.7*	*B (1.307,0.5689, 28254,80963)*	*12.43*
80	*B (1.1473,0.666, 28256,72533)*	*7.8*	*B (1.594,1.083, 27962,54890)*	*9.3*	*B (1.307,0.5689, 28254,80963)*	*10.77*

7.2.3 Correlation among Packet Delays

Next, we look at the correlation among delay values in the packet stream. If the delay values are correlated, i.e., the delay of the current packet depends on the delay of the previous packets, then correlation values are required to fully characterize the packet delay process. Correlation coefficient is used to study the correlation between the data values.

A correlation coefficient is a number between -1 and 1 which measures the degree to which two variables are linearly related. If there is perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is positive correlation, whenever one variable has a high (low) value, so does the other. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is negative correlation, whenever one variable has a high (low) value then the other has a low (high) value. A correlation coefficient of 0 refers to non-existence of a linear relationship between the variables. The script to compute correlation coefficient for data set to the lag of twenty values in provided in Appendix D.

Fig. 7.3 shows a correlogram, i.e., correlation between the delay values for 64-byte packets of Net-1 up to 20 lags. For lag equal to 'j', the correlogram denotes the correlation between delays of all pairs of packets separated by 'j' positions, e.g., packet 'k' and packet 'k+j'. It can be seen that the correlation coefficients are high for consecutive delay values for higher rates, i.e., 80, 60, 40, 20 Mbps. For these rates, the dependency among the delay values extends beyond 20 positions. However, for lower rates, e.g., 1Mbps and 0.5 Mbps, it can be observed that the correlation is negligible. In addition, if the packet size is higher the correlations are lower for the same data rate.

To understand the reason for correlations, let us consider a packet of a data stream being processed at an intermediate node. If more packets of this stream arrive before the first packet's departure, then the delays of new arrivals depend on the current packet's processing component of the delay. These waiting times produce a cumulative effect at higher data rates, which leads to high correlations. At low rates such as 0.5 Mbps, the correlation is

almost zero, and hence the delays can be safely assumed to be independent and identically distributed (i.i.d). Thus, the delay distribution of the packets is sufficient to model the delay at lower data rates. However, for higher data rates, the correlation between the values has to be characterized. These correlations can be computed using IPG as shown in Section 7.5.

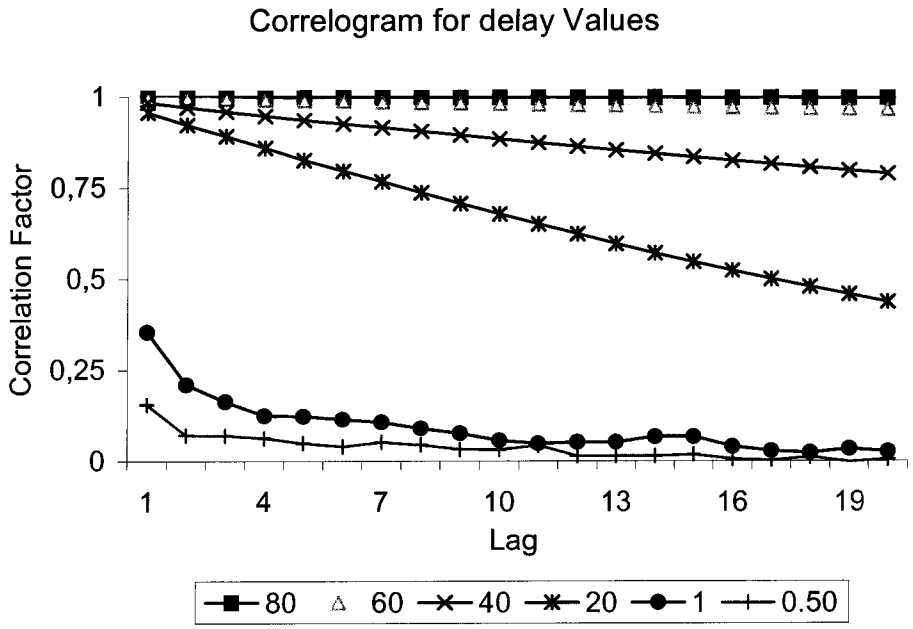


Fig. 7.3. Correlograms for lag = 1 to 20 of delay values in Net-1 for varied data rates (80Mbps – 0.5 Mbps) and 64-byte frame size

7.3 Inter-Packet Gap Measurements

To measure IPG values from the captured trace, we identify the consecutive packets by using the sequence number fields in the data and then compute the difference between arrival times of consecutive packets. The sender side inter-packet gap is subtracted from this difference to obtain G_i, the random variable that denotes the change in the inter-packet gap

between packets 'i' and 'i+1'. Though this offset shifts the curve by an amount equal to the sending gap, it has no effect on the analysis.

Interestingly, the correlation coefficients for IPGs are negligible for higher rates. For lower rates (e.g., 0.5 Mbps, 1 Mbps), there is a correlation between consecutive IPGs for a lag one as shown in Fig. 7.4. When sending rates are very low, the probability of cross-traffic affecting two consecutive packets is very low. Thus, if an IPG of two packets is increased in transit, the next IPG is likely to decrease leading to dependence between two consecutive IPGs. Fig. 7.5 illustrates scatter plots of consecutive IPG pairs for Net-1 data for sending rates of 90, 60 and 30 Mbps. It can be seen that there is no relation between these values. Thus, IPG can be safely assumed to be i.i.d at higher rates.

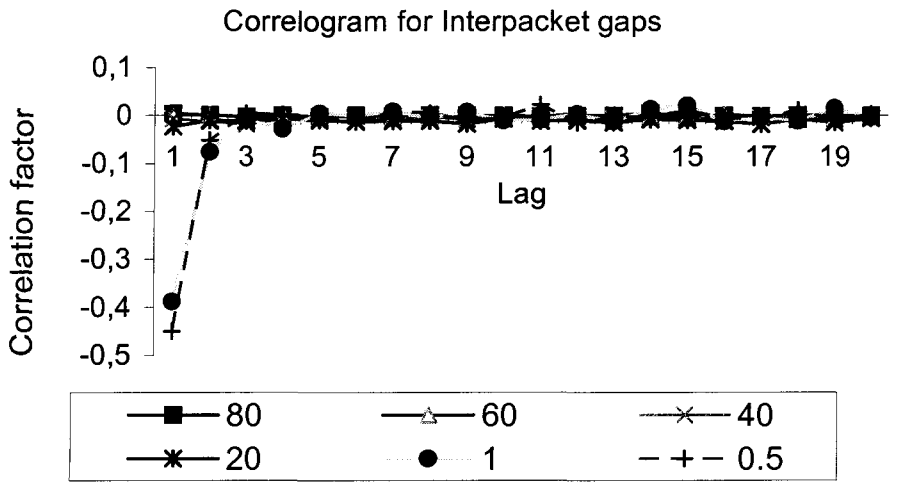


Fig. 7.4. Correlograms for lag = 1 to 20 of IPG values for varied data rates (80 Mbps – 0.5 Mbps) in Net-1

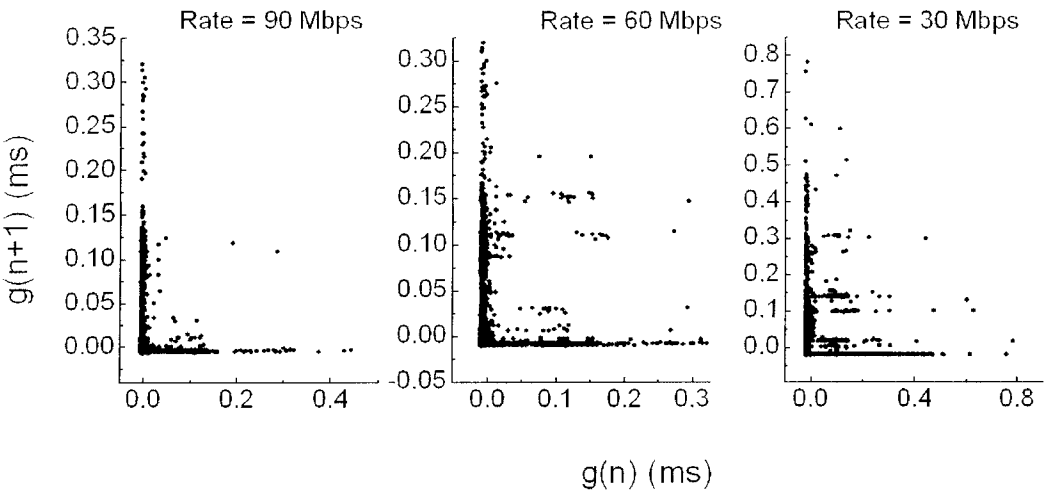


Fig. 7.5. Scatter plots for IPG with lag 1 of Net-2 at different sending rates

7.4 Relation between Delay and IPG

Let $D = \{D_i, i = 1,2...\}$ denote a random process corresponding to the end-to-end delay values of a packet stream, where D_i is the delay of the i^{th} packet. As shown in Section 7.4, at low data rates, the correlation between successive delay values is negligible, and thus D_i is i.i.d. for different 'i'. Now let $G = \{G_i, i = 1,2,...\}$ denote a random process where G_i is the IPG between i^{th} and $(i+1)^{th}$ packets, with offsets as explained earlier. As shown earlier, for higher data rates, G_i can be considered as i.i.d for different 'i'.

The delay and IPG are related by $D_{i+1} = D_i + G_i$, resulting in $D_{i+1} = D_1 + G_1 + \ldots + G_i$. This expression is similar to the 'random walk' problem. Since the increments are i.i.d and D_1 and G_i are independent, at higher data rates, D is a Markov process, i.e., dependence of D_{i+1} on the delays of prior packets is captured by that of the previous packet namely, D_i. Given the mean delay of packets, we can derive the distribution of delay along with the

correlation values. Therefore, for higher data rates, the delay can be modeled by using IPG distributions alone. Though the discussion here is limited to delay model, one can exploit the properties of Markov chain to understand the delay random process better.

Next, we derive the relationship between the IPG process, the delay process and the correlation among IPG and delay values. Let Cov (D_i, D_k) and Cov (G_i, G_k) be the covariance between i^{th} and k^{th} samples of delay and IPG respectively. Let $(\sigma_D)^2$ and $(\sigma_G)^2$ be the variances of delay and IPG respectively. The correlation coefficient for D_i and D_k, is given by

$$R^{(D)}_{i,k} = Cov(D_i, D_k) \Big/ (\sigma_{D_i} \sigma_{D_k}) \tag{7.1}$$

At low data rates, since D_i, D_k ($i \neq k$) are i.i.d, $R^{(D)}_{i,k} = 0$.

For higher rates, D_i and D_k are dependent; however, the packet gaps are independent. Since $D_k = D_i + G_{i+1} + \ldots + G_{k-1}$ (for $k > i$):

$$R^{(D)}_{i,k} = Cov(D_i, D_i) \Big/ (\sigma_{D_i} \sigma_{D_i+G_i+\ldots+G_k}) + \sum_{j=i}^{k} Cov(D_i, G_j) \Big/ (\sigma_{D_i} \sigma_{D_i+G_i+\ldots+G_k}) \tag{7.2}$$

The second term on right hand side is equal to zero as D_i and G_j are independent.

Therefore,

$$R^{(D)}_{i,k} = \left(\sqrt{1 + k\left(\sigma_G / \sigma_D\right)^2} \right)^{-1/2} \tag{7.3}$$

Table 7.3 shows the correlation coefficients calculated with this expression for $R^{(D)}_{1,k}$ for k =1, 2, ...8 against the observed correlation through measurements for a data rate

of 20 Mbps through Net-1. In addition, with the determination of 'k' for which $R^{(D)}_{I,k}$ is negligible, we can compute the lag for which the delay dependency ceases to exist.

Let $R^{(G)}_{i,k}$ be the correlation coefficient between i^{th} and k^{th} samples of IPG. For low data rates ($i \neq k$),

$$R^{(G)}_{i,k} = Cov(D_{i+1}, D_{i-1}) \Big/ 2\sqrt{(\sigma^2_{D_{i-1}} + \sigma^2_{D_{i-1}})} = -1/2 \qquad (7.4)$$

for k-i = 1.

As $G_i = D_{i+1} - D_i$, $Gi_{+1} = D_{i+2} - D_{i+1}$, and $\{D_i\}$ are i.i.d., $R^{(G)}_{i,k} = 0$ for k-i > 1. For higher data rates as $\{G_i\}$ are i.i.d., $R^{(G)}_{i,k} = 0$ for ($i \neq k$). Using Fig. 7.4, the correlation coefficients for lower rates, 0.5 Mbps and 1 Mbps, are -0.4 and -0.45 respectively as compared to -0.5 computed in Eq. (7.4).

Table 7.3. For Net-1, measured Vs. computed correlation coefficients, data rate = 20Mbps

Lag	*Measured Value*	*Computed* $R^{(D)}_{I,k}$
1	0.95	0.95
2	0.92	0.91
3	0.89	0.86
4	0.86	0.83
5	0.79	0.78
6	0.76	0.76
7	0.73	0.73
8	0.70	0.71

7.5 Remarks

This chapter discussed the packet delay, correlation among delay values and its impact on modeling. Due to the Markov nature of delay at high data rates and i.i.d process

at low data rates, it can be better modeled and also understood in terms of subnets. In addition, this research provides a consolidation of studies done on delay modeling and authenticates seemingly contradictory data, explaining how the distribution varies with .... . The observed spectrum of delay distribution depicts a clear picture of variation of delay model under different operating conditions.

Chapter VIII

CONCLUSIONS

The existing metrics to measure reordering are vague and insufficient to characterize packet reordering. We presented a formal representation of reordering using the concept of reorder event. With this representation, the sum of all displacements in a sequence is conserved. As packets that are lost are considered to have zero displacement, the conservation of total displacement is possible even when packets are lost.

Reorder events are of two types, namely primary event (p-event) associated with a packet that is delayed or arrives early and secondary event (s-event) associated with packets whose displacement is caused by a packet undergoing a p-event. Reorder events can be intertwined, and an event formed by two or less p-events intertwining is called a basic pattern. A non-tangled single p-event is called Independent Reordering whereas two p-events intertwined are known as embedded reordering (ER) if embedded in each other, and overlapping reordering (OR) if they are overlapping with each other. Depending on the direction of displacements of p-events in basic patterns, there exist ten different forms of basic patterns. However, lateness p-event based IR, lateness overlapping lateness, and lateness embedded in lateness are most common patterns observed in the Internet. This observation points to the dominant choice of shortest path as preferred path in the Internet.

A metric for packet reordering, Reorder Density (RD) was derived based on the formal representation. In certain cases, the shapes of RD can be used to match reordering to probable causes. For example, the property involving IR patterns and RD shapes indicate that if a network has only earliness then the reorder events of the reorder set will have non-zero components for less than two. RD characterizes and measures packet reordering comprehensively, is simple, and orthogonal to losses and duplicates. RD follows a principle of conservation of amount of reordering, thus maintaining the interdependence of displacements between p- and s- events. In addition, it can be used as the probability density function corresponding to the displacement of an arbitrary packet. The metric can be evaluated in real-time as the packets arrive at a node. A threshold D_T limits the complexity of implementation, by considering a packet that is late by D_T to be lost. This also provides a concrete definition for losses, and the threshold can be selected based on application or protocol or both. For both Stay-back and Go-back D_T methods, the computation complexity of the algorithm is O(N), where N is the number of packets in the sequence. The memory requirement for the implementation is proportional to D_T . The use of D_T also makes it robust, allowing it to recover from cases such as very early or very late packets, and sequences of duplicates, or bursty losses. Further, the metric can be used to characterize the reordering introduced by a network, and under a fairly broad set of conditions, the reorder measurement of different subnets can be combined to predict the end-to-end reorder characteristics of a network, using convolution operation. Reorder response of a network depends on factors such as the network load, background traffic, and the distribution of the inter-packet gap.

The second metric, Reorder Buffer Density (RBD) while characterizing packet reordering can also be used for making decisions related to transport protocols and applications. Using RBD, an application or a transport protocol can estimate the buffer allocation to deal with reordering. In addition, based on the topology and load-balancing strategy, jitter buffer management, network suitability, etc., can be evaluated for real-time applications. As RBD also contains the usage information along with the extent of buffer requirement, impact of buffer size vs. loss of packets can be evaluated. In addition, protocols like TCP can tune the parameters such as 'dupthresh' using RBD measure, to better differentiate lost packets from reordered packets, leading to an improvement in performance of the protocols as well as reducing unnecessary retransmissions. The buffer-occupancy threshold (B_T) associated with RBD allows the management of the complexity of measurement. RBD can be implemented so that the measure is orthogonal to losses. Both forms of RBD have computation complexity of O(N), where N is the number of packets in the received sequence and a memory requirement proportional to B_T. Similar to the value of D_T, the value of B_T can be determined based on the precision required for a reorder measure. Moreover, this threshold also improves the robustness of the reorder measure.

Internet measurements indicate that reordering on the Internet occur in localized clusters, and that primary events of reordering rarely occur in groups with more than two primary events in a subsequence, i.e., basic patterns (OR, IR and OR) form a major fraction of the reordered segments. The expressions for RD and RBD for basic patterns are derived and a methodology is presented that uses the models reorder characteristics of networks. The methodology can be extended to the model cases where more than two primary events are clustered in overlapped or embedded manner. The developed models, for a simple load-balancing scenario that contains such multiple reordering, agree with the simulated results

for low amounts of reordering, even when there is a significant amount of variance within the delay values of different paths used for balancing the load. However, with increased reordering expected in future high-speed networks, more complex multiple reordering patterns are inevitable, and the general models of reordering presented in Chapter IV consider such instances.

The general models of RD for two-path and multi-path load splitting are derived and verified against the reordering in sequence received through emulated load splitting topologies. In the case of two-path load splitting, the RD distributions are mirror images for complementary probabilities of packets taking alternate route. In addition, as the delay of the alternate path increases the non-zero components of RD increase. Therefore to reduce the extent and amount of reordering, it is suggested to use the next available shortest path, for load-splitting scenarios. Also, as the variation of delays on paths increase the RD distribution spreads out including more non-zero components than that from the model, thus indicating the limitations of models. Though the limited and general models depict reordering due to load splitting, the methodology can also be used for parallelism within the nodes, such as the parallelism due to queuing and the parallelism due to processing in a router.

The above models have been verified by sending packet streams with constant inter-packet gaps leading to a clear and single value of displacement for each path. But, the inter-packet gaps are not always constant at the sending side, e.g., TCP streams, which are sent at different rates based on congestion in the network. There is a need to obtain the distribution of the inter-packet gaps in TCP over the Internet. The distribution of displacement suffered by a packet in an alternate path can then be modeled according to inter-packet gap distributions. In addition to the issues with inter-packet gaps and delay deviations, the

models may also be extended to take into account the QoS criteria where the choice of alternate path is dependent on the criteria in SLA and is not probabilistic.

Furthermore, the spread of RD distribution is indicative of disorder and an atomic unit such as entropy derived from RD can provide a simplistic view for comparing the impacts of various load-splitting scenarios. As RD is a probability density function of displacement, it could be used in Shannon's formula of entropy to derive a measure of disorder in the sequence. Preliminary measurements of reordering show that the entropy can clearly distinguish various magnitudes of reordering. For example, with increasing displacement associated with an alternate path, there is more out-of-orderliness as higher number of packets has non-zero displacement, this is clearly captured in the value of reorder entropy. This is a promising measure as it is an atomic metric like percentage reordering, yet contains more useful information.

The metrics for reordering, Reorder Density (RD), Reorder Buffer-occupancy Density (RBD), Reorder extent and n-Reordering were compared in Chapter VI with respect to relevant attributes. Attributes are classified into (i) essential attributes that must be satisfied by any reorder metric, and (ii) desirable attributes, the presence of which makes a metric more useful for extensive analysis and characterization of networks. Essential attributes include capturing reordering in a sequence, low sensitivity to packet loss and duplication, and metric's usefulness to evaluate behavior and performance of a network. RD is unique among the metrics for reordering in that it satisfies all the essential requirements.

Simplicity/informativeness, evaluation complexity, robustness, extension to cascaded networks and models form a set of desirable attributes. Lack of complete reordering in the measure based on n-reordering makes it less informative. The evaluation complexities of RD and RBD are characterized by constant buffer requirement and

computation complexities of O(N), where N is the size of the sequence under consideration. Reorder extent and n-reordering have a buffer requirements of O(N) and computation complexities higher than O(N), making them less attractive for real-time measurements. Due to DT and BT in RD and RBD respectively, these metrics are robust to peculiar sequences. In addition, methodology to derive packet reordering models from metrics is available only for RBD and RD.

Using network measurements, it is shown that RD provides a comprehensive picture of reordering and contains all the usages that other metrics provide except the frequency of occupancies of buffer for de-reordering, which is contained in RBD. Due to the comprehensive nature of RD, perhaps it may be possible to derive all the other metrics from RD. If this is the case, then it will clearly be the choice for measuring reordering in packet sequences. This opens many opportunities for future research.

Measurements shown in Chapter VII, indicate that for a given packet size, the delay distributions of packets in data streams form a spectrum of distributions, which ranges from beta (for higher data rates) to gamma (for lower data rates). However, the transition from one distribution to the other depends on the packet separation within the data stream. At low data rates, because of independence in delay values, the delay distribution itself characterizes the delay process completely. On the other hand, delay values have significant correlations at high data rates. Thus, it is necessary to include the correlation values along with the delay distribution for its characterization. Since, the IPG values are not correlated at these rates, the delay process is shown to approximate a Markov process.

The delay measurements used UDP streams and as TCP changes its rate based on the congestion, in future, one can map the delay distribution for TCP streams as a weighted average of delay distributions for UDP at various rates. Though, delay process shown to be a Markov process, for TCP streams, this claim may not remain true. Thus, there is scope for extending the delay and IPG study for TCP streams.

BIBLIOGRAPHY

[Aa04] I. Aad, J-P. Hubaux, E. W. Knightly, "Denial of Service resilience in ad hoc networks," *Proc. MobiCom'04*, Sep./Oct. 2004, pp. 202-215.

[Ab02] P. Abry, R. Baraniuk, P. Flandrin, R. Riedi and D. Veitch, "Multiscale Nature of Network Traffic," *IEEE Signal Processing Magazine*, 2002, pp. 28-46.

[Ba02] T. Banka, A. A. Bare and A. P. Jayasumana, "Metrics for Degree of Reordering in Packet Sequences," *Proc. 27th IEEE Conference on Local Computer Networks*, Nov. 2002, pp. 333-342.

[Ba04] A. A. Bare, "Measurement and Analysis of Packet Reordering using Reorder Density," *MS Thesis*, Dept. of Computer Science, Colorado State University, 2004.

[Be02] J. Bellardo and S. Savage, "Measuring Packet Reordering," *Proc. Internet Measurements Workshop*, Nov. 2002, pp. 97-105.

[Be99] J. C. R. Bennett, C. Partridge and N. Shectman, "Packet Reordering is Not Pathological Network Behavior," *Trans. on Networking IEEE/ACM*, Dec. 1999, pp. 789-798.

[Best] Bestfit 5.0, Ver. 4.5.2. Palisade Corporation © 2002.

[Bh05] S. Bhandarkar, A. L. N. Reddy, M. Allman and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events," IETF draft (work in Progress).

[Bl01] M. S. Blumenthal and D. D. Clark, "Rethinking the Design of the Internet: The End-to-End Arguments Vs. the Brave New World," *ACM Trans. On Internet Technology*, Vol. 1, Aug. 2001, pp. 70-109.

[Bl02] E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering", *ACM Computer Comm. Review*, Jan. 2002, pp. 20-30.

[Bo02] C.J. Bovy, H.T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal and P. Van Mieghem, "Analysis of End-to-end Delay Measurements in Internet," *Proc. Passive and Active Measurements Conference*, Colorado, March 2002.

[Bo03] S. Bohacek, J. Hespanha, J. Lee, C. Lim and K.Obraczka, "TCP-PR: TCP for Persistent Packet Reordering," *Proc. IEEE 23rd ICDCS*, May 2003, pp. 222-231.

[Bo93] J-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. ACM SIGCOMM*, 1993, pp. 289-298.

[Ca02] O. Cappe, E. Moulines, J-C. Pesquet, A. Patropulu, and X. Yang, "Long-Range Dependence and Heavy-Tail Modeling for Teletraffic Data," *IEEE Signal Processing Magazine*, 2002, pp. 14-26.

[Ca03] M. Carson and D. Santay, "NIST Net- A Linux based Network Emulator Tool," *ACM SIGCOMM Computer Communications Review*, Vol.33, Jan 2003, pp. 111-126.

[Ci03] L. Ciavattone, A. Morton, and G. Ramachandran, "Standardized Active Measurements on Tier 1 IP Backbone," *IEEE Communication Magazine*, June 2003, pp. 90-97.

[Cl88] D. D. Clark, "The Design Philosophy of the DARRPA Internet Protocols," *ACM Computer Communication Review*, Vol. 18, Aug. 1988, pp. 106-114.

[Co02] M. Coates, A. D. Hero III, R. Nowak and B. Yu, "Internet Tomography," *IEEE Signal Processing Magazine*, May 2002, pp. 47-65.

[Cr97] M. E. Crovella, A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *Trans. IEEE/ACM on Networking*, Vol. 5, Dec. 1997, pp. 835-846.

[e204] e2e mail forum:http://www.postel.org/pipermail/end2end-interest/2004-August/004233.html.

[Fl01] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Trans. on Networking*, 2001, pp. 392-403.

[Fl03] S. Floyd and E. Kohler, "Internet Research Needs Better Models," *ACM SIGCOMM Computer Communications Review*, Vol. 33, Jan 2003, pp. 29-34.

[Fu02] Z. Fu, B. Greenstein, X. Meng and S. Lu, "Design and Implementation of a TCP-friendly Transport Protocol for Ad Hoc Wireless Networks," *Proc. IEEE ICNP*, Oct. 2002, pp. 216-225.

[Fu95] C. Fulton and S. Q. Li, "Delay Jitter Correlation Analysis for Traffic Transmission on High Speed Networks," *Proc. INFOCOM*, 1995, pp. 717-727.

[Ga03] M. Garetto and D. Towsley, "Modeling, Simulation and Measurements of Queuing Delay under Long-tail Internet Traffic," *Proc. ACM Sigmetrics*, Jun. 2003, pp. 47-57.

[Gh04] L. Gharai, C. Perkins and T. Lehman, "Packet Reordering, High Speed Networks and Transport Protocol Performance," *Proc. IEEE ICCCN*, Oct. 2004, pp. 73-78.

[Gr93] D. V. Groebner and P. W. Shannon, "Business Statistics: A Decision-Making Approach," *4th ed.: Macmillan Publishing Company*, New York, NY, 1993.

[Gu91] F. Guilleman and J. W. Roberts, "Jitter and Bandwidth Enforcement," *Proc. Global Telecommunication Conference (GLOBECOM),* 1991, pp. 261-265.

[Gu05] O. Gurewitz, I. Cidon and M. Sidi, "Maximum Entropy based One-Way Link Delay Estimation," *Proc. SPECTS 2005*, Jul. 2005, pp. 804-812.

[Ha05] E. Hasenleither, P. Krueger and T. Ziegler, "A Performance Evaluation of Software Tools for Delay Emulation," *Proc. SPECTS 2005*, Jul. 2005, pp. 904-912.

[Ho02] ``Internet Traffic Tends *Toward* Poisson and Independent as the Load Increases", *Nonlinear Estimation and Classification,* Eds. C. Holmes, D. Denison, M. Hansen, B. Yu, and B. Mallick, Springer, New York, 2002.

[Hi94] History of ARPANET - http://www.dei.isep.ipp.pt/docs/arpa.html, Last Accessed Aug. 16, 2005.

[Ho00] O. Hodson, C. Perkins, and V. Hardman, "Skew Detection and Compensation for Internet Audio Applications," *Proc. IEEE International Conference on Multimedia and Expo*, July 2000.

[Ho01] G. Hooghiemstra and P. Van Mieghem, "Delay distributions on fixed Internet paths," *Delft University of Technology*, 2001, report20011020.

[Hu00] L. Huang and K. Sezaki, "An Analysis of the characterization and prediction of network delay," *Proc. IEICE General Conference*, SB-9-7, Japan, Mar. 2000.

[Ho04] N. Hohn, D. Veitch, K. Papagiannaki and C. Diot, "Bridging Router Performance and Queuing Theory," *Proc. ACM Sigmetrics/Performance*, Jun. 2004, pp. 355-366.

[Ia01] G. Iannaccone, S. Jaiswal and C. Diot, "Packet Reordering Inside the Sprint Backbone," *Tech. Report, TR01-ATL-062917*, Sprint ATL, Jun. 2001.

[IPLo] IP2Location™, http://www.ip2location.com. Last accessed Aug. 16, 2005.

[Ixia] Ixia Inc.,

http://www.ixiacom.com/products/chassis/ch_display.php?skey=ch_1600t_400t_100, Last Accessed Aug. 16, 2005.

[Ja03] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley, "Measurement and Classification of Out-of-sequence Packets in Tier-1 IP Backbone," *Proc. IEEE INFOCOM*, Mar. 2003, pp. 1199-1209.

[Ja05] A. Jayasumana, N. Piratla, A. Bare, T. Banka, R. Whitner and J. McCollom, "Reorder Density and Reorder Buffer-occupancy Density - Metrics for Packet Reordering Measurements,"http://www.ietf.org/internet-drafts/draft-jayasumana-reorder-density-05.txt. Last Modified in Sept. 2005 (work in progress).

[Ji03] G. Jin and B. Tierney "Netest: A Tool to Measure the Maximum Burst Size, Available bandwidth and Achievable Throughput," *LBNL report # 48350.*

[Kh99] D. A. Khotimsky, "A Packet Resequencing Protocol for Fault-Tolerant Multipath Transmission with Non-Uniform Traffic Splitting," *Proc. Global Telecommunication Conference (GLOBECOM)*, Dec. 1999, pp. 1283-1289.

[La02] M. Laor and L. Gendel, "The Effect of Packet Reordering in a Backbone Link on Application Throughput," *IEEE Network*, Sep/Oct. 2002, pp. 28-36.

[Le94] W. E. Leland, M. S. Taqqu, W. Willinger and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Transactions on Networking*, Vol.2, Feb. 1994, pp. 1-15.

[Li01] Light Reading – Networking the Telecom Industry, http://www.lightreading.com/document.asp?doc_id=4009&page_number=8, Last Accessed Aug. 16, 2005.

[Li02] H. Liu, "A Trace Driven Study of Packet Level Parallelism," *Proc. International Conference on Communications (ICC)*, New York, NY, 2002, pp. 2191-2195.

[Ma94] W. Matragi, C. Bisdikian and K. Sohraby, "Jitter Calculus in ATM Networks: Single Node Case," *Proc. INFOCOM*, 1994, pp. 232-241.

[Mi03] D. L. Mills, "A Brief History of NTP Timer: Memoirs of an Internet Timekeeper," *ACM SIGCOMM*, Vol. 33, Apr. 2003, pp. 9-21.

[Mo03] P. Molinero-Fernandez, N. McKeown and H. Zhang, "Is IP Going to Take Over the World (of Communications)?" *ACM SIGCOMM Computer Communication Review*, Vol. 33, Jan 2003, pp. 113-118.

[Mo05] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov and J. Perser, "Packet Reordering Metric for IPPM," IETF draft (work in progress).

[Mo98] S. B. Moon, J. Kurose, P. Skelly and D. Towsley, "Correlation of Packet Delay and Loss in the Internet," UM-CS-1998-011, http://www.cs.umass.edu/Dienst/UI/2.0/Describe/ncstrl.umassa_cs/UM-CS-1998-011, Mar. 1998.

[Mo99] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and Removal of Clock Skew from Network Delay Measurements," *Proc. IEEE INFOCOM*, Mar. 1999, pp. 227-234.

[Mu94] A. Mukherjee, "On the Dynamics and Significance of Low Frequency Components of Internet Load," *Internetworking: Res. and Experience*, vol. 5, Dec. 1994, pp. 163-205.

[Ns-2] The Network Simulator – ns –2, http://www.isi.edu/nsnam/ns/. Last Accessed Aug. 16, 2005.

[Oh01] H. Ohsaki, M. Murata, and H. Miyahara, "Modeling End-to-End Packet Delay Dynamics of the Internet Using System Identification," *Proc. 7th International Teletraffic Congress*, Dec. 2001, pp. 1027 1038.

[Ok05] B. Oklander and M. Sidi, "Delay Jitter Analysis," *Proc. SPECTS 2005*, Jul. 2005, pp. 814-821.

[Pa03] K. Papagiannaki, S. Moon, C. Fraleigh. P. Thiran and C. Diot, "Measurement and Analysis of Single-Hop Delay on an IP Backbone Network," *IEEE Journal on Selected Areas of Communications*, Vol. 21, Aug. 2003, pp. 908-921.

[Pa95] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, Vol. 3, 1995, pp. 226-244.

[Pax97] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. Thesis*, University of California, Berkeley, Apr. 1997.

[Pa97] V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," *Proc. Winter Simulation Conference*, 1997, pp. 1037-1044.

[Pax98] V. Paxson, G. Almes, J. Mahdavi and M. Mathis, "Framework for IP Performance Metrics," RFC 2330.

[Pa98] V. Paxson, "On Calibrating Measurements of Packet Transit Times," *Proc. SIGMETRICS,* 1998, WI USA, pp. 11-21.

[Pr05] M. Przybylski, B. Belter and A. Binczewski, "Shall we worry about Packet Reordering," *Proc. TERENA 2005,* June 2005.

[Ri97] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *ACM Communication Review*, vol. 27, 1997, pp. 31-41.

[Ru01] M. Ruiz-Sanchez, E. W. Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, Mar/Apr. 2001, pp. 8-23.

[Sa84] J. Saltzer, D. Reed, and D. Clark, "End-to-end arguments in system design," *ACM Trans. on Computer Systems (TOCS)*, vol. 2, 1984, pp. 195-206.

[Tcpd] TCPDUMP Public Repository, http://www.tcpdump.org/. Last accessed Aug. 16, 2005.

[Vi98] S. Vincent, "Input Data Analysis," *in Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, Edited by J. Banks, John Wiley & Sons, Inc, New York, 1998, pp. 55-91.

[Xi03] Y. Xia and D. Tse, "Analysis on Packet Resequencing for Reliable Network Protocols," *Proc. IEEE INFOCOM*, Mar. 2003, pp. 990-1000.

[Ya04] M. Yang, X. R. Li, H. Chen and N.S. V. Rao, "Predicting Internet End-to-End Delay: An Overview," *Proc. 36th Southeastern Symposium on System Theory*, Mar. 2004, pp. 210- 214.

[Ya99] M. Yajnik, S. Moon, J. Kurose and D. Towsley, "Measurement and Modelling of the Temporal Dependence in Packet Loss," *Proc. IEEE INFOCOM*, 1999, pp. 345-352.

[Zh01] L. Zheng, L. Zhang and D. Xu, "Characteristics of Network Delay and Delay Jitter and its Effects on Voice over IP (VoIP)," *Proc. IEEE International Conference on Communications (ICC)*, 2001, pp. 122-126.

[Zh02] L. Zhang, Z. Liu, and C. H. Xia, "Clock Synchronization Algorithms for Network Measurements," *Proc. IEEE INFOCOM*, Jun. 2002, pp. 1-10.

[Zh03] M. Zhang, B. Karp, S. Floyd and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK," *Proc. 11th IEEE International Conference on Networking Protocols (ICNP 2003)*, Atlanta, GA, Nov. 2003, pp. 95-106.

APPENDIX A

I. Algorithm to compute RD

Variables used:

DT : Displacement Threshold.
S: Arrival under consideration for lateness/earliness computation.
D: Lateness or earliness of the packet being processed.
RI : Receive index.

window[1..DT] : Incoming sequence numbers window.

buffer[1..DT] : Buffer to hold early arrivals.
FLE[-DT..DT]: Frequency of lateness and earliness.

```
1. Initialize.
   Store first unique DT+1 sequence numbers in arriving order into window;
   RI = 1;
   Empty buffer.

2. Repeat.
   If (window or buffer contains sequence number RI)
   {
      Copy first sequence number in window to S;
      Delete first sequence number from window;
      D = RI - S; # compute displacement

      If (absolute(D) <= DT) # Apply threshold
      {
         FLE[D]++; # Update frequency
         If (buffer contains sequence number RI) Delete RI from buffer;
         If (D < 0) add S to empty slot in buffer; # Early Arrival
         RI++; # Update RI value
      }

      Else Discard S; # Displacement beyond threshold

      # Get next incoming non-duplicate sequence number, if any.
      # A sequence number < RI or already present in buffer or window is
      # duplicate
      Do
      {
         newS = get next incoming sequence number;
      }
      While (newS is valid and (newS < RI or newS in buffer or newS in
      window));
      If (newS is valid)
      {
         Add newS to window;
      }

      If (window is empty) go to step 3;
   }
   Else # RI not found. Get next RI value
   {
      # Next RI is the minimum among window and buffer contents
```

```
        m = minimum (minimum (window), minimum (buffer));

        If (RI < m) RI = m;
        Else RI++;
    }
3. Normalize FLE to get RD.
```

II. Perl Script for RD (Stay-back D_T)

```
#!/usr/bin/perl -w
#########################################################################
#
# RD Perl Script provides RD from a file containing sequence numbers
#
# Usage:
# RD.pl input_file DT_value [-debug]
#
# 1. The input file must contain sequence numbers
# 2. If DT value is not provided then the deafult value is 10
#
#########################################################################

my($DT,$pl);
my(@seq,@window,@buffer);
my(%FLE);

my $debug=0;

sub get_new_seq()
{
   my $new_seq = shift @seq;
   return $new_seq;
}

# Utility subroutine: returns index of an element if it exists in
# the array given else -1
sub exists_in_array()
{
   my ($key,@in) = @_;
   my $i = 0;
   if ($key && ($#in >= 0))
   {
       foreach $k(@in)
       {
           return $i if ($key == $k);
           $i++;
       }
   }
   return -1;
}

# Utility subroutine: returns minimum seq. number in the array

sub min_in_array()
{
   my $arr_r = shift , $min=-1;
```

```
    my @arr = @$arr_r;
    if ($#arr > 0)
    {
        $min = $arr[0];
        foreach (@arr)
        {
            $min = $_ if ($_ < $min);
        }
    }
    return($min);
}

# Utility subroutine: prints contents of windows - for debug only
sub print_windows()
{
    print "\t\twindow = @window\n";
}

if ($#ARGV == 2) { $DT = $ARGV[1]; $file = $ARGV[0]; if ($ARGV[2] eq
"-debug") { $debug = 1; }}
elsif ($#ARGV == 1) { $DT = $ARGV[1]; $file = $ARGV[0];}
elsif ($#ARGV == 0) { $DT = 10; $file = $ARGV[0]; }
else { print "Format:$0 input-file [DT value] [-debug]\n"; exit;}

if ($debug) { print "Reading input file...";}
open IN , $file or die "Error: Could not open input file";       #
Open input file with sequence numbers
while (<IN>) { chomp; push @seq,$_; }
close IN;
if ($debug) { print "Read ".($#seq+1)." sequence numbers\n"; }

$pl = 1;

# Read in first DT+1 non-duplicate sequence numbers or whole
sequence if lesser than DT+1
foreach (0..$DT)
{
    do {$new_seq = &get_new_seq; }
    while ($new_seq &&
           ((&exists_in_array($new_seq,@buffer) != -1) ||
            (&exists_in_array($new_seq,@window) != -1) ||
            ($new_seq < $pl)));
    if ($new_seq) { push @window,$new_seq; } else { last; }
}

if ($#window == -1) {print "No sequence numbers to process\n";
exit;}

while(1)
{
    $pl_source = 0;
    if ($debug)
    {
        if ($debug) { print "Currently:\n";
        print "\tPl = $pl\n\twindow=@window\n\tbuffer=@buffer\n"; }
    }
    $pl_position = &exists_in_array($pl,@window);
```

```
    if ($pl_position == -1) {$pl_position =
    &exists_in_array($pl,@buffer); $pl_source = 1;}

    if ($pl_position != -1)
    {
        $reordering = $pl - $window[0];
        if ($debug) {print "Found reordering = $reordering for
        $window[0]\n";}
        if (abs($reordering) <= $DT)
        {
           $FLE{$reordering}++;
           if ($pl_source == 1)
           {
              if ($debug) { print "Pl = $pl found in buffer\n"; }
              splice @buffer,$pl_position,1;
           }
           if ($reordering < 0) { push @buffer , $window[0]; }
           $pl++;
        }
        else { if ($debug) {print "Reordering more than DT, ignoring
        the packet...\n";}}
        shift @window;
        do {$new_seq = &get_new_seq; }
        while ($new_seq &&
               ((&exists_in_array($new_seq,@buffer) != -1) ||
                (&exists_in_array($new_seq,@window) != -1) ||
                ($new_seq < $pl)));
        if ($new_seq) {push @window,$new_seq;}
        last if ($#window < 0);
    }
    else
    {
        $m = &min_in_array(\@window);
        if ($pl < $m) {$pl = $m;} else {$pl++;}
    }
}

if ($debug) { print "\n\nFinal RD\n";
print "DT = $DT\n"; }

$sum = 0;
foreach $i(keys(%FLE))
{
   $sum += $FLE{$i};
}
if ($sum)
{
   foreach (sort {$a <=> $b} keys(%FLE))
   {
       $freq = $FLE{$_}/$sum;
       print "RD[$_]\t $FLE{$_}\t $freq\n";
   }
}
else
{
       print "All zero frequency\n";
}
```

APPENDIX B

I. Script to generate sequence numbers from readable tcpdump

```
#!/usr/bin/perl

#########################################################################
#
# tdparse parses tdpdump text output to generate "index" type
# sequence numbers,
# e.g. 1,2,3...
#
# Usage:
# tdparse input_file [output_file]
#
# 1. The input file must contain tcpdump output from only one TCP
# session i.e. strictly from one IP:port number pair to another
# IP:port number pair. It does
# not check if all packets in the input file confirm to this
# requirement. If the input file contains output of multiple TCP
# sessions, the sequence numbers will be mixed up and the output is
# undefined.
# 2. If output_file is not provided, it prints output to stdout.
#########################################################################

die "Input file not provided\n" if ($#ARGV < 0);
open IN, $ARGV[0] or die "Could not open file $ARGV[0] for
reading.\n";

# Parse input file to collect sequence numbers
$line_no = 0;
<IN> # discard first line
while (<IN>) {
  @parts = split;
  next if ($parts[5] =~ /ack/i); # Ignore "only ACK" packets
  ($seq, $ignore, $len) = split /[:\(\)]/, $parts[5];
  push @seq_nos, $seq;
  print "Line $line_no: Packet of length 0 detected.\n" if ($len ==
  0);
  $line_no++;
}
close IN;

# Create a mathematical "set" of sequence numbers to remove
# duplicates and sort them in order for mapping to "index" sequence
# numbers 1,2,3...
# This does not mean that duplicate packets are removed from output.
# This'll help them get same index sequence number.
```

```
%seq_nos_set = map { $_ => 1 } @seq_nos;
@sorted_seq_nos = sort {$a <=> $b} keys(%seq_nos_set);

# Map arrived sequence numbers to sorted sequence numbers to find
# their index.

foreach $arrv (@seq_nos) {
  for ($n = 0; $n <= $#sorted_seq_nos; $n++) {
    if ($arrv eq $sorted_seq_nos[$n]) {
      push @indices, $n+1;
      last;
    }
  }
}

# Print output. If output file is not given, print to stdout

if ($ARGV[1]) {
  open OUT, ">$ARGV[1]" or die "Could not open $ARGV[1] for
writing.\n";
}
else {
  *OUT = STDOUT;
}
foreach (@indices) { print OUT "$_\n"; }
```

APPENDIX C

I. Algorithm to compute RBD

```
E: Next expected sequence number.
S: Sequence number of the packet just arrived.
buffer: Hypothetical buffer.
B: Current buffer occupancy.
BT: Occupancy threshold.
FB[i]: Frequency of buffer occupancy i (0 <= i <= BT).
in_buffer(N): True if the packet with sequence number N is
already stored in the buffer.

1. Initialize E = 1, B = 0 and FB[i]=0 for all values of i.
2. Do the following for each arrived packet.

If (in_buffer(S) || S < E) /* Do nothing */;
/* Case a: S is a duplicate or delayed packet. Discard the packet */
Else
{
   If (S == E)
   /* Case b: Expected packet has arrived */
   {
      E = E + 1;
      While (in_buffer(E))
      {
         B = B - 1; /* Free buffer occupied by E */
         E = E + 1; /* Expect next packet */
      }
      FB[B] = FB[B] + 1; /* Update frequency for buffer occupancy B */
   } /* End of ElseIf (S == E)* /
   ElseIf (S > E)
   /* Case c: Arrived packet has a sequence number higher than expected
   */
   {
      If (B < BT) /* Store the arrived packet in a buffer*/
         B = B + 1;
      Else /* Expected packet is delayed beyond the BT. Treat it as
      lost */
      {
         Repeat
         {
            E = E + 1;
         }
         Until (in_buffer(E) || E == S);

         While (in_buffer(E) || E == S)
         {
            If (E != S) B = B - 1;
            E = E + 1;
         }
      }
      FB[B] = FB[B] + 1; /* Update frequency for buffer occupancy B */
```

```
    } /* End of ElseIf (S > E)*/
}

3. Normalize FB[i] to get RBD[i] for all values of i using

                        FB[i]
    RBD[i] = ----------------------------------
                Sum (FB[j] for 0 <= j <= BT)
```

IV. Algorithm to compute Loss Orthogonal RBD, Stay-back Method

```
S, buffer, B, BT, FB[i] : Same as in algorithm for RBD
E: Currently expected sequence number used for processing S.
newS: Next arriving sequence number.
newE: Next expected sequence number used for updating EW.
window[1..BT+1]: List of incoming sequence numbers.
TW[1..BT+1]: Threshold window.
EW[1..BT+1]: Expected sequence numbers queue.

1. Initialize.
   Store first unique BT+1 sequence numbers in arriving order into
   window;
   Copy all elements in window to TW;

2. Repeat.
   # The smallest element in TW is taken as newE to neglect lost packets
   newE = smallest sequence number in TW;
   Delete newE from TW;
   Add newE to EW;

   # Following is regular RBD processing
   S = first sequence number in window;
   Delete first sequence number from window;
   E = first sequence number in EW;
   If (S == E) # Next expected packet arrived
   {
      Delete E from EW; # Sequence number is not expected anymore
      k = first sequence number in EW;
      While (buffer contains sequence number k)
      {
         Delete k from both EW and buffer;
         k = first sequence number in EW;
      }
   }
   Else add S to buffer; # S is an early arrival. Add it to the buffer
   B = number of occupied slots in buffer;
   FB[B]++; # Increment frequency of current buffer occupancy

   # Get next incoming non-duplicate sequence number, if any
   # A sequence number <= newE or already present in TW is duplicate
   do
   {
      newS = get next incoming sequence number;
   }
   while (newS is valid and (newS <= newE or newS is in TW));
   if (newS is valid)
   {
      Add newS to window;
      Add newS to TW;
   }
```

```
if (window is empty) go to step 3;
Normalize FB to get LOSS ORTHOGONAL RBD.
```

V. Algorithm to compute Loss Orthogonal RBD, Go-back Method

```
E, S, buffer, B, BT, FB[i] : Same as in algorithm for RBD
go_back_buffer: Queue to store packets for reprocessing.
go_back_index: Index to first packet in buffer to reprocess.
lost_packet: Highest lost sequence number

1. Initialize
   E = 1;
   go_back_index = 1;

2. Repeat
   /* Try to get next S from go-back buffer first */
   If (go_back_buffer is not empty) S = dequeue element from
   go_back_buffer;
   Else S = get new incoming sequence number;
   If (S is not valid) Go to step 3.
   If (buffer contains S or S < E) discard S; /* Duplicate packet */
   ElseIf (S == E)
   {
      E++;
      While (buffer contains E)
      {
         Delete E from buffer.
         E++;
      }
      B = current buffer occupancy;
      go_back_index = B + 1;
      FB[B]++;
   }
   Else
   {
      B = current buffer occupancy;
      If (B < BT)
      {
         Append S to buffer.
         FB[B+1]++;
      }
      Else
      {
         /* Find the lost packet */
         lost_packet = smallest sequence number in the buffer - 1;
         If (lost_packet > S - 1) lost_packet = S - 1;
         /* To process the lost packet as if it actually arrived */
         Enqueue lost_packet into go_back_buffer.
         /* Prepare for reprocessing packets in buffer */
         FB[go_back_index - 1]--;
         For (k = go_back_index to BT)
         {
            FB[k]--;
            Enqueue kth element from buffer into go_back_buffer;
         }
         Delete elements at indices (go_back_index to BT) from buffer.
         Enqueue S into go_back_buffer;
      }
   }

3. Normalize FB[B] to get L-RBD[B].
```

(The following programs/scripts are authored by Abhijit A. Bare)

IV. C Listing to Compute RBD

```
/*
Description of variables used

   E : Next expected sequence number.
   S : Sequence number of packet just arrived.
   D : Estimated displacement of the arrived packet from its
expected
       position in sequence.
   DT: Displacement threshold. (DT is also the size of the buffer
       allocated.)
   F[i]: Frequency of occurrence of D = i.
   in_buffer(N) : True if the packet with sequence number N is
already stored in the buffer.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define FIRST_SEQUENCE_NUMBER 1
#define DEFAULT_DISPLACEMENT_THRESHOLD 10
#define TRUE 1
#define FALSE 0

FILE *fr,*fw;
long *F,*buffer;
int DT;

void compute_histogram();
void write_output();
void store_in_buffer(long);
void free_buffer(long);
int in_buffer(long);

void print_buffer();

/***
Main function
***/
int main(int argc, char *argv[])
{
   int count = 0,i;
   if (argc<2)
   {
         printf("COMMAND FORMAT IS : command sequence-file
         [displacement-threshold Default : %d] [output-
         file]\n",DEFAULT_DISPLACEMENT_THRESHOLD);
      printf("The input file should contain the sequence numbers
      one on each line, just as integers, followed by newline.");
      printf("Default first expected sequence number : %d\n",
      FIRST_SEQUENCE_NUMBER);
      exit(1);
```

```
    }

   fr = fopen(argv[1],"r");
   if (argc >= 3) DT = atoi(argv[2]);
   else DT = DEFAULT_DISPLACEMENT_THRESHOLD;
   if (argc == 4) fw = fopen(argv[3],"w");
   else fw = NULL;

   /* Frequency array */
   F = (long*) calloc(DT + 1 , sizeof(long));
   buffer = (long*)calloc(DT , sizeof(long));
   for (i = 0;i <= DT;i++) buffer[i] = -1;
   compute_histogram(count);
   write_output();
}

/***
Function to get the next number in arrived sequence. Here, the
program reads the sequence from a file.
***/

long nextSequenceNumber()
{
     int pkt;
     if (fscanf(fr,"%d",&pkt)!=EOF) return pkt;
     else return -1;
}

/***
Takes the arrived array with length = count and computes the
histogram in array F of length DT.
***/

void compute_histogram(int count)
{
   int i;
   long E = FIRST_SEQUENCE_NUMBER,S;
   int D = 0;

   /* For each packet in the sequence */
   while ((S = nextSequenceNumber()) != -1)
   {
      //printf("%d %d\n",S,D);
      /*Case a: S is a duplicate or delayed packet. Discard the
packet.*/

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

      /*Case b: Expected packet has arrived.*/

      else if (S == E)
      {
         E++;
         while (in_buffer(E))
         {
            D--;
            free_buffer(E);
```

```
            E++;
         }
         F[D]++;
      }

      /*Case c: Arrived packet has a sequence number higher than
expected.*/

      else if (S > E)
      {
         if (D < DT)
         {
            D++;
            store_in_buffer(S);
         }
         else
         {     /*Expected packet is delayed beyond DT. Treat it
as lost.*/
            do
            {
               E++;
            }
            while(!(in_buffer(E)||(E==S)));
            while(in_buffer(E)||(E==S))
            {
               if (E!=S)
               {
                  D--;
                  free_buffer(E);
               }
               E++;
            }
            if (E < S)
            {
               D++;
               store_in_buffer(S);
            }
         }
         F[D]++;
      }
      //F[D]++;
    }

}

/***
Following are the functions used to manage the hypothetical
buffer.
***/

void store_in_buffer(long sequence_number)
{
     int i;
     for (i = 0; i < DT;i++)
         if (buffer[i] == -1)
         {
            buffer[i] = sequence_number;
```

```
            return;
         }
}

void free_buffer(long sequence_number)
{
     int i;
     for (i = 0; i < DT;i++)
         if (buffer[i] == sequence_number)
         {
            buffer[i] = -1;
            return;
         }
}

int in_buffer(long sequence_number)
{
     int i;
     for (i = 0; i < DT;i++)
         if (buffer[i] == sequence_number) return TRUE;
     return FALSE;
}
/***
Following function prints the output either to the screen or to
the specified output file.
***/

void write_output()
{
     int i;
     long F_sum = 0;
     for (i=0;i<=DT;i++) F_sum+=F[i];
     if (fw == NULL) printf("%s %s %s
\n","displacement","absolute","normalized");
     else
        fprintf(fw,"%s %s %s
\n","displacement","absolute","normalized");
     if (F_sum!=0)
     {
        for (i=0;i<=DT;i++)
        {
            if (fw == NULL) printf("%d %d %5.3lf\n" , i , F[i] ,
            (double)F[i]/(double)F_sum);
            else fprintf(fw , "%d %d %5.3lf\n" , i , F[i] ,
            (double)F[i]/(double)F_sum);
        }
     }
     else
     {
        if (fw == NULL) printf("All zero frequency, cannot
        compute normalized frequency.\n");
        else fprintf(fw ,"All zero frequency, cannot compute
       normalized frequency.\n");
     }
}

/***
```

```
Use following function to print the buffer, for debugging
purpose.
***/

void print_buffer()
{
     int i;
     printf("buffer = ");
     for (i = 0; i < DT;i++)
         printf("%ld ",buffer[i]);
     printf("\n");
}
```

V. Perl Script to Compute Loss Orthogonal RBD using Go-back Method

```
#!/usr/bin/perl -w

### Perl script to compute RBD using goback method
### The script cannot handle the sequences for which
### packets remain in buffer even at the end of the sequence

### The algorithm is expected to generate same results as the
stay-back RBD

my (@buffer , @gobackbuffer , @F);
my ($S,$E,$BT,$buffer_size_exp_rcvd);

if ($#ARGV == 2) { $BT = $ARGV[1]; $file = $ARGV[0];
if ($ARGV[2] eq "-debug") { $debug = 1; } }
elsif ($#ARGV == 1) { $BT = $ARGV[1]; $file = $ARGV[0];}
elsif ($#ARGV == 0) { $BT = 3; $file = $ARGV[0]; }
else { print "Parameters: input-file [BT value] [-debug]\n";
exit;}

if ($debug) {print "Reading input file...";}
open IN , $ARGV[0] or die "Error:Could not open input file";    #
Open input file with sequence numbers
while (<IN>) { chomp; push @seq,$_; }
close IN;
if ($debug) {print "Read ".($#seq+1)." sequence numbers\n"; }

$lost_flag = 0;
$buffer_size_exp_rcvd = 0;
$E = 1;
foreach (0..$BT) { $F[0] = 0; }

while(1)
{
    if (!$lost_flag)
    {
        if (! defined($S = shift @gobackbuffer)
           && ! defined($S = shift @seq))
        { last; }
    }
    else { $lost_flag = 0; }
```

```
    if ($debug) { print "S = $S E = $E -> "; }

    if ((&exists_in_array($S,@buffer) != -1) || ($S < $E) ) {
print "Duplicate $S\n" if ($debug); } # Do nothing

    # Following is normal RD computation code

    if ($S == $E)       # Equal condition
    {
        while(1)
        {
            $E++;
           if (($buffer_i = &exists_in_array($E, @buffer)) != -1)
            # if the new expected exists in RD buffer
            {
                if ($debug) { print "Freeing buffer of $E at pos
                $buffer_i\n" if $debug;}
                splice @buffer,$buffer_i,1; # Remove k from
                buffer
            }
            else { last; }
        }
        $buffer_size_exp_rcvd = $#buffer + 1;
        $F[$#buffer + 1]++;
    }
    elsif ($S > $E)
    {
        if ($#buffer + 1 < $BT)
        {
           # if S != E, we need to keep it in RD buffer
           push @buffer,$S;
           $F[$#buffer + 1]++;
        }
        else
        {
            $lost_packet = minimum((@buffer , $S)) - 1;
            print "\nLost = $lost_packet\nRemoving frequency
            increments for packets to be reprocessed...\n" if
            ($debug);
            $F[$buffer_size_exp_rcvd]--;
            foreach ($buffer_size_exp_rcvd..$BT-1)
            {
                $F[$_ + 1]--;
                push @gobackbuffer , $buffer[$_];
            }
            splice @buffer , $buffer_size_exp_rcvd , ($BT -
               $buffer_size_exp_rcvd);
            push @gobackbuffer , $S;
            $E = $lost_packet;
            $S = $lost_packet;
            $lost_flag = 1;
            print "Gobackbuffer = @gobackbuffer\n" if ($debug);
        }
    }

    if ($debug) { print "S = $S E = $E Buffer = @buffer RD =
@F\n"; }
```

```
}

&print_output;

# Utility subroutine: returns index of an element
# if it exists in the array given else -1
sub exists_in_array()
{
    my ($key,@in) = @_;
    my $i = 0;
    if (defined($key) && ($#in >= 0))
    {
        foreach $k(@in)
        {
            return $i if ($key == $k);
            $i++;
        }
    }
    return -1;
}

sub minimum
{
    @arr = @_;
    if ($#arr < 0) { return (-1,-1); }
    $min = $arr[0];
    $pos = 0;
    foreach $i (1..$#arr)
    {
        if ($arr[$i] < $min)
        {
            $min = $arr[$i];
            $pos = $i;
        }
    }
    return $min;
}

sub print_output
{
    foreach (@F) { $sum += $_; }
    print "Final RD:\n";
    if ($sum)
    {
        foreach (0..$#F)
        {
            print "$_ $F[$_] ".$F[$_]/$sum."\n" if ($F[$_]);
        }
    }
    else { print "Freq sum ZERO !!\n"; }
}
```

VI. Perl Script to compute Loss Orthogonal RBD using Stay-back Method

```
#!/usr/bin/perl -w

# Version 2

#tw Thershold Window
#sw Sequence Window
#ew Expected Window
#buffer RD Buffer

my($BT,$i,$tw_i,$newE,$new_seq,$buffer_i,$seq_i);
my(@tw,@sw,@ew,@buffer,@seq);

my $debug = 0 ,$step=1;

if ($#ARGV == 2) { $BT = $ARGV[1]; $file = $ARGV[0];
if ($ARGV[2] eq "-debug") { $debug = 1; } }
elsif ($#ARGV == 1) { $BT = $ARGV[1]; $file = $ARGV[0];}
elsif ($#ARGV == 0) { $BT = 3; $file = $ARGV[0]; }
else { print "Format: bd input-file [BT value] [-debug]\n";
exit;}

if ($debug) {print "Reading input file...";}
open IN , $ARGV[0] or die "Error:Could not open input file";   #
Open input file with sequence numbers
while (<IN>) { chomp; push @seq,$_; }
close IN;
if ($debug) {print "Read ".($#seq+1)." sequence numbers\n"; }
$seq_i = 0;

# Read in first BT+1 non-duplicate sequence numbers or whole
sequence if lesser than BT+1
foreach (0..$BT)
{
    $new_seq = shift @seq;
    if (!defined($new_seq)) { last; }
    elsif (&exists_in_array($new_seq,@tw) == -1)
    {
   push @tw,$new_seq;
    }
}

@sw = @tw;

if ($debug) { print "Initial Windows::\n"; &print_windows(); }

if ($#sw == -1) {print "No sequence numbers to process\n";exit;}

while(1)       # Do forever
{
    ($newE,$pos) = &minimum(@tw);
    if ($debug) { print "Min $newE at $pos in TW\n"; }
    if ($pos != -1)
    {
```

```
        # Append the anticipated packet at the end of EW
         push @ew,$newE;
        # Remove the anticipated packet from the TW, pos is its
        # index as returned by sub minimum
         splice @tw,$pos,1;
    }

# Following is normal RD computation code

    $S = $sw[0];            # First packet in window
    $E = $ew[0];            # First packet in EW
    if ($S == $E)           # Equal condition
    {
   shift @ew;         # Remove the expected packet, E, from EW
   @curr_ew = @ew;          # Satisfy perl for foreach operation
   foreach $k(@curr_ew)
   {
       if (($buffer_i = &exists_in_array($k, @buffer)) != -1)
       # if the new expected exists in RD buffer
       {
         if ($debug) { print "Freeing buffer of $k at pos
         $buffer_i\n";}
         shift @ew;  # Remove k from EW
         splice @buffer,$buffer_i,1;   # Remove k from buffer
       }
       else
       {last;}
   }
    }
    else
    {
   push @buffer,$S;
   # if S != E, we need to keep it in RD buffer
    }
    $F[$#buffer + 1]++;
    # Increment frequence of current buffer occupancy
    shift @sw;              # Remove first packet from window
    if ($new_seq = &get_new_seq)
    {
   push @tw,$new_seq;
   # Pull in new packet from the source and append at TW end
   push @sw,$new_seq;# Append the new packet to SW
    }

    $seq_i++;         # Move to next packet in sliding window

    if ($debug)
    {
   print "At end of Step $step:\n";
   &print_windows();
        $step++;
    }

    if ($#sw == -1)  # if there is nothing in SW
    {
   foreach (0 .. $#buffer)
   {
```

```
        $F[0]++;        # Just increment frequency of 0 for each
                        packet in buffer
    }
    last;               # And BREAK main loop
     }
}

if ($debug) { print "\n\nFinal RD\n";
         print "BT = $BT\n";}
$sum = 0;
foreach $i(0 .. $#F)
{
     $sum += $F[$i];
}
if ($sum)
{
   foreach $i(0 .. $#F)
   {
      $freq = $F[$i]/$sum;
            print "RD[$i] $F[$i] $freq\n";
   }
}
else
{
   print "All zero frequency\n";
}


sub get_new_seq()
{
     my $new_seq;
     while (1)
     {
   $new_seq = shift @seq;
   if (!defined($new_seq)) { return $new_seq; }
   elsif ((exists_in_array($new_seq,@tw) == -1) &&
   # packet does not alreay exist in TW
          ($new_seq > $newE))
   # packet seq is greater than or equal to current anticipated
   value
   {
       return $new_seq;
   }
   else {print "\n****\nFound duplicate (or ignored)
$new_seq\n****\n";}
     }
}

sub print_windows()
{
     print "\t\tTW = @tw\n";
     print "\t\tEW = @ew\n";
     print "\t\tSW = @sw\n";
     print "\t\tbuffer = @buffer\n";
     print "\t\tbuffer size = ".($#buffer + 1)."\n";
}
```

```
sub exists_in_array()       # Utility subroutine: returns index of
an element if it exists in the array given else -1
{
    my ($key,@in) = @_;
    my $i = 0;
    if (defined($key) && ($#in >= 0))
    {
   foreach $k(@in)
   {
       return $i if ($key == $k);
       $i++;
   }
    }
    return -1;
}

sub minimum
{
    @arr = @_;
    if ($#arr < 0) { return (-1,-1); }
    $min = $arr[0];
    $pos = 0;
    foreach $i (1..$#arr)
    {
         if ($arr[$i] < $min)
         {
             $min = $arr[$i];
             $pos = $i;
         }
    }
    return ($min,$pos);
   }
```

APPENDIX D

I. Perl Script to extract delay from tcpdump at the UMass receiver (sender is an ixia box).

```
#!/usr/bin/perl

# This program extracts delay from the tcpdump ran with -x and -
# tt option for 64 bytes. Avg. one-way delay to UMass is 28.5ms

use Math::BigInt;
open(IN,$ARGV[0])|| die "Could not open file $ARGV[0] : $!\n";
open(OUT,">$ARGV[1]");
$fir_delay = 0;
$rec_time = 0;
$umass_avg_delay = 28500;
while($str=<IN>)
{
    $recvtime = new Math::BigInt $rec_time;
    $recvtime = substr($str, 4, 10);
    $recvtime = $recvtime*1000000;
 #  print $recvtime;
     $sen_time = "0x";
    $str1=<IN>;
    $str2=<IN>;
    $str3=<IN>;
    $sen_time0 = substr($str3, 24, 4);
    $sen_time1 = substr($str3, 29, 4);
    $sen_time2 = substr($str3, 34, 4);
    $sen_time = "0x".$sen_time0.$sen_time1.$sen_time2;
#   print $sen_time;
    $sendtime = new Math::BigInt $sen_time;
    $sendtime = $sendtime*20/1000;
#   print $sendtime;
#   print "\n";
    $delay = $recvtime - $sendtime;
    if($fir_delay == 0)
    {
         $fir_delay = $delay;
    }
#   print $delay;
    $delay = $delay - $fir_delay + $umass_avg_delay;
    print OUT ($delay)."\n";
    #if ($delay == 1) {print $recvtime,$sendtime,$delay,"\n";}
    #if ($SHOW) {print "\n";}

}
close(IN);
close(OUT);
```

```
close(OUT);
```

VII. Perl Script to extract inter-packet gap from tcpdump

```
#!/usr/bin/perl

# This program extracts delay from the tcpdump ran with -x
# and -tt option for 64 bytes.

open(IN,$ARGV[0])|| die "Could not open file $ARGV[0] : $!\n";
open(OUT,">$ARGV[1]");
$first=1;
while($str=<IN>)
{

 $recv_time = substr($str,0,17);
 $seq1 = "0x";
 #print "$recv_time\n";
 <IN>;
 <IN>;
 $str = <IN>;
 $tokens = substr($str,4,4);
 $seq1=$seq1.$tokens;
 $sequence = hex($seq1);
 #print "$sequence\n";
 if (!($first))
 {
     if($sequence-$prev_seq == 1)
     {

       $diff = $recv_time-$prev;
       #$diff = substr($diff,0,8);
       print OUT "$diff"*1000,"\n";
     }
     else
     {
    print "Lost found $sequence\n";
      }
 }
 else
 {
     $first=0;
     printf("\nFound first seq %d\n",$sequence);
 }
 $prev = $recv_time;
 $prev_seq = $sequence;
}
printf("Found last seq %d\n\n",$sequence);
close(IN);

close(OUT);
```

VIII. Matlab code extract correlogram from a file containing values

```
% This script reads the values in a file and returns
% correlation coefficient up to 20 lags
```

```
clear all;
close all;
lag_limit=20;

filename = input('Enter Filename: ','s');
fid=fopen(filename);
values=fscanf(fid,'%g',[1,inf]);
fclose(fid);
n=length(values);
for (j=1:lag_limit)
    corrf = corrcoef(values(1:n-j),values(j+1:n));
    corr(j)=corrf(1,2);
end;
outfile = strcat(filename,'.corr');
fid=fopen(outfile,'w');
for (j=1:lag_limit)
    fprintf(fid,'%d %f\n',j,corr(j));
end;
fclose(fid);
```