Chapter 5

RESULTS OF PERFORMANCE EXPERIMENTS

5.1 Introduction

This chapter presents the core of this dissertation. While many of the side-products of this investigation have proved useful and significant, it is this chapter that directly addresses the problem statement first presented in Section 1.1, and repeated here for the reader's convenience:

> ***To determine through experimentation with real systems the extent to which PO/PR transport service can provide performance benefits for real applications***.

In this chapter, we present results from performance experiments that address this question.

5.1.1 Goals and limitations of our investigation

This section describes the limitations of our investigation and clarifies our goal with respect to the problem statement cited above.

First, consider the term *real applications*. In this investigation, we examine only one application for PO/PR service, namely multimedia document retrieval. Using PO/PR service to support multimedia document retrieval was first proposed in (Amer et al., 1994), and developed further in (Conrad et al., 1996). Only one other concrete application has been proposed for PO/PR transport service:

(RFC1693) proposes using PO/R service to transmit the results of database queries. Section 7.2 comments further on the applicability of our results to this application.

For other hypothetical applications, we can produce only negative results. For example, we can show a range of network conditions under which the benefits of partial order in terms of delay or throughput are *zero*. However, if the benefit is *anything* other than zero, we cannot determine the significance of the gain except in the context of a particular application, because *any* gain, no matter how arbitrarily large or small, might be significant, or insignificant in a particular set of circumstances. An improvement in delay or throughput that is significant in one context may be meaningless in another. The fact that we cannot extrapolate the significance of a performance improvement from one application to another highlights the advantage—as well as the disadvantage—of examining PO/PR transport service in the context of a *specific* application.

We should also recognize that even within the context of ReMDoR, our experiments are limited to specific documents. We have tried to use documents for our experiments that are reasonable examples of multimedia, similar to those seen in practice. However, we make no claim that our documents are representative in any scientific sense. One might imagine that someday we might be able to perform experiments on randomly selected multimedia documents from a huge corpus of literature, as is the practice of researchers in natural language processing. However, no such corpus of PMSL documents exists—not yet, in any case.

Thus, our goal is to find some specific circumstances under which PO/PR service provides benefits for some specific documents, and then to investigate these circumstances fully, examining the effect of varying network parameters on the gain in

performance. By understanding these effects, we can better evaluate the potential of PO/PR service as a useful addition to the range of services provided by the transport layer.

5.1.2 Organization of this chapter and overview of performance experiments.

This section describes the organization of this chapter. It also provides an overview of the performance experiments included in this dissertation, with a brief description of the role that each plays in our investigation of PO/PR transport protocols. (The reader may wish to review the introduction in Section 1.7 and the notation in Table 1.3 before proceeding.)

Section 5.1 provides background information about the performance experiments in general. It includes:

- a description of the overall experimental framework (Section 5.1.3), and
- specific sections on the lossy router and the packet reflector (Sections 5.1.4 and 5.1.5), including a discussion of how the correct operation of these components was verified.

The core of the chapter, Sections 5.2 through 5.7, describes the results of our experiments. In each case, the ReMDoR system is used to retrieve a document using two or more transport protocols. We present a comparison of the performance statistics for each protocol to see if one protocol outperforms the other(s). Depending on the experiment, we investigate the effect of packet loss rate, bitrate, propagation delay, and/or window size.

- Section 5.2 describes Experiment **N1**, comparing ordered/reliable service with unordered/reliable service using network-conscious images produced with **N**ETCICATS (Iren,1999b). The improvement of an unordered/reliable service over an ordered/reliable service represents

the limiting case for gains from any partially ordered/reliable transport service.

- Sections 5.3, 5.4 and 5.5 describe Experiments **R1**, **R2** and **R3**, which use the **R**eMDoR application to compare ordered/reliable service to partially-ordered/reliable service for retrieval of a document with eight images presented in parallel. The main difference between these experiments and Experiment **N1** is that for **R1**, **R2** and **R3,** images are compressed using the traditional GIF file format. GIF requires ordered/reliable delivery for each image, so unordered service cannot be used. However, partially-ordered service can be used because the data for each image can be interleaved in eight parallel streams.

- Section 5.6 describes Experiment **R4**, which compares ordered/reliable service to partially-ordered/reliable service for retrieval of a document with three images in parallel with a single audio clip. We consider performance statistics related both to the display of the images and the smoothness of the audio presentation.

- Section 5.7 describes Experiment **R5,** which compares ordered/reliable service to partially-ordered/reliable service for retrieval of entire documents. Where **R1, R2** and **R3** focus on small documents, here, we look at a full document in its entirety.

Following the experiment results, Section 5.8 surveys common problems that can arise in measuring system performance and how we dealt with these. Section 5.9 concludes the chapter, with some overall conclusions, an assessment of the significance of our results.

5.1.3 Experimental setup

Section 1.7 provided a high-level view of the experimental framework developed as part of this dissertation. In this section, we provide a more detailed view of this setup. Figure 1.2 showed a server and a client with a cloud abstracting the unreliable network connecting them. Figure 5.1 illustrates the details of that cloud.

The network always includes a 100Mbps Ethernet at the client side. In addition, it may include combinations of the following components:

(1) a ***lossy router*** (described in Section 5.1.4): an IP router that can, on request, purposely drop certain packets according to one of several loss models.[50]

(2) a ***packet reflector*** (described in Section 5.1.5): a traffic shaping program to simulate packet transmission delays, propagation delays, and queuing delays for a two-way UDP packet flow

(3) a ***PPP link***: a serial connection using the Point-to-Point Protocol (RFC1661) over an RS–232 null modem cable

(4) A ***wide-area Internet path.*** Although not used in any of the experiments described in this dissertation, the experimental framework includes provisions to do experiments between UD and other locations on the Internet. This capability was used in (Iren 1999b) to do experiments between UD and the Middle East Technical University in Ankara, Turkey. Planned future work involves using this capability for experiments between UD and Temple University, which although a short distance in geographical terms, is actually thirteen hops away in Internet distance.

[50] The initial implementation of the lossy router was completed by Golden as part of (Golden, 1997). The author of this dissertation designed the SLRP control protocol (described later in this section), and implemented a GUI client. The Bernoulli loss model was implemented by Golden; the deterministic and Gilbert loss models were added by Iren (Iren, 1999b).

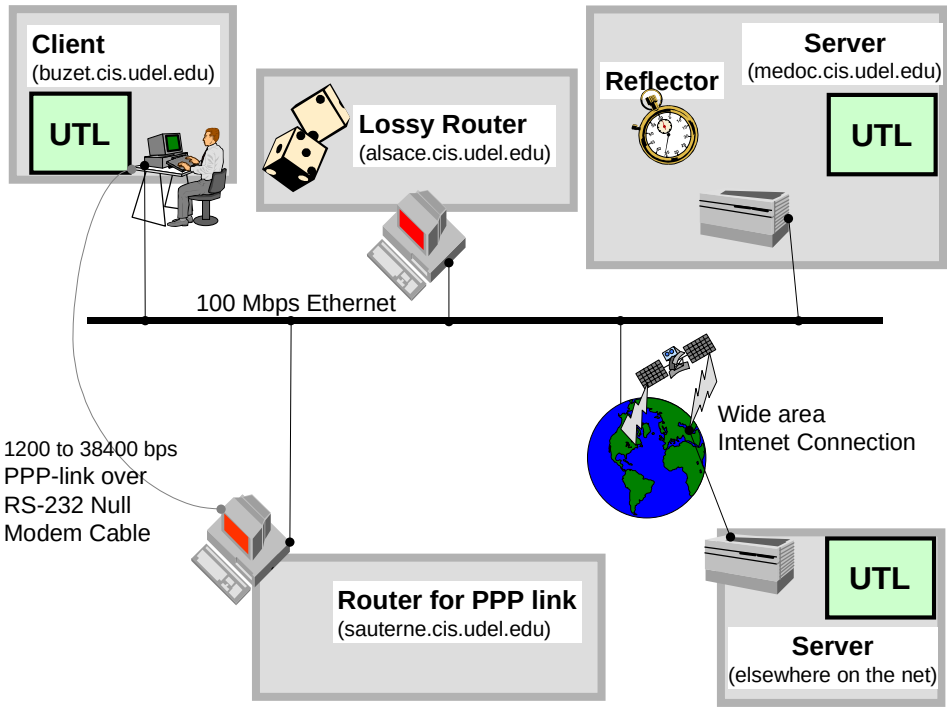


Figure 5.1 Detail of experimental environment

5.1.4 The lossy router

To simulate delay and loss, we use a lossy router, developed at the UD Protocol Engineering Lab (PEL). While the hosts used in our experiments have only one Ethernet interface, attached to a single 100Mbps Ethernet hub, each host is assigned *two* distinct IP addresses: one for normal traffic, and one for experimental traffic on the so-called *loss network*. The loss network (domain name: `loss.udel.edu`) is a virtual network that runs in parallel with the regular network (`cis.udel.edu`) over the same Ethernet hardware. The router for this network is a machine called `alsace.loss.udel.edu`. When a host on the loss network (say,

`buzet.loss`)[51] has a packet destined for another host on the loss network (say, `medoc.loss`), rather than placing the packet on the Ethernet with the destination host's Ethernet address, the host instead sends the packet to the lossy router (`alsace.loss`). Although `alsace.loss` acts as a router for the `loss` network, the normal IP forwarding mechanisms on `alsace` are disabled. Instead, a user-level program developed at UD intercepts incoming packets and performs the routing function, deliberately introducing packet loss according to one of three available models:

(1) Bernoulli: a uniformly distributed pseudo-random number is generated, and the packet is either forwarded or dropped, based on the outcome (parameter: packet loss probability *p*)

(2) Deterministic: every kth packet is dropped (parameter: *k*)

(3) Gilbert-Elliot: loss is modeled by a Markov chain with 2 states: *bad* (packets are dropped) and *good* (packets are forwarded). (parameters: p and q, as in Figure 5.2)(Gilbert, 1960; Elliot, 1963; Ebert and Willig, 1999)

To enable automated experimentation, a control protocol was developed called the Simple Lossy Router control Protocol (SLRP). This protocol is an ASCII protocol similar in structure to FTP, NNTP or SMTP:

- request PDUs begin with four character command codes, and
- response PDUs begin with three digit response codes indicating the success or failure of the command, and in some cases returning statistics.

[51] Henceforth, we drop the `.udel.edu` suffix; for example, `buzet.cis` refers to `buzet.cis.udel.edu`, and `buzet.loss` refers to `buzet.loss.udel.edu`

Commands allow experimenters to change the loss model and parameters either interactively via a telnet or GUI client, indirectly from a Perl or Tcl/Tk script, or from a program using TCP sockets.

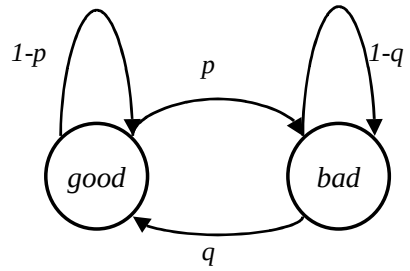


Figure 5.2 Gilbert-Elliot loss model (two state Markov chain)

The experiments described in this chapter use only the Bernoulli loss model. Future work includes performing experiments based on the Gilbert-Elliot loss model to study the effect of burst losses, such as might be caused by queue overflows, momentary power outage in a router, or noise bursts on a wireless channel.

Validation of the lossy router

We took several steps to ensure that the lossy router operates correctly for the Bernoulli loss model. First, we tested the lossy router using the standard Unix `ping` utility at a variety of loss rates ranging from 0% to 100%. Second, we performed a code review involving several members of the PEL staff, in which we reviewed the source code line-by-line to increase our confidence in its correctness. Third, we measured the actual packet loss obtained in each of our experiments by analyzing a packet trace. The remainder of this section provides more information

about how we took this measurement, and some sample data to back up the claim that the lossy router operates correctly.

Unless otherwise noted, every experiment reported in this dissertation involves repeating an experiment many times, and taking the average and standard deviation across many repetitions of the experiment. The lossy router maintains counters of packets forwarded and dropped that can be reset and inquired against. Before each repetition of every experiment, we reset these counters, and following every repetition, we retrieved the values of these counters. Therefore, we know the actual number of packets forwarded and dropped in each experiment, as reported by the lossy router. Based on the evidence in these logs, we conclude that the mean and standard deviation of the actual loss rate for traffic through the lossy router is reasonably close to the values predicted by the formulas for the mean and standard deviation of a Bernoulli distribution.

Of course, the conclusion in the previous sentence is predicated on the assumption that the lossy router is providing accurate reports of its own behavior. To validate that the counters reported by the lossy router are accurate, we also take a packet trace of every experiment, and use a Perl script to analyze this output. The script counts the number of packets sent from client-to-server and server-to-client in each experiment, and the number of packets dropped by the lossy router, dividing to get the actual loss rate for each direction. We can then cross check this against the report by the lossy router. Our analysis of the packet traces for the experiments leads us to conclude that the lossy router's reporting about itself is accurate.

5.1.5 The packet reflector

To simulate various bandwidths we used the UDP packet reflector developed at the UD Protocol Engineering Lab. The packet reflector is a user-level process that forwards UDP packets between a client and a server, simulating the effects of queuing delays at a slow PPP link, and propagation (speed of light) delays for long-haul transmission.

The traffic shaping done by the reflector has characteristics of both the classic *leaky bucket* and *token bucket* methods (Tanenbaum, 1996). The goal of both of the classic schemes is to limit traffic to a maximum bit rate of *b*. The difference between the schemes is that:

- In the leaky bucket scheme, the upper bound on the bit rate is enforced on a packet-by-packet basis. When the queue is non-empty, packets leave the queue at a steady rate of *b* bits per second, similar to the flow of water through a hole in the bottom of a bucket. Bursts of transmission higher than bitrate *b* are disallowed.
- In the token bucket scheme, tokens are added to a bucket at a steady rate, and are used to bound the long-term average bitrate. Bursts of packets may be sent at a bitrate higher than *b* to make up for past underutilization of the allocated bandwidth.

As the remainder of this section explains, the model for the PEL reflector is the leaky bucket scheme. However, due to context switching, the reflector may occasionally fall behind in sending out packets according to the transmission times calculated for the leaky bucket model. In this case, bursts of packets may be sent out to allow the reflector to catch up; this is more akin to the token bucket scheme.

The remainder of this section is in two parts: we first provide details of the modified leaky bucket scheme, and then describe how the reflector was validated.

Modified leaky bucket scheme used in the packet reflector

Each direction of the link is modeled independently. When each packet arrives, it is placed in the outgoing packet queue. The current time (noted as *now* in the equations below) is measured. Then three timestamps are calculated and associated with the arriving packet:

- t_{start}: first bit of the packet begins transmission
- t_{end}: last bit of the packet ends transmission
- t_{arr}: last bit of the packet arrives at the receiver, given the propagation delay t_{prop} that is being modeled by the reflector.

These values are calculated as follows. We start by calculating the value *totalLengthInBytes*, which is the sum of:

- the bytes of UDP data in the arriving packet, including the application layer data, and all UTL headers,
- 8 bytes for the UDP header,
- 20 bytes for the IP header, and
- 8 bytes for the PPP header.

This calculation is valid only assuming that IP fragmentation does not take place; which is a valid assumption for the experiments in this dissertation. We then calculate the timestamps for the arriving packet as follows:

if queue.isEmpty()

arrivingPacket.t_{start} ← *now*

else

arrivingPacket.t_{start} ← queue.tail.t_{end}:

arrivingPacket. t_{end} ← arrivingPacket.t_{start} + (8 * *totalLengthInBytes* / *b*)

arrivingPacket. t_{arr} ← arrivingPacket.t_{end} + arrivingPacket.t_{prop}

The reflector then operates on the assumption that the actual transmission and propagation delay is negligible on the 100Mbps Ethernet to which all the machines in the experimental setup are attached, and transmits each packet in the outgoing packet queue over the network at time packet. t_{arr}.

To ensure timely delivery, the reflector implements an infinite loop that blocks on a `select()` system call (see Section 3.5.7) with a delay value *d* calculated as follows:

- if both queues are non-empty, *d* is the earlier of t_{arr} values for the head packets
- if only one queue is non-empty, *d* is the t_{arr} value for the head of that queue
- if both queues are empty, *d* is NULL indicating that `select()` should block waiting for incoming data.

Provided that a context switch does not occur in the middle of the loop, and provided the `select()` system call does not return late, this algorithm correctly implements a leaky bucket model. On the other hand, if the `select()` system call returns late, the reflector may find that one or more packets have missed their deadlines for transmission. In this case all of these packets will be sent out in a burst as in the token bucket model, to allow the reflector to catch up.

Validation of the packet reflector

To verify that the packet reflector correctly models propagation delays and queuing delays, we performed tests at various bitrates and propagation delays, and took `tcpdump`s of the resulting traffic. We then calculated the correct sending times by hand, using a spreadsheet, and compared these with those shown on the tcpdump, to verify that the sending times were calculated and carried out correctly. Based on these results, we have confidence that the packet reflector correctly models bitrate, propagation delay, and queuing delay.

5.2 Experiment N1: showing the upper bound for performance gain (U/R vs. PO/R vs. O/R Service using NETCICATS)

This section describes Experiment **N1**, which compares unordered service with partially ordered service and ordered service using a document produced with the aid of the Network-Conscious Image Compression and Transmission System (NETCICATS) of (Iren, 1999b). This experiment is designed to investigate the following hypotheses:

> **Hypothesis 5.2.1:** At 0% loss, the progressive display of a document containing eight parallel network-conscious images will not change significantly, regardless of whether unordered, partially-ordered, or totally-ordered service is used.
>
> **Hypothesis 5.2.2**: As the loss rate increases from 0% to 10% to 20%, there will be increasing benefits to using unordered service over partially ordered service, and partially-ordered service over totally ordered service.
>
> **Hypothesis 5.2.3**: At the bitrates, propagation delays and loss rates tested, variations in processing and packet header overhead among the UTL mechanisms X2E, R2E, S2E and T2E will be measurable, but small in terms of end user impact.

The NETCICATS system allows us to produce image files where every ADU of the image can be decoded and displayed out of sequence, regardless of the reception of other ADUs. Therefore, using NETCICATS we can produce a document that allows us to compare the performance of ordered service vs. unordered service. The significance of this comparison is that *the improvement provided by unordered service over ordered service represents the upper bound* on any improvement that partially ordered service could ever provide.

This experiment uses the document `ncg8par.pmsl`, which contains eight images from the `military.pmsl` document (described in the appendix), all

sent in parallel. This simple document, illustrated in Figure 5.3, has no temporal aspects; it is similar to a static web page consisting of multiple images. Each of the eight images is compressed using the Network-Conscious Graphics Interchange Format (NCGif) format defined in (Amer et al., 1998).

Organization of Section 5.2

- Section 5.2.1 describes previous and concurrent work related to this experiment, specifically (Diot and Gagnon, 1999) and (Iren 1999).
- Section 5.2.2 describes the parameters for Exp. **N1**.
- Section 5.2.3 describes the format of the performance graphs we use to present our data for this experiment, and those throughout the chapter.
- Section 5.2.4 describes our observations for Exp. **N1** while Sections 5.2.5 through 5.2.7 explain the conclusions we draw based on these observations.
- Section 5.2.8 explains the significance of sample screen dumps of the ReMDoR performance experiments that we use throughout the chapter to illustrate the end-user impact of the numerical performance gains we report.
- Section 5.2.9 interprets a set of sample screen dumps for Exp. **N1** at 10% loss.
- Section 5.2.10 summarizes our conclusions for Exp. **N1**.

5.2.1 Unordered vs. ordered service: related work

In addition to looking at partially-ordered service, the experiments in this section also consider the benefits of unordered service. Our interest in unordered service is chiefly the upper bound on any gain that may be obtainable from partially-ordered service. However, the comparison of unordered vs. ordered service is also an interesting question in its own right (Iren, 1999b; Diot and Gagnon, 1999).

Unordered vs. ordered service (Iren, 1999b)

(Iren 1999b) considers the performance benefits of using an unordered transport service with network-conscious image compression techniques vs. using an ordered service with traditional image compression techniques. The central focus of Iren's work is the tradeoff between using:

- better compression (fewer bits per pixel) at the expense of having to use an ordered/reliable transport service which may experience increased delay when there is significant packet loss, vs.
- a more flexible transport service (unordered/reliable) which provides reduced delay and progressive display, at the expense of worse compression (more bits per pixel.)

Thus, each of Iren's experiments involves comparing

- the transmission over an ordered/reliable transport service (e.g., S2E) of a single image in a format requiring ordered/reliable delivery (e.g., GIF or SPIHT), with
- the transmission of the same image over unordered/reliable service (e.g.,X2E) in a network-conscious format (e.g., NCGif, or network-conscious SPIHT), that is, one that provides slightly worse compression, but allows ADUs to be processed and displayed out of sequence.

Thus, in the experiments comparing ordered and unordered service in Iren's work, an *unequal number of bytes* is transmitted over the two protocols, because the focus is on the progressive display of an *equal number of pixels*. Experiment **N1** differs from Iren's experiments with network-conscious image compression in that, since our focus is on studying the transport layer, per se, rather than the tradeoffs involved in various compression techniques, we desire equal numbers of both bytes and pixels in our experiments.

Unordered vs. ordered service (Diot and Gagnon, 1999)

(Diot and Gagnon, 1999) presents a more abstract comparison of unordered vs. ordered service in general. A serious limitation of this work is that it considers the benefits of unordered delivery only in terms of improvements in throughput, buffer utilization, and jitter, with the main emphasis decidedly on throughput. This viewpoint overlooks a key benefit of out-of-sequence delivery, namely the progressive display (or in the general case, the progressive processing) of information.

Nevertheless, (Diot and Gagnon, 1999) does contain several useful observations. A major theme of their work is that a crucial factor in determining the *throughput* benefit of out-of-sequence delivery is the relationship among the round-trip delay, the bitrate, and the application's ADU processing time. This relationship can be best understood by considering what happens when there is a packet loss with an ordered service. A packet loss results in a *gap* in the sequence number space of bytes or packets. Until this gap is filled, data delivery is suspended, and packets that follow the gap must be buffered. The impact of this gap on throughput depends on the relationship between application processing time, and round-trip delay. There are three cases:

Case 1*: If the application processing time is small compared to the round-trip delay, then there is little benefit to out-of-sequence delivery.* In this case, if ordered delivery is used, when a retransmission fills a gap, the resulting burst of packets will be processed very quickly. Thus, overall throughput does not suffer significantly.

On other hand, this argument ignores the benefit of progressive display. Even if the application processing time is arbitrarily small, out-of-sequence delivery is

useful if the elapsed time needed to fill a gap is sufficiently large to produce noticeable delays for the end user in access to information. For many applications, such as web browsing, it does not matter if the application can decode 60MB of information in a microsecond; the end user would rather be presented with 1MB of information each second for 60 seconds, than to receive nothing for 59.999999 seconds, and then receive all 60MB in the final microsecond. This is where the throughput-centric argument breaks down.

Case 2: *If the application processing time is large compared to the round-trip time, then there is little benefit to out-of-sequence delivery.* By the time the application is finished with a particular ADU, enough time has elapsed that any gap that may have existed can be filled by retransmission.

Unlike the previous argument, *this argument also pertains to progressive display*. No progressive display is obtained from delivering packets out-of-sequence if the transport layer can fill gaps more quickly than the application can process successive packets.

Case 3: *If the application processing time, the round-trip time, and the interpacket arrival time are of the same order of magnitude, out-of-sequence delivery can improve throughput.* In this case, when a gap occurs, the application may be blocked waiting for a retransmission. If the application processing time is only slightly smaller than the interpacket arrival time, when the retransmission fills the gap resulting in a burst delivery of buffered data, the application processing may be too slow to compensate for the time that was lost while waiting for the retransmission. Any lost time increases the delay of the final packet, thus reducing throughput. By

keeping the application pipeline flowing when there is a gap, this extra delay is avoided, and throughput is increased.

One useful application of Diot and Gagnon's principles is to point the way towards where we can expect to find benefits from partially-ordered service, and where those benefits are unlikely to be found. In particular, from Diot and Gagnon's work, we can derive the hypotheses that we are more likely to find gains from partially-ordered service:

- at low bit rates (because of the large round trip times),
- at high bit rates only where there is a significant propagation delay (again, resulting in a significant round-trip time), and
- when the application processing time is not large as compared to the round-trip-time.

On the other hand:

- we are unlikely to find any benefit from partial order on networks where the bit rate is high and the propagation delay is small, such as local area networks.

These hypotheses are developed more formally in the experiments throughout the remainder of this chapter.

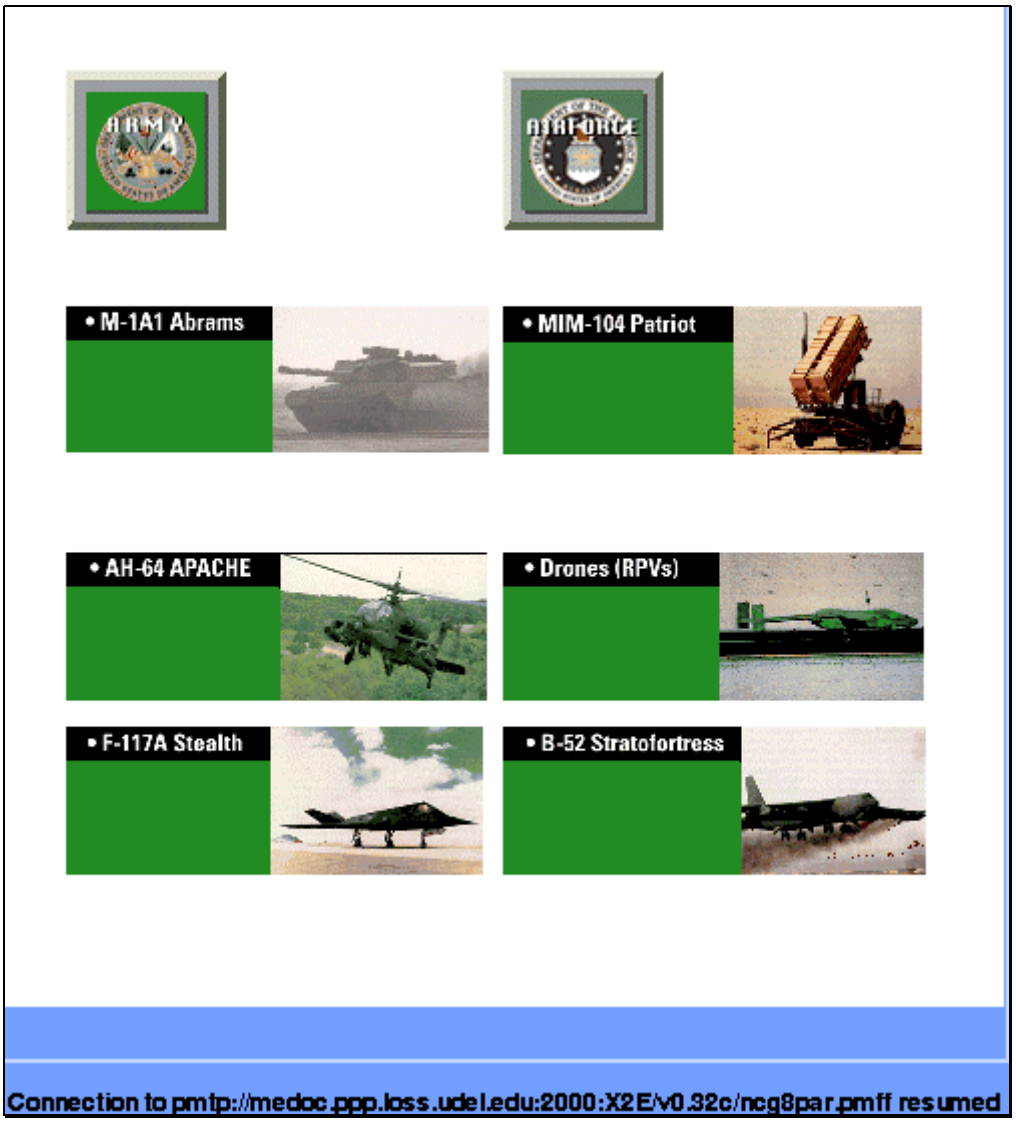


Figure 5.3 Illustration of `ncg8par.pmsl`

5.2.2 Experiment N1: parameters

Tables 5.1 and 5.2 shows the parameters and UTL mechanisms (i.e., the transport protocols) used for experiment **N1**.

Table 5.1 Parameters for Experiment N1

Parameter	***Values***
Mechanisms	**X2E,R2E,S2E,T2E**
Loss Rates	**0,10,20**
Network	PPP
Bitrate	38.4kbps
Propagation Delay	negligible: distance is less than 5 meters
Document	`img8par.pmsl`
Window Size	32

Bold indicates the parameters that are the focus of the experiment.

Table 5.2 UTL Mechanisms used in Experiment N1

Mechanism[52]	**Service**	**Layers**	**TPDU header size**
X2E	unordered/reliable[53]	KX2	20 bytes
R2E	partially-ordered/reliable	KX2,NUL,POL	52 bytes
S2E	ordered/reliable	KX2,TOL	24 bytes
T2E	ordered/reliable	KX2,TOL,POL	52 bytes

Note: Congestion control disabled for all four mechanisms (See Section 5.11.2)

5.2.3 Format of performance graphs

A consistent format will be used for performance graphs throughout this dissertation; in this section, we explain this format by referencing Figure 5.5, showing the first set of graphs from Experiment **N1**.

[52] See Table 3.1, 3.1a for more details about the entries in this table.

[53] X2E normally provides U/PR*k* service (see Table 3.1, 3.1a); for **R1** and **R2** the default value of *k*=0 is left unchanged by the application, resulting in U/R service.

The graphs in the left hand column show the average progressive delivery of bytes for mechanisms X2E, R2E, S2E and T2E, while the right hand column shows the average progressive display of pixels for mechanisms X2E, R2E, S2E and T2E.

The main independent variable in this experiment is the loss rate. The top row shows results for 0% loss, while the subsequent rows show results for 10% and 20% loss. In general, except where specifically noted, this layout of:

- multiple protocols on each graph,
- BYTES on the left, PIXELS on the right
- values of the independent variable in rows down the page

will be used throughout for the presentation of performance graphs. Except where there is a deviation from this layout, we will not repeat this explanation in subsequent sections.

Note that both the bytes and pixels metrics have significant implications. The pixels metric allows us to assess the impact on the human end user for this particular application. The bytes figure allows us to understand the advantage of out-of-sequence delivery from a transport layer viewpoint, and may allow us to predict the impact of out-of-sequence delivery on other applications.

5.2.4 Experiment N1: observations

We would like to highlight three observations concerning the graphs for Experiment **N1**:

(1) Each of the bytes graphs begins an upward curve from time zero, while the pixels graphs all have a delay of around 3 seconds before any delivery begins.

(2) There is a noticeable performance difference among the protocols at 0% loss, with X2E and S2E providing the best performance, R2E and T2E providing slightly worse performance.

(3) There are larger performance differences at 10% and 20% loss:

- T2E provides the worst performance,
- S2E provides slightly better performance than T2E,
- R2E provides considerably better performance than S2E, and
- X2E provides the best performance of all.

The next three sections interpret these observations in light of our hypotheses.

5.2.5 Exp. N1 analysis: delay in delivery of pixels vs. bytes

Our first observation is to note an interesting difference between the bytes graph and the pixels graph. The bytes graph shows that we begin receiving bytes nearly from time 0, while in the case of the pixels graph, there is a delay of about 3 seconds before we start recording any progressive display of pixels. This delay reflects the fact that the first two ADUs of each GIF image contain a GIF header and the color table for the image. For the 0% loss case, the GIF header and color table ADUs for each of the eight parallel images are guaranteed to be the first sixteen ADUs to arrive. Thus we will not see any progressive display of pixels until the seventeenth ADU arrives. In the case of the `img8par.pmsl` document, the first sixteen ADUs contain a total of 6,918 bytes of data. Therefore, we expect that the bytes graph would reach 6,918 at about the same time (actually one ADU time sooner) as the first rise of the pixels graph from zero—and indeed, as we compare the two graphs, this is the case.

Exp. N1 analysis: differences in performance at 0% loss

Next, we find evidence to support Hypotheses 5.2.1 and 5.2.3. For the network conditions tested, with 0% loss, the progressive display of the images is almost the same, regardless of the delivery order enforced by the transport service, but there is a measurable difference among the four services. The green and red lines, representing simple unordered (X2E) and ordered service (S2E), are nearly identical to one another, and show that the entire document is presented in approximately 22 seconds on average. The red and purple lines, representing partial order service (R2E), and total order service (T2E) with explicit release and identical overheads, are nearly identical to one another, with the red and purple lines reflecting an overhead that results in approximately 2 seconds of extra delay. The difference in performance is due to the processing requirements and TPDU headers that differentiate X2E and S2E from R2E and T2E, as shown in Table5.2.2:

- X2E has the smallest amount of processing overhead (1 layer) and the smallest TPDU header (20 bytes). Thus, when there is no loss, X2E provides the best performance.
- S2E has the next smallest amount of processing overhead (2 layers) and the next smallest TPDU header (24 bytes). Therefore, S2E performs next best, with nearly identical, but slightly worse performance than X2E for 0% loss.
- R2E and T2E have the largest processing and TPDU header overheads, and thus perform the worst at 0% loss. Because the overhead for R2E and T2E is nearly identical, so is their performance at 0% loss.

However, note that for this document, at this bitrate, the performance gap among all four protocols is small. We make the following observations:

(1) All four protocols start delivering bytes and/or pixels at virtually the same instant.

(2) All four provide smooth data delivery, with no noticeable interruptions in the flow.

(3) The maximum gap between the average performance of the X2E and R2E protocols occurs at 22.1 seconds after the document request. At this point, X2E is, on average, finished with the entire document, while R2E still has on average 56,833 pixels remaining (about 8.7% of the total pixels). R2E does not complete the document until, on average, time 24.1 seconds. We can make a similar observation about any comparison of either X2E or S2E with either R2E or T2E.

The left hand side of Figure 5.4 shows the completed document, representing the view the user would have at 22.1 seconds with X2E. The right hand side of this figure shows the document with 56,831 pixels, which is close to the average of 56,833 that would be on the screen at time at time 22.1 seconds for R2E. As the reader can see, there is a significant difference in appearance between the two screen dumps. However, both protocols deliver pixels at a steady rate, and two seconds after this picture, the entire document is finished for both protocols. The main difference is the slope of the line, which would probably not be noticed by most users. We can draw the following conclusions from these results:

(1) The TPDU and processing overhead of R2E and T2E as compared to X2E and S2E is significant enough to be measured, but probably not significant enough to cause a noticeable performance penalty for `ncg8par.pmsl` at 38.4kbps.

(2) We believe that we can safely extrapolate this result to documents that are smaller than or equal in size to `ncg8par.pmsl`, and bit rates equal to or larger than 38.4kbps; we should not draw any particular conclusions about larger documents or slower bitrates, since the “lines” that appear in the first row of Figure 5.5 may not be lines at all, but may be a flat-appearing region of a non-linear function.

(3) Most importantly, we can conclude therefore that the TPDU and processing difference between R2E and T2E is much smaller

than the difference between the R2E,T2E mechanisms and the X2E,S2E mechanisms. It is important to the significance of results later in the chapter that R2E and T2E have virtually identical processing overheads. This experiment provides the first indication that R2E and T2E do have virtually identical processing overheads, at least for this document at bitrates ≥ 38.4kbps.

5.2.7 Exp. N1 analysis: differences in performance at 10% and 20% loss

Next, we see evidence for Hypothesis 5.2.2 in the second and third rows of Figure 5.5. When the loss rate is increased to 10% and 20%, we make the following observations:

(1) In terms of progressive display of pixels, unordered service (the green line) is the clear winner over partially-ordered service (the blue line) or either of the ordered services (red and purple)

(2) In terms of both bytes and pixels, partially-ordered service (blue) is better than ordered service (red and purple)

A question that arises, however, is: what is the significance of these gains to the end user? In Section 5.2.9, we address this question by relating the graphs to the example screen dumps in Figures 5.6 through 5.8. These illustrations provide tangible proof that in this experiment, there are significant performance gains for unordered service vs. partially-ordered service, and partially-ordered service vs. ordered service. We will rely on illustrations such as these throughout the chapter. Nevertheless, as helpful as these illustrations are, some important caveats are involved in their interpretation.

5.2.8 Caveats regarding interpretation of example ReMDoR screen dumps

Throughout this chapter, we present screen dumps[54] from the ReMDoR browser for comparison purposes. For example, Figures 5.6 through 5.8 show the `ncg8par.pmsl` document at various stages in its progressive display over three different protocols. As the figures' captions show, each screen dump illustrates, for a given experiment with a given document:

- an image *close to* that with the average number of pixels on the screen, (momentarily, we explain what we mean by "close to")
- across all repetitions of that experiment,
- at a particular point in time,
- using a particular protocol,
- at a given loss rate.

In this section we explain what we mean by "close to."

As an example, consider the screen dump in the lower left-hand corner of Figure 5.6. The text under screen dump reads "avg. 1151 (1169 pixels shown)". This label "avg. 1151" indicates that when protocol S2E is used to retrieve the `ncg8par.pmsl` document at 10% loss, five seconds after the document is requested, there are, on average, 1151 pixels on the screen. However, the phrase "(1169 pixels shown)" indicates that the screen dump that appears in Figure 5.6 actually shows 1169 pixels, not 1151. The rest of this section explains why this is the case, and indeed, why this discrepancy is necessary if these illustrations are to be used to make a fair evaluation of the benefits of unordered and partially ordered transport service.

---

[54] The term *screen dump* refers to a graphic representation of the entire ReMDoR display at a given moment in the display of the document. This term avoids any ambiguity that might result if we called these screen dumps *images*; we reserve the term *image* for single image elements in a PMSL document.

The *p(t)* function: pixels on the screen as a function of time for ordered service

For any given experiment, let *p(t)* as a function represent the number of pixels on the screen at time *t*. Because all pixels from each ADU are placed on the screen instantaneously by a single operation, *p(t)* is a step function that can only take on a finite number of discrete values. For ordered protocols, given the arrival time of each ADU, arr_i we can characterize this step function precisely by the following formula:

For ordered protocols, given arr_i for all $i \in \{1,2,\ldots,n\}$

$$p(t) = \sum_{j=1}^{k(t)} \text{number of pixels in packet } i$$

where *k(t)*, is the index of the most recently arrived packet:

$k(t) = \text{maximum } i \in \{1,2,...,n\}$ such that $arr_i \le t$, or 0 if no such *i* exists.

On the other hand, in our experimental results, we report an *average* number of pixels present on the screen at some point in time. For a given number of experiments, say, 30, there are again a finite number of values that the average of 30 instances of this step function (as determined by the particular set of arr_i values) can take on. However, it is likely that few, if any of these average values will correspond *exactly* to any of the possible pixel values in the range of the function *p(t)*:

$$\text{range set of } p(t) = \{0\} \cup \bigcup_{i=1}^{n} \left(\sum_{j=1}^{i} \text{number of pixels in packet } i \right)$$

Therefore, the images we will show as representative images are ones with pixels that come from the range set and are *close to* the average number of pixels reported.

Returning to our example, for protocol S2E, Figure 5.6 shows that, on average, after 5 seconds, 1151 pixels are displayed. However, in the actual runs it never occurs that exactly 1151 pixels are displayed. Therefore, to represent the value

1151 with a screen dump, we use a screen dump with 1169 pixels, which is a value close to 1151 representing an actual screen dump that can occur in practice.

For unordered or partially-ordered service, the *p(t)* function is problematic

In the case of totally ordered service, since the delivery order is fixed, there is a direct mapping from a given number of pixels to a unique screen dump, with a specific number of pixels allocated to each of the eight parallel images. However, for unordered, or partially ordered service, there is a finite, but astronomically huge set of possible delivery orders for the given ADUs. In particular, the ncg8par.pmff file contains 119 ADUs, therefore there are ($119! \approx 5.57 \times 10^{196}$) possible delivery orders for the unordered case. Therefore, as a practical matter, we cannot compute the most likely delivery order for any given number of pixels; and we consequently cannot determine the most likely image to be presented on the screen.

Therefore, to produce sample screen dumps with a given number of pixels for the unordered or partially ordered protocols, instead of trying to come up with a most-likely screen dump with a certain number of pixels, we took the following steps:

- we ran the browser with the same document and mechanism as the original experiment.
- we enabled a special option on the browser that causes it to freeze when a certain number of pixels is reached.
- we chose a value as close as possible to the pixel value reported in the table.

This produces *a* real screen dump with close to the same number of pixels as the average reported in the experimental results. However, unlike the totally ordered case, where *the* exact screen dump shown did actually occur in *every* experiment at some point in time (on average, close to the time interval with which it is associated in the

picture), the screen dumps we show for the unordered and partially ordered cases may or may not have occurred in any particular experiment, since the number of possible screen dumps is astronomical.

Avoiding bias in the choice of screen dumps

This method of selecting images close to a particular average number of pixels for the unordered or partially-ordered case risks introducing bias. For each protocol and time instant, it is usually the case that an image must be chosen that is either higher or lower than the actual average number of pixels. The choice of the higher or the lower value could tend to exaggerate or minimize the benefit of partially-ordered transport service. Therefore, to guard against bias we have chosen always to err on the side that would reduce, rather than exaggerate, the benefits of partial order. In essence, then, the benefit seen in the images, is a *lower bound* on the actual benefit represented by the averages reported for the experiments.

For example, in comparing R2E and S2E at 10% loss, our claim is that R2E provides better progressive display than S2E. Whenever we had to choose images to represent S2E, we always chose the image with the next *higher* number of pixels, rather than the image with the closest number of pixels. Similarly, for R2E, we always chose the image with the next *lower* number of pixels rather than the one with the closest number of pixels. The labels under the screen dumps in Figures 5.6 through 5.8 show this avoidance of bias.

In summary:

- The actual number of pixels shown in a screen dump for ordered service (S2E) is always more than the corresponding average number of pixels for the given loss rate and time.

- Conversely, the actual number of pixels shown in a screen dump for unordered or partially-ordered service (X2E, and R2E, respectively) is always less than the corresponding average.

5.2.9 Exp. N1 analysis: interpretation of sample screen dumps

Figures 5.6 through 5.8 shows representative images of the R2E and S2E graphs at 10% loss. Starting with Figure 5.6, we see that after 5 seconds, the partially-ordered protocol, R2E has started placing portions of four images on the screen, while the ordered protocol, S2E has only begun to place one image on the screen. By 10 seconds, R2E is working on all eight images, while S2E still has only two. By 15 seconds, R2E has slightly obscured images of all six weapons systems, and has started to unveil the Army and Air Force seals, while S2E still has only obscured images of all the equipment and has not yet begun to unveil the seals. At 20 seconds, R2E has *entirely finished* two of the images, while S2E has not yet completely finished *any* of the images. By 25 seconds, R2E has finished five images, to S2E's three images complete. Finally, after 30 seconds S2E essentially catches up with R2E, and the protocols provided essentially equivalent performance for the final 1-3% of the pixels that remain.

5.2.10 Exp. N1: conclusions and summary

The results of Experiment **N1** support Hypotheses 5.2.1, 5.2.2 and 5.2.3. The results indicate that for a document with parallel images being sent over a PPP link at 38.4kbps, when there is significant[55] packet loss (10% or 20%), unordered service provides noticeable improvements in progressive display over partially ordered

[55] See Section 1.4.1 for a motivation of various rates of packet loss.

service, which in turn provides noticeable improvements in progressive display over ordered service.

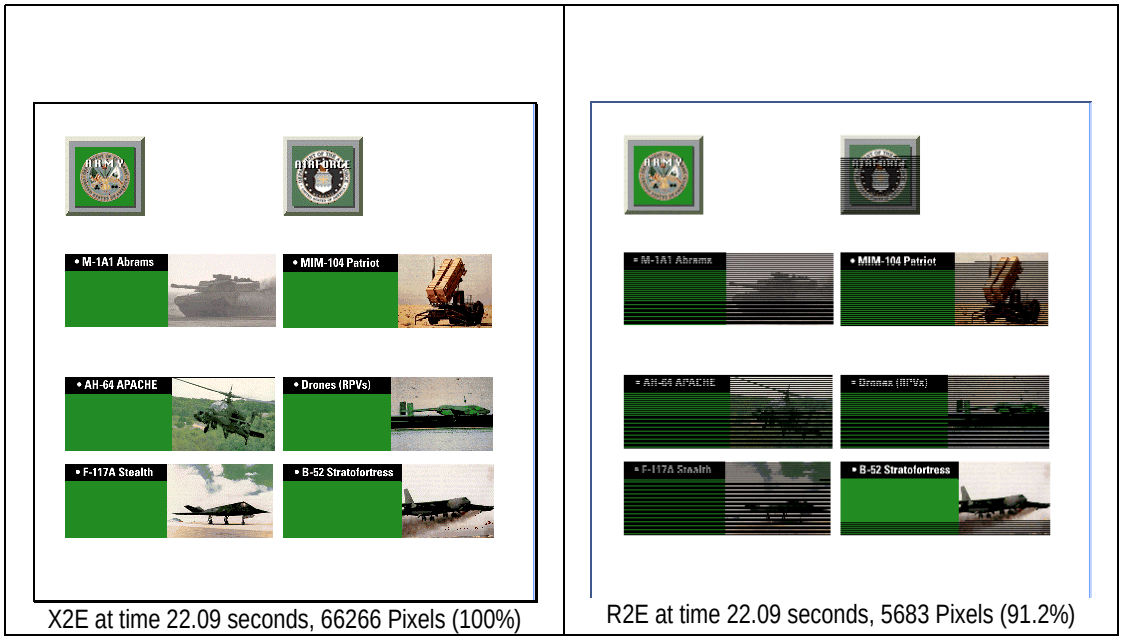


Figure 5.4 Illustration of maximum performance gap for Exp. N1 at 0% loss

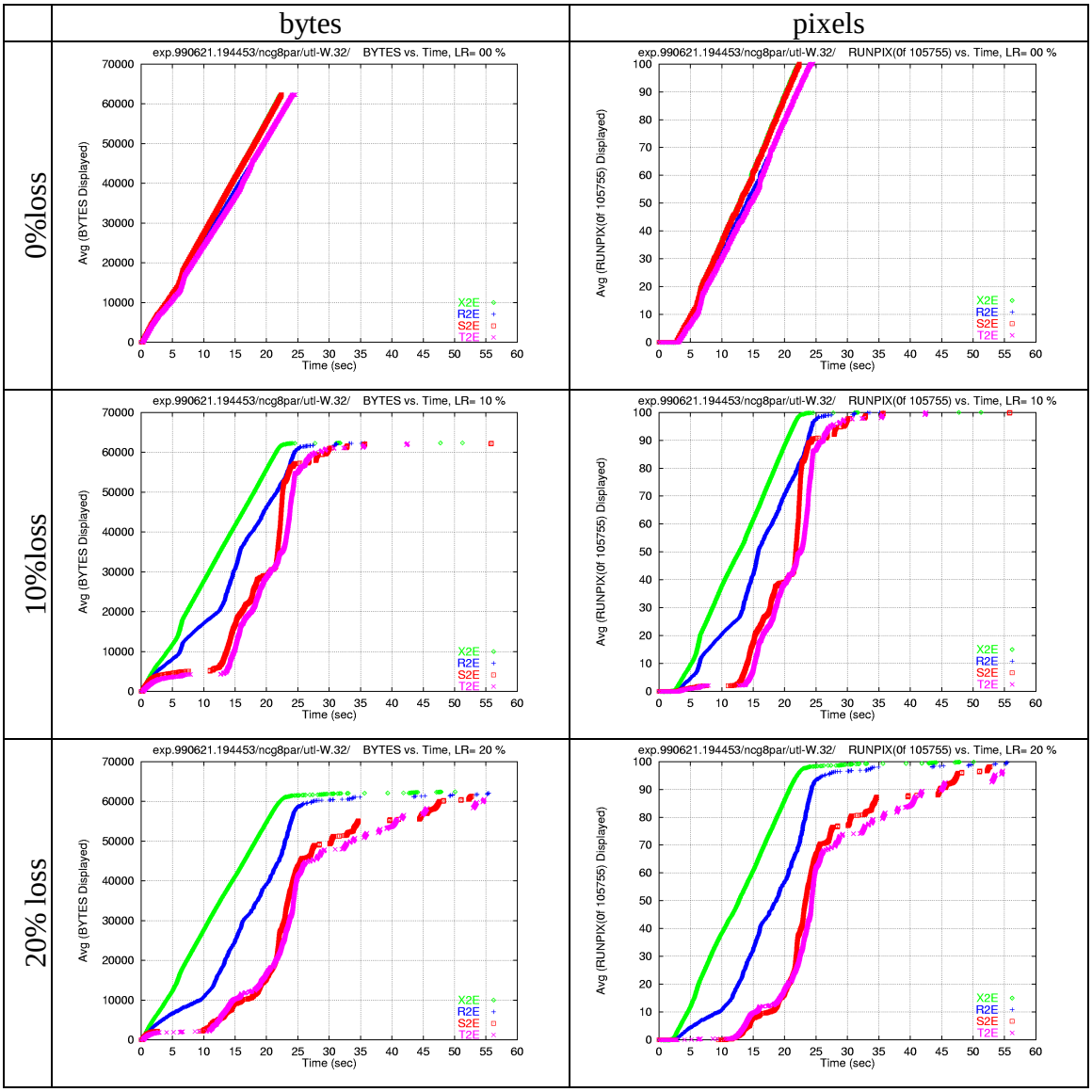


Figure 5.5 Experiment N1: Performance graphs

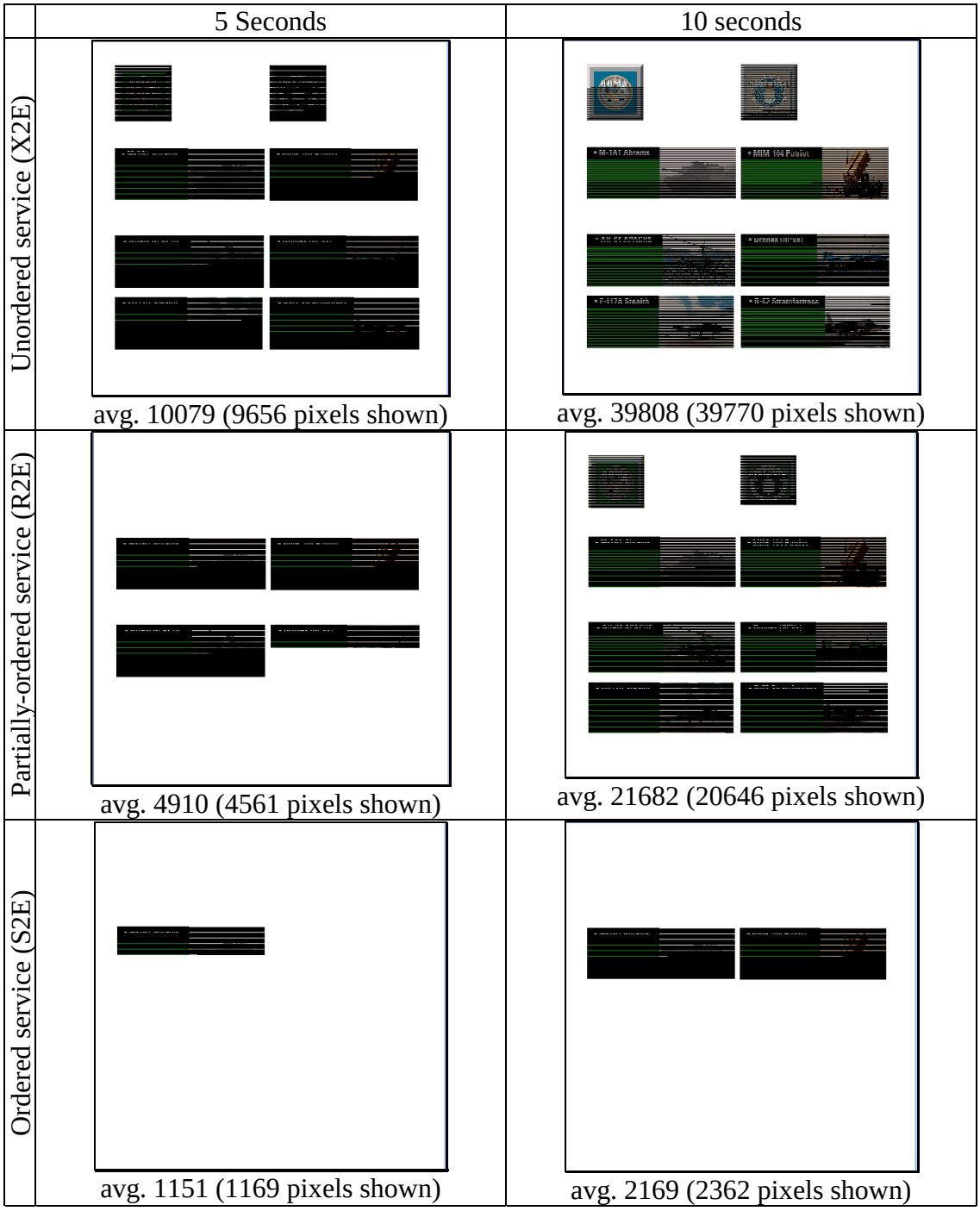


Figure 5.6 Screen dumps: `ncg8par.pmsl`, 10% loss, 38.4kbps PPP link

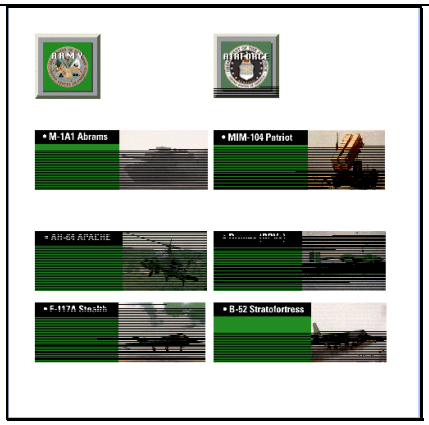
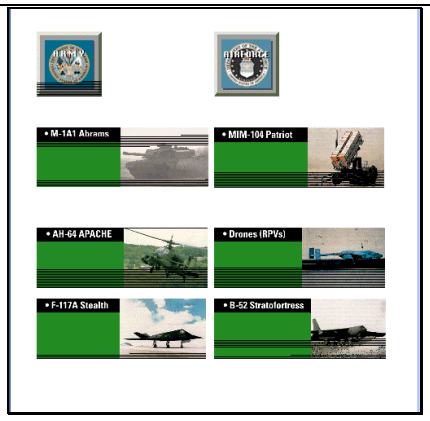
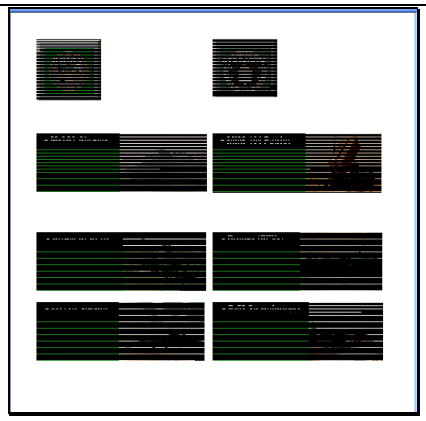
	15 seconds	20 seconds
Unordered service (X2E)	 avg. 64978 (64068 pixels shown)	 avg. 93181 (92898 pixels shown)
Partially-ordered service (R2E)	 avg. 44325 (43840 pixels shown)	 avg.: 74765 (73396 pixels shown)
Ordered service (S2E)	 avg. 18452 (18670 pixels shown)	 avg. 41367 (41594 pixels shown)

Figure 5.7 Screen dumps: `ncg8par.pmsl`, 10% loss, 38.4kbps PPP link

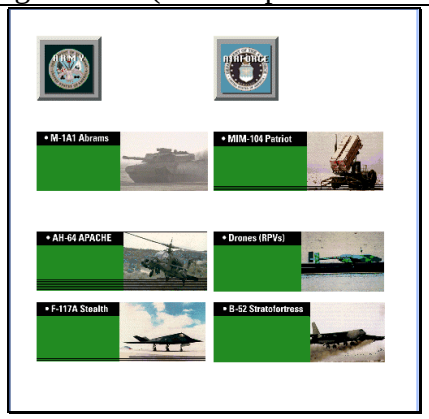
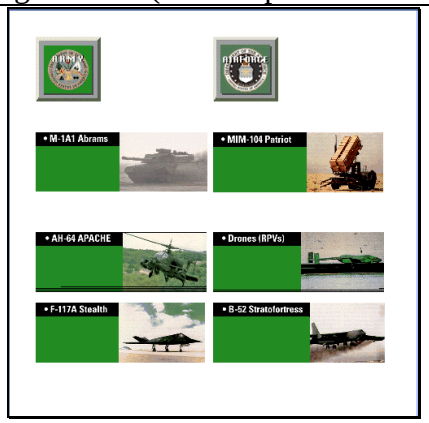
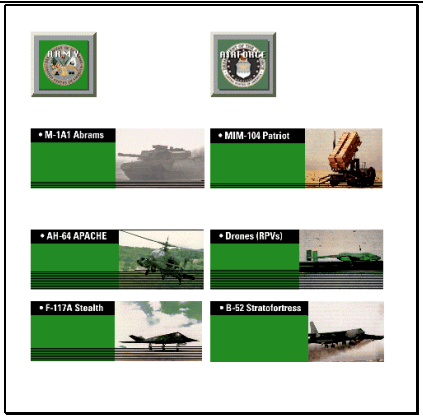
	25 seconds	30 seconds
Unordered service (X2E)	 avg. 105567 (105329 pixels shown)	 avg. 105610 (105329 pixels shown)
Partially-ordered service (R2E)	 avg.: 103315 (102400 pixels shown)	 avg.: 104853 (104529 pixels shown)
Ordered service (S2E)	 avg.: 95925 (96686 pixels shown)	 avg.: 102929 (103587 pixels shown)

Figure 5.8 Screen dumps: `ncg8par.pmsl`, 10% loss, 38.4kbps PPP link

5.3 Experiment R1: O/R vs. PO/R for eight parallel GIF images at 9.6kbps.

This section describes Experiment **R1,** the first of five sets of experiments evaluating partially-ordered/reliable transport service for remote multimedia document retrieval.

Experiments **R1, R2,** and **R3** all involve retrieval of the same document; the difference among the three is the bitrates that are used. The document used in **R1** through **R3** is called `img8par.pmsl`. This document contains eight images (taken from `military.pmsl`) presented in parallel. To the end user, the *final* appearance of this document is identical to that of the ncg8par document used in Experiment **N1 (**illustrated in Figure 5.3). However, the images are coded differently, and hence result in significantly different progressive display. In the `ncg8par.pmsl` document, we used the NCGIF file format (Iren, 1999b). For the `img8par.pmsl` document, the same images are compressed using the traditional GIF file format. The GIF format requires ordered/reliable delivery for each image, because the entire block of pixels is compressed using Lempel/Ziv compression. Therefore, since the interpretation of every bit in every ADU depends on all the bits that precede it in the entire encoding, we cannot use unordered/reliable service for GIF images. However, because the data for the eight separate images can be processed in eight parallel streams (one stream per image), we can use partially-ordered service with the stream abstraction (Section 2.3). This idea is captured in the hypothesis for this experiment:

> For all loss rates > 0%, , partially-ordered/reliable (PO/R) service provides, on average, better progressive display for parallel GIF images than ordered/reliable (O/R) service.

For practical reasons, it is necessary to refine this hypothesis somewhat. It would be absurd to evaluate the gain at *very* low loss rates (<1%). For very low loss rates, the gain will be so small as to be insignificant for all practical purposes. It would be equally absurd to evaluate PO/R service vs O/R service at loss rates approaching 100%. At these loss rates, the performance of both services would be unacceptable. So instead of trying to characterize the gains of partial order over the entire range of loss rates between 0% and 100%, we focus on a few loss rates, and on the trend in performance as the loss rate increases:

> Hypothesis 5.3.1: When there is no packet loss, partially-ordered/reliable service and ordered/reliable service should perform identically, apart from any difference in overhead. Specifically, since R2E and T2E have virtually identical overheads, their performance should be nearly identical.
>
> Hypothesis 5.3.2: For packet loss rates of 10%, 20% and 30%, partially-ordered/reliable (PO/R) service provides better progressive display for parallel GIF images than ordered/reliable (O/R) service.
>
> Hypothesis 5.3.3: As the packet loss rate increases from 0% to 10%, 20% and 30%, the gain from partially-ordered service for parallel GIF images will increase.

Organization of Section 5.3

- Section 5.3.1 provides background concerning this experiment, including an explanation of the drawbacks of using parallel TCP connections for this application (as is done in many current Web browsers.)
- Section 5.3.2 describes the parameters for Experiment **R1**.
- Section 5.3.3 explains the choice of R2E and T2E as the best available UTL mechanisms for comparing partially-ordered/reliable and ordered/reliable service for multimedia documents.

- Sections 5.3.4 through 5.3.8 describe our results and conclusions for Experiment **R1.**
- Section 5.3.9 provides an overall summary for Section 5.3.

5.3.1 Related Work: other ways of providing parallel flows

When the author and his colleagues have made presentations on partially-ordered transport service, a frequently asked question is why a partially-ordered service is needed at all, when opening multiple TCP connections in parallel would provide all or most of the putative benefits of partially-ordered service. In this section, we briefly discuss the idea of parallel TCP connections, and discuss another protocol that provides parallel streams

The parallel TCP connections approach has serious drawbacks

Given that the document used in Experiments **R1** through **R3** is non-temporal and contains parallel images, it is essentially equivalent to many web pages having a similar structure. The popular *Netscape* web browser opens multiple TCP connections to retrieve these images, in essence, implementing a crude form of partially-ordered transport service over TCP. However, this approach has serious disadvantages. It increases the likelihood of congestion, and requires extra round-trips and control packet overhead for connection establishment (Mogul, 1995).

The Multi-Stream Protocol

(LaPorta and Schwartz, 1991) describes the performance analysis of a MultiStream Protocol (MSP). This protocol provides for the transfer of up to seven parallel streams of packets. However, in MSP, each of these streams has a particular set of characteristics; for example, STREAM1 provides ordered/reliable/no-duplicates

service, STREAM2 provides ordered/unreliable/no-duplicates service, STREAM3 provides unordered/partially-reliable/no-duplicates service, and so forth. Thus while parallel streams are provided, the number of streams and type of service provided by each stream is fixed. This is fundamentally different from the parallel streams that can be provided by PO/R service, where an arbitrary number of parallel, ordered-reliable streams can be specified.

5.3.2 Experiment R1: parameters

Table 5.3 shows the parameters used for Experiment **R1**. Table 5.4 shows the UTL mechanisms used in **R1** (these mechanisms are also used in Experiments **R2** through **R4**.)

Table 5.3 Parameters for Experiment R1

	Values for Experiment			
Parameter	***R1.1***	***R1.2***	***R1.3***	***R1.4***
Mechanisms	**R2E,T2E**			
Loss Rates	**0, 10, 20**	**0, 10, 20**	10	10
Network	PPP	reflector	PPP	reflector
Bitrate	9.6kbps	7.68kbps (=0.8×9.6kbps)	9.6kbps	7.68kbps (=0.8×9.6kbps)
Propagation Delay	negligible: distance is less than 5 meters			
Document	`img8par.pmsl`			
Window Size	16	16	**8, 16, 32**	**8, 16, 32**

Bold indicates the parameters that are the focus of the experiment.

Table 5.4 UTL Mechanisms used in Experiments R1 (also Exps. R2–R4)

Mechanism[56]	Service	Layers	TPDU header size
R2E	partially-ordered/reliable	KX2,NUL,POL	52 bytes
T2E	ordered/reliable	KX2,TOL,POL	52 bytes

Note: Congestion control disabled for both mechanisms (See Section 5.3.3)

5.3.3 Why the R2E and T2E mechanisms are used for Experiments R1–R5

Experiments **R1** through **R5** compare exactly two transport services:

- UTL mechanism R2E, providing partially-ordered/reliable service, and
- UTL mechanism T2E, providing totally-ordered/reliable service.

There are several reasons these two particular services were chosen from among the dozens available in UTL:

(1) These mechanisms provide the services we want to compare. To evaluate PO/R service, the crucial comparison is against O/R service, since in the absence of the provision of partial order, applications requiring partial order must use O/R service (e.g.,TCP).

(2) This comparison is fair in terms of overhead. R2E is composed of (KX2, NUL, POL); T2E is composed of (KX2, TOL, POL). Thus the top and bottom layers are identical. The middle layer, in both cases, adds a four byte sequence number; thus the total header lengths are identical. The only difference between the two middle layers is that one of them reorders out-of-sequence packets while the other does not. This is more fair than, for example, a comparison of P2E (KX2, POL) vs. T2E (KX2, TOL, POL).

[56] See Table 3.1, 3.1a for more details about the entries in this table.

(3) These mechanisms support explicit release synchronization. Placing a partial order layer (POL) on top of a total order layer may seem superfluous until one recognizes the need to provide explicit release synchronization for both the PO/R and O/R cases. While explicit release synchronization is not strictly necessary for the simple documents in Experiments **R1—R4**, it is essential for the `paris.pmsl` document used in Experiment **R5**. Furthermore, Experiments **R1**–**R4** are intended to represent excerpts from larger documents. Therefore, it makes sense to perform all the ReMDoR related experiments using transport services supporting explicit release. We argue that this does not bias our results in favor of either total order or partial order because this processing is necessary regardless of whether it is performed by the application layer or the transport layer. The need for explicit release motivates having POL as the top layer in both cases.

(4) These mechanisms use KX2, with the slow-start and cwnd congestion avoidance features disabled, which is the best choice among current options available. From a standpoint of best practice, the ideal experiment would compare PO/R vs. O/R service using a protocol offering sender and receiver application-transport flow control, and TCP-friendly congestion control—that is, it would be based on KX3. However, at the time these experiments were conducted, KX3 had not been tested and validated, while KX2 had been. The next best choice was KX2 with congestion control totally disabled.

5.3.4 Experiment R1.1: observations and conclusions

Experiment **R1.1** illustrates the performance of the PO/R service provided by the R2E protocol, with the O/R service provided by the T2E protocol. Figure 5.3.1

shows the average performance graphs for Exp. **R1.1.** We make the following observations concerning these graphs:

(1) As in Exp. **N1**, at 0% loss, R2E and T2E have virtually identical performance.

(2) Also, as in Exp. **N1**, there is an initial startup delay before the first pixels can be presented.

(3) As the loss rate increases to 10% and 20%, the performance degrades for both R2E and T2E, as indicated by the fact that the progressive display curves move to the right. This shift indicates that the bytes (or pixels) are being presented to the application (or to the user) at later points in time.

(4) While both R2E and T2E experience worse performance as the loss rate increases, the performance of R2E degrades more slowly than that of T2E.

(5) At nearly every point in time, on average, R2E provides more pixels to the end-user.

From these observations, we conclude that this set of experimental data supports Hypotheses 5.2.1, and Hypotheses 5.2.2, 5.2.3 as regards 10% and 20% loss over a PPP link at 9.6kbps.

To provide an end-user perspective, Figures 5.9 and 5.10 show the difference between R2E and T2E performance at a few sample points, for 10% loss and 20% loss, respectively. As can clearly be seen, at each of these points, partially-ordered service provides better performance than totally-ordered service. One of the most dramatic differences is at 25 seconds for the 20% loss case, where totally-ordered service provides no pixels at all to the user, while partially-ordered service provides almost as many pixels, on average, at 20% loss as it did at 10% loss. While human factors studies (which we suggest as future work) would be necessary to establish this scientifically, we hypothesize that the initial delivery of at least a few pixels will prove

to be highly correlated with user satisfaction. Seeing at least *some* progress provides hope to the user, while seeing a screen that does not change for a long period of time (especially a blank one) can be discouraging.

5.3.5 Experiment R1.2: observations and conclusions

Experiment **R1.2** repeats the same parameters as **R1.1**, except that instead of using the PPP link, the UDP reflector is used. The parameter settings for the reflector are intended to reproduce as accurately as possible the conditions on the PPP link. The propagation delay on the reflector is set to zero, since the propagation delay of a PPP cable of less than 2 meters is negligible. The bitrate is set to 7.68kbps which is 80% of 9.6kbps, reflecting the fact that an RS-232 connection uses 1 stop and 1 start bit to send each character; thus only 80% of the bitrate is effectively available to the data link layer.

With this in mind, we would expect the results from the reflector to match those of PPP. The data are similar, albeit not as close as we would like. Figures 5.12 and 5.13 show an overlay of the performance results from Experiment **R1.2** on top of the matching results for Experiment **R1.1** We make the following observations:

(1) All five observations from Section 5.3.4 concerning **R1.1** apply equally to the performance graphs for **R1.2**.

(2) At 0% loss, both experiments product a straight line as the average, however the slope of the line for **R1.2** is steeper, indicating that the effective bandwidth of the reflector is higher than the effective bandwidth of the PPP link, in spite of the fact that the reflector was set to a lower bandwidth.

(3) However, as the loss rate increases to 10% and then 20% loss, the results for **R1.2** appear to move closer to those observed in **R1.1.**

Observation (2) was a disappointment, as we had hoped to use the PPP link to validate the operation of the reflector. Further investigation revealed that PPP uses byte stuffing for the flag byte (binary value 0111 1110) that marks the beginning and end of each frame. We therefore tried to account for the difference by dumping the raw data and IP and UDP packet headers, and counting the number of occurrences of the flag byte, however this did not account for the difference. We then calculated, using a spreadsheet, the time that each packet should be delivered, based on an idealistic assumption of zero processing time. (In reality, both the PPP link and the packet reflector actually do have to perform some processing.) We found that the results calculated by the spreadsheet matched the reflector results exactly, while the PPP link always provided less than the bandwidth at which it was configured. We tried to find a model for the PPP overhead through running experiments at different bitrates with different packet sizes, and determining by solving a set of linear equations, values that we could add to the processing of each packet, and each byte that would allow us to accurately model the behavior of the PPP link. In the end, we abandoned this goal, since specifically modeling a PPP link was not central to our work, and the spreadsheet results provided sufficient evidence of the correctness of the reflector calculations.

Having noted the inaccuracy at 0% loss, surprisingly, at higher loss rates the results from the reflector and the PPP link are close to one another. Moreover, the general trends do not change from one graph to another. Therefore, while our failure to closely model the performance of the PPP link at 0% loss was disappointing, nevertheless the fact that the results match qualitatively increases our confidence in conclusions drawn from experiments with the packet reflector.

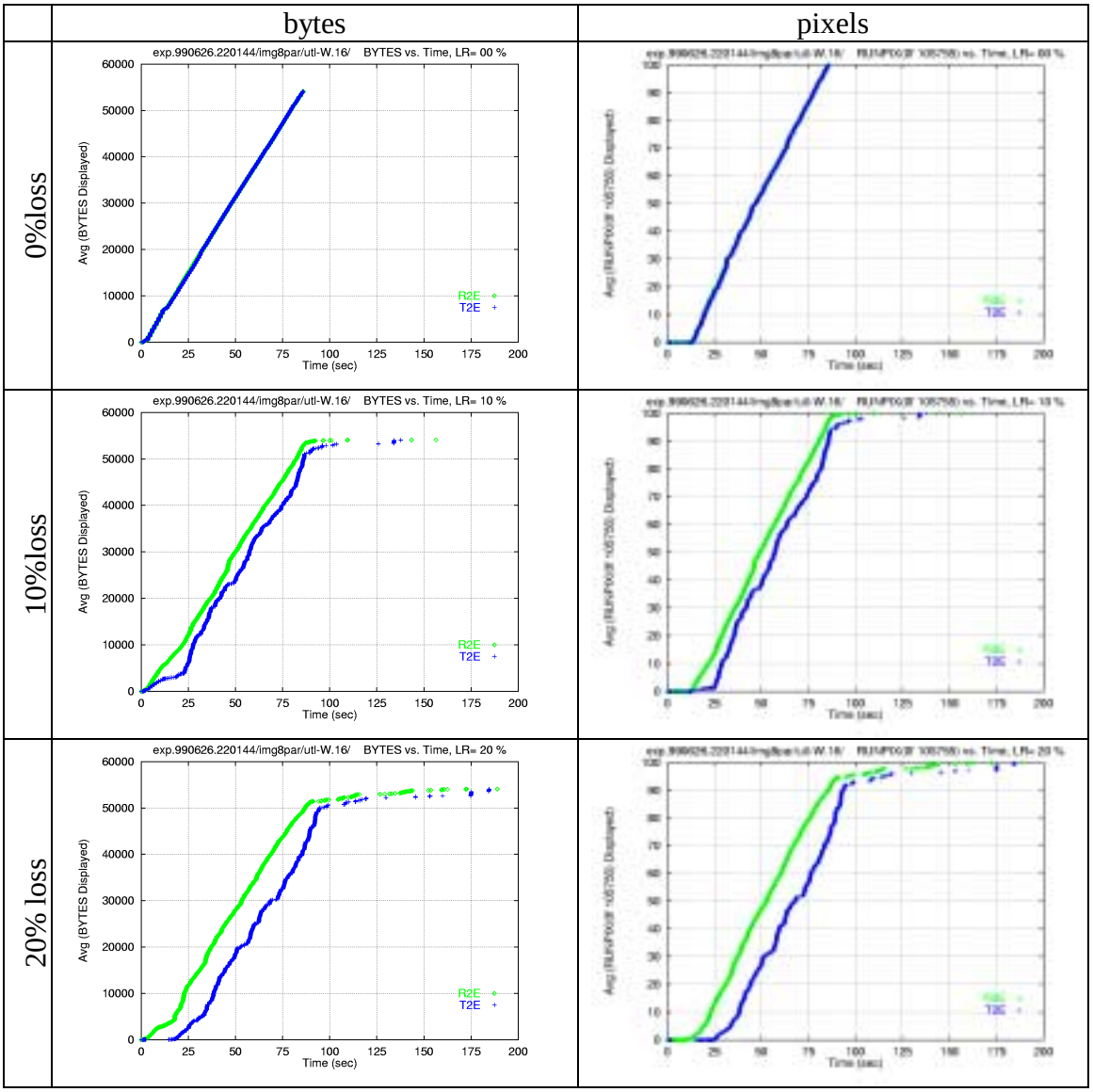


Figure 5.9 Experiment R1.1: Performance graphs

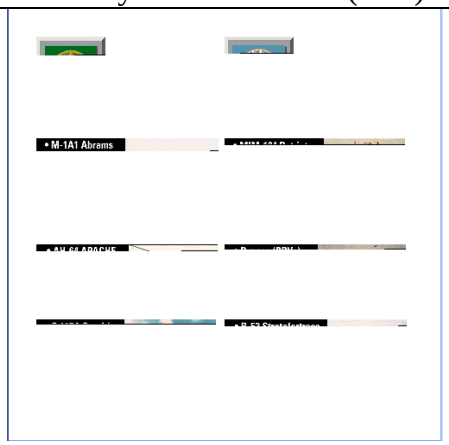
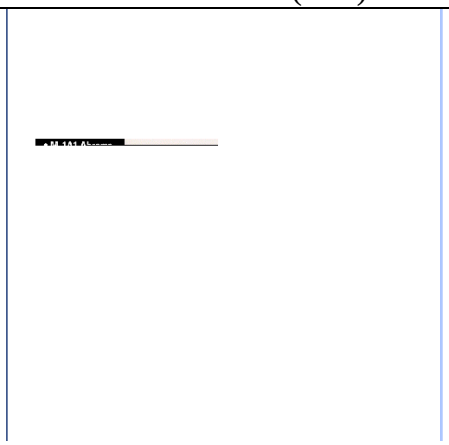
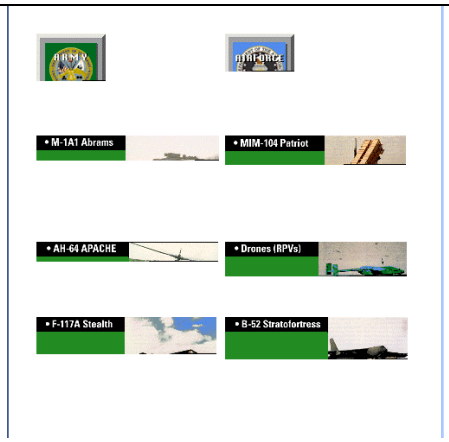
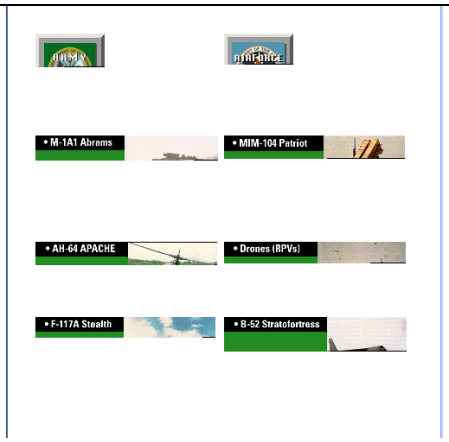
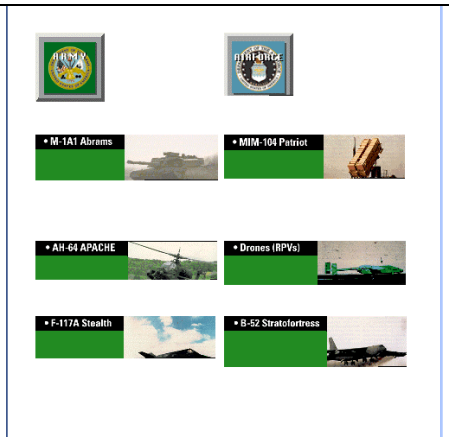
	Partially-ordered service (R2E)	Ordered service (T2E)
25 seconds		
	avg. 14447 (13508 pixels shown)	avg. 1530 (1535 pixels shown)
50 seconds		
	avg. 53767 (53294 pixels shown)	avg. 40437 (41745 pixels shown)
75 seconds		
	avg. 87345 (86446 pixels shown)	avg. 75665 (76506 pixels shown)

Figure 5.10 Screen dumps: `R1.1`, 9.6kbps PPP link at 10% loss

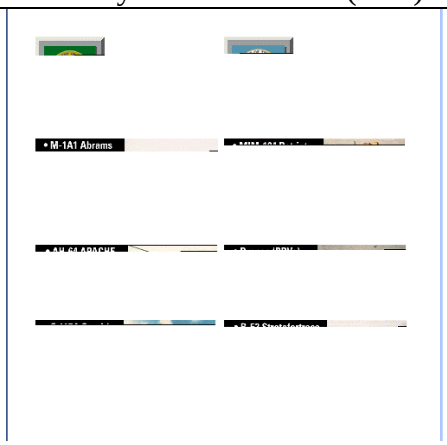
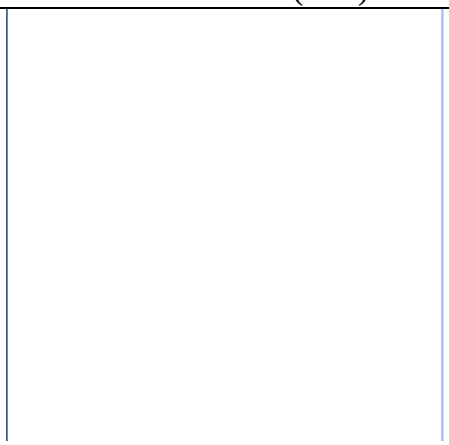
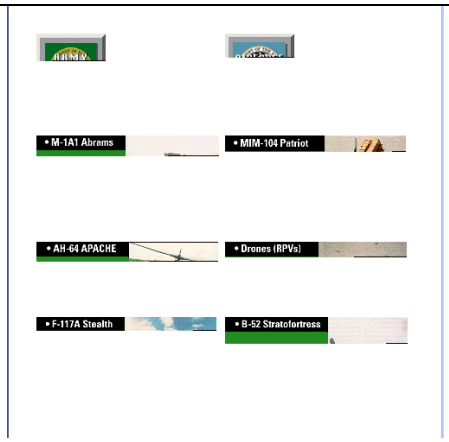
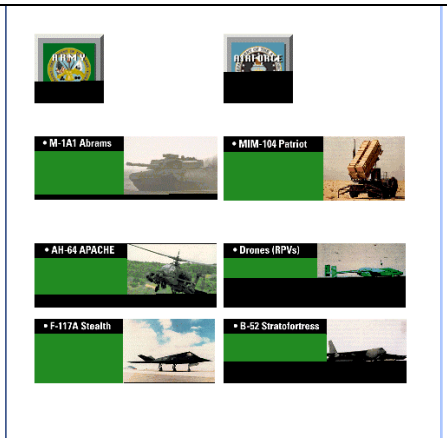
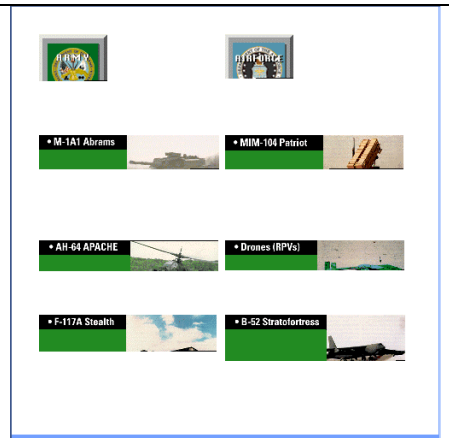
	Partially-ordered service (R2E)	Ordered service (T2E)
25 seconds at 20% loss	 avg. 13679 (13508 pixels shown)	 avg. 0 (0 pixels shown)
50 seconds at 20% loss	 avg. 49862 (48584 pixels shown)	 avg. 29066 (31929 pixels shown)
75 seconds at 20%loss	 avg. 83811 (83671 pixels shown)	 avg. 59615 (60359 pixels shown)

Figure 5.11 Screen dumps: `R1.1`, 9.6kbps PPP link at 20%loss

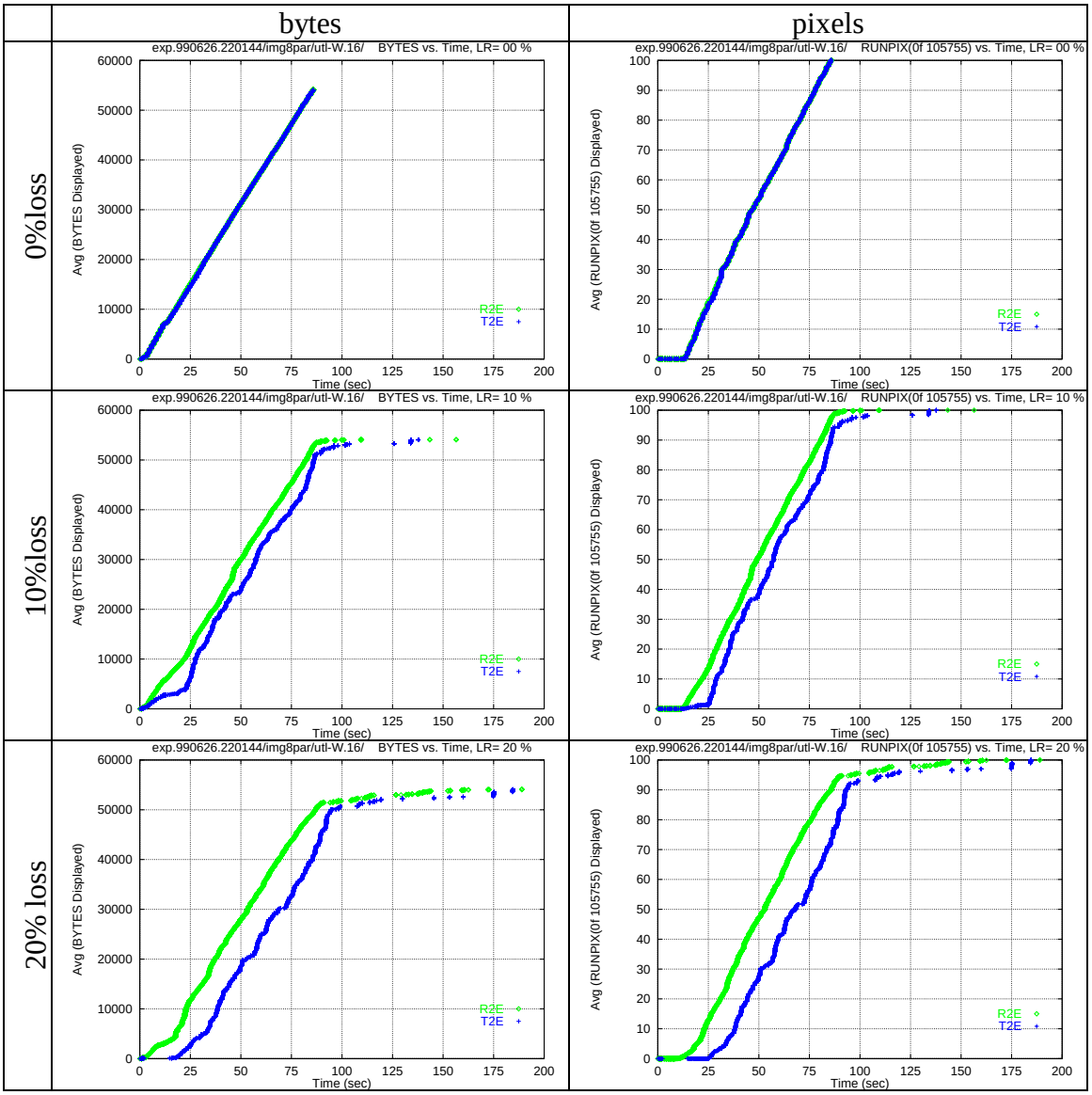


Figure 5.12 Experiment R1.1 performance graphs (overlay for Fig. 5.13)

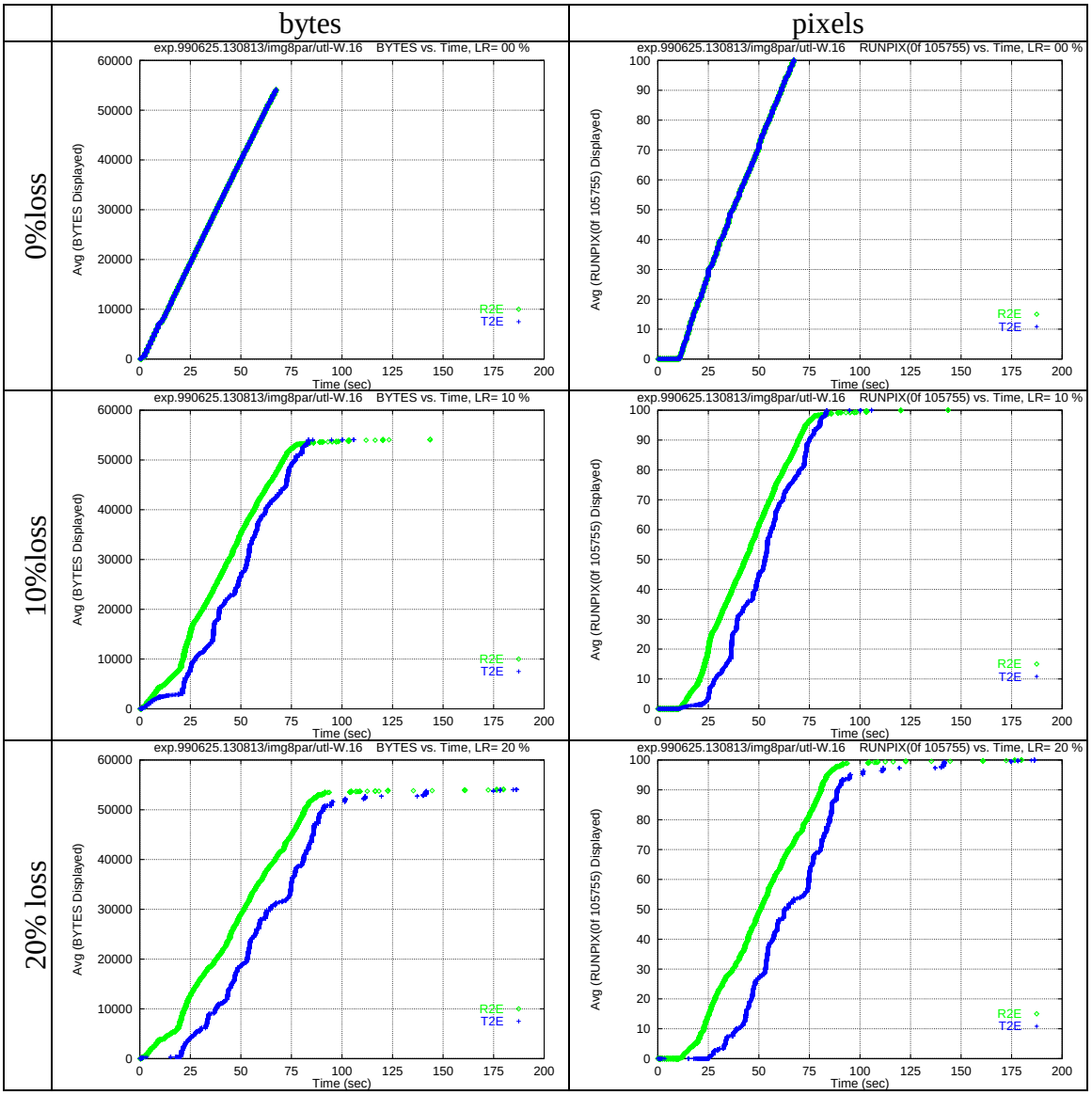


Figure 5.13 Experiment R1.2 performance

5.3.6 Experiment R1.3: observations and remarks

Experiment **R1.3** investigates the effect of window size on performance at 10% loss. Figure 5.14 shows the average performance graphs for Exp. **R1.3.** We first make several observations concerning these graphs:

(1) As the window size increases, performance gets worse for both protocols. However, the performance degradation is worse for T2E than it is for R2E.

This observation seems counter-intuitive at first. We would normally expect that providing more of a resource to a process would improve performance rather than degrading it. Classic analysis of Automatic Repeat Request (ARQ) type protocols (e.g., Tanenbaum 1986; Stallings 1997) indicates that larger windows (as in go-back-N and selective repeat) provide better performance than smaller windows (as in stop-and-wait, in the limiting case), up to a point of diminishing return. This point occurs when the window size equals the bandwidth-delay product. Thus, while we would expect that beyond a certain point, larger windows would not help performance, it seems surprising at first that they would hurt performance (in Section 5.3.7 we explain why they do).

We might also expect that the choice of window size would not be an important factor in evaluating the benefit of partially-ordered service, provided that the same window size is chosen for both transport services. The results here show that at least for the document and network conditions that apply in this experiment, the apparent benefit of partial order is highly dependent on the window size. When other parameters are fixed, partial order provides more benefit at window size 32 than at 16, and more benefit at 16 than at 8. (Again, in Section 5.3.7, we explain why.)

(2) At window size 32, for T2E, there is a gap of 12.6 seconds between time 29.8 and time 42.4 (relative to the initial document request) where no packets were delivered in *any* run of T2E [57] at loss rate 10%, window size 32. Under the same circumstances, R2E does not seem to have any such gap in packet delivery.

This surprising observation motivates a closer look at the individual runs that produced this average graph. Figure 5.15 shows graphs of bytes versus time for the 31 runs that are averaged to make up the single line for T2E in Figure 5.14, with 10 runs in each of the four smaller graphs. These individual runs lead us to the following observation:

(3) At window size 32 for T2E, the individual runs have gaps of around 20 seconds in each case; the gap of 12.6 seconds in the average graph represents the intersection of these larger gaps in the individual runs.

We also present an individual runs graphs for the 8 and 16 window cases (Figures 5.16 and 5.17) We see in these graphs that while there is no apparent gap in the average graph, in fact:

(4) Gaps in delivery for T2E are present at every window size, but are smaller and more spread out in time at window sizes 8 and 16. Therefore, there is no visible gap in the average graph at the smaller window sizes.

These results for T2E raise the question as to whether these gaps are present in the case of R2E. Figure 5.18 through 5.20 show the performance of individual runs for R2E. We see that gaps do occur in runs with R2E, but more rarely than with T2E. The next section presents our interpretation of these observations.

---

[57] Keep in mind that these graphs plot the average number of bytes or pixels delivered across all runs, at every point at which there is a packet delivery for *any* run. (That is, any run, with that protocol, at that loss rate and window size.)

5.3.7 Experiment R1.3: conclusions

Why does performance decrease as window size increases, and why does this affect T2E without seeming to affect R2E? Because this is a phenomenon we will see throughout the experiments in this chapter, we provide a thorough discussion of it here; in later experiments, we will only reference this discussion. Following this explanation, we provide a higher level interpretation of experiment **R1.3**.

Why larger windows hurt performance more for T2E, and less for R2E

Our explanation has two parts. First, we must explain why the gaps occur more for ordered service vs. unordered service. Second, we explain why the gaps are longer as the window size increases.

For ordered service, when any packet loss occurs, data delivery (at the receiver) of newly arriving packets is suspended until a retransmission of the missing packet is received. Therefore, for loss rates > 0, we would expect to see many gaps in the data delivery for ordered service. On the other hand, for partially ordered service, data delivery is suspended in only two cases:

(1) when the initial TPDU containing the service profile is missing, or,

(2) when no TSDU is deliverable according to the partial order, which in the case of this document means that within a single round-trip time, there is a missing TSDU on *each* of the eight parallel streams.

For partially-ordered service, each single TPDU loss affects only the delivery of the stream to which it belongs, roughly one-eighth of the traffic. Thus with high probability, at least one of the eight streams can deliver some data at all times. Total gaps in the data delivery are therefore rare with partially-ordered service for this document. What is more common is to see changes in the slope of the curve

representing the delivery of bytes; this can be seen especially reflected in the individual runs of R2E for window size 32 (Figure 5.20).

The length of the gaps is determined by the time it takes for the sender to detect a missing packet and successfully send a retransmission. This time is at least a round-trip time, but in practice somewhat more. (See the discussion of RTO calculation in Section 3.6.7). The reason the length of the gap grows with window size is explained by queuing in the PPP link. Packets are submitted to the PPP link as quickly as the server (host `medoc`) can read data from the disk file and put the packets on the Ethernet (see Figure 5.1). PDUs are thus placed on the network by `medoc` much faster than the 9.6kbps bitrate of the PPP link. PDUs travel from `medoc` over the Ethernet to the lossy router (`alsace`), and then (unless dropped by the lossy router) to `sauterne`, and are queued in `sauterne`'s outgoing PPP link. This outgoing queue is the bottleneck in the system.

How large will this queue grow? If there were not flow control in place, it would grow arbitrarily long, since the arrival rate for this queue is higher—orders of magnitude higher, in fact—than the departure rate. However, because R2E and T2E, both based on KX2, use window flow control, the length of this queue will be limited by the sending window size (w). For loss rates > 0, eventually a packet is lost, and times out. When this occurs, the retransmission may have to wait in line behind as many as w packets.[58] Thus for larger w, we see larger gaps.

[58] There will tend to be a full window of packets even for the first retransmission, because the initial retransmission timeout tends to be large. The Van Jacobson algorithm has an initial RTO value of 6 seconds, which then goes up before it comes down, because of the contribution of the large deviation between 6 seconds and the first sample.

Why finding the optimal window size is a hard problem

One suggestion of these results is that one should avoid using a window size that is larger than the bandwidth delay product. Using a window larger than this cannot improve performance, and as these results show, it can hurt performance because of queuing effects.

However, unlike at the data-link layer, where the bandwidth and delay are typically fixed or have small variance (often negligible) due to processing times, at the transport layer in the Internet, bandwidth and delay can be highly variable. Variation in bandwidth available to a transport layer protocol in the Internet results from variability in the number of packet flows using each intermediate link in the path. Variation in delay arises from variations in queuing delays in intermediate routers. Less often, variations in delay and bandwidth may result from routing changes that may take place during the lifetime of a connection or flow (Paxson, 1996).

Therefore, transport layer protocols use various techniques to choose an appropriate window size. The TCP congestion control algorithms of Jacobson (slow start and cwnd) are one approach to this problem. While TCP does not *directly* attempt to measure the bandwidth delay product, the net effect of the algorithm is to dynamically determine a window size that balances several goals. Essentially, the overall goal is to maximize throughput, while maintaining fairness, and avoiding packet drops due to congestion—that is, due to excessive queuing in intermediate routers.[59] Even when such techniques are used, because the bandwidth and delay can

[59] Evaluating the effectiveness of the TCP congestion control algorithm at estimating the true bandwidth-delay product (among other goals, such as fairness) is actually a subject of considerable controversy. Researchers have come to different conclusions regarding three major variants of the TCP congestion control algorithms known as *Tahoe, Reno* and *Vegas.* A full comparison of these competing approaches is beyond the scope of this chapter; it suffices for our purposes that all three algorithms take

vary, sometimes the current window size will not be the best one at some particular moment.

With partially-ordered service, choosing an oversized window is less detrimental

The results from this experiment (and later ones in this chapter involving window size) show that one of the advantages of partially ordered service is that it can be more robust to an incorrect choice of window size. That is, if the window size ends up being larger than the bandwidth-delay product, then partially-ordered service degrades less than totally-ordered service under the same circumstances. This reduction in degradation occurs because the increased delay in receiving a retransmission of a missing packet affects only the successors of the missing packet, rather than necessarily affecting *all* packets.

Future work: evaluating gain for dynamic windows, correlating gain with density

Future work should use the KX3 TCP-friendly algorithms to evaluate the gain from partial order when the window size is adjusted dynamically. In particular, it would be interesting to determine what happens when there are one or more major step changes in the underlying network (e.g., in bandwidth or delay) during the lifetime of a connection. Such changes would have the effect that the transport layer's estimate of the window size would suddenly become inaccurate. In those cases, would partially-ordered service provide benefit during the time it takes the transport layer to detect that change and adjust the window size appropriately?

---

steps to dynamically adjust the window based on feedback. In particular, all three shrink the window to varying degrees in the presence of retransmissions.

Also, we would expect the precise degree of benefit from partially-ordered service to depend on the degree of parallelism in the partial order. In this experiment, we have eight parallel chains. We would expect the gain from partial order to be less when there are fewer chains in parallel. Simulation studies (Marasli et al., 1998; Marasli 1999b) show that the *density* of the partial order correlates well with expected performance gains from partially-ordered service. Future work can investigate whether the correlation found in these simulation studies can be replicated with empirical results using ReMDoR.

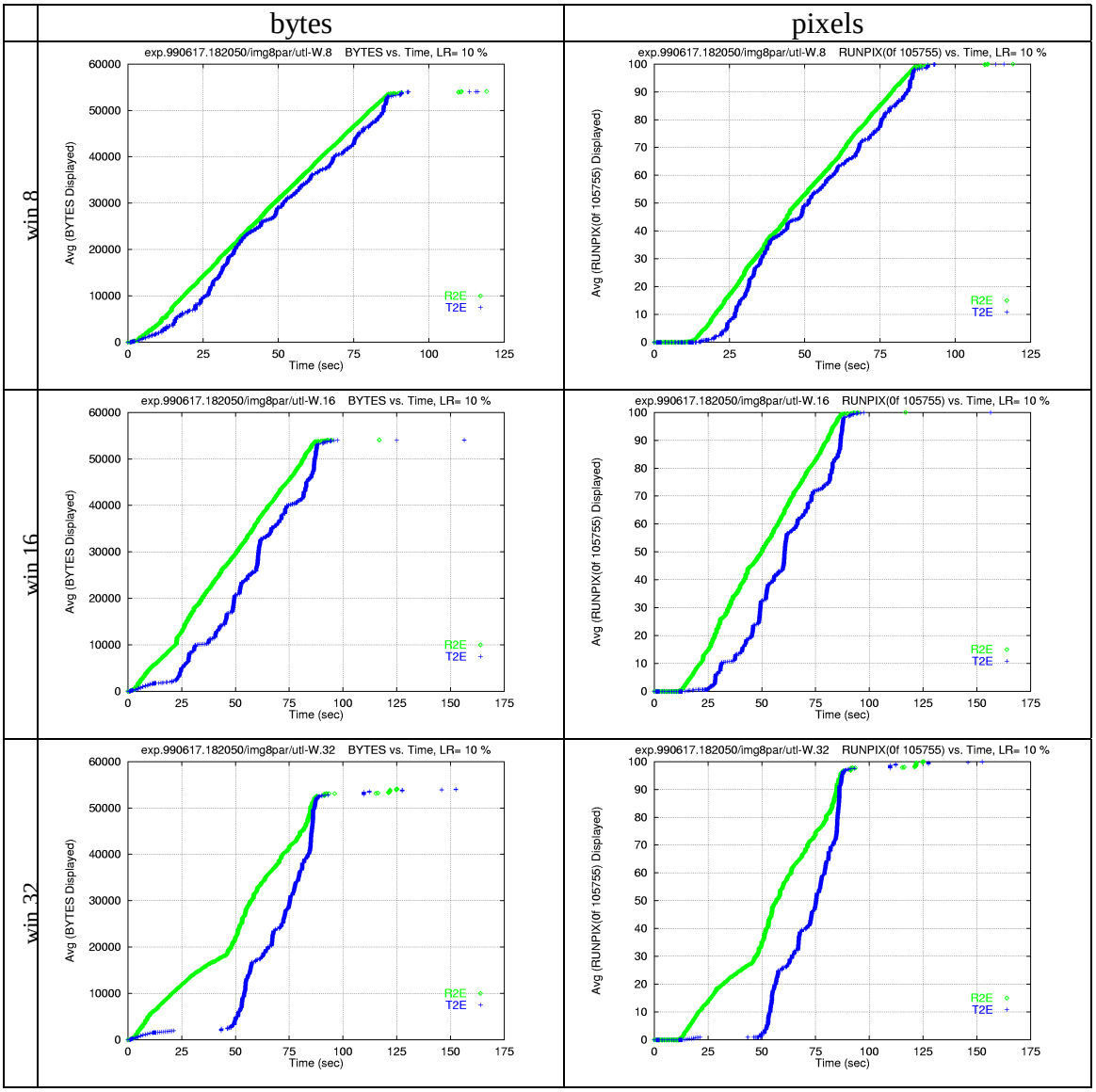


Figure 5.14 Experiment R1.3: Performance Graphs

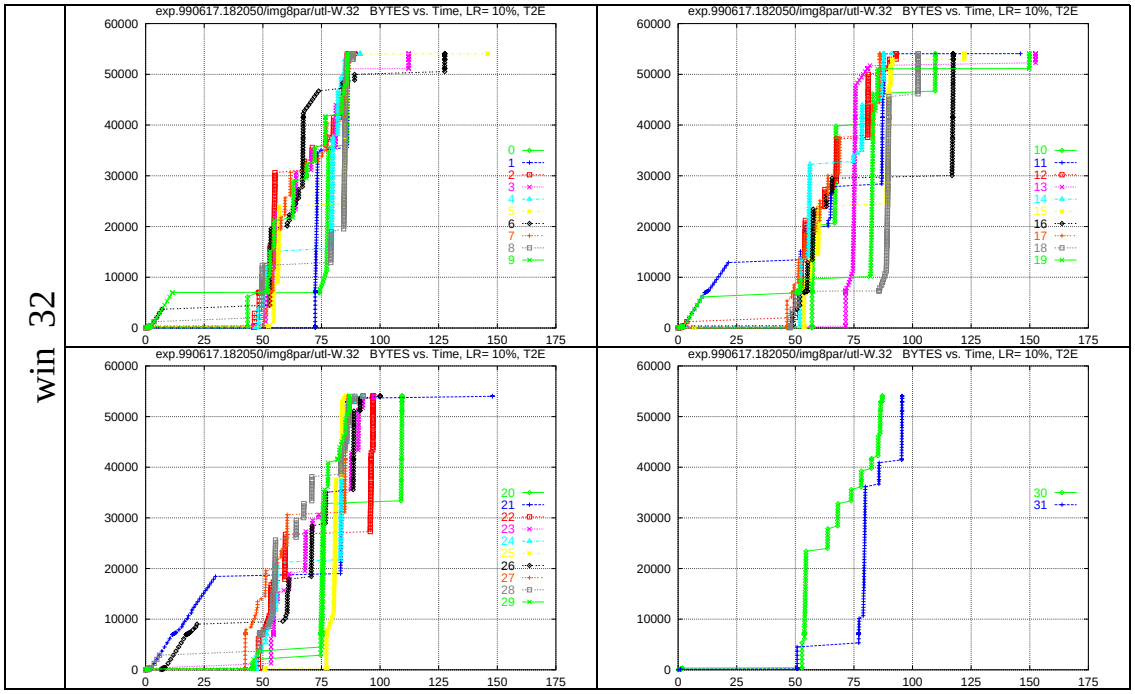


Figure 5.15 Exp. R1.3, Ind. Runs, T2E, win 32, bytes

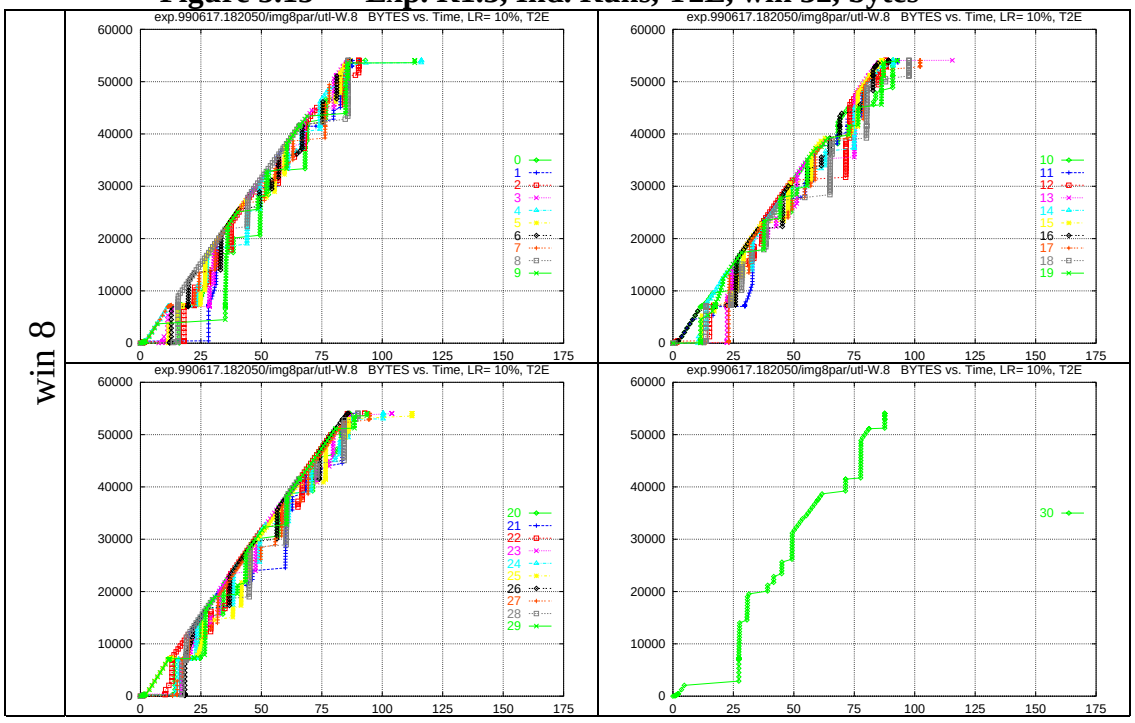


Figure 5.16 Exp. R1.3, Ind. Runs, T2E, win 8, bytes

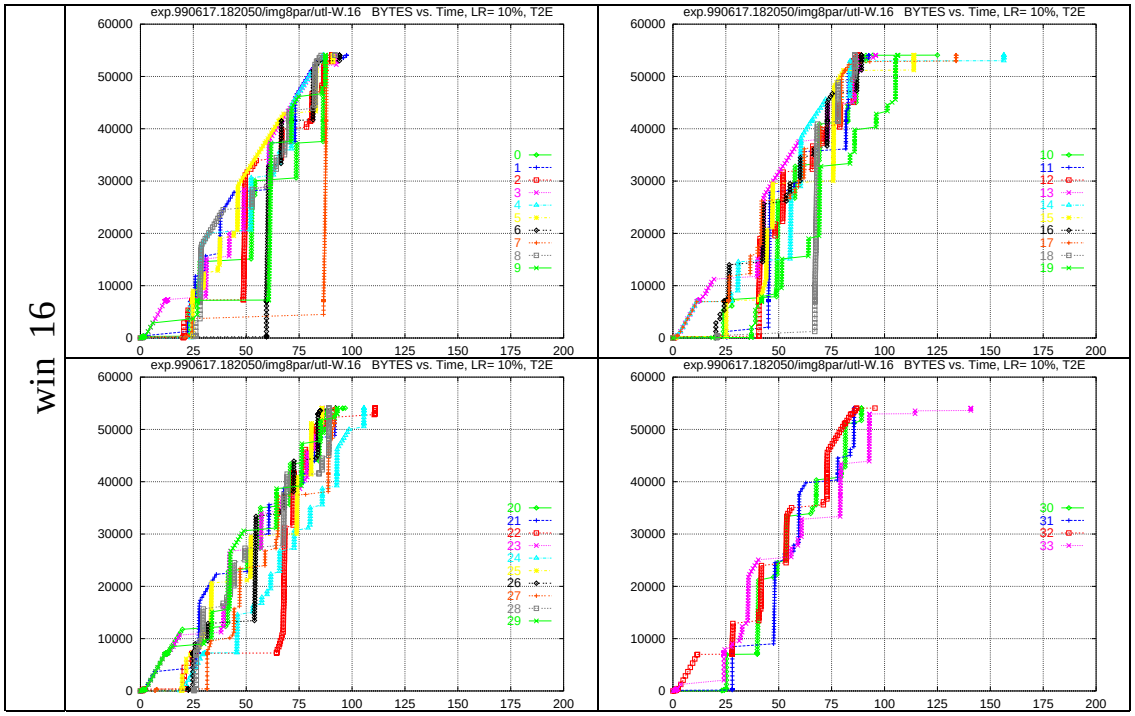


Figure 5.17 Exp. R1.3, Ind. Runs, T2E, win 16, bytes

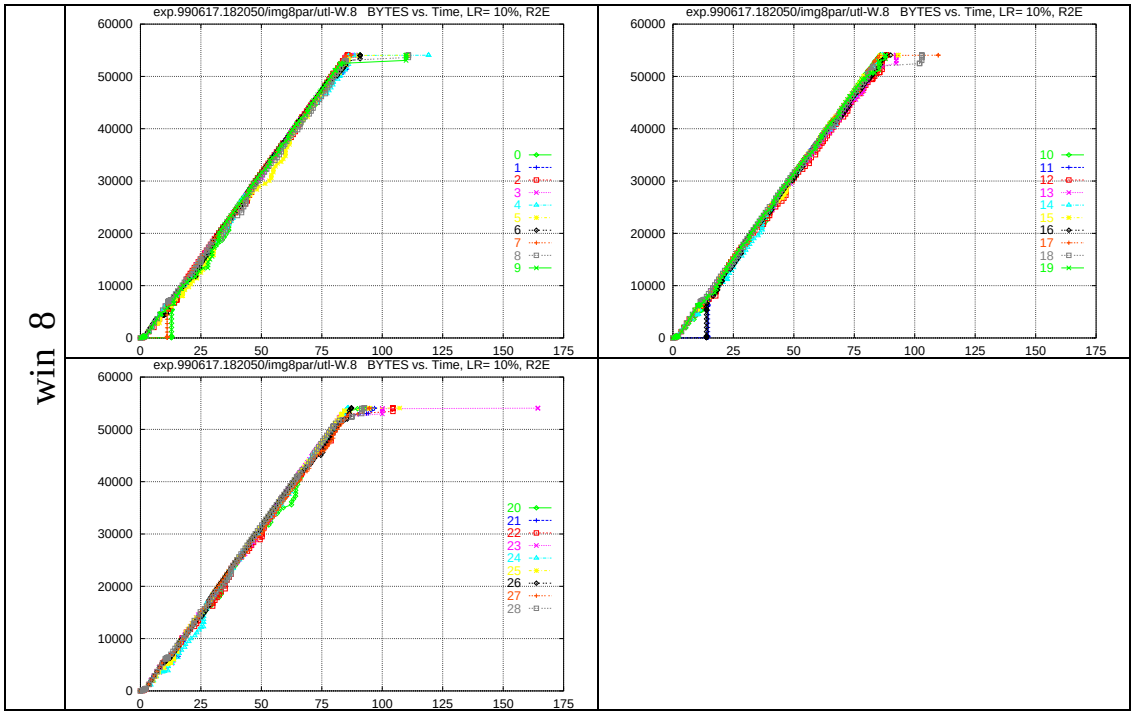


Figure 5.18 Exp. R1.3, Ind. Runs, R2E, win 8, bytes

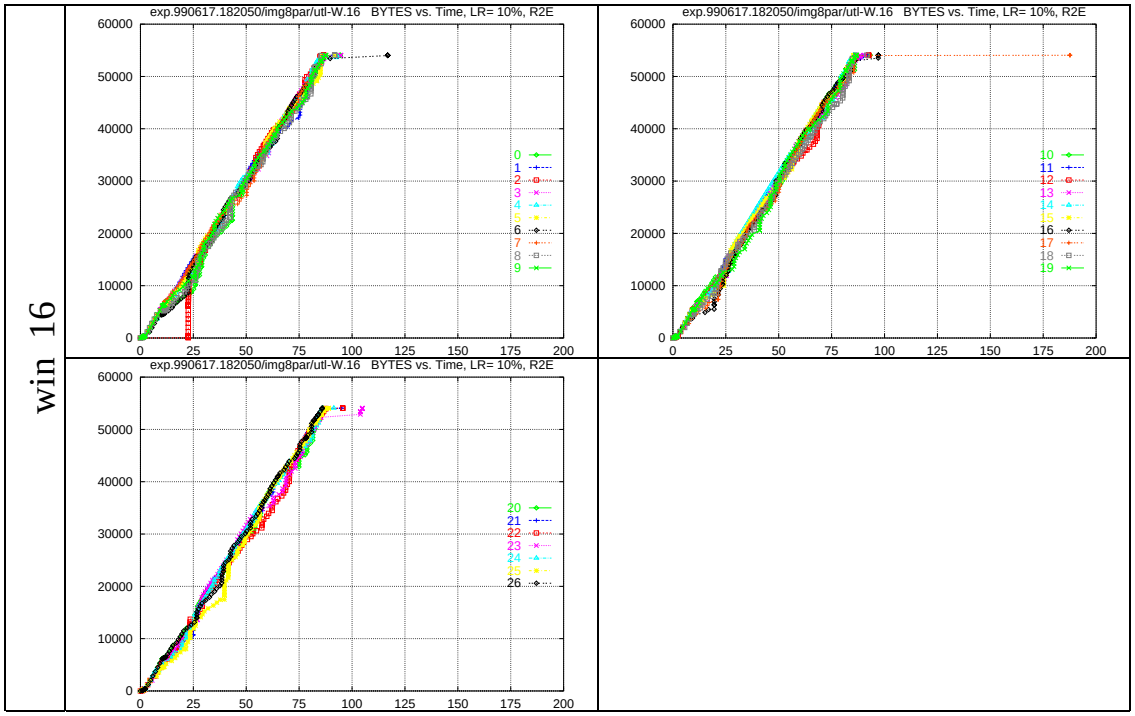


Figure 5.19 Exp. R1.3, Ind. Runs, R2E, win 16, bytes

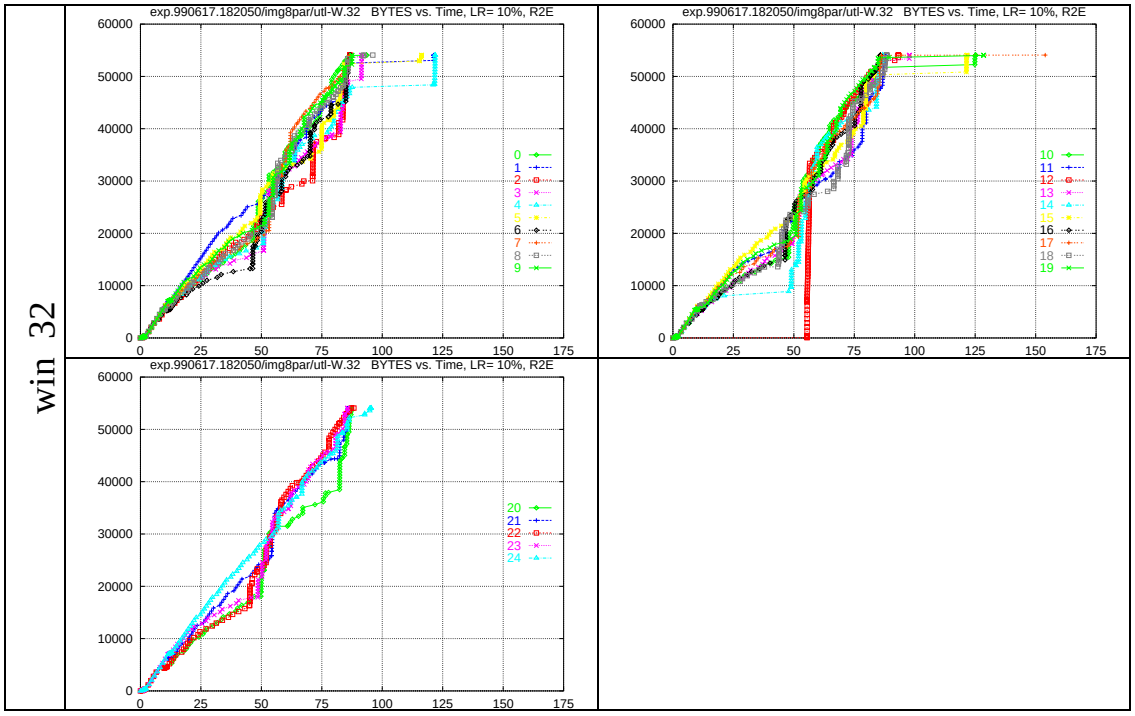


Figure 5.20 Exp. R1.3, Ind. Runs, R2E, win 32, bytes

5.3.8 Experiment R1.4: observations and conclusions

Experiment **R1.4** repeats Experiment **R1.3** using the reflector instead of the PPP link. Figure 5.3.x shows the bytes results for **R1.3** and **R1.4** side by side, plotted at the same scale. We omit the pixels results for this experiment, since for the observations we make in this section, they would add little additional information to the discussion.

Our experience here is similar to that of comparing **R1.2** with **R1.1**. We find that the results do not match quantitatively, which we attribute to overheads in the PPP implementation that the reflector does not model. Nevertheless, we find that the results do match qualitatively, in that all the observations we applied to the results from **R1.3** apply equally to **R1.4.** Again, this increases our confidence in interpreting results based on the packet reflector.

5.3.9 Experiment R1: summary

In Section 5.3 describing Experiment **R1**, we have shown the existence of a set of network conditions (9.6kbps, window size 16, loss rate 10 or 20%) where partially-ordered/reliable transport service provides significant benefits in progressive display over ordered/reliable service. We have shown that window size is an important parameter in determining system performance, and windows that are too large can be detrimental because queuing at the bottleneck link delays retransmissions. Moreover, we have shown that partially-ordered service can ameliorate the negative effects of a window size that is inappropriately large, providing better performance overall because it is less susceptible to delays in retransmissions.

Future Work

The fact that partially ordered service can provide performance benefits over ordered service for parallel images has significant implications for work on network-conscious image compression. The idea of network-consciousness is to replace image encodings that require ordered/reliable service with encodings that allow out-of-sequence delivery (for example, unordered/reliable service). As (Iren, 1999b) shows, network-conscious image formats can improve progressive display at higher loss rates, at the expense of a modest penalty in compression, and hence a modest penalty in performance at 0% loss. Our results show that for documents with parallel images, *some* gains in performance from out-of-sequence delivery (partially-ordered delivery, in this case) are possible *without* paying *any* penalty in compression. Further study of network consciousness and partial order in tandem is therefore indicated. Future work can also investigate the correlation between performance gains and the density of the partial order, by performing experiments with documents that have various densities.

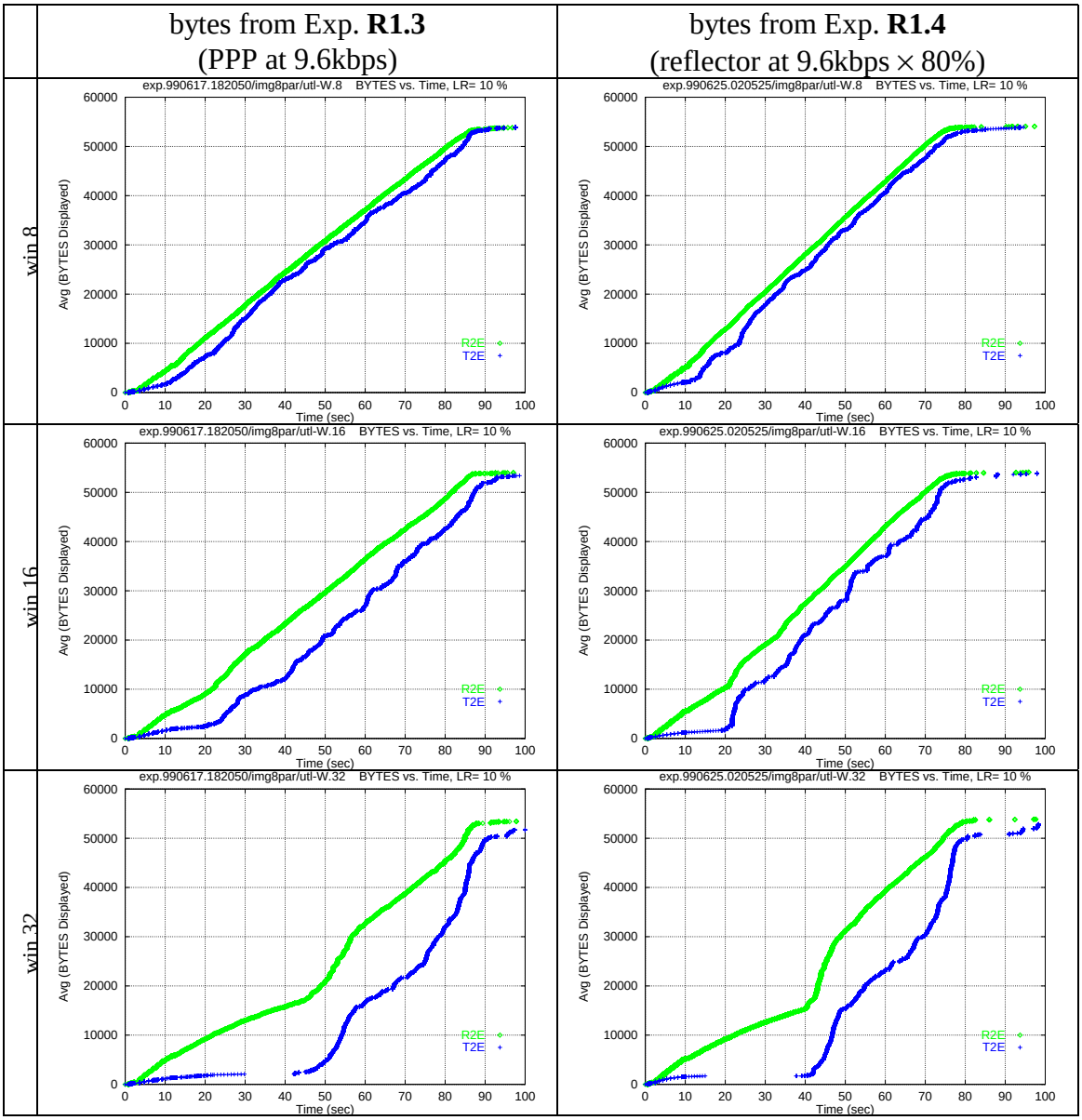


Figure 5.21 Experiment R1.3 vs. R1.4: Performance Graphs

5.4 Experiment R2: O/R vs. PO/R for eight parallel GIF images at 128kbps

As in Experiment **R1,** Experiment **R2** involves the retrieval of the `img8par.pmsl` document, a document containing eight images presented in parallel. However, in this experiment, we investigate the performance of PO/R service vs. O/R service at a single bit rate of 128kbps, using the reflector. There are several real-world scenarios that can motivate this bit rate:

- narrowband ISDN service,
- xDSL service, or
- the effective throughput available to some particular connection over a wide-area network such as the Internet.

We investigate the following hypotheses in Experiment **R2**:

> **Hypothesis 5.4.1** For img8par, at 128kbps, there will be more gain from PO/R service vs. O/R service at higher round-trip delays than at lower round-trip delays.
>
> **Hypothesis 5.4.2** For img8par, at 128kbps, there will be neither a significant gain nor a significant penalty for using PO/R service vs. O/R service at 0% loss.
>
> **Hypothesis 5.4.3** For img8par, at 128kbps, there will be increasing gains from using PO/R service vs. O/R service at 10%, 20% and 30% loss.
>
> **Hypothesis 5.4.4** For img8par, at 128kbps, as the window size is increased from a value below the bandwidth-delay product to values near the bandwidth-delay product, performance will increasingly improve for both PO/R service and O/R service.
>
> **Hypothesis 5.4.5** For img8par, at 128kbps, as the window size is increased from values near the bandwidth-delay product to values above the bandwidth-delay product, performance will (a) degrade for both PO/R and O/R service, but it will (b) degrade more, and (c) degrade faster for O/R service than for PO/R service.

Organization of Section 5.4

- Section 5.4.1 describes the parameters for the experiments that comprise R2.
- Sections 5.3.2 through 5.4.4 describe our observations and conclusions for R2.
- Section 5.4.5 provides an overall summary for Section 5.4.

5.4.1 Experiment R2: parameters

As Table 5.5 shows the parameters that all the **R2** Experiments have in common is a bit rate of 128kbps, the `img8par.pmsl` document, and the comparison of R2E and T2E. Experiments **R2.1** and **R2.2** focus on the effects of loss rate for two different one-way delays, 250ms and 500ms. Experiment **R2.3** focuses specifically on the effect of delay, adding the values 0ms and 125 ms to the values 250ms and 500ms already studied in the previous two experiments. Finally, Experiment **R2.4** examines the effect of seven different window sizes on performance.

Table 5.5 Parameters for Experiment R2

	Values for Experiment:			
Parameter	***R2.1***	***R2.2***	***R2.3***	***R2.4***
Mechanisms	**R2E,T2E**			
Loss Rates (%)	**0,10,20,30**	**0,10,20,30**	0,20	10
Network	reflector			
Bit rate (kbps)	128			
One-way delay (ms)	250	500	**0, 125, 250, 500**	250
Document	`img8par.pmsl`			
Window Size (pkts)	32			**4, 8, 16, 32, 64, 128**

Bold indicates the parameters that are the focus of the experiment

5.4.2 Experiments R2.1 and R2.2: observations and conclusions

Experiments **R2.1** and **R2.2** investigate the effect of loss rate on progressive display at 128kbps, with a one-way delay of 250ms and 500ms, respectively. These represent realistic delays that might occur if a satellite link is in place between two endpoints, or if a connection involves communication across multiple continents. Figure 5.22 shows the average number of pixels that are displayed at every point in time where there is a change in the number of pixels for *any* run, with **R2.1** on the left, and **R2.2** on the right. We also provide data about the median case for progressive display of pixels in Table 5.6. We make the following observations from this data:

(1) When the loss rate is zero, the entire document takes roughly 5 seconds to display—slightly less when the one-way delay is 250ms, and slightly more when the one-way delay is 500ms.

Observations (1) stands in contrast to our observations of Experiment **R1**, where the document took over a minute to display at 0% loss. This difference affects our interpretation of the impact of performance gains on end-user satisfaction. When the entire document takes over a minute to download, one might expect that the user's attention will wander. When the entire download takes only 5 seconds, we can expect that the user will remain focused on the progressive display. We also note that while there is a measurable impact of increasing the one-way delay from 250ms to 500ms, the impact is negligible at 0% loss in terms of end-user perception.

(2) As the loss rate increases from 0% to 10%, 20% and 30%, the performance of both PO/R and O/R service degrades. However, the performance of O/R degrades more severely than that of PO/R.

(3) As the loss rate increases, both documents tend to finish at the same time. However, for the non-zero loss rates, from 1-2

seconds after the document is requested until the end, PO/R provides more pixels than O/R at every instant in time (in the average case.)

(4) For **R2.1**, at each loss rate, in the median case for PO/R service, pixels begin appearing in less than 1.2 seconds. In the median case for O/R service, no pixels appear until after:

3.3 seconds for 10% loss,

5.0 seconds for 20% loss, and

7.1 seconds for 30% loss.

The results for **R2.2** are similar.

(5) There does seem to be an increase in the delay of pixel delivery for the experiments at 500ms vs. the corresponding ones at 250ms, however it is unclear whether this increase is a direct result of the differing one-way delays, or is due to experimental variance.

Observation (4) is particularly significant result in terms of user satisfaction. Human factors studies summarized in (Mogul 1995) suggest that response times of two to four seconds are preferred to those exceeding four seconds, for reasons related to attention spans. In conclusion, we note that the data from Experiments **R2.1** and **R2.2** provide evidence to support Hypotheses 5.4.2 and 5.4.3. There is somewhat less evidence to support Hypothesis 5.4.1.

Table 5.6 First pixel delivered (median) for R2.1

loss rate	R2E	T2E
0%	1.1	1.2
10%	1.2	3.3
20%	1.3	5.0
30%	1.5	7.2

Table 5.6 shows the earliest time (measured in elapsed seconds from the document request) at which the median over all experiments of the statistic "pixels delivered" is non-zero.

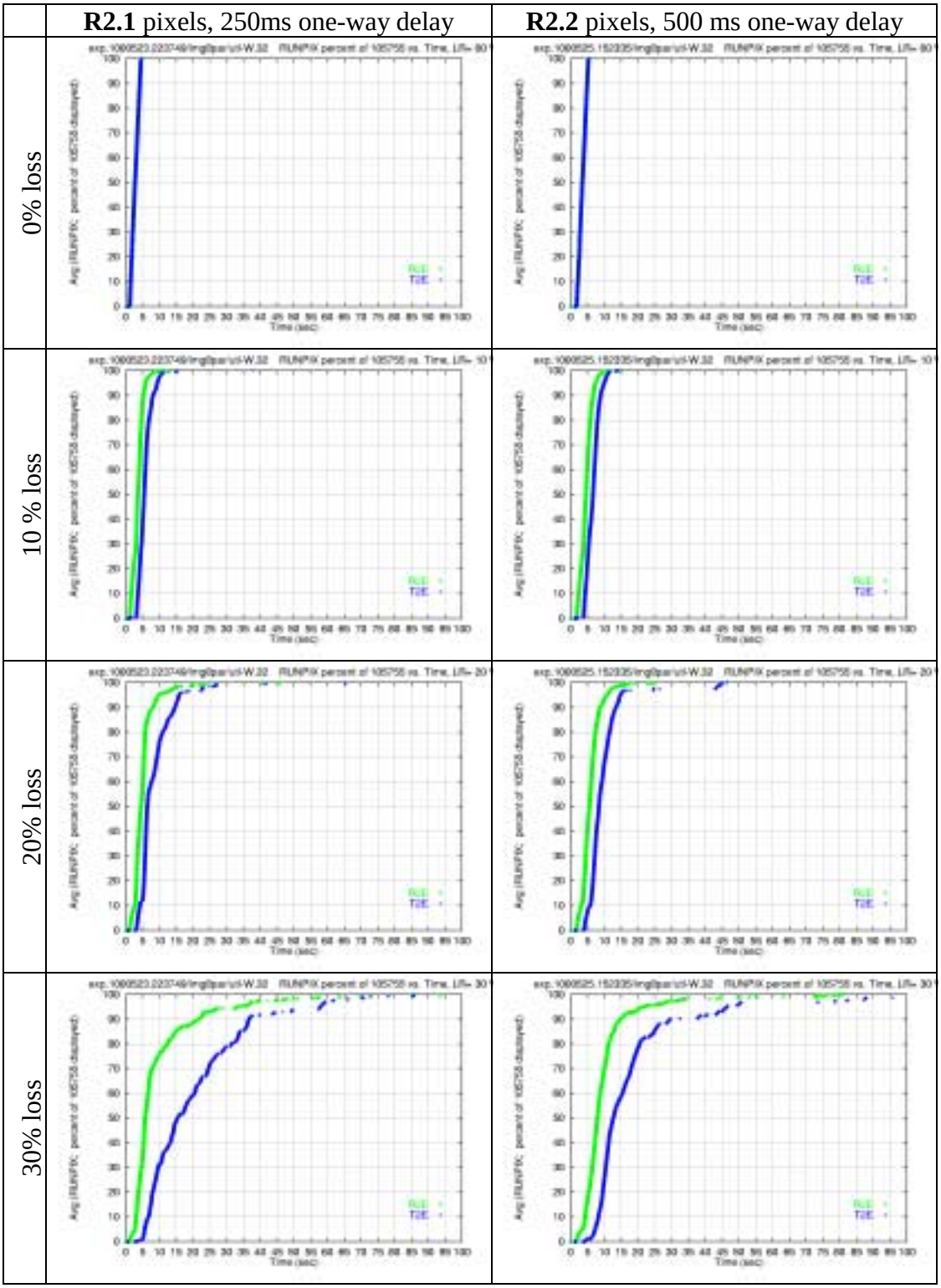


Figure 5.22 Exp. R2.1, R2.2, pixels, 128kbps, 4 loss rates × 2 delays

5.4.3 Experiment R2.3: observations and conclusions

Experiment **R2.3** examines the effect of round-trip delay on performance, and on the gap between PO/R and O/R service. We model round-trip delay by changing the propagation delay parameter of the reflector. The actual round-trip delay encountered by the PDUs in the experiment includes not only this propagation delay (times two) but also any queuing delay at the reflector.

Figure 5.23 illustrates the progressive delivery of pixels for both PO/R and O/R service at 0% and 20% loss for four different one-way round-trip delays. We first consider the left column of Figure 5.23, showing results for 0% loss. As predicted in Hypothesis 5.4.2, there is no difference between the average case for PO/R and O/R service at any delay. The difference among the delays is visible by a shifting of the entire graph to the right as the delays increase. Also as expected, the amount of the shift corresponds precisely to the double the increase in the one-way round-trip delay introduced in each experiment; for example, the 250ms case is shifted 500ms to the right as compared to the 0ms case, and the 500ms case is shifted another 500ms to the right as compared to the 250ms case. The fact that these results come out precisely as we expect increases our confidence in the accuracy of the experimental framework.

We now consider the results in the right column, showing the effect of increased network delay at 20% loss. The first observation is that at every network delay, there is a significant improvement in the progressive display for PO/R service vs. O/R service. Consider the points at which each graph crosses the line y=80% for both PO/R and O/R service. For PO/R service at each of the four network delays, the 80% line is crossed at times (5.4, 5.9, 6.0, 7.2) respectively. By contrast, for O/R service, the corresponding figures are (10.1, 10.3, 10.6, 10.6). The clear advantage in terms of providing early response belongs to PO/R service. At the time that PO/R has

80% of the pixels for each of the respective network delays, the corresponding O/R cases have only (13%, 30%, 27%, 27%) respectively.

As a final observation, we provide throughput figures based on the median case for progressive display. For each network delay at 20% loss, Table 5.7 shows the time at which the median number of pixels delivered was 100%. PO/R service provides an improvement in throughput at each network delay. This observation is consistent with the general observation made in (Diot and Gagnon, 1999) that modest throughput gains may be obtained from out-of-sequence delivery. We suggest that more significant benefits can be illustrated by focusing on progressive display, a benefit not considered by Diot and Gagnon.

Table 5.7 R2.3: Last pixel delivered (median)

delay	R2E	T2E
0ms	11.5	13.3
125ms	11.1	11.7
250ms	11.1	12.8
500ms	12.6	13.7

Table 5.7 shows the earliest time (measured in elapsed seconds from the document request) at which the median over all experiments of the statistic "pixels delivered" is non-zero.

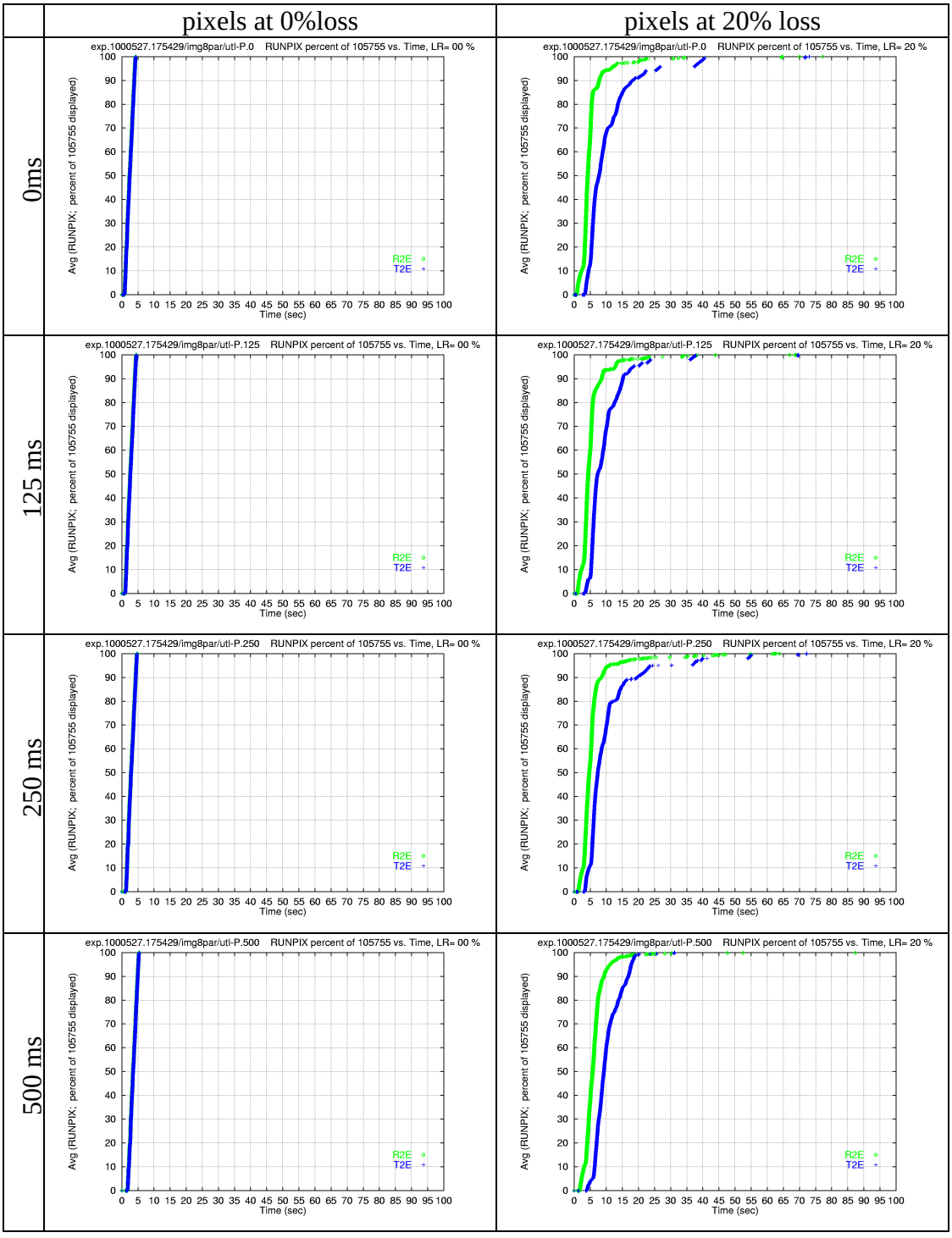


Figure 5.23 Experiment R2.3: Four one-way delays at 0% and 20% loss

5.4.4 Experiment R2.4: observations and conclusions

Experiment **R2.4** examines the effect of window size on performance, and on the gap between PO/R and O/R service. Figure 5.24 shows the average cases for progressive display of pixels for six different window sizes. Unlike other similar figures in this dissertation, Figures 5.24 and 5.25 show results for six different window sizes all on the same page, for 0% and 10% loss, respectively. There are two observations to make about the results of this experiment.

(1) The optimal sending window size for this transmission lies somewhere between 16 and 32.

(2) PO/R service provides better progressive display then O/R service over a range of window sizes, including those closest to the optimum window size.

We now explain both of these observations in more detail.

The optimal window size for Experiment R2.4 lies between 16 and 32

Suppose that we fix the transport service, and compare, say, just the PO/R results for each window size, or alternatively, compare just the O/R results for each window size. Looking first at the results for 0% loss in Figure 5.24, we notice that as the window size moves from 4 up to 128 (exponentially by powers of 2), that the performance improves as we move from 4 to 8 to 16 to 32, then stays about the same as we move from 32 to 64 and 128.

These results are exactly those that are predicted by the usual analytic model of sliding window protocols (Tanenbaum, 1996; Stallings 1998). As the window size of such protocols increase—for example, as one moves from the classic stop-and-wait, to go-back-n, and finally to selective-repeat schemes—performance

improves. However, there is a limit on the potential improvement; when the window size reaches 1+2*a*, where *a* is the ratio of one-way network delay to packet transmission time, no further improvement is feasible. At this point, the pipeline stays full of packets continuously, and the usable window is limited to 1+2*a*.

In the classic textbook analysis of sliding window schemes, the performance simply levels off after the window size is increased to the 1+2*a* level. Our results for 0% loss correspond exactly to this model. However, our results for 10% loss, shown in Figure 5.25, differ from what the classic analysis predicts, in that after the 1+2a level is surpassed, performance then begins to degrade. Observe that just as with the 0% loss case, performance improves as the window size is increased from 4 to 8 to 16, but then degrades as the window size is moved from 16 to 32, and continues to degrade as the window size is increased from that point. This is counterintuitive in the sense that one generally expects computer system performance to stay the same or improve when available resources (in this case, memory) are increased.

To understand why performance degrades as the window size is increased beyond the optimum point, it is necessary to review the experimental design, and in particular, the topic of flow control. Recall that the experimental environment (see Figure 5.1) consists of a fast link between the server and the packet reflector (the loopback interface of the machine `medoc`) which feeds a simulated slow link (the packet reflector, operating at 128Kbps), which then empties into fast links (100Mbps Ethernet) to the lossy router (`alsace`) and ultimately, the client (`buzet`). Since the TCP slow-start mechanism is not used in these experiments, as soon as the server starts to transmit the multimedia document, the server fills up the input buffer of the

packet reflector with an entire window's worth of packets at 100Mbps. This initial burst of *w* packets (where *w* is the sending window size in packets) creates a queuing delay that affects the retransmission of any subsequent packets in two ways. First, for the O/R case, the first missing packet, p_i will have to wait behind this initial burst, which delays not only p_i but also all packets from p_i up to p_w. Second, in both the O/R and PO/R cases, the adaptive retransmission timers will be tuned based on RTT measurements that include this queuing delay. Thus, as the window size increases, the RTO value of the transport sender also increases, causing the recovery from errors to take longer, thus decreasing throughput when the loss rate is non-zero.

We conclude that that the observed performance degradation for the 10% loss case as the window size is increased is an artifact of two aspects of the experimental environment: (1) having a single bottleneck link with no traffic other than our own, and (2) not employing slow-start. However, from the standpoint of evaluating the usefulness of PO transport service, the observation we explain next is more significant.

PO/R outperforms O/R over a range of window sizes at 10% loss

We now turn to an explanation of observation (2). Note that for 10% loss, with the exception of window size 4 (which is clearly too small to provide reasonable performance) PO/R service outperforms O/R service. This result is important because: (1) it is often difficult for a transport protocol to determine the optimal window size, and (2) this result indicates that PO/R service can outperform O/R service at a wide range of window sizes, including those near the optimal value. In fact, the optimal window size is a function of the round-trip delay, which is essentially a random variable, or more precisely, a stochastic process: a random variable which is a function

of time. Therefore, the best the transport layer can do is to estimate the optimal window size, and constantly refine its estimate based on measurements of the delay, loss rate, and effective throughput. The results of Experiment **2.4** indicate that PO/R service has the potential to improve progressive display as long as the transport layer is reasonably close to the optimal value, which we would hope to be the normal case.

As a final observation about the run where the window size is 4, we observe that PO/R service at least does no harm when the transport services window size estimate is much too small. The situation of a "much too small window size" occurs frequently during the slow-start phase of normal TCP congestion avoidance. Since TCP's congestion avoidance mechanisms dynamically change the window size as a response to congestion, an interesting topic for future work would be to investigate the effect of these window size changes on the performance difference between PO/R and O/R service. Such an investigation might plot measured RTT, effective window size, and the performance differential as a function of time.

5.4.5 Experiment R2: summary

Overall, the results for Experiment **R2** have shown a variety of conditions under which a PO/R service can provide better progressive display than O/R service for a particular kind of document: specifically, a document with parallel streams of data containing pixels, where each stream can be independently decoded and displayed. Specifically, with respect to our five hypotheses:

- The results of experiment **R2.3** provide little support for Hypothesis 5.4.1. At 0% loss, there is no gain or loss for PO/R service vs. O/R service, while at 20% loss, the benefit of PO/R service vs. O/R service seems to be fairly constant regardless of the round-trip delay.

- Hypothesis 5.4.2, on the other hand, is supported by the results of Experiments **R2.1**, **R2.2** and **R2.3**.

- Hypothesis 5.4.3 is supported by the results of Experiments **R2.1** and **R2.2**.

- Finally, Hypotheses 5.4.4 and 5.4.5 are supported by the results of Experiment **R2.4**.

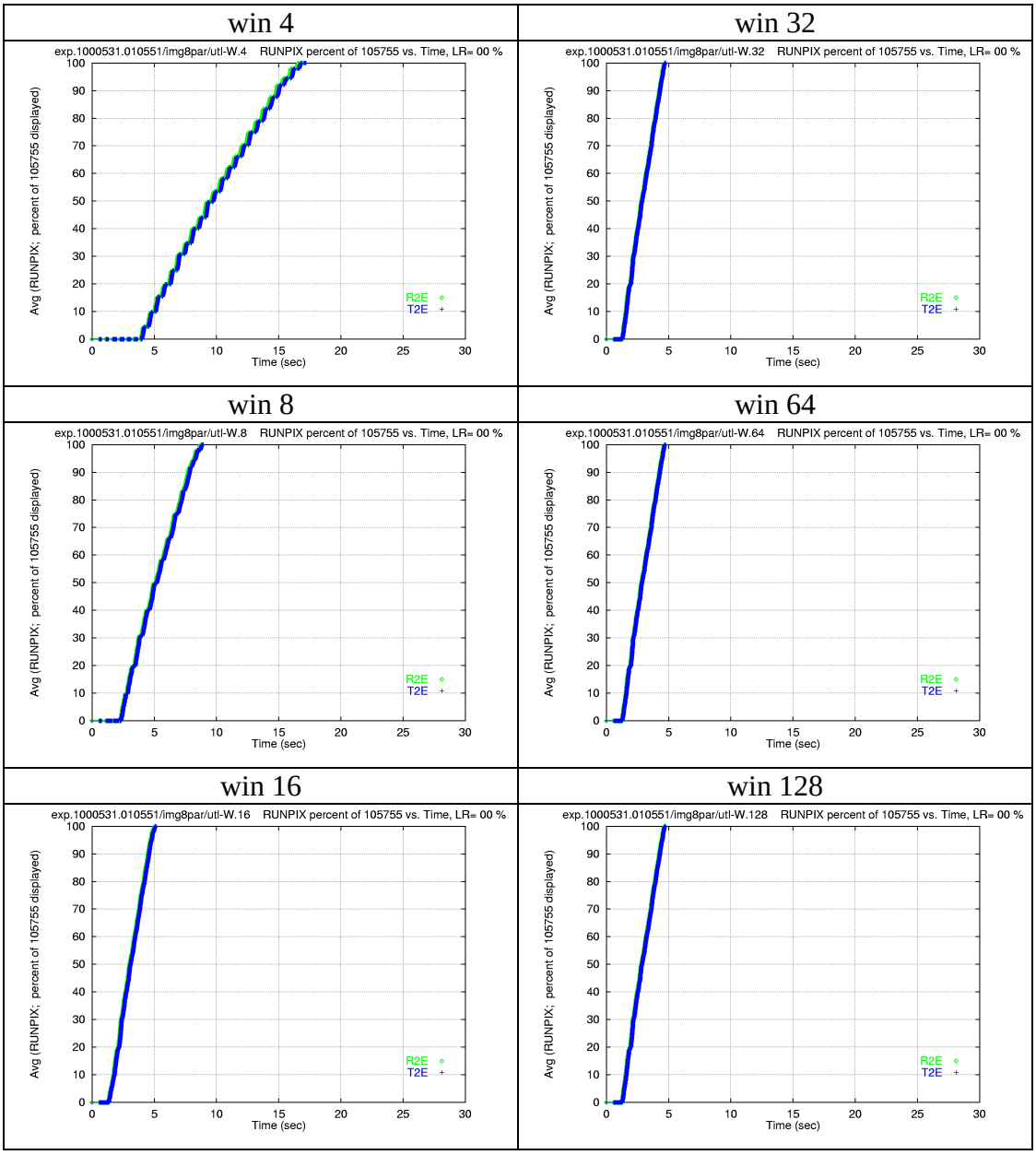


Figure 5.24 Exp. R2.4: Performance Graphs (128kbps at 0% loss)

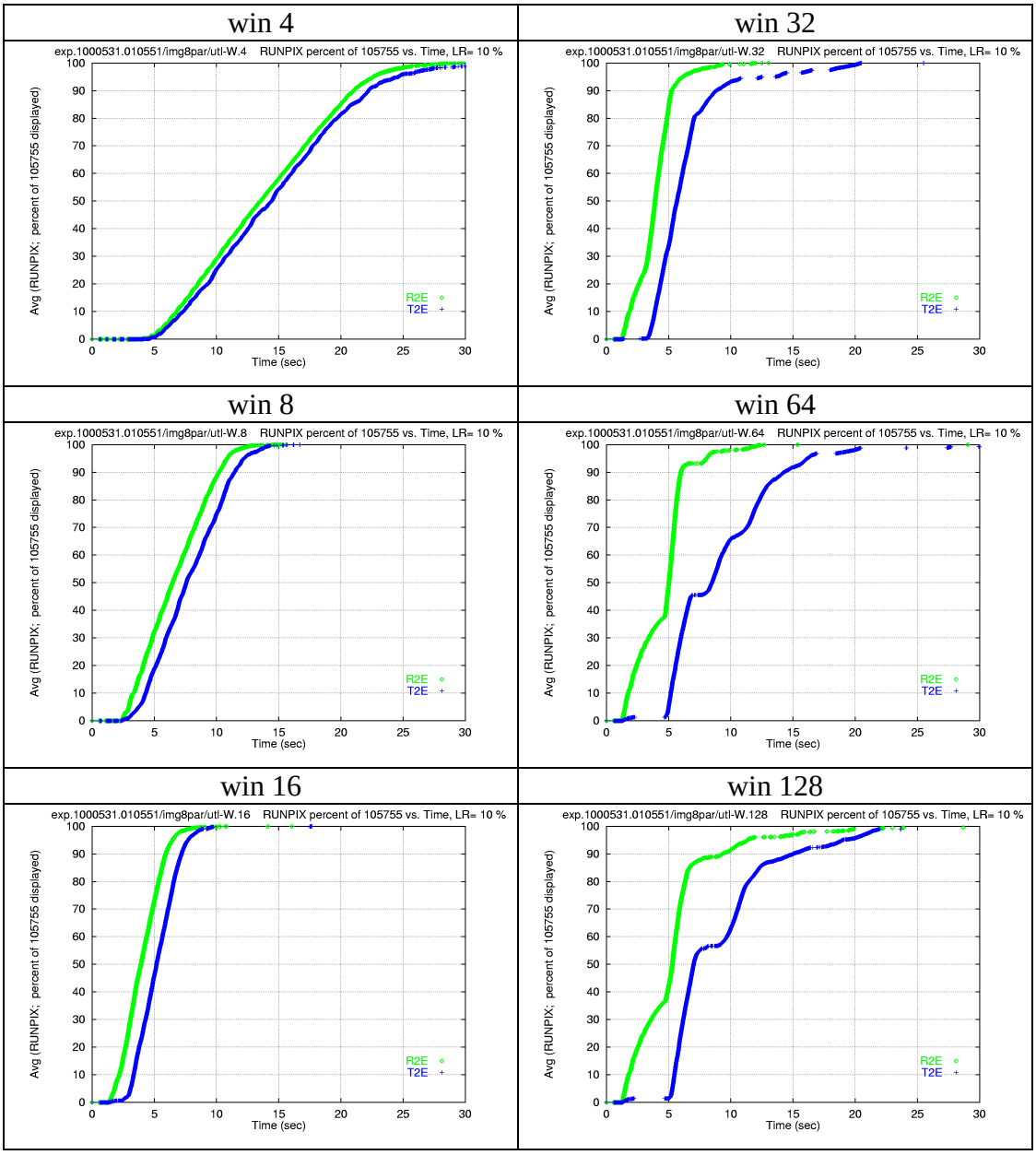


Figure 5.25 Exp. R2.4: Performance Graphs (128kbps at 10% loss)

5.5 Experiment R3: O/R vs. PO/R, eight parallel GIF images, various bit rates

5.5.1 Experiment R3: motivation

In **R3**, we vary the bit rate to put the advantage of PO/R service over O/R service into context. In contrast to **R1** and **R2**, where we chose two fixed bit rates and varied the loss rate, propagation delay, and window size, in **R3** we choose fixed values for loss rate, propagation delay and window size, and vary the bit rate. We have one hypothesis:

> Hypothesis 5.5.1: At lower bit rates, the absolute gain (measured in seconds) of PO/R service over O/R service will be larger than the gain at faster bit rates.

In addition, we expect to find evidence to support the following conjecture:

> Conjecture 5.5.2: For any fixed values of {loss rate, propagation delay, window size}:

a) there is a range of bit rates where the gain in progressive display has a significant impact on end-user perceived performance, and conversely,

b) when the bit rate is below some threshold b_1, or above some higher threshold b_2, the impact of the gain on end-user perceived performance is negligible.

The basis of Conjecture 5.5.2 is that for any given application context, there exists some lower bound, perhaps different for each individual user, below which the performance of both O/R and PO/R service would be considered unacceptable, therefore the end-user is unlikely to care about any qualitative difference. Conversely, above a certain bit rate, the gain is so small as to be imperceptible. We cannot rigorously prove or disprove this conjecture without human subject experimentation, however we can offer results to illustrate that this conjecture is reasonable.

5.5.2 Experiment R3: parameters

Table 5.8 shows the parameters for Experiment **R3**. The 2.4kbps bit rate is motivated by SINCGARS combat net radios; these radios provide a low bit-rate data channel as a side-feature to their main function as radios for voice communication. (Estimates of effective bit rates for SINCGARS vary considerably; 2.4kbps is a reasonable target value for that domain.) The 33.6kbps bit rate is motivated by dial-up modem service (V.34, 1998). The 128kbps bit rate is motivated by residential or small business ISDN or DSL service, or by the available throughput for a particular connection on a best-effort WAN such as the Internet.

Table 5.8 Parameters for Experiment R3

Parameter	***Values for Experiment R3:***
Mechanisms	**R2E,T2E**
Loss Rates (%)	0,20
Network	reflector
Bit rate (kbps)	**2.4, 9.6, 33.6, 128**
One-way delay (ms)	0
Document	`img8par.pmsl`
Window Size (pkts)	8

Bold indicates the parameters that are the focus of the experiment.

5.5.3 Experiment R3: observations and summary

For **R3**, we provide a more detailed analysis than that given in the previous experiments: for **R3**, we compare not only averages, but also several rank statistics: min, 25th percentile, median, 75th percentile, and maximum (these can also be described as the quartile boundaries of the data.) Comparing these rank statistics adds additional insight into the implication of the results for end-user perceived performance. Specifically, we provide the following graphs:

- Figure 5.26 shows the average results for the progressive display of pixels at these four bit rates, for both 0% loss and 20% loss.
- Figure 5.27 shows the median results for progressive display of pixels at 20% loss only.
- Figures 5.28 and 5.29 shows the quartile boundaries for progressive display of pixels at 20% loss.

Referring to Figure 5.26, we first note that at 0% loss, as in previous experiments, there is no significant difference between the average performance of R2E vs. T2E at 0% loss; the median graphs (omitted) show the same results. Therefore, the remainder of our analysis will focus on the 20% loss case.

Referring to Figures 5.26 and 5.27, overall, we note that at each bit rate, there is a significant improvement in the progressive display of R2E (PO/R service) over T2E (O/R service.) We now make some more detailed observations about these graphs for each bit rate.

Observations for bitrate 2.4kbps

Both the average case graph (upper right, Figure 5.26) and median case graph (Figure 5.27, upper left as viewed in landscape) show an advantage for PO/R over O/R service. In the median case, pixels begin appearing on the screen at 39.2 seconds for PO/R service. By the time, in the median case, that O/R service is presenting the first pixel (32 seconds later, at 71.2 seconds), PO/R service has delivered 13% of the pixels in the median case. The PO/R median case crosses the 50% mark (that is, at least 50% of the pixels have been delivered) at time 156.0, while the O/R median case requires 8.1 seconds longer to reach this point in the document. Similar figures could be cited for the average case graph. An even more interesting observation is that when one looks at the quartile boundaries (top left of Figure 5.28),

the advantage of PO/R service of O/R service tends to increase as one moves from the maximum towards the minimum rank statistic. The lines representing the maximum number of pixels displayed at each point in time (best case performance) for O/R vs. PO/R service are not significantly different from one another. However, as one examines the respective quartile boundaries in sequence—that is, the 25^{th}, 50^{th}, 75^{th}, and 100^{th} percentile of pixels displayed at each point in time—one observes that the advantage of O/R vs. PO/R service increases at each boundary. Examination of the worst cases for both O/R and PO/R service at various loss rates revealed long runs of successive losses of the same packet early in the run. PO/R service is able to recover sooner than O/R service in these cases. We can make the general observation that "the worse things get, the more advantage out-of-sequence delivery can offer."

Observations for bitrate 9.6kbps

The results for 9.6kbps are similar to those for 2.4kbps. Both the average case graph (right column, 2^{nd} from top, Figure 5.26) and median case graph (Figure 5.27, lower left when viewed in landscape) show curves that are shaped very much like those for 2.4kbps—the likeness is particularly striking in Figure 5.27, where the two graphs have been scaled to allow a fair comparison between the bitrates. Because the shape is similar, we will not provide detailed analysis of points on the median graphs, but will point out only that the advantages at this bitrate are 25% of the duration of the advantages for the 2.4kbps case. This makes the advantages less significant in terms of end-user perceived performance, although relative to the entire size of the document, they are essentially equivalent. The trend regarding the quartile boundaries that was observed at the 2.4kbps loss rate holds here as well, with the exception of the minimum lines; we can expect that there will be more variance in the

maximum lines, since these represent an extreme point in the data set. Nevertheless, there is an increasing advantage for PO/R service as one moves from the 75th to the 50th and 25th percentiles.

Observations for bitrate 33.6kbps

While for 2.4kbps and 9.6kbps, we could reach roughly the same conclusions regardless of whether we considered the average or the median graphs, for the higher bitrates in Experiment **R3**, the shape of the average and median graphs do not agree. Consider, in particular, the average graph for 33.6kbps (Figure 5.26, right column, 3rd from top.) We might conclude from this graph that the key advantage of PO/R service is experienced during the delivery of the final 5% of the pixels. However, when we examine the median and quartile boundaries, a different picture emerges. In particular, consider the top row of Figure 5.29, and the top row of Figure 5.30, where the same graph is presented at four different scales. We observe that at all of the quartile boundaries except the minimum, the graphs show a 2-4 second advantage for PO/R service. When the worst case performance is compared, PO/R service shows what appears to be a dramatic advantage, with an improvement in progressive display of more than 30 seconds. However, it should be borne in mind that the extreme case is more subject to variance, so we should be careful about claiming this as an advantage for partial order. Instead, we highlight it to illustrate how the average was affected by an extreme case. We will therefore sometimes use the median rather than the average to summarize datasets in the remainder of this chapter, because of the greater robustness of the median statistic.

Observations for bitrate 128kbps

An interesting feature of the results for 128kbps is revealed by considering the progression of normalized graphs in Figure 5.28 and 5.29, particularly in the left hand column, where each tick mark represents the delivery 12000 bytes of data. We observe that while the *absolute* gain for PO/R service decreases, the *relative* gain in terms of the document size actually increases. We also observe that at each percentile, PO/R service outperforms O/R service, with the median case running 1 to 3 seconds ahead throughout most of the document. Another interesting feature of the 128kbps results is that, with the exception of the worst-case results, for each of the percentiles, there is a particular shape to the gain. First, PO/R service jumps out to an early lead. Then O/R service catches up, narrowing the gap (to zero gain, in the case of the 100^{th}, 75^{th} , and 50^{th} percentile graphs). Finally, PO/R service pulls ahead again, restoring its earlier lead. It would be interesting to run more experiments at this bitrate and other nearby bitrates (64kbps, 256kbps), with this document and other documents, to determine whether this shape is feature of this dataset only, or is consistent across many runs.

Experiment R3: summary

In summary, we conclude that for 20% loss, significant gains for PO/R service can be shown over a range of bit rates from 2.4kbps to 128kbps.

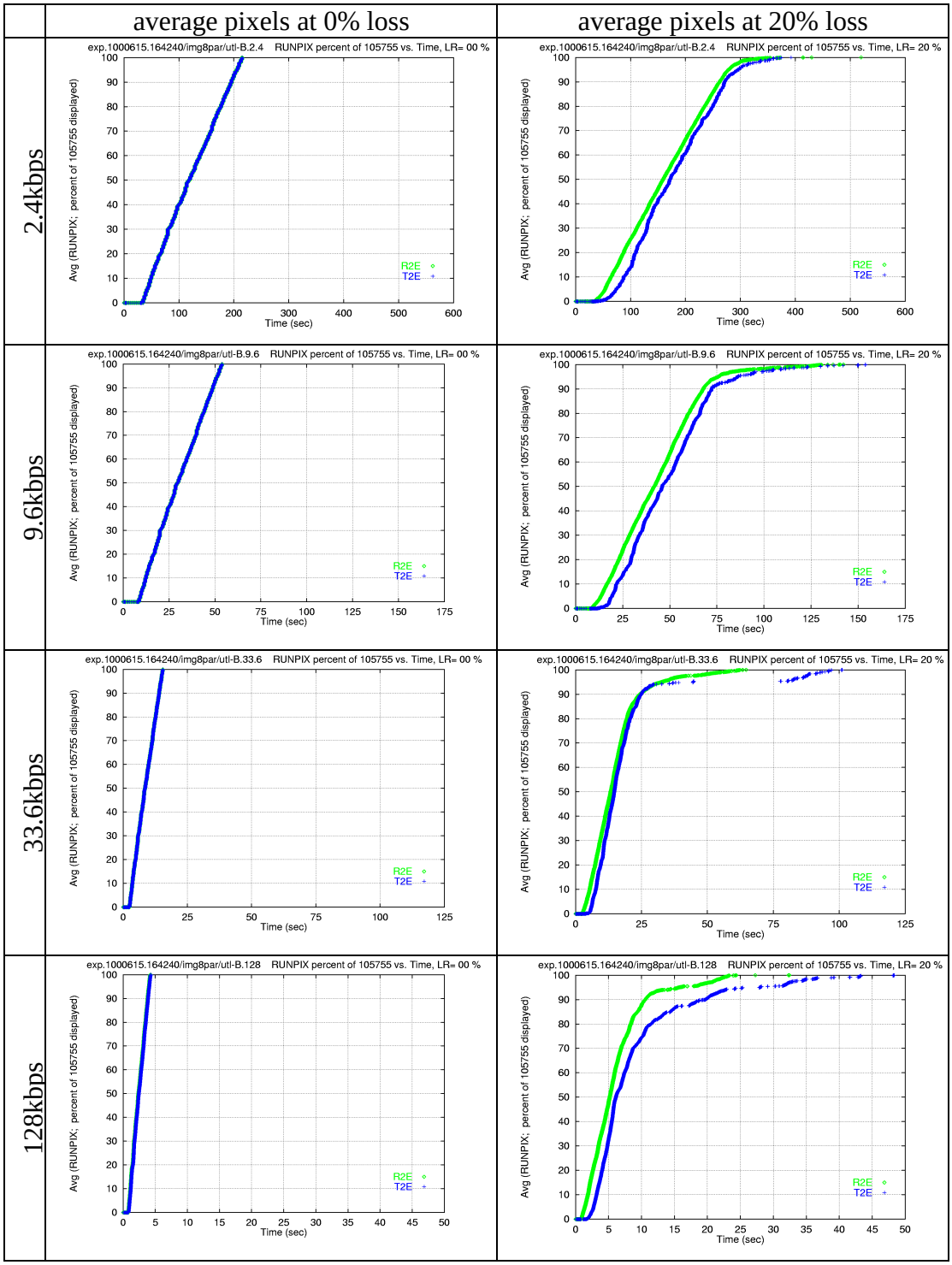


Figure 5.26 Experiment R3: Average Performance Graph

Graphs in Figure 5.26 are scaled for each bitrate so that the last data point (last PDU arrival of the run with the slowest response time) is included. The scales are not normalized w.r.t. throughput, as they are in Figure 5.4.

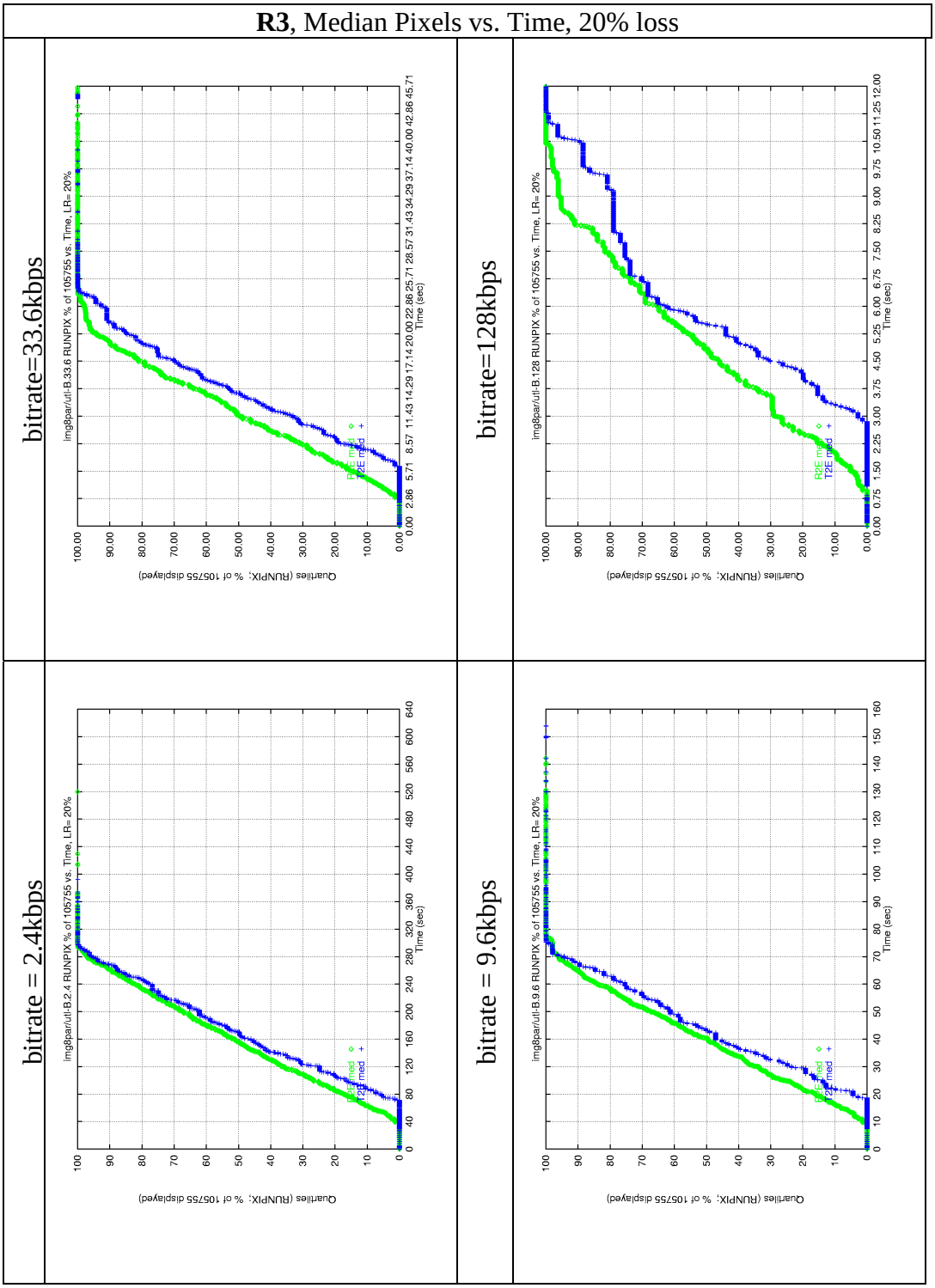


Figure 5.27 Experiment R3: Median Performance Graph

The graphs in Figure 5.27 are normalized with respect to throughput: each of the twelve tick marks on the x-axis represents the time it would take to send 12000 bytes at the respective bitrates.

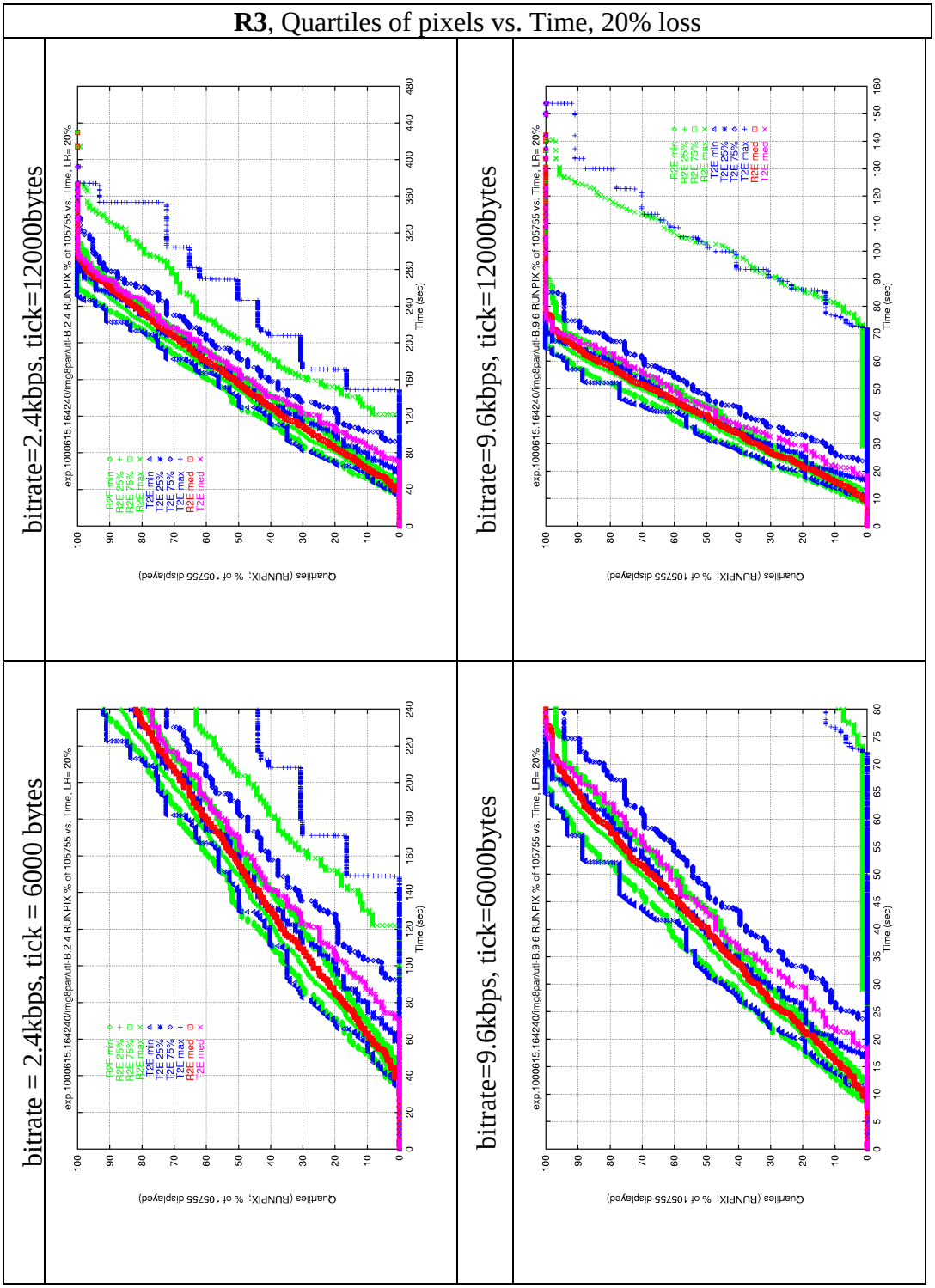


Figure 5.28 Experiment R3: Quartiles Performance Graph

The graphs in Figure 5.28 are normalized with respect to throughput: each of the twelve tick marks on the x-axis represents the time it would take to send 6000 bytes or 12000 bytes (see column labels) at the respective bitrates.

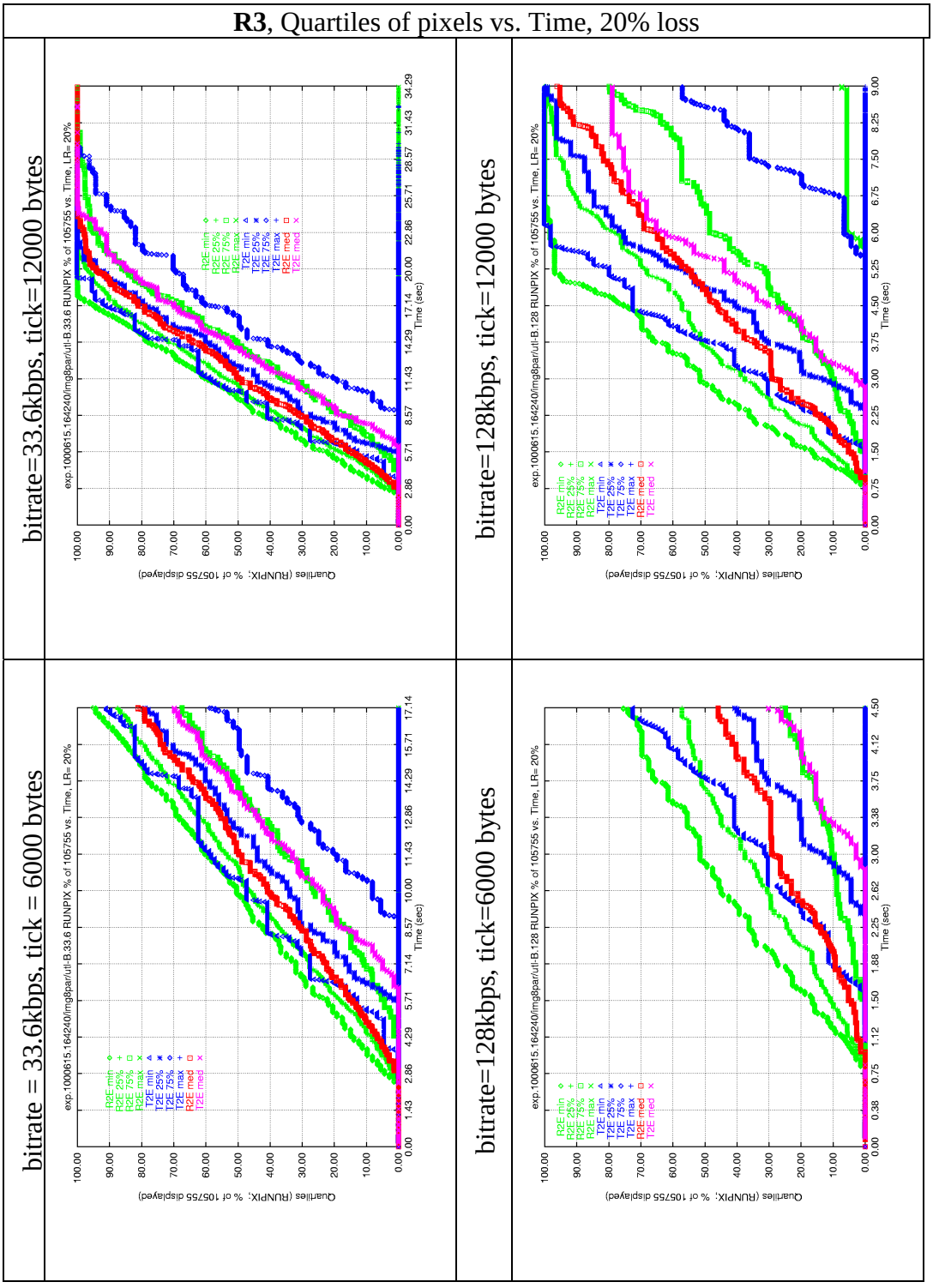


Figure 5.29 Experiment R3: Quartiles Performance Graph

The graphs in Figure 5.29 are normalized with respect to throughput: each of the twelve tick marks on the x-axis represents the time it would take to send 6000 bytes or 12000 bytes (see column labels) at the respective bitrates.

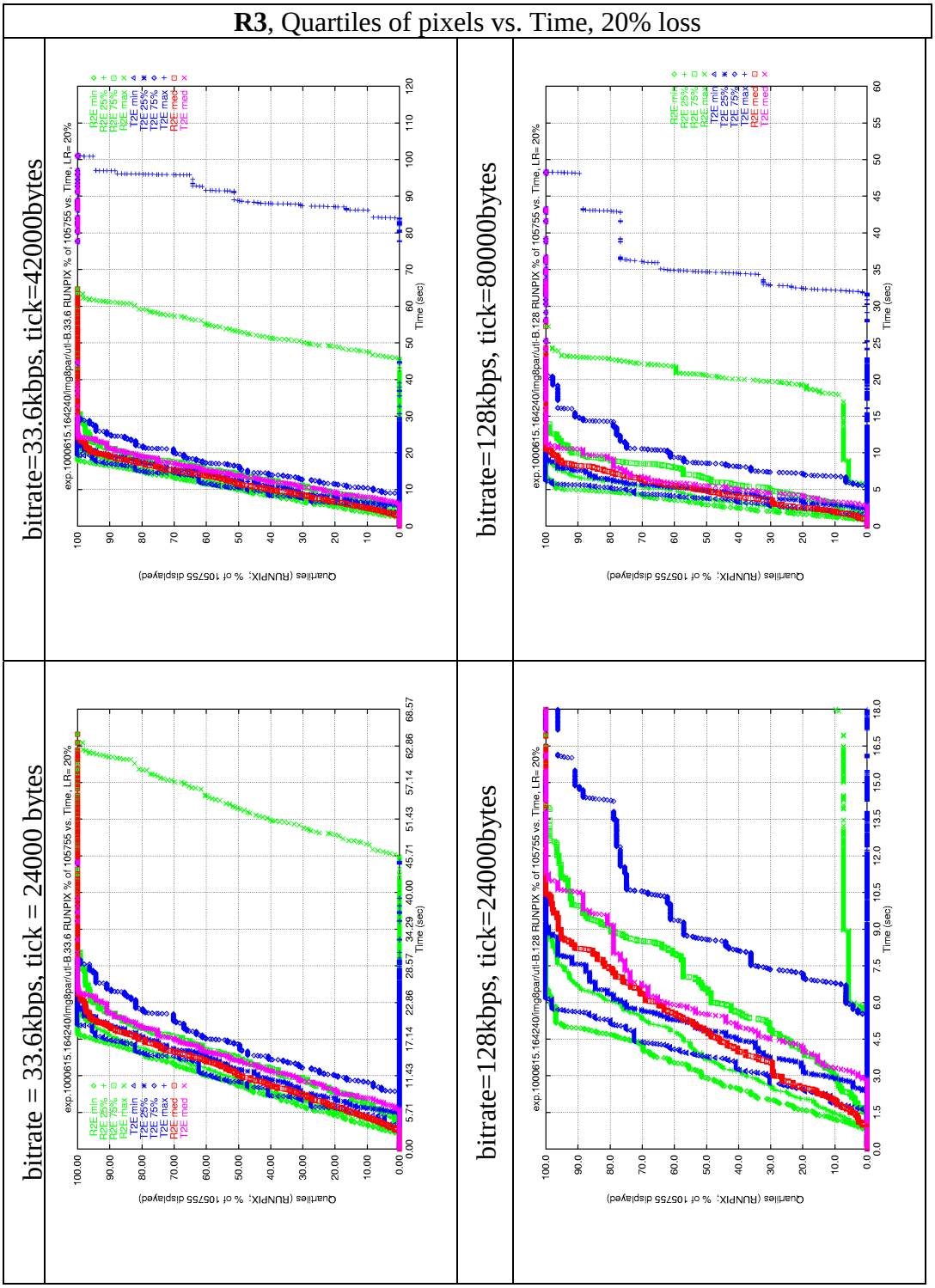


Figure 5.30 Experiment R3: Scaled Quartiles Performance Graphs

The graphs in Figure 5.30 are scaled versions of the 33.6kbps and 128kbps graphs from Figure 5.29.

5.6 Experiment R4: images in parallel with audio (O/R vs. PO/R for images and audio from `paris.pmsl`)

This section describes Experiment **R4** which extends the investigation of ReMDoR performance to documents that include audio. For this experiment, we retrieve a document called `parisMap0.pmsl`, which contains three GIF images and an audio clip (taken from `paris.pmsl`), all presented in parallel. In this experiment, we measure performance statistics that capture both the progressive display of the images and the smoothness of the audio presentation. The audio encoding used is the standard SUN`.au` format (8Khz μ-law), which requires 64Kbps of throughput. Because our PPP link provides a maximum throughput of less than 38.4Kbps, we use the reflector to investigate bitrates ranging from 80Kbps to 256Kbps, with 128Kbps as the nominal target bitrate.

Section 5.6.1 provides background concerning this experiment, including a description of our metrics for audio performance. Section 5.6.2 describes the parameters for Experiment **R4**, and our hypotheses about the results. Section 5.6.3 describes our results and conclusions.

5.6.1 Background: Three proposed metrics for audio performance

When audio is streamed over a network with a fully reliable service, the goal is to ensure that the audio device never underflows, because underflows introduce interruptions during playout. The method typically used is described in (Dempsey, 1994; Dempsey et al., 1996.). A small initial playout delay is introduced, during which a queue is allowed to accumulate packets. Once audio playout begins, this queue is drained at a constant rate—for example, at 64Kbps in the case of the 8Khz μ-law encoding used in the ReMDoR system. Since the service rate is constant, we can

measure the queue length in seconds rather than in bits. The length of the queue in seconds determines how much time is available for the transport layer to achieve a retransmission of any missing packets.

If the queue length is too short, using a fully reliable service with audio has the potential to introduce defects in the form of *interruptions* when the audio queue underflows. Interruptions can be measured in two ways: the number of interruptions that occur, and the length of each interruption.

Suppose the network delay is constant, or has relatively low variance (say, a standard deviation of 5% of the mean value). In this case, a fixed playout delay of, say, twice the round-trip time should provide enough time for a single retransmission of a missing packet. However, several factors may cause a reliable service to underflow:

- The network delay may vary, causing the playout delay to be too small for even a single retransmission.
- The loss rate may be sufficiently high that multiple retransmissions are required.
- Retransmissions may rob bandwidth from original transmissions, causing the source to be unable to provide packets fast enough.

We would expect that for the case where the audio stream is transmitted in parallel with other streams (e.g., image data) that PO/R service would result in fewer underflows than O/R service. This is because missing packets in the non-audio streams will impact the delivery of the audio stream for O/R service, while with PO/R service, only missing audio packets would cause underflows. Therefore, we would like to measure the impact of underflows on audio quality, to assess the performance improvement offered by PO/R service.

There is a complicating factor, however. We can measure audio underflows in several ways, and it is unclear which way correlates best with perceived end-user audio quality. For example, suppose a user is listening to an excerpt from the French National Anthem lasting approximately 120 seconds. Which scenario would that user prefer?

(a) a single audio interruption of 3 seconds, occurring right in the middle of the piece

(b) 9 interruptions of one-third of a second each, uniformly distributed across the 120 seconds

(c) 3000 interruptions of 1 millisecond each, occurring every 40 ms (that is, between every single audio packet)?

The author's anecdotal experience is that scenario (c) is perceived only as a slowing in the tempo of the music, and for some listeners may be the least objectionable defect. On the other hand, (a) is probably much less annoying than (b), since with (a) once the defect has passed, it is easily forgotten, while with (b), there is a constant reminder of noticeable problems.

This simple example illustrates that, when considered in isolation, neither the number of interruptions, nor the total duration of the interruptions (which is the same for all three cases above) nor the mean duration of the interruptions is necessarily a good indicator of the impact on quality.

Further, the impact on perceived quality of various kinds of defects will vary among listeners. The impact may also depend on the media content, and the media purpose. A user retrieving a clip solely for entertainment purposes may be intolerant of even slight defects, and may give up on the transmission altogether rather than listen to less than perfect playback. On the other hand, a student replaying a

lecture the night before an exam, or a soldier retrieving useful intelligence information in a hostile environment may prefer a clip with fewer defects, but may nevertheless be grateful for any information at all.

Because of the subjective nature of perceived audio quality, a subjective metric called the Mean Opinion Score (MOS) has often been used. The MOS metric has frequently been applied to investigate defects in audio quality introduced by distortion resulting from A-D or D-A conversion, quantization, and lossy compression schemes. However, we are not aware of previous work that assigns Mean Opinion Scores to reliable playback of audio with interruptions. Such a study would be useful as future work, but is out-of-scope for this dissertation.

Therefore, pending the outcome of such a study, this dissertation introduces three objective metrics for defects introduced by interruptions of reliable audio streams: The purpose of these metrics is to compare the difference between using O/R and PO/R service for documents containing audio.

1) INT (Absolute number of interruptions)

Zero interruptions represents perfect playback. The more interruptions there are, the worse the performance. While this is a useful metric, it does not capture all the information we might find useful. In particular, it would assign a much worse metric to a playout with 10 barely perceptible (or possibly imperceptible) interruptions of 1 millisecond, than it would to a playout with five interruptions of 3 seconds each, which might be more annoying. This fact motivates the next metric.

2) FRACPLAY (Fraction Playing).

FRACPLAY is defined as the fraction of time during the playout of the audio that the user is actually hearing the audio playing, as opposed to hearing the silence of an interruption.

$$\text{FRACPLAY} = \frac{\text{playing time of clip}}{\text{playing time of clip} + \text{duration of audio interruptions}}$$

If there are no interruptions, then FRACPLAY = 1; this represents perfect playout. But if, for example, a 9-second clip is interrupted once for 1 second, than the metric would be 9/10, since the total playout time will now be 10 seconds.

The FRACPLAY metric makes a useful distinction between an 18 second clip interrupted twice for one second each, and an 18 second clip interrupted 10 times for .001 seconds each time, we assume that users will notice the former, and barely notice (or be altogether unaware of) the latter.

However, the FRACPLAY metric also fails to capture exactly what we might want. It does not distinguish between a 18 second clip interrupted once for 2 seconds, or the same 18 second clip interrupted 10 times for 0.2 seconds each time. We assume[60] that most users would find the second case more annoying.

The need to capture both the number of interruptions and the length of the interruptions motivates the third metric:

[60] Testing the validity of this assumption is outside the scope of this dissertation; human subject research in this area is suggested as future work.

3) FRACPLAY^{INT}

To capture both the influence of the number of interruptions as well as the size of the interruptions, we propose the metric FRACPLAY^{INT} (signifying FRACPLAY raised to the INT power). The intuition behind this formula is that each time there is an interruption, there is a cumulative effect on the degree to which the user is annoyed; i.e., we suggest that *annoyance multiplies*. As with the INT and FRACPLAY metrics, we can assert that a value 1 represents perfect performance, and that interruptions will cause the value to tend towards zero.

5.6.2 Experiment R4: parameters and hypotheses

Table 5.9 presents our parameters for the Experiments that make up **R4**; as in the previous experiments, we investigate the effects of loss rate, bit rate, propagation delay, and window size on transmission of a single document, `parisMap0.pmsl`. This document consists of three images in parallel with a short audio stream lasting several seconds.

Table 5.9 Parameters for Experiment R4

	Values for Experiment:			
Parameter	***R4.1***	***R4.2***	***R4.3***	***R4.4***
Mechanisms	← **R2E,T2E** →			
Loss Rates (%)	**0,10,20**	**0,20**	0,20	10
Network	← reflector →			
Bitrate (Kbps)	128	128	**80, 96, 128, 256**	128
One-way delay (ms)	250	**0,125,250, 500**	250	250
Document	`parisMap0.pmsl`			
Window Size (pkts)	128	128	128	**64, 128, 256**

Bold indicates the parameters that are the focus of the experiment

Our hypotheses about these experiments are as follows[61]

Hypothesis 5.6.1 No difference at 0% loss: There will be no significant gain or penalty for using PO/R service vs. O/R service at 0% loss from the standpoint of (a) progressive display of pixels,(b) or in any of the audio metrics. (all experiments)

Hypothesis 5.6.2 Better graceful degradation of progressive display of pixels: At loss rates greater than zero, progressive display of pixels will be better when PO/R service used rather than O/R service. (all experiments)

Hypothesis 5.6.3 Gain increases with loss rate: In terms of progressive display of pixels, there will be increasing performance gains from using PO/R service vs. O/R service at 20% loss vs 10% loss. (Exp. **R4.1**)

Hypothesis 5.6.4 Better graceful degradation of audio: At loss rates greater than zero, all three audio metrics will degrade more slowly when PO/R service is used rather than O/R service.

Hypothesis 5.6.5 Gain increases with delay: With respect to the improvements in progressive display of pixels, or audio performance, there will be less gain from PO/R service vs. O/R service at lower round-trip delays, and more gain at higher round-trip delays. (Experiment **R4.2**)

Hypothesis 5.6.6 More robust when bit rate is too low. As the bit rate is decreased from values that can support the audio stream in the presence of loss to values that are insufficient to support the audio stream in the presence of loss, PO/R service will degrade less rapidly than O/R service in terms of both (a) progressive display of pixels, and (b) audio. (Experiment **R4.3**)

Hypothesis 5.6.7 More robust in face of inappropriately large windows. As the window size is increased from values near the bandwidth-delay product to values above the bandwidth-delay product,

[61] The titles in bold such as "No difference at 0%loss" refer to the benefits of PO/R service over O/R service captured in each hypothesis.

performance will degrade for both PO/R and O/R service, but it will degrade more for O/R service than for PO/R service. (Exp. **R4.4**)

As the next section shows, our results find clear evidence some of these hypotheses, while the results are mixed for others. In general, the beneficial impact of PO/R service vs. O/R service on the progressive display of pixels that was observed in Experiments **R1**-**R3** is maintained when audio is introduced in parallel. In some cases, PO/R service provides better audio performance than O/R service. In other cases, however, there is no clear evidence that PO/R service provides better quality than O/R service for the audio portion of the document. At the same time, neither is there any evidence that PO/R service significantly impairs the audio quality when compared to O/R service.

5.6.3 Experiment R4.1: observations and conclusions

In this experiment, we make observations about both pixels, and audio performance. Figure 5.31 shows performance graphs similar to the ones that were presented for Experiments **R1–R3;** each graph shows the median (across all experiments) of the number of bytes or pixels presented at each point in time. Figures 5.32 through 5.34 show performance graphs for audio based on the metrics described in Section 5.6.1.

Observations and conclusions for pixels and bytes

We make the following observations concerning these graphs:

(1) At 0% loss, there is little difference between the bytes and pixels graphs for PO/R service vs. O/R service. This offers support for Hypothesis 5.6.1(a).

(2) The pixels graphs for the 0% loss case has a sharp bend upwards towards the end.

Observation (2) is explained by the fact that at this point in the document, the transmission of audio is finished. Audio is given preferential treatment in the linear extension selection algorithm (see Chapter 6.)

(3) For bytes, the gain of PO/R service over O/R service is rather modest, but when this is translated into pixels, the gain is more impressive.

(4) The fact that there is a gain for PO/R service, and that the gain increases for 20% vs. 10% loss provides evidence to support Hypotheses 5.6.2 and 5.6.3

Observation (3) is worth further exploration for two reasons: (1) it is counterintuitive: how can there be an advantage in pixels when there is no advantage in bytes? (2) We will see this pattern in Experiments **R4.2**, **R4.3** and **R4.4** as well.

In the BYTES graph, the advantage of PO/R service appears rather modest, with the O/R service frequently showing identical or even slightly better progressive display. Meanwhile, on the PIXELS graph, the PO/R service seems to be the clear winner. For example, for 10% loss, at time 11.3 seconds, PO/R service displays 36850 pixels in the median case (80% of the total pixels) while O/R service presents only 11962 pixels (26%). It takes O/R service 5.1 additional seconds to deliver 80% of the pixels. The gain for 20% loss, as predicted, is larger: in this case, PO/R service displays 36889 pixels in the median case (80%) at time 13.8 seconds, while O/R service displays only 7967 pixels (17%) after 13.8 seconds. In the 20% loss case, O/R service requires an additional 16.2 seconds to deliver 80% of the pixels.

The explanation for the discrepancy between the BYTES and PIXELS results lies in the fact that the BYTES metric includes both audio and image data, while the PIXELS metric focuses only on image data. When ordered service is used, it is necessary to hold back the presentation of image data because of out-of-sequence

audio data, and vice-versa. With PO/R service, the two kinds of data can be interleaved. As a result, when PO/R service, image data can overtake the audio data, and get presented earlier, while with O/R service, this is impossible.

There is clear evidence in the graphs that pixels are in fact overtaking audio when PO/R service is used, as compared to the fixed linear extension enforced by O/R service. To see this, one can consider, for both PO/R and O/R service, the point at which 50% of the total bytes have been displayed vs. the point at which 50% of the total pixels have been displayed. For PO/R, the point at which 50% of the pixels are displayed occurs before the point at which 50% of the bytes are displayed, while for O/R service, the opposite is trued. Similar observations can be made at other percentages.

Interpreting the graphs for the audio metrics

The meaning of the CDF graphs for the audio metrics may not be immediately intuitively obvious. This section provides some general help in interpreting the results in these graphs. It is generally helpful to compare the CDF for each loss rate with the CDF for 0% loss, and ask the question: how is perfect playback represented?

For the INT metric, the ideal case is that the CDF goes immediately to 1 when the *x*-axis is at 0 interruptions. As the *x*-axis value increases, representing an increasing number of audio interruptions:

- If the CDF rises *slowly* to one, this indicates that there were a *large* number of audio interruptions.
- If the CDF rises *quickly* to one, this indicates that there were *fewer* audio interruptions.

Thus, for the INT metric, the transport service represented by the *higher* curve is the one that provides superior audio quality.

For the FRACPLAY and FRACPLAY^{INT} metrics, the ideal case is that the CDF remains at zero until the *x*-axis reaches 1, then it jumps to 1, indicating that 100% of the experiments had perfect playback (i.e., the entire duration of the audio consists of audio playback.) If the CDF moves above zero any earlier than when the *x*-axis reaches 1, this indicates that some portion of the experiments experiences sub-optimal playback. Thus, the longer the CDF remains low, the better, and the transport service represented by the *lower* curve is the one with the superior playback.

Observations and conclusions for audio metrics

(5) Audio performance is identical, and perfect at 0% loss. This offers support for Hypothesis 5.6.1(b).

(6) At 10% loss, PO/R clearly experiences fewer interruptions that O/R service. On the other hand, the fraction of time spent playing is slightly worse for PO/R service than for O/R service. When these results are combined using the FRACPLAY^{INT} metric, the combined metric assigns a higher quality score to the performance of PO/R service.

(7) The results for 20% loss are similar to those for 10% loss, with the exception that there is no clear winner between the two services in terms of the FRACPLAY metric.

(8) Comparing the results for 10% and 20% loss, we observe that the advantage of PO/R service over O/R service increases as the loss rate goes from 0% to 10%, but then diminishes as the loss rate goes from 10% to 20%.

Taken together, Observations (6) through (8) offer some evidence to support Hypothesis 5.6.6. It would be interesting to investigate, in both objective and human

factors studies, loss rates between 0% and 10%, and between 10% and 20% to determine the following:

- whether, as the loss rate increases from 0% to 20%, there is a trend of an increasing gap between PO/R and O/R service followed by a narrowing gap, and

- whether the gap is perceivable in terms of end-user quality, within a range where the quality is still usable/acceptable for some applications. It may be the case, for example, that above some loss rate, the performance of both services is considered by most end users to be equally unacceptable.

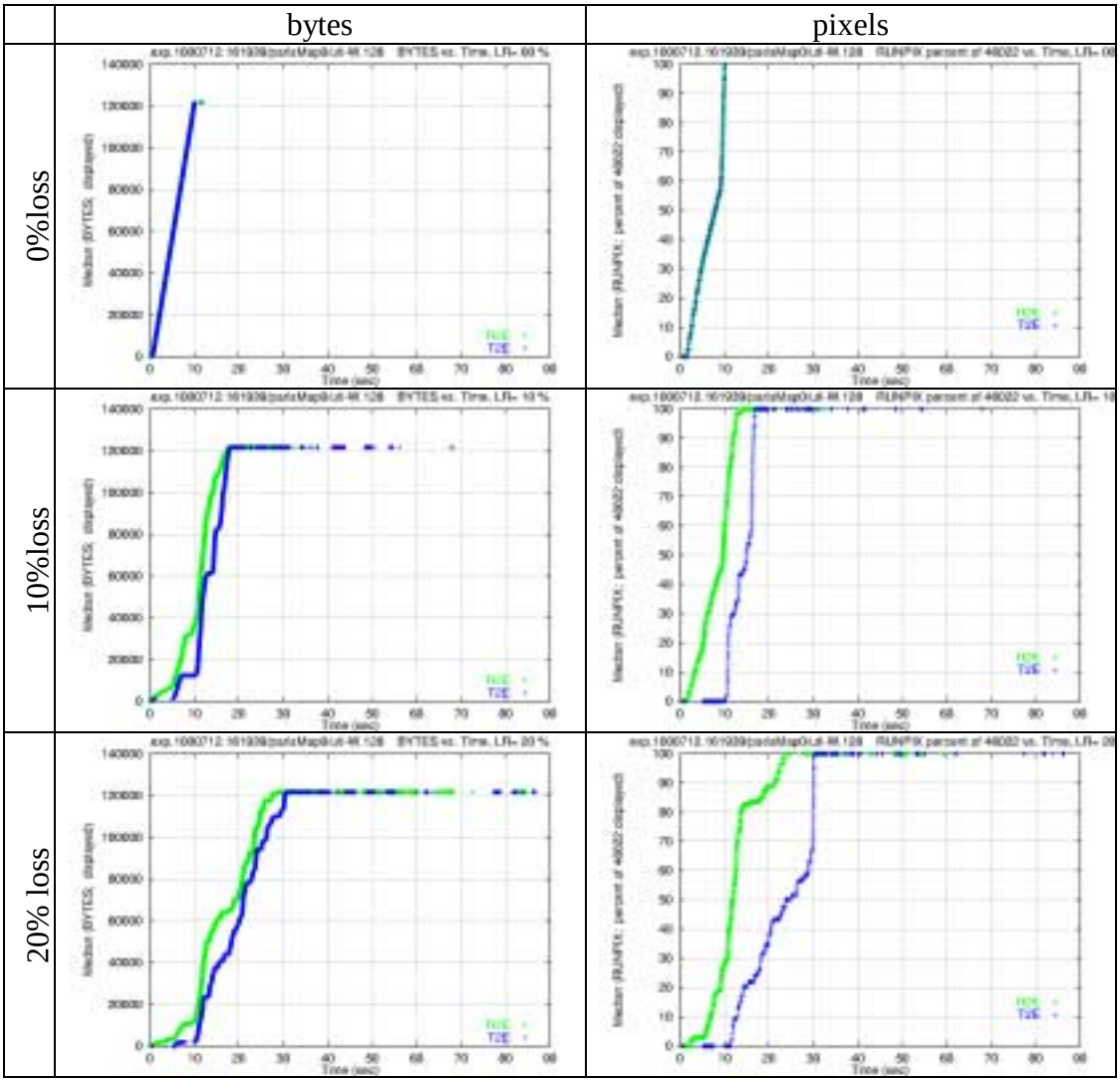


Figure 5.31 Exp. R4.1: bytes, pixels performance graphs

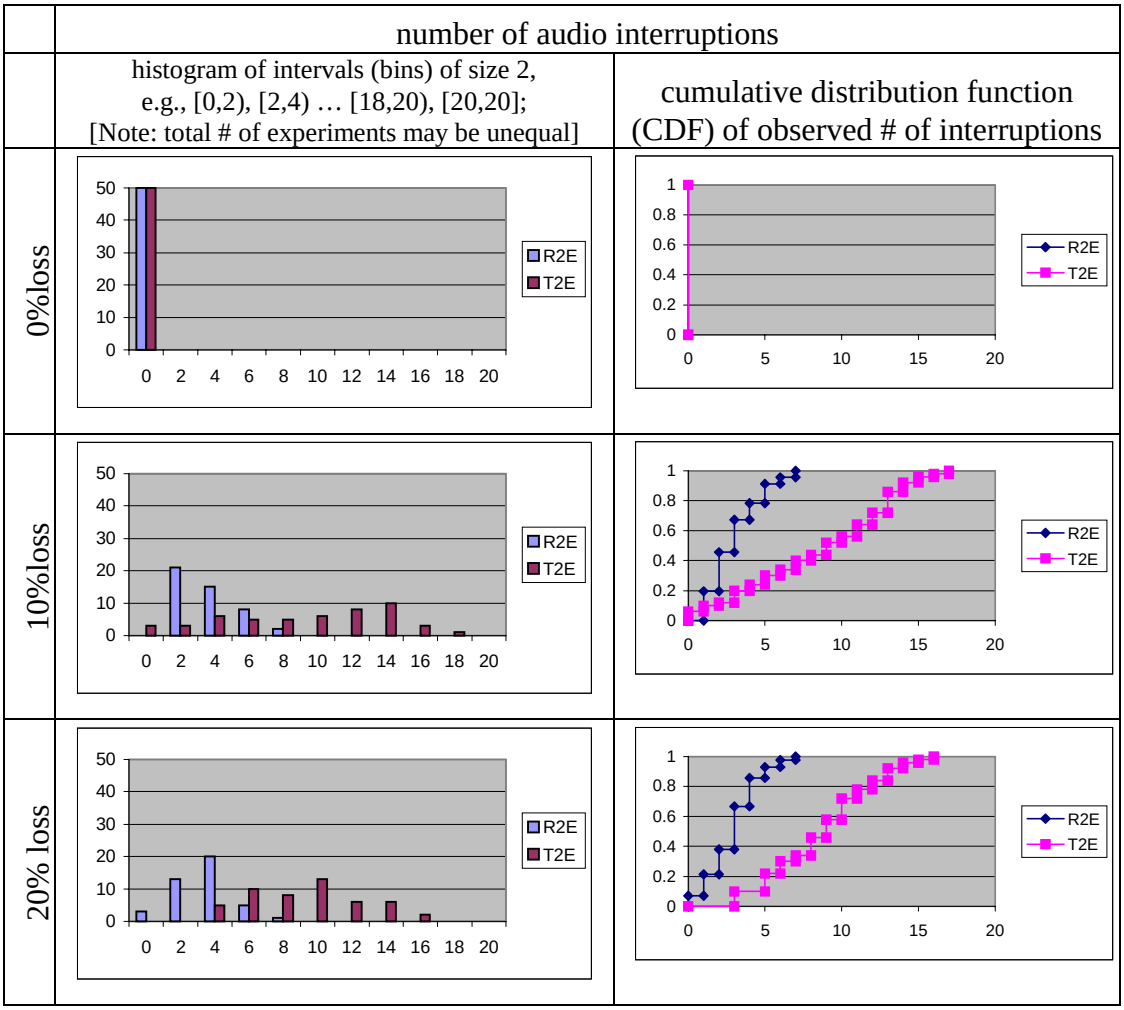


Figure 5.32 Experiment R4.1: audio interruptions

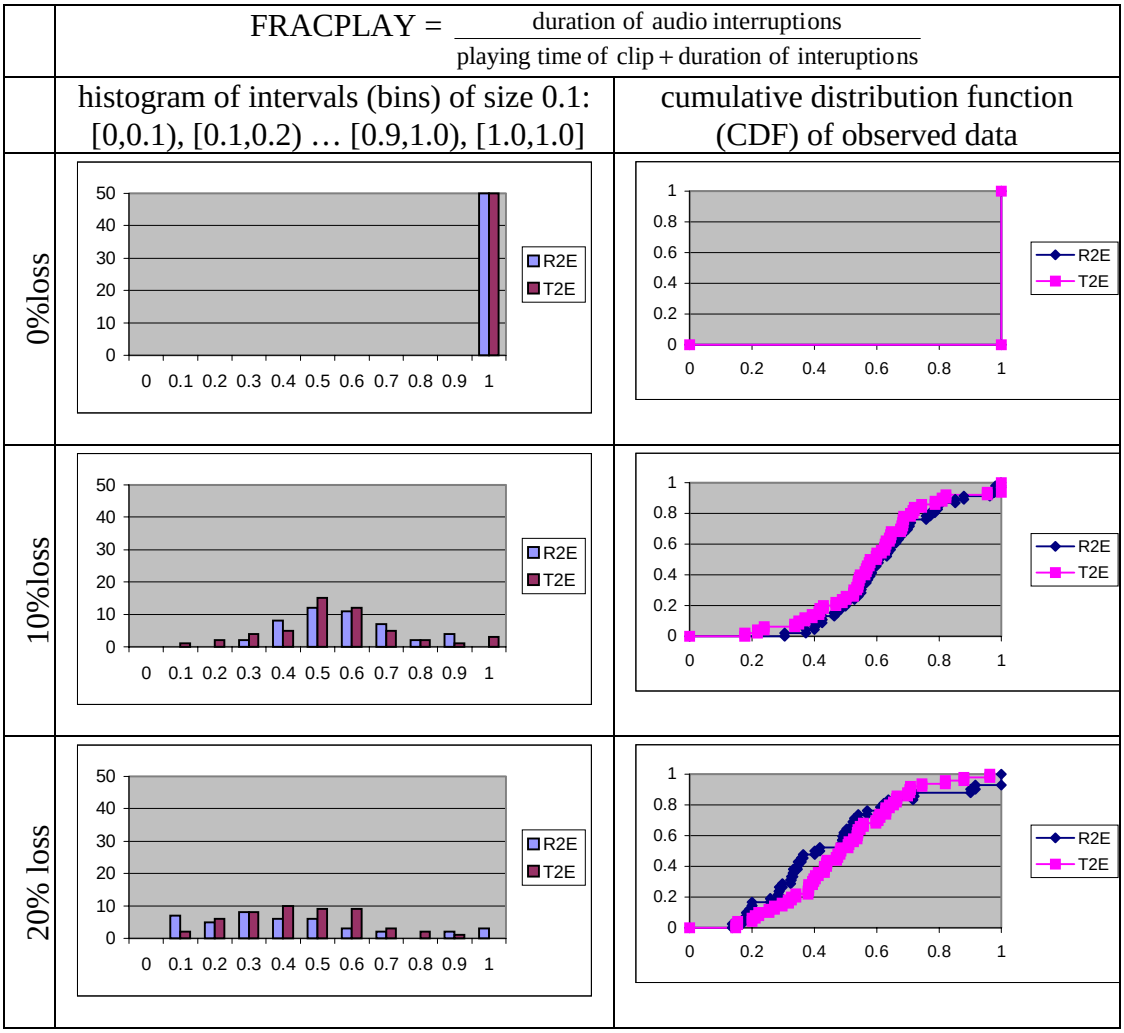


Figure 5.33 Experiment R4.1: FRACPLAY metric

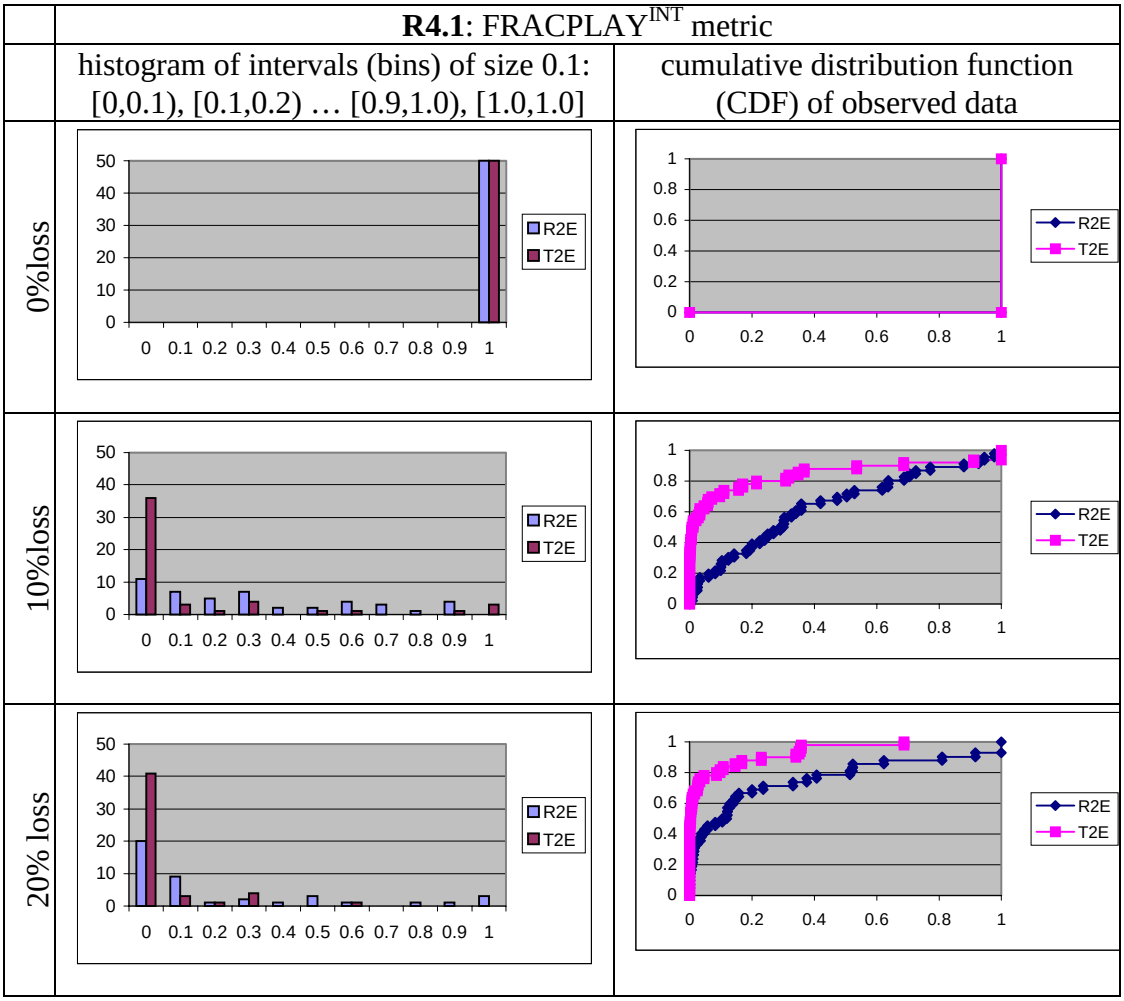


Figure 5.34 Experiment R4.1: FRACPLAY^{INT} metric

5.6.4 Experiment R4.2: observations and conclusions

Experiment **R4.2** investigates Hypothesis 5.6.5, which predicts that round-trip delay will have an impact on the gain in performance of PO/R service over O/R service. Round-trip delay is modeled by one-way delays in each direction through the reflector. The hypothesis states that with respect to the improvements in progressive display of pixels, or audio performance, there will be less gain from PO/R service vs. O/R service at lower round-trip delays, and more gain at higher round-trip delays. We make the following observations regarding Experiment **R4.2**:

(1) In general, the results are consistent with those of Experiment **R4.1** w.r.t. hypotheses 5.6.1 and 5.6.2; there is no significant difference between PO/R and O/R service for bytes, pixels, or audio metrics.

(2) In terms of progressive display of pixels, while at each round-trip delay, PO/R service is identical to O/R service at 0% loss, and outperforms O/R service at 20% loss,(further supporting hypotheses 5.6.1 and 5.6.2.) However, there is no significant difference among the performance graphs for bytes or pixels for the various round-trip delays chosen. Thus, there is no evidence to support Hypothesis 5.6.5

(3) At 20% loss, there is no clear winner between PO/R and O/R service for audio.

It might be interesting in future work to repeat this experiment at 10% loss, where there is more likelihood of a gap between the audio performance or PO/R and O/R service. However, the results of this experiment, taken together with those of Experiment **R2.3**, seem to provide little evidence that round-trip delay, per se, is a significant factor in determining the degree of benefit provided by PO/R service over O/R service (at least for the parameter ranges studied)

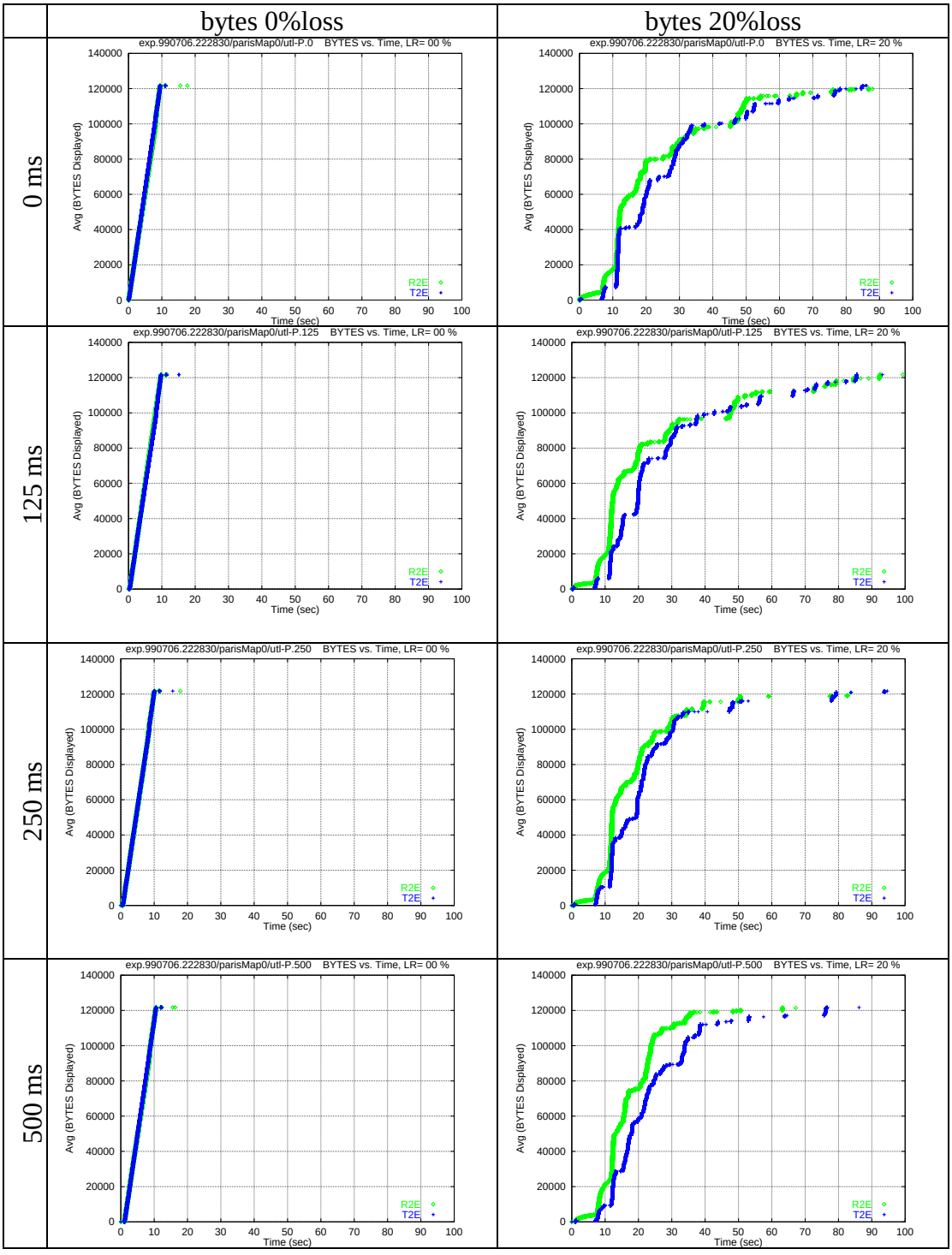


Figure 5.35 Experiment R4.2: bytes performance graphs

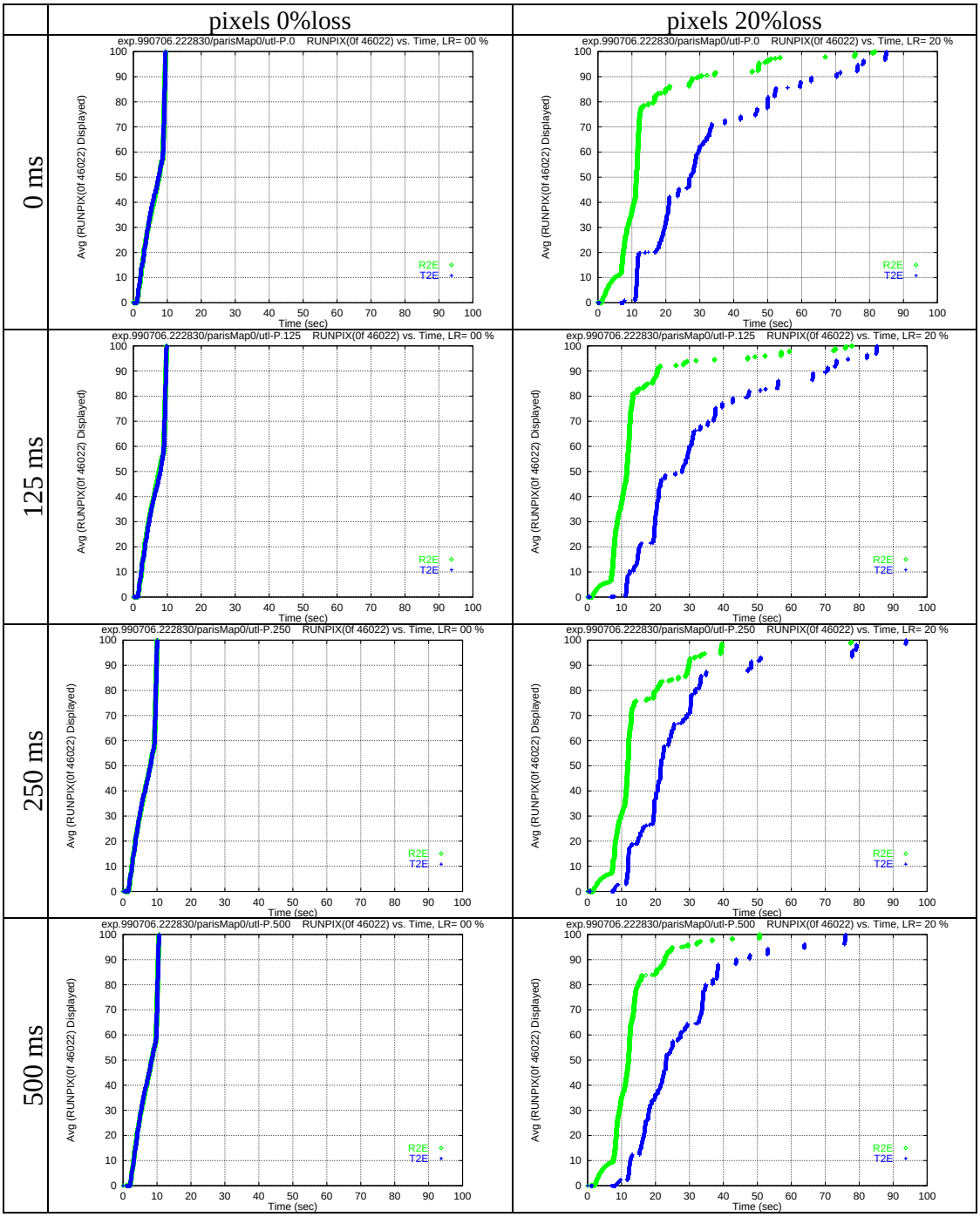


Figure 5.36 Experiment R4.2: pixel performance graphs

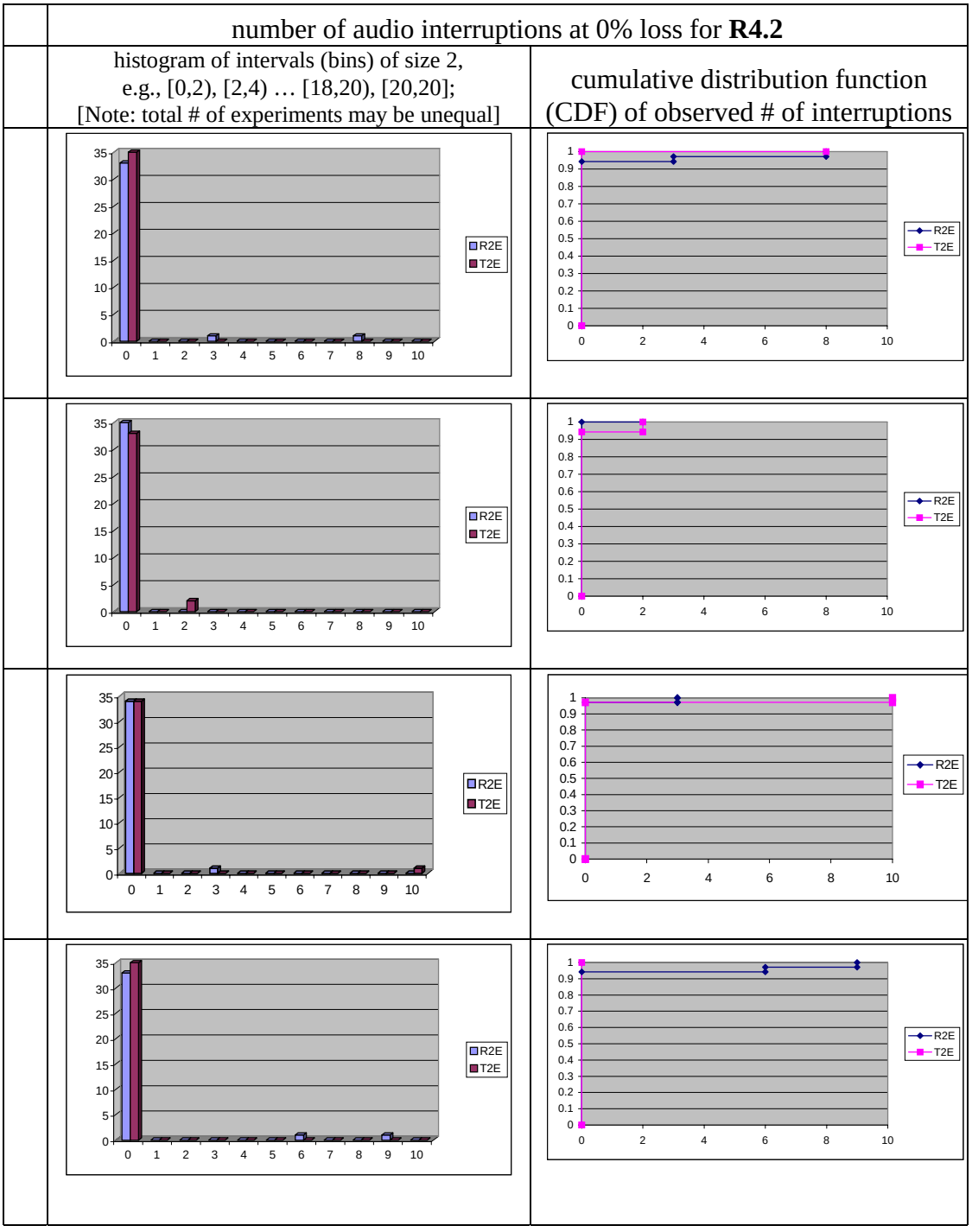


Figure 5.37 Experiment R4.2: audio interruptions at 0% loss

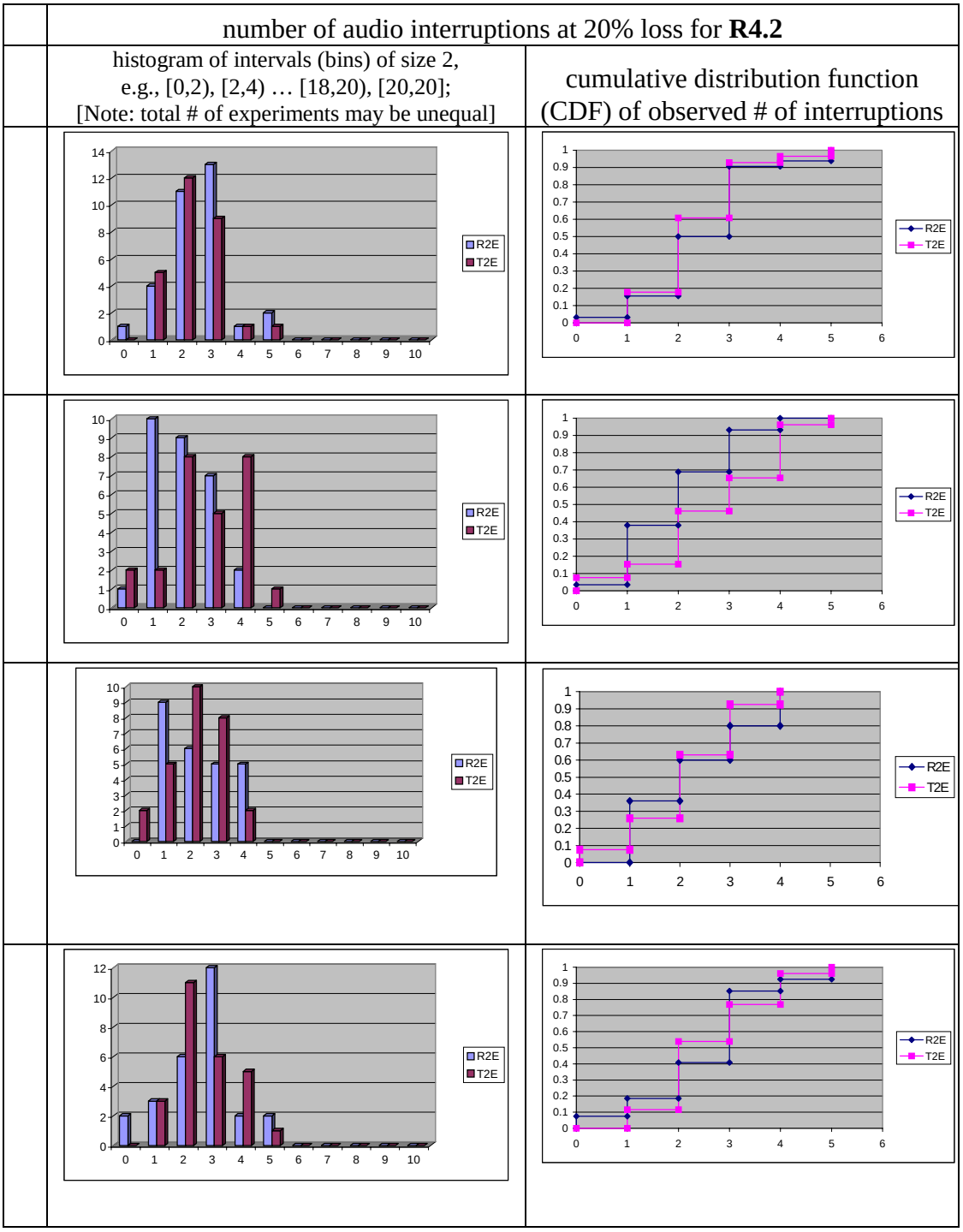


Figure 5.38 Experiment R4.2: audio interruptions at 20% loss

R4.2 at 0% loss, FRACPLAY metric = $\frac{\text{duration of audio interruptions}}{\text{playing time of clip} + \text{duration of interuptions}}$

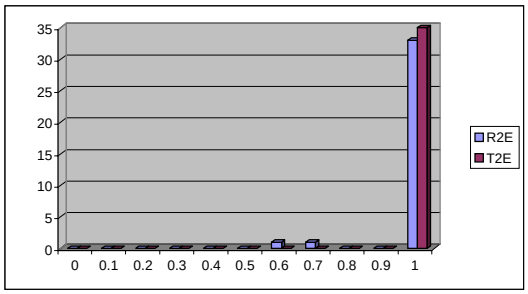
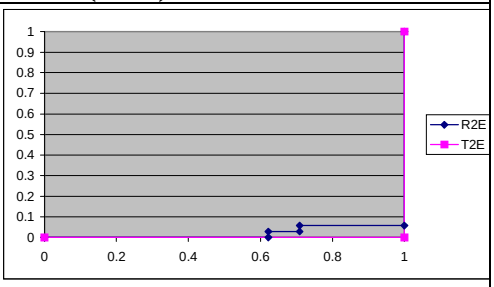
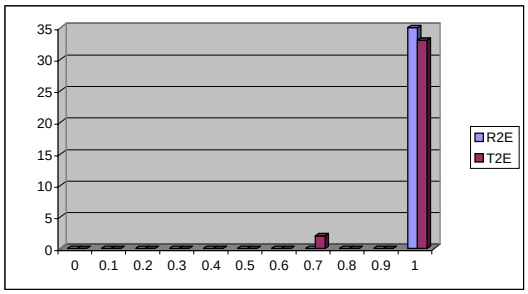
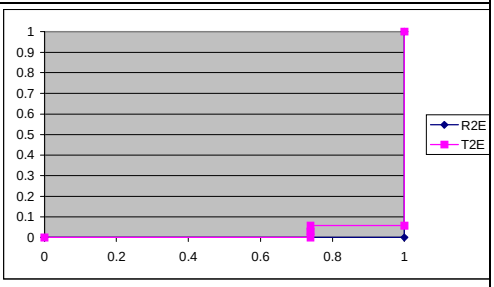
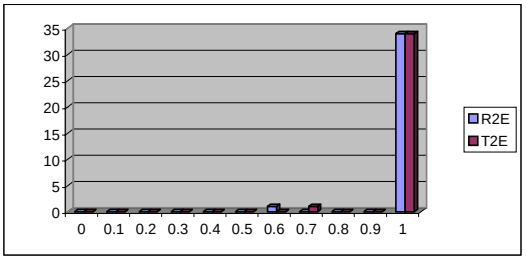
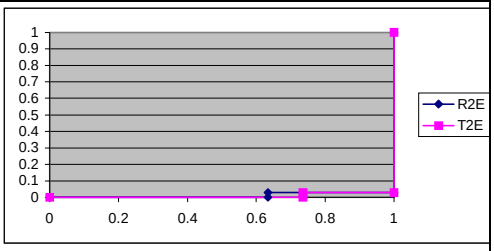
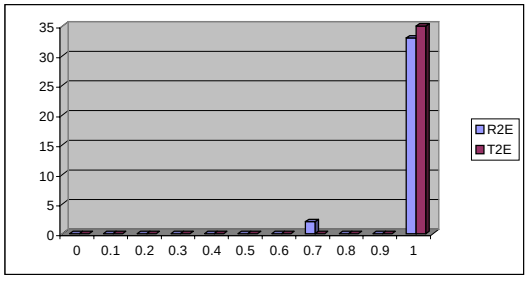
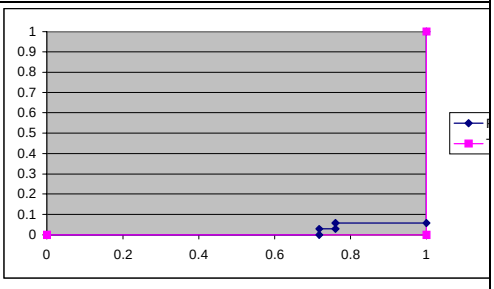
histogram of intervals (bins) of size 0.1: [0,0.1), [0.1,0.2) … [0.9,1.0), [1.0,1.0]	cumulative distribution function (CDF) of observed data
	
	
	
	

Figure 5.39 Experiment R4.2: FRACPLAY metric at 0% loss

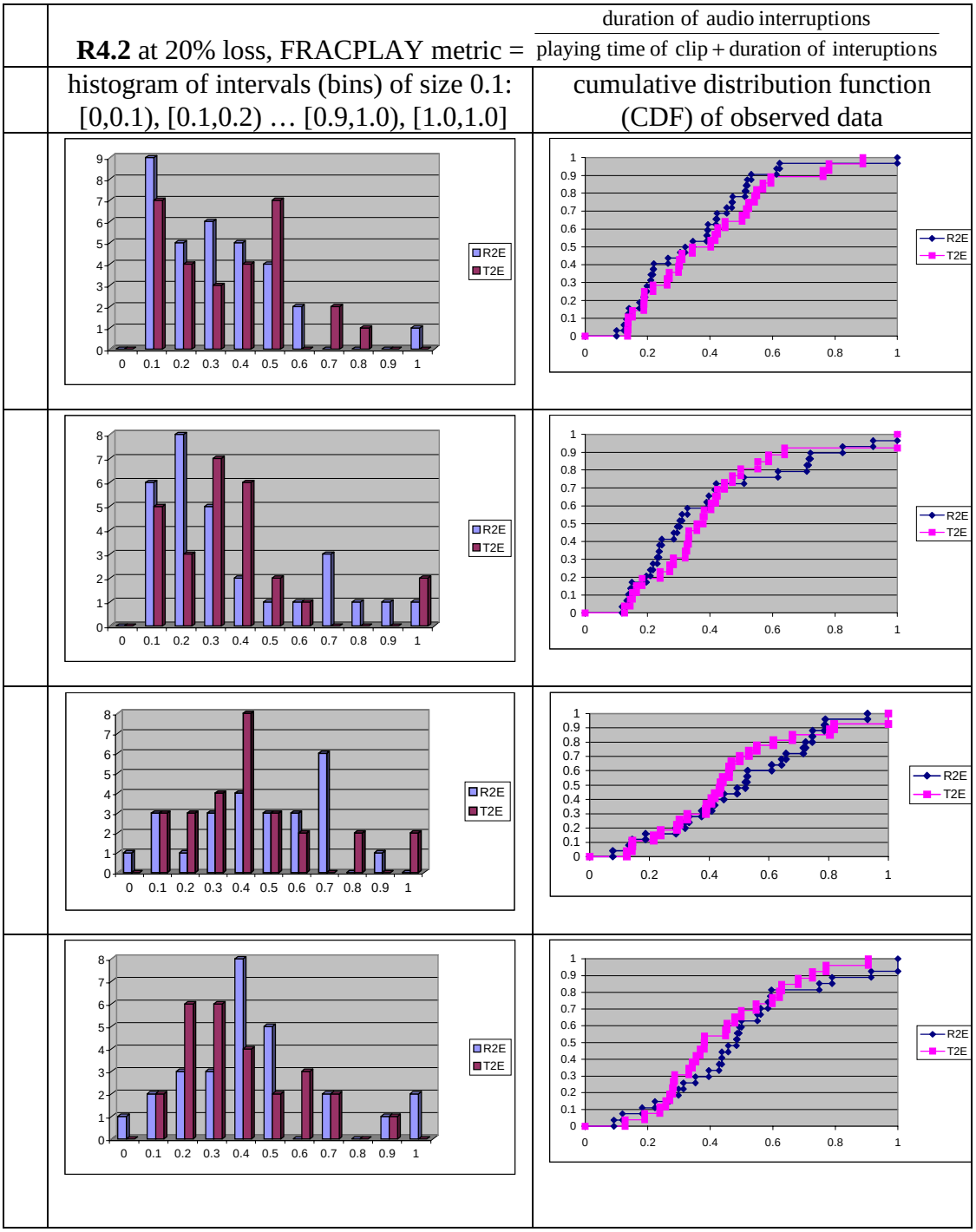


Figure 5.40 Experiment R4.2: FRACPLAY metric at 20% loss

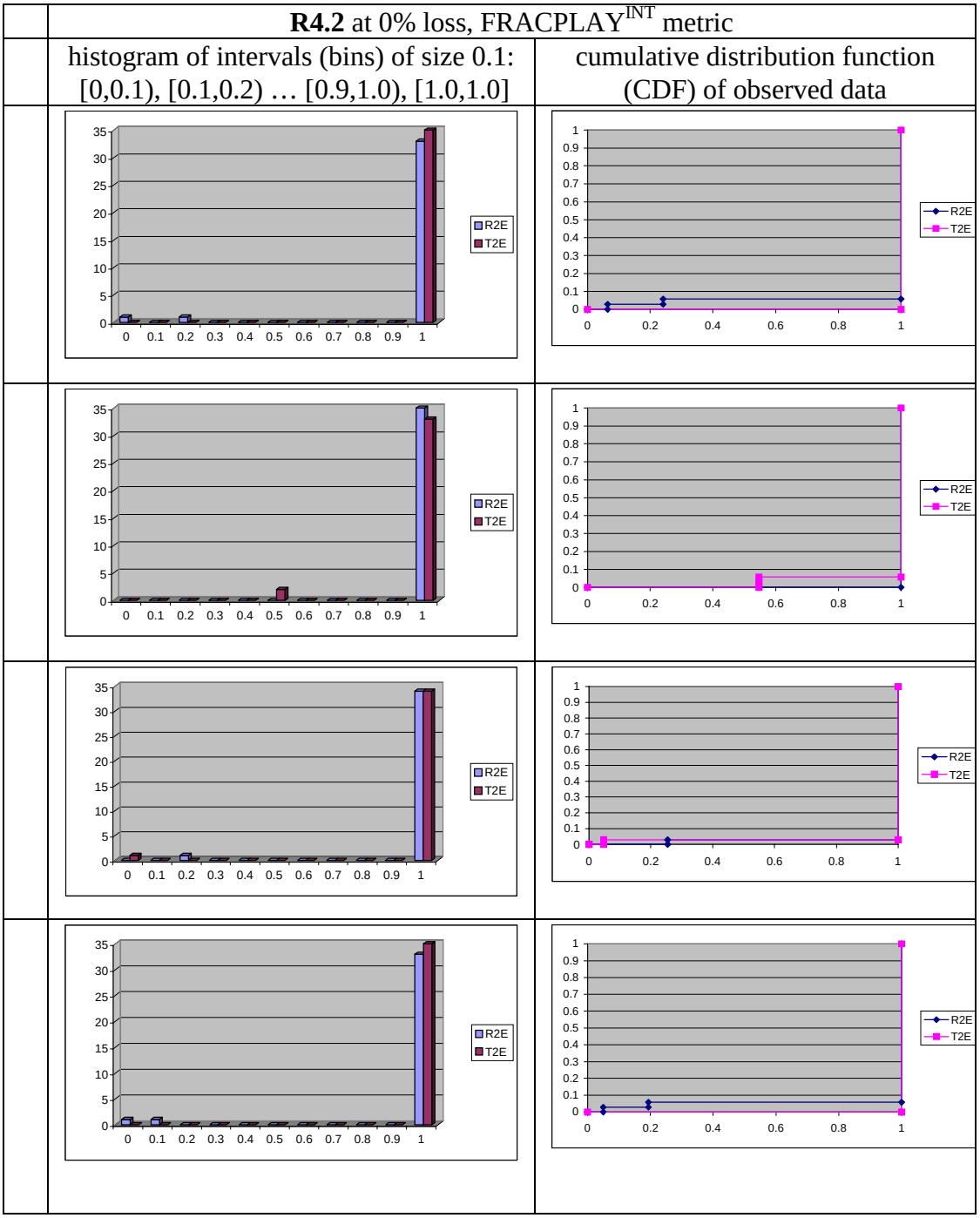


Figure 5.41 Exp. R4.2: FRACPLAY^{INT} metric at 0% loss

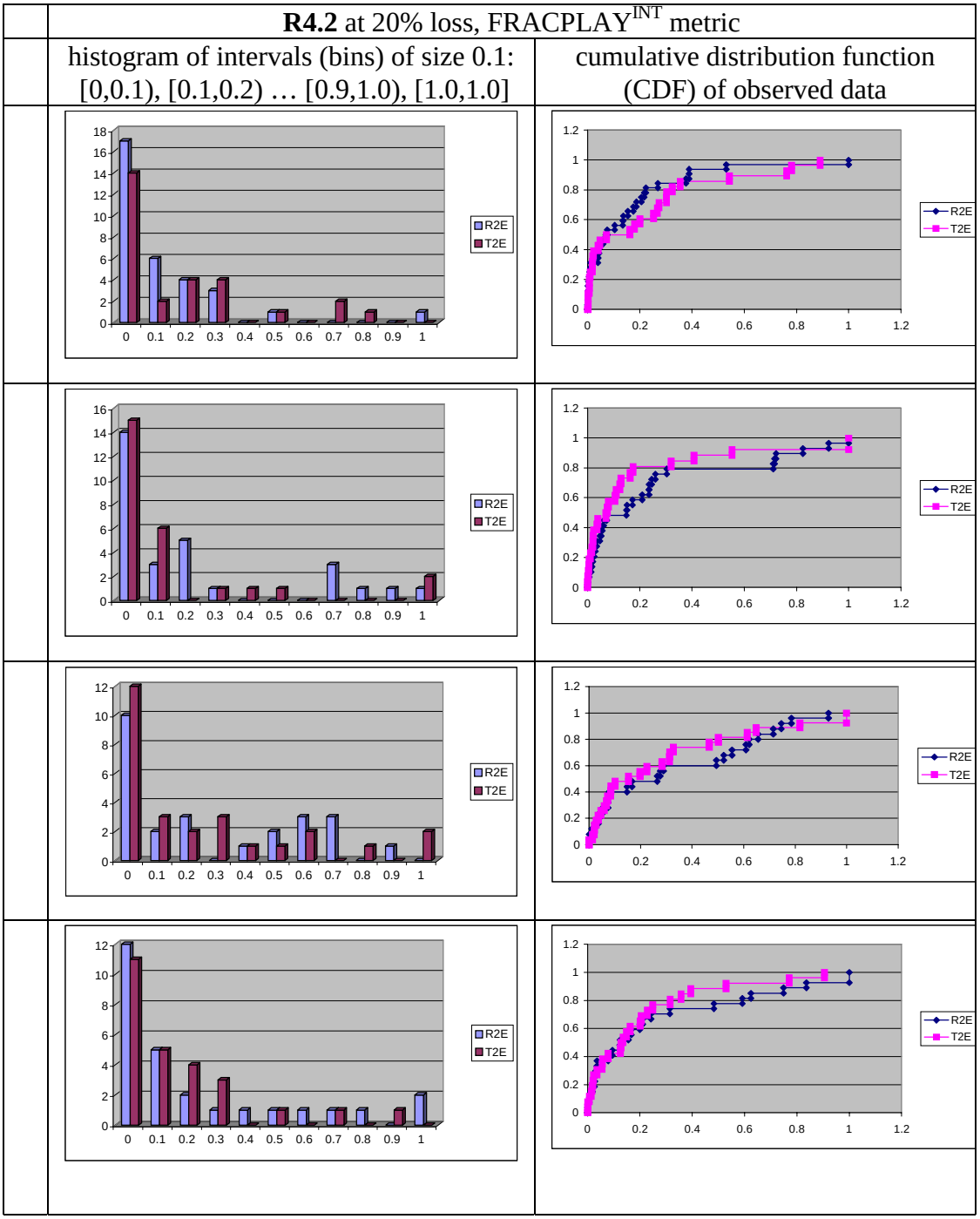


Figure 5.42 Exp. R4.2: FRACPLAY^{INT} metric at 20% loss

5.6.5 Experiment R4.3: observations and conclusions

Experiment **R4.3** looks at the effect of bitrate on performance of parallel audio and image streams. In this experiment, we perform a stress test by deliberately reducing the bit rate below that at which presentation of the document is feasible (because of the presence of the audio stream). We investigate the difference in robustness between PO/R service and O/R service. Hypothesis 5.6.6 predicts that as the bit rate is decreased, PO/R service will degrade less rapidly than O/R service in terms of both (a) progressive display of pixels, and (b) audio. We make the following observations regarding Experiment **R4.3**:

(1) In general, the results are consistent with those of Experiment **R4.1** w.r.t. hypotheses 5.6.1 and 5.6.2; there is no significant difference between PO/R and O/R service for bytes, pixels, or audio metrics, and when the loss rate is non-zero, PO/R service outperforms O/R service w.r.t. progressive display of pixels.

(2) At each bit rate, PO/R service provides better progressive display than O/R service in terms of pixels, providing evidence to support Hypothesis 5.6.6 part (a).

(3) At 128Kbps, which is the bit rate for which the document was optimized, at 20%loss, PO/R service provides better audio performance than O/R service for all three metrics. (This result is consistent with Experiment **R4.1**, supporting Hypothesis 5.6.4).

(4) As the bitrate decreases from the value for which the document was optimized (128Kbps), the gain of PO/R over O/R service at 20% loss vanishes.

A common sense observation about this data can be summarized by comparing PO/R service to an umbrella. In a hurricane, an umbrella is not of much use. However, in a light to moderate rain storm, an umbrella can be valuable indeed. Our data suggests that PO/R service can function analogously. If network conditions are *extremely* bad,

as represented by the 80Kbps and 96Kbps bit rates, PO/R service does not help much. Note that even at 0% loss, PO/R and O/R service are both providing far less than optimal audio performance. However, when network conditions are less than optimal, but not horrible, PO/R service improves performance considerably, as represented by the 128Kbps case.

It would be useful, as future work, to investigate bit rates that are less than 128Kbps, but closer to it, such as 120, 112 and 104Kbps to determine how the audio performance curve at 0% loss relates to the change in bandwidth. If the audio performance curve is perfect or near perfect for any value less than 128kbps, this might provide a more interesting value at which to further investigate Hypothesis 5.6.4.

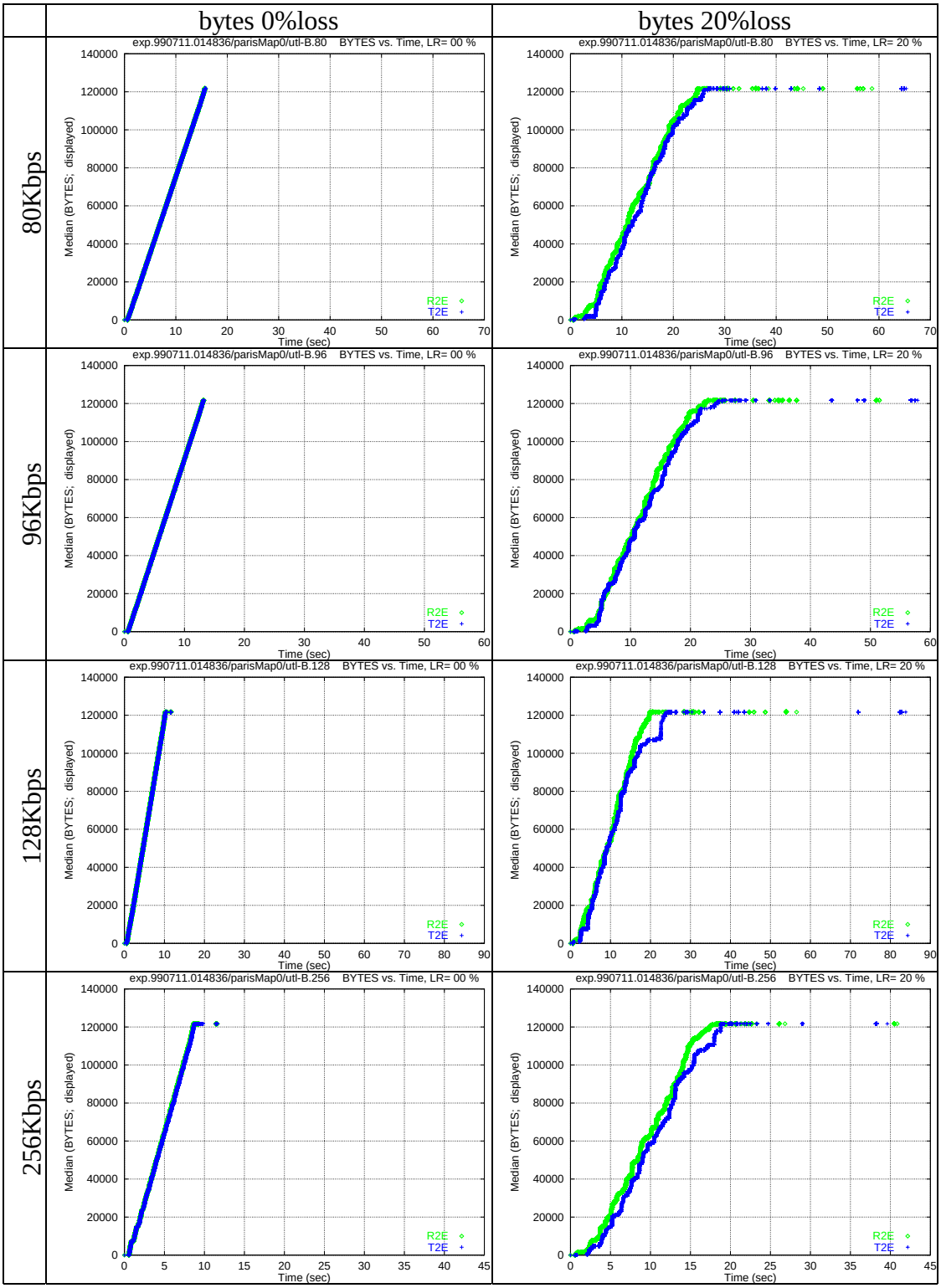


Figure 5.43 Experiment R4.3: bytes performance graphs

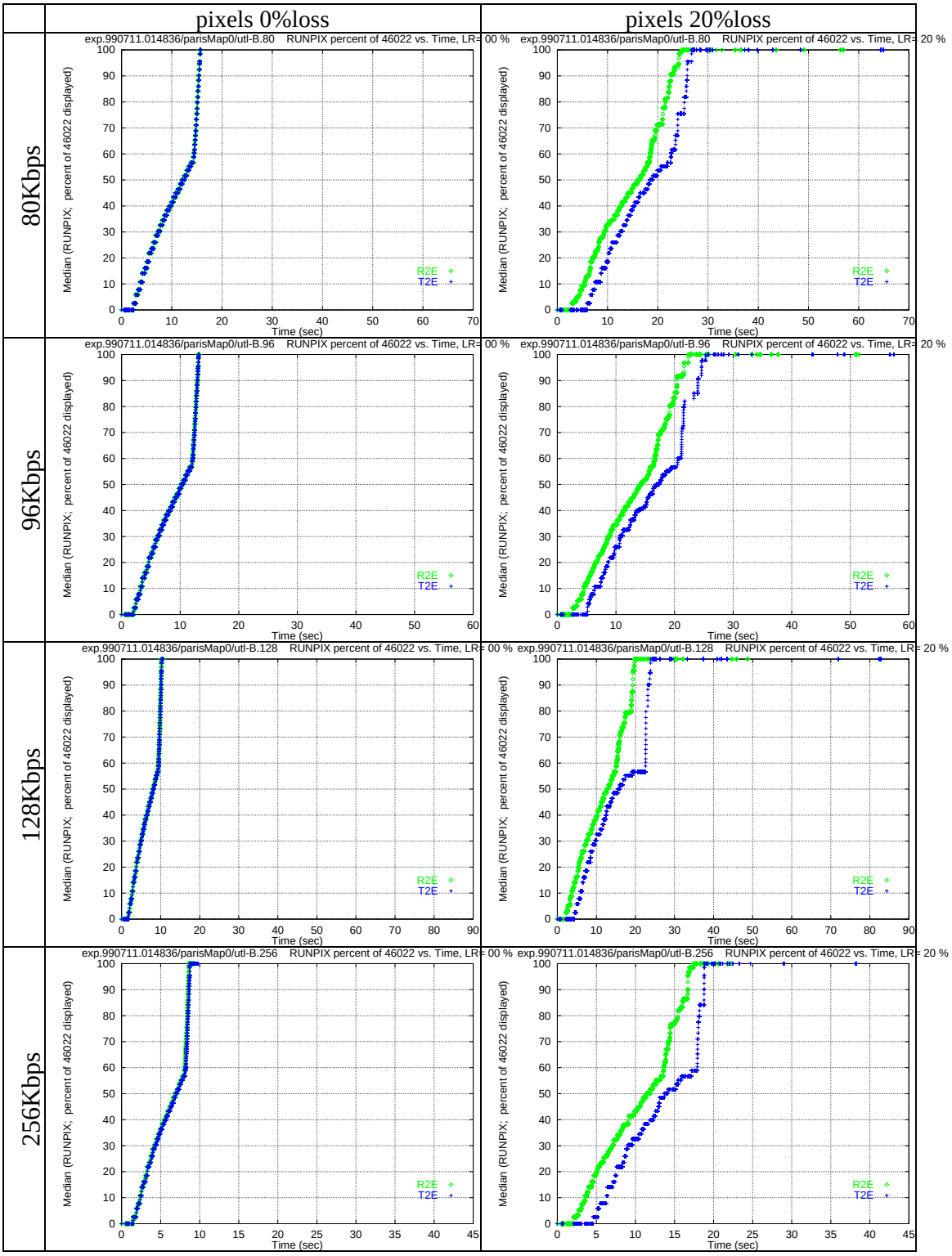


Figure 5.44 Experiment R4.3: pixel performance graphs

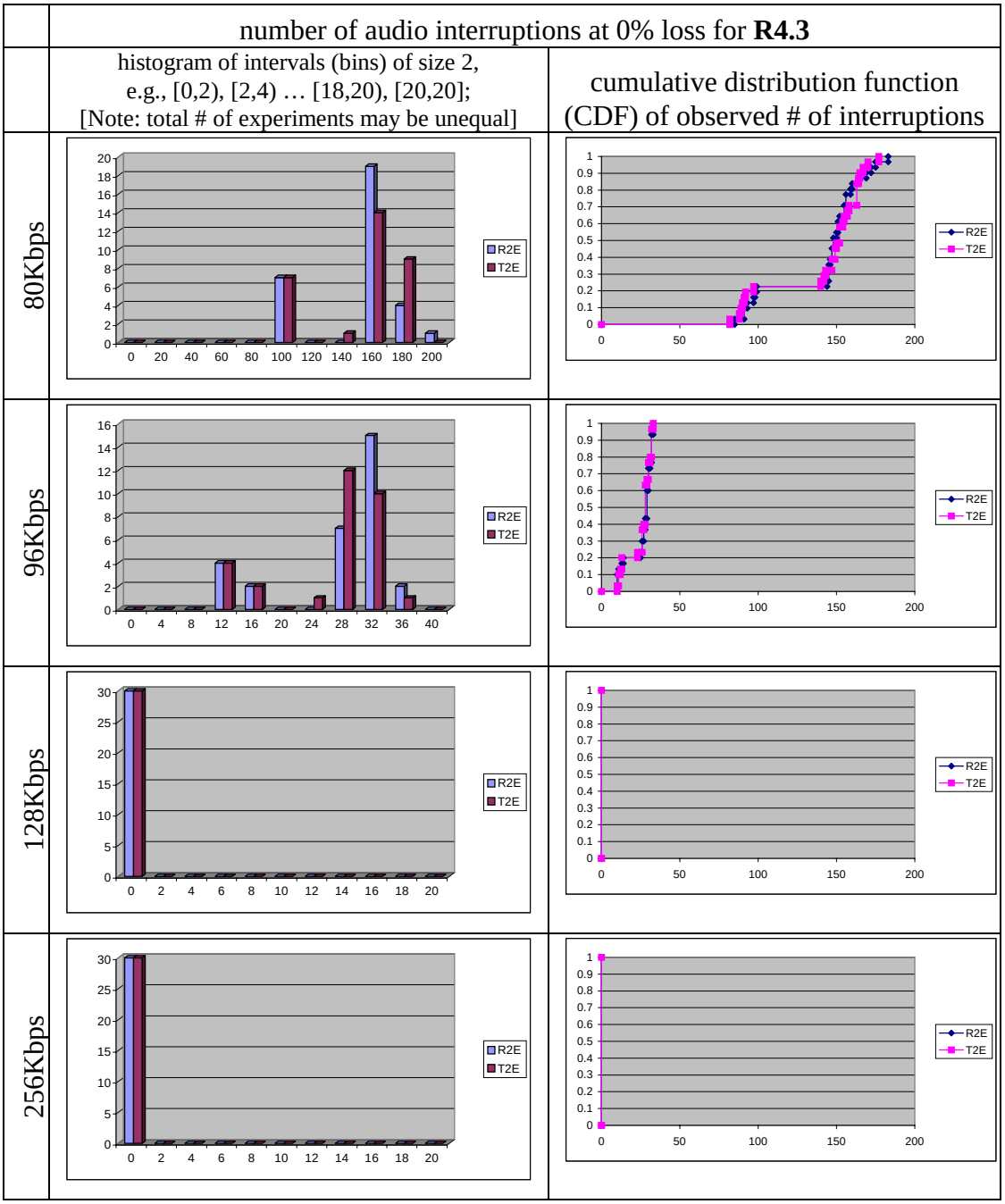


Figure 5.45 Experiment R4.3: audio interruptions at 0% loss

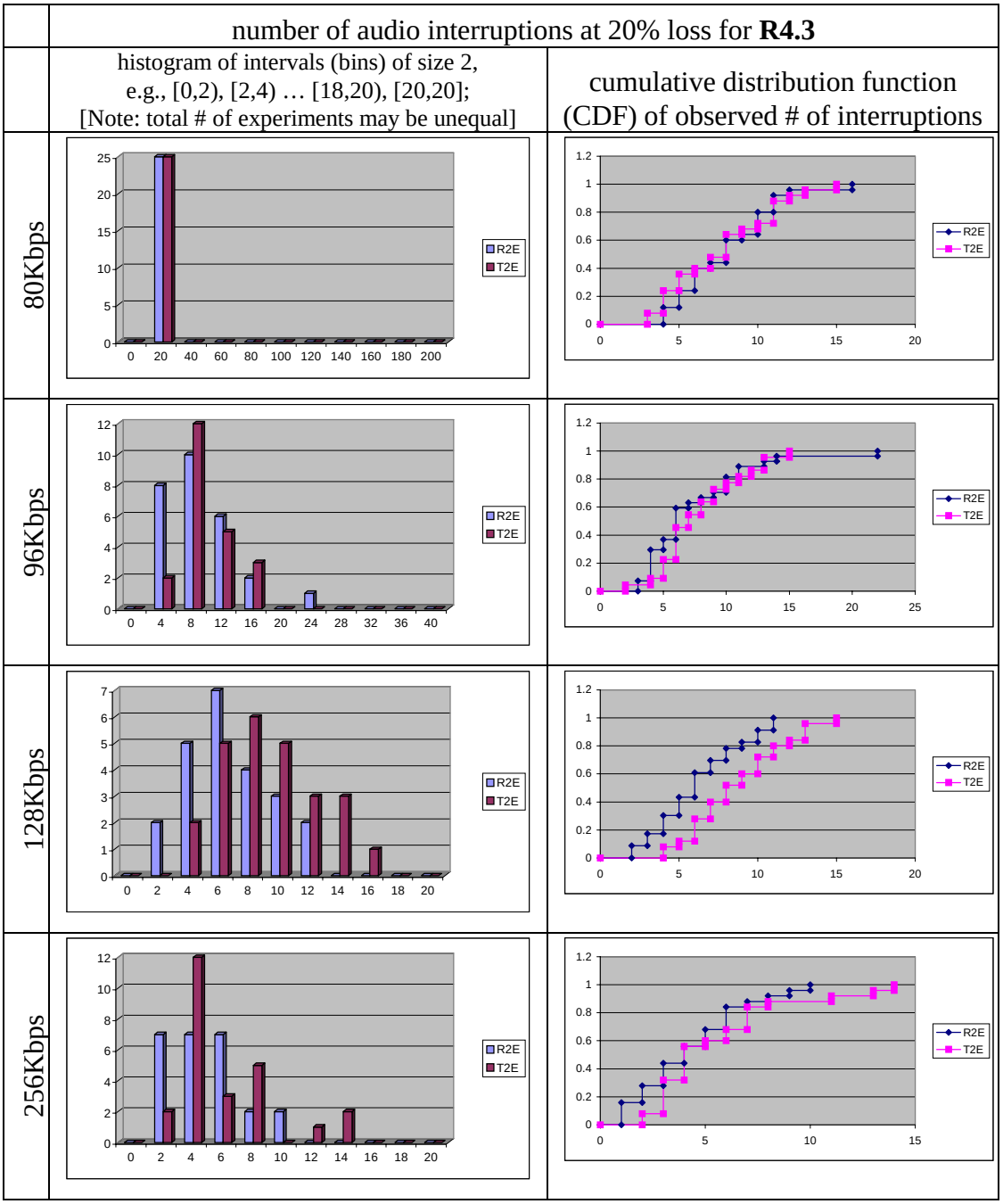


Figure 5.46 Experiment R4.3: audio interruptions at 20% loss

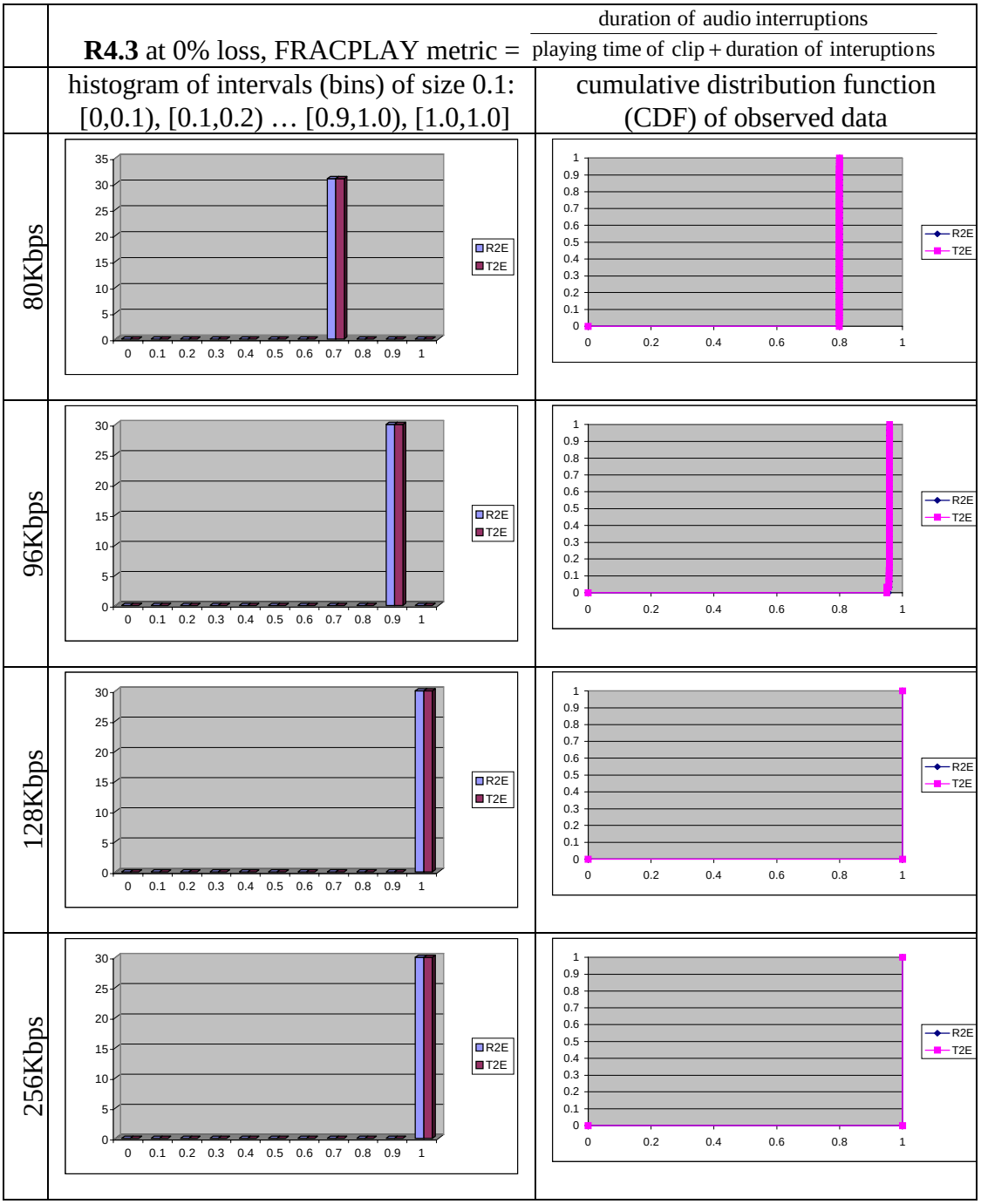


Figure 5.47 Experiment R4.3: FRACPLAY metric at 0% loss

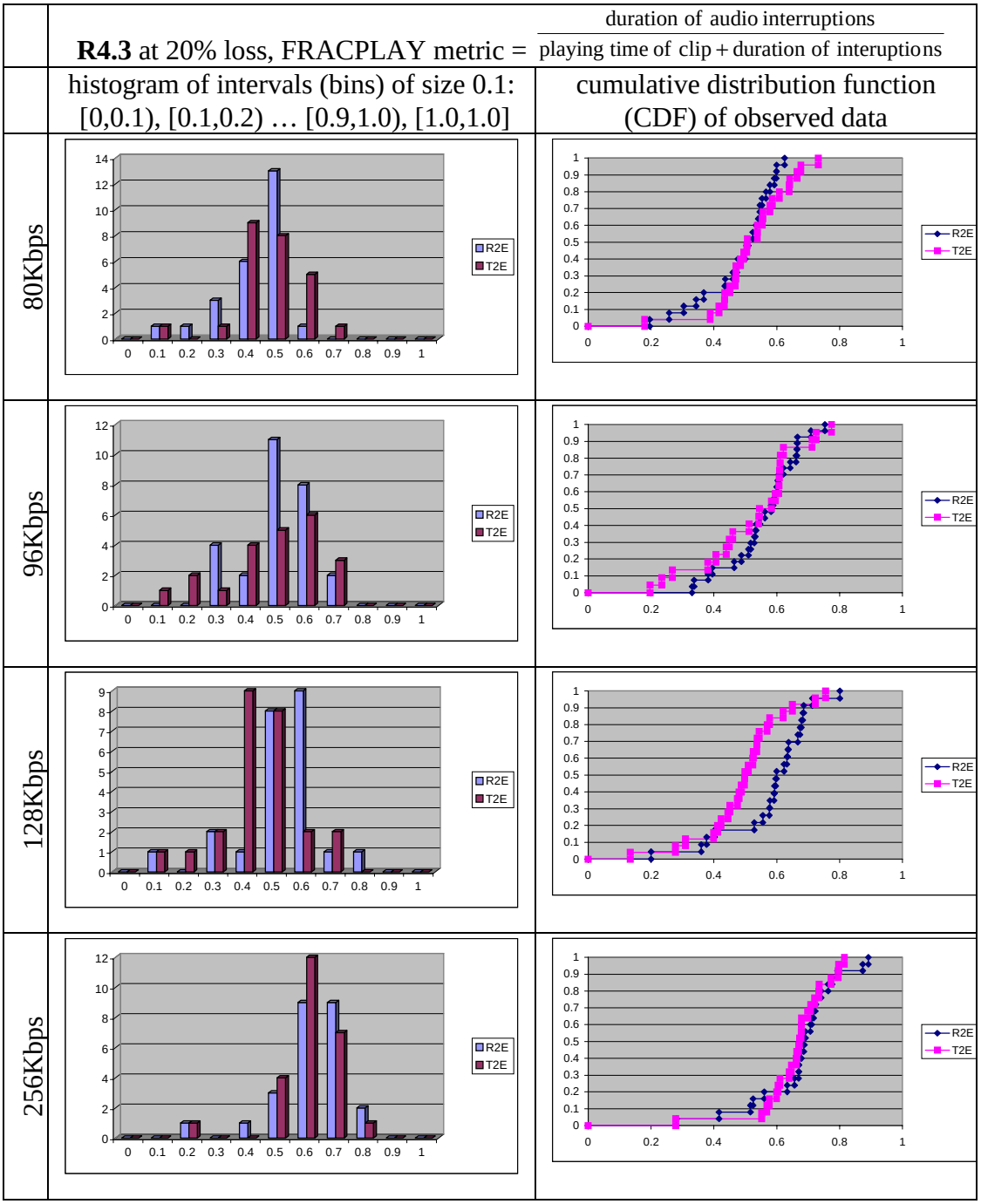


Figure 5.48 Experiment R4.3: FRACPLAY metric at 20% loss

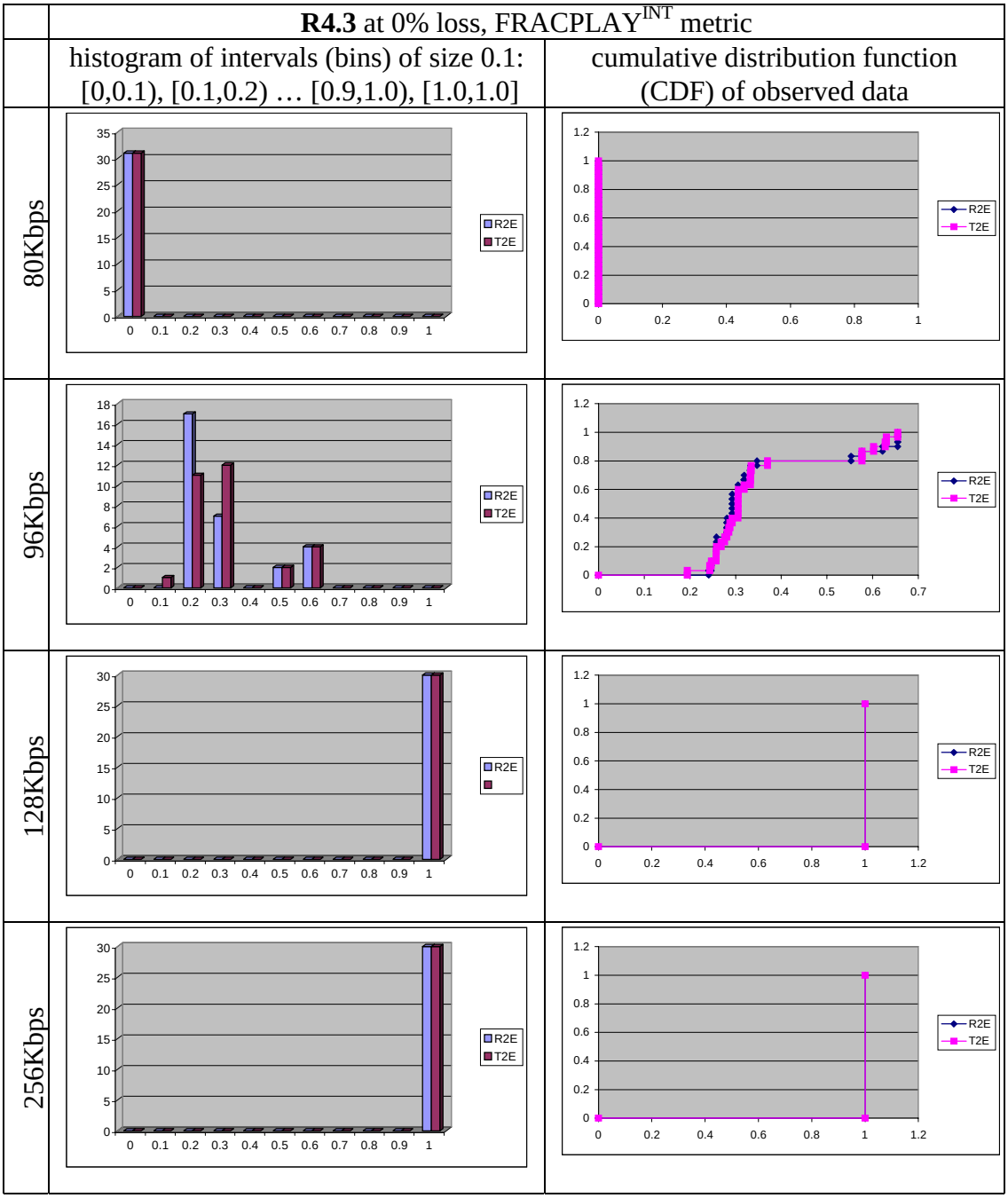


Figure 5.49 Experiment R4.3: FRACPLAY^{INT} metric at 0% loss

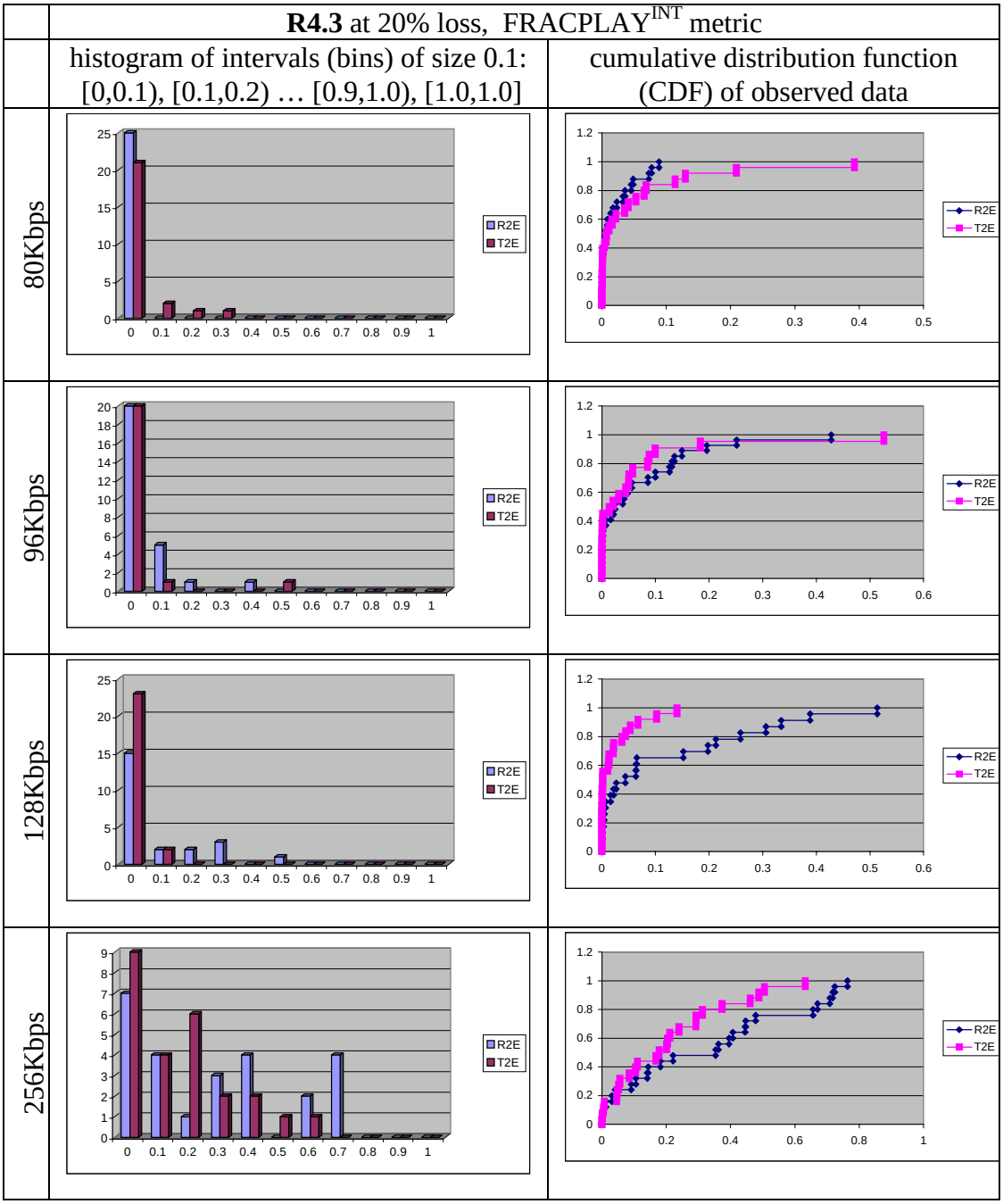


Figure 5.50 Experiment R4.3: FRACPLAY^{INT} metric at 20% loss

5.6.6 Experiment R4.4: observations and conclusions

Experiment **R4.4** explores Hypothesis 5.37, that PO/R service is more robust than O/R service when the window size is increased. We investigate three window sizes, 64, 128 and 256 packets, where the effect of increasing the window size is to reduce performance (see explanation accompanying Experiment **R2.4** in Section 5.4.5). 0% loss is used to validate the results, while 20% loss is used to investigate the effects of large window sizes in the presence of loss. Our observations:

(1) In general, the results are consistent with those of Experiment **R4.1** w.r.t. hypotheses 5.6.1; at 0% loss: (a) there is no significant difference between PO/R and O/R service for bytes, pixels, or audio metrics, and (b) when the loss rate is non-zero, PO/R service outperforms O/R service w.r.t. progressive display of pixels.

(2) For 20% loss, while at the window size of 64, the audio INT metric and $\text{FRACPLAY}^{\text{INT}}$ metric both show better performance for PO/R service, this advantage vanishes as the window size is increased.

The fact that PO/R service shows an advantage at the window size of 64 (a suboptimal window size), and that this advantage vanishes as the window size increases, provides another instance of the umbrella principle described in the previous section: if the window size is suboptimal (as may occur early in the slow-start phase of a TCP connection, for example, or if the receiver's window is less than the bandwidth-delay product), then PO/R offers a performance advantage over O/R service.

We also conclude that in terms of progressive display of pixels, PO/R service offers advantages over a wide range of window sizes, even when pixels occur in parallel with audio. However, as compared with pixels, the advantages of PO/R service for audio are limited to a narrower range of parameter values.

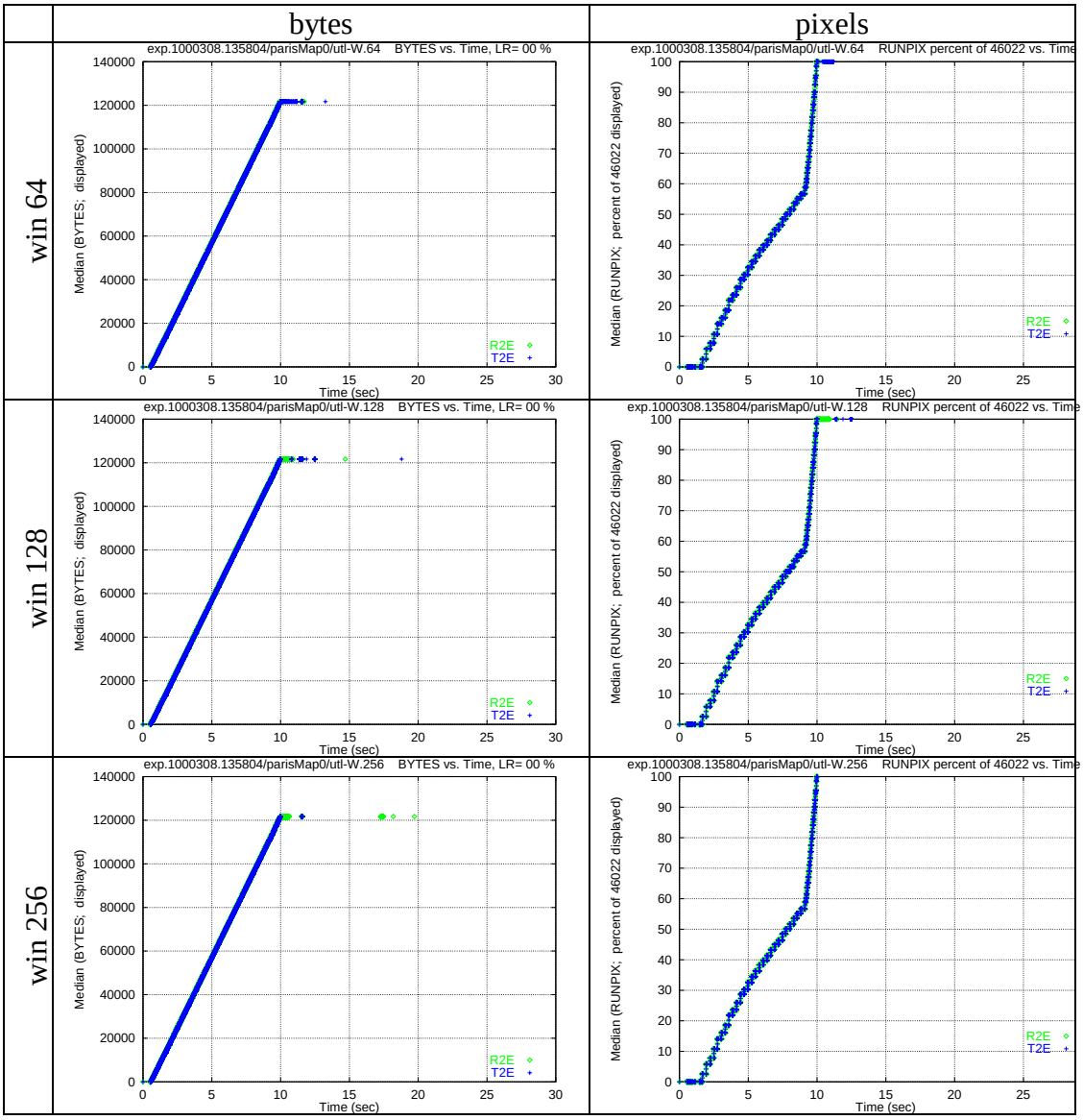


Figure 5.51 Exp. R4.4: LR 0% bytes, pixel perf. graphs

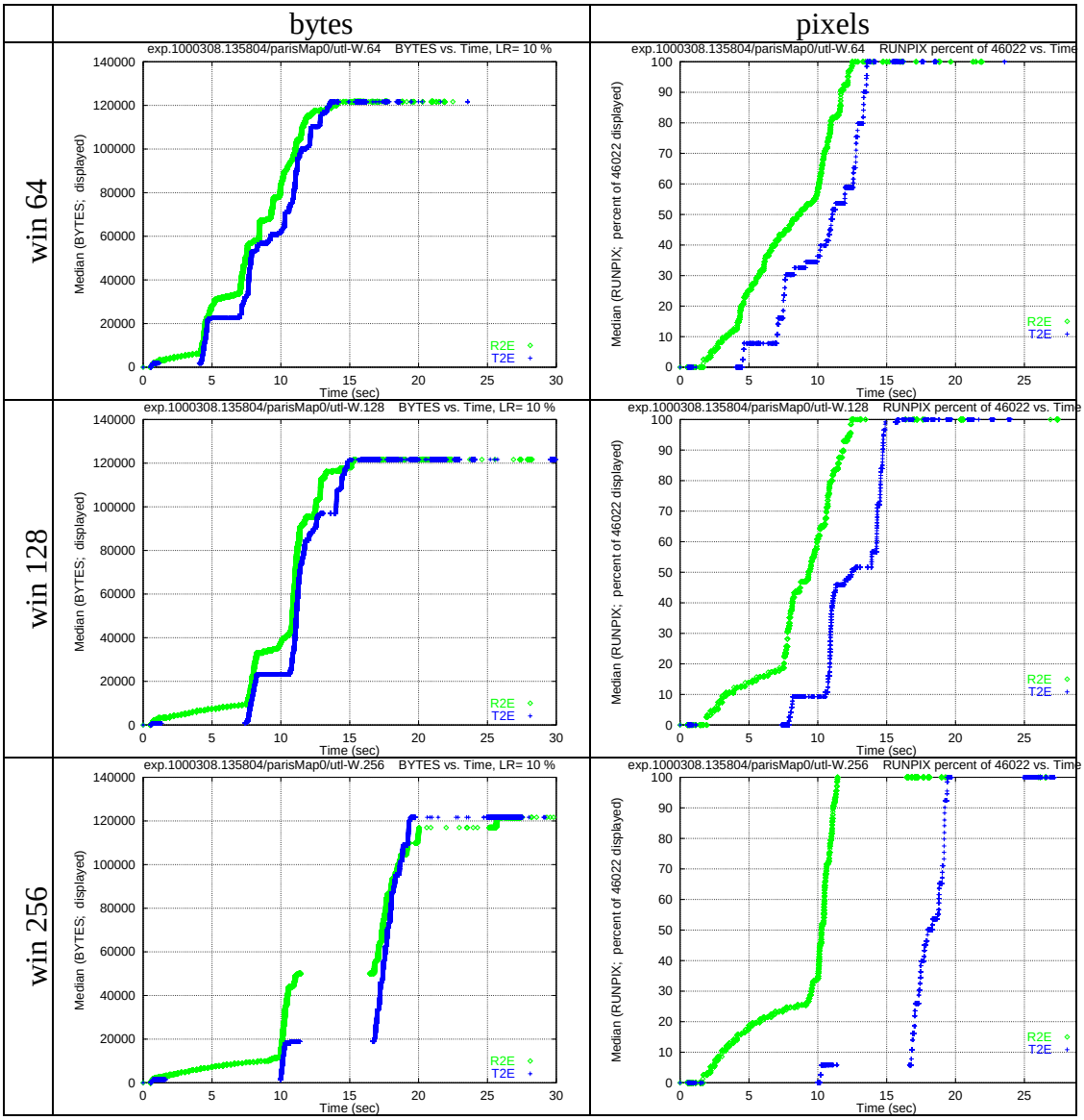


Figure 5.52 Exp. R4.4: LR 10% bytes, pixel perf. graphs

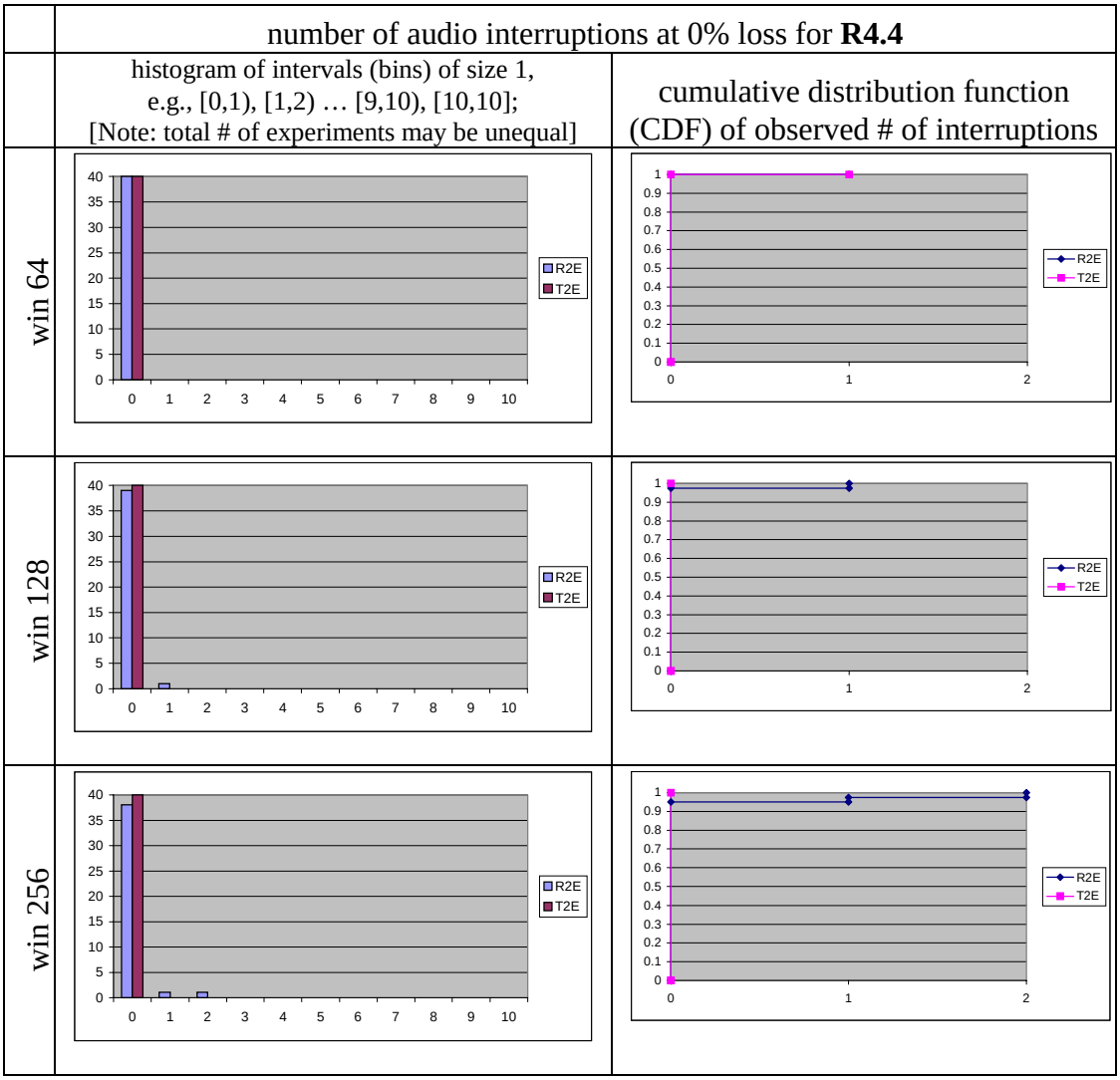


Figure 5.53 Experiment R4.4: audio interruptions at 0% loss

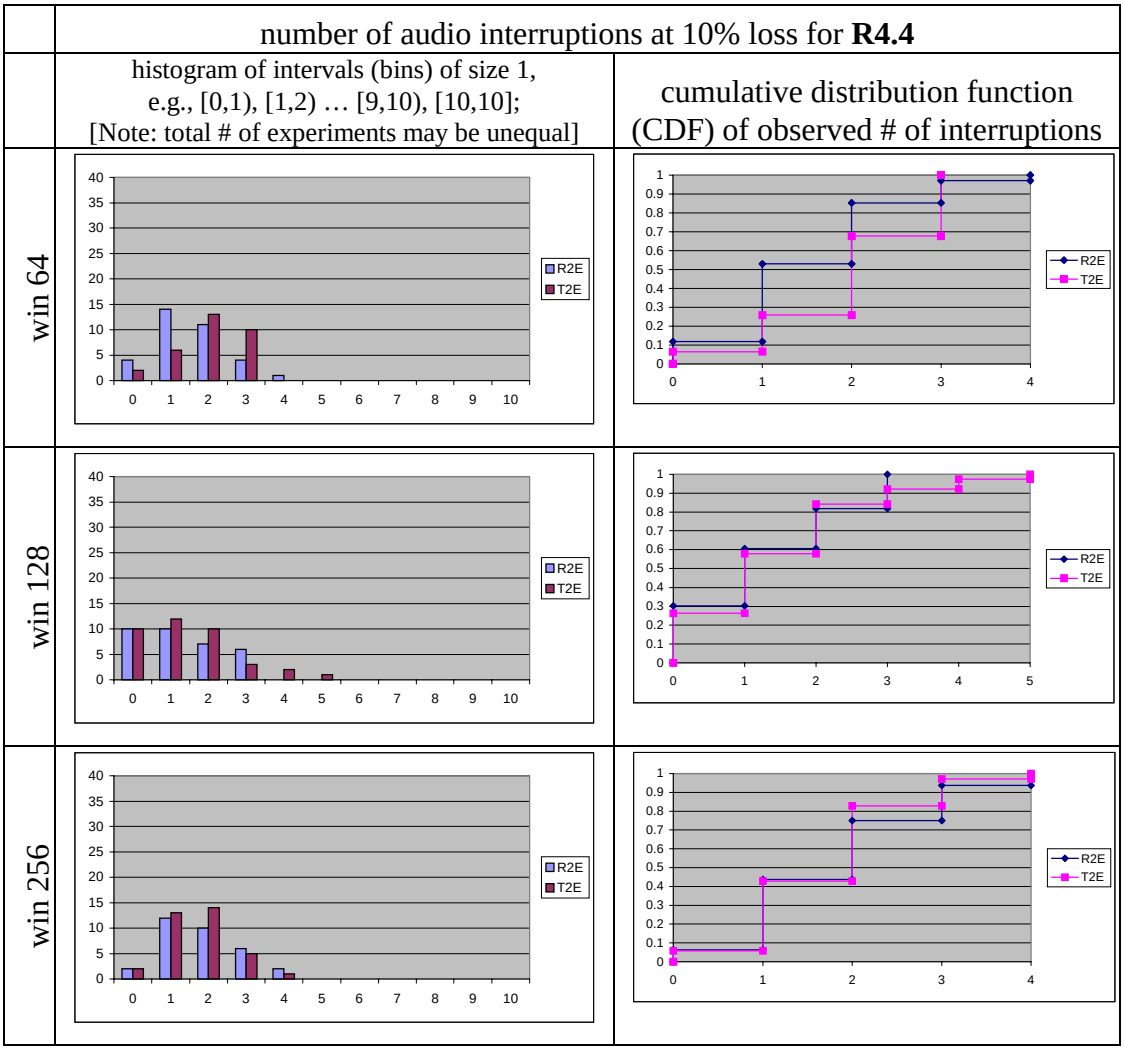


Figure 5.54 Experiment R4.4: audio interruptions at 10% loss

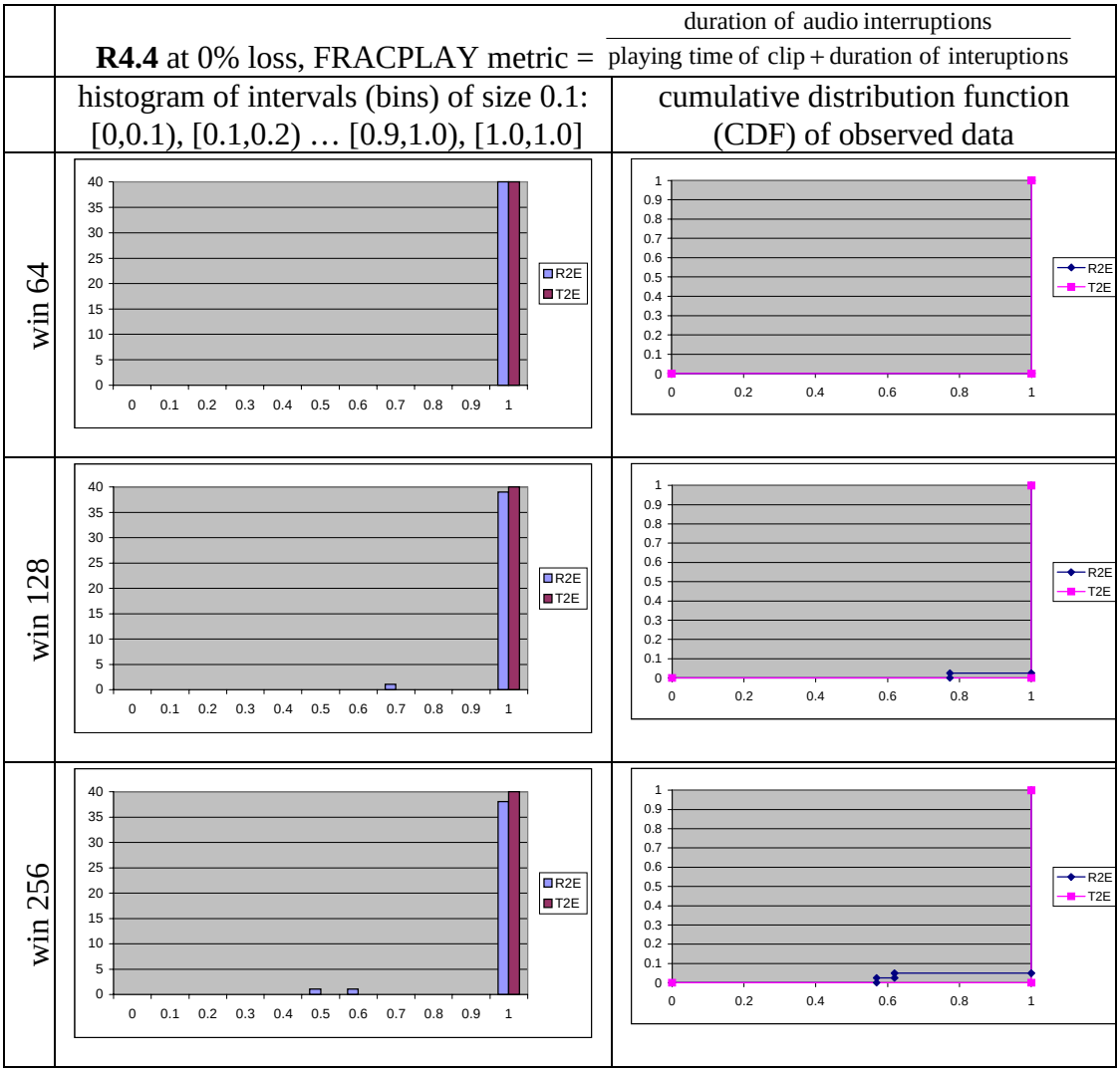


Figure 5.55 Experiment R4.4: FRACPLAY metric at 0% loss

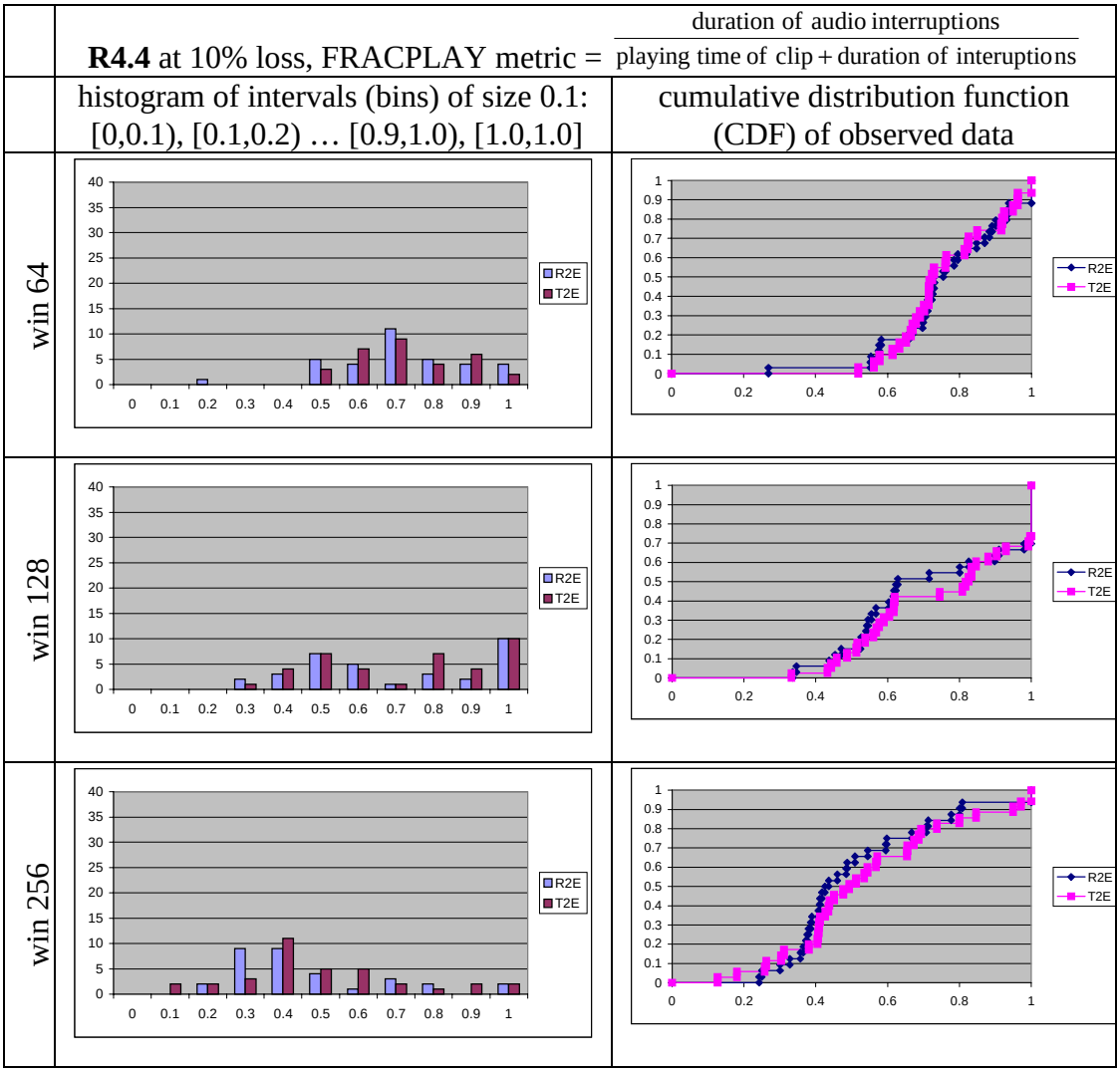


Figure 5.56 Experiment R4.4: FRACPLAY metric at 10% loss

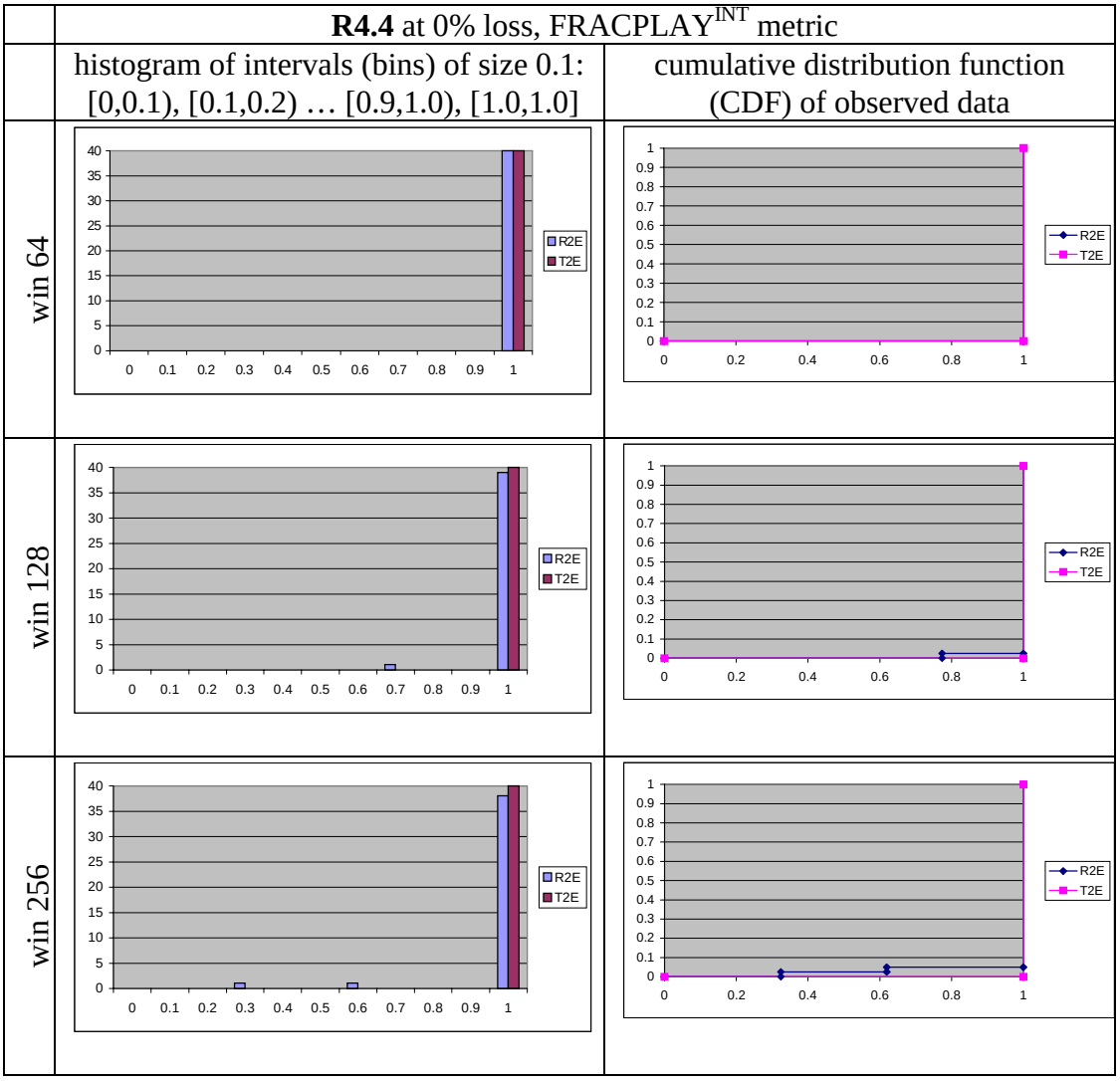


Figure 5.57 Exp. R4.4: FRACPLAY^{INT} metric at 0% loss

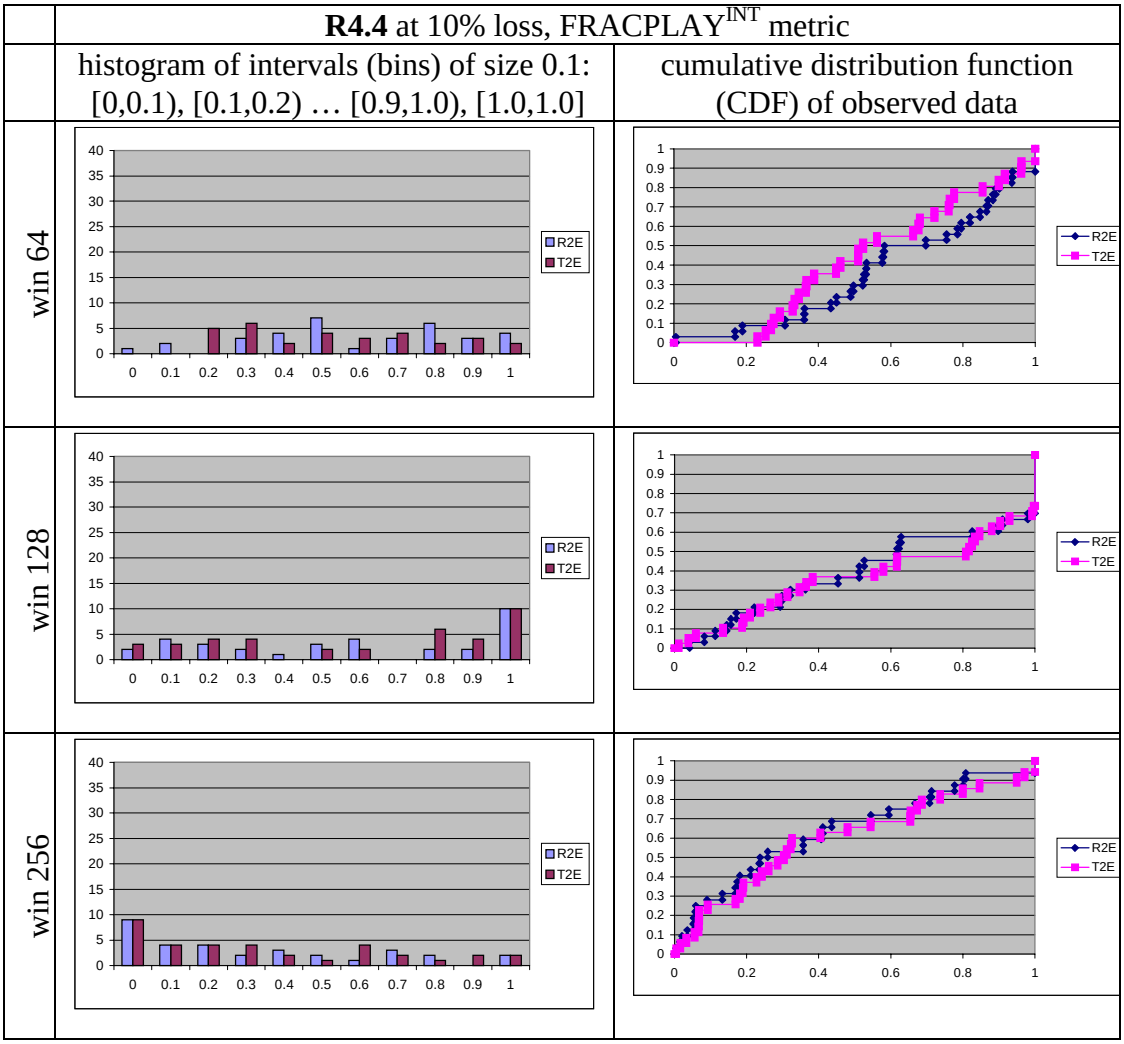


Figure 5.58 Exp. R4.4: FRACPLAY^{INT} metric at 10% loss

5.7 Experiment R5: a complete multimedia document (PO/R vs. O/R for `parisScene1.pmsl`)

In this section, we describe Experiment **R5**, which represents the culmination of our evaluation of ordered/reliable service vs. partially-ordered/reliable service for multimedia document retrieval. In contrast to Experiments **R1, R2**, **R3** and **R4**, which focus on small documents and excerpts from larger documents, Experiment **R5** looks at a full document. The document `parisScene1.pmsl` represents the first section of the `paris.pmsl` document (see appendix).

Section 5.7.1 describes the `parisScene1.pmsl` document. Section 5.7.2 discusses the issue of flow control. Section 5.7.3 describes the parameters and hypotheses for Experiment **R5.** Section 5.7.4 describes our results and conclusions regarding progressive display of bytes and pixels while Section 5.7.5 discusses our results and conclusions related to audio performance**.**

5.7.1 Experiment R5: description of document `parisScene1.pmsl`

In the `parisScene1.pmsl` document, the entire French national anthem is played in parallel with two image streams. The first image stream is a chain of maps zooming in to the city of Paris. In parallel with the maps, and with the audio, a series of eight postage-stamp sized scenes from Paris is presented. Images 1 through 4 are presented, by default, sequentially; this ordering is accomplished by specifying each image as a successor of a particular part of the linear audio stream. However, no precedence relationship exists among images 1–4, which means that when partial order service is used, if an image is delayed by a retransmission, later images can be presented without delay. Images 5–8 follow images 1–4, with 5 following 1, 6 following 2, etc.

After the anthem, map sequence, and scenes have all been presented, an audio clip proclaims "Welcome to Paris", in parallel with the presentation of the final map of Paris. A continue button is then presented, which for purposes of the experiment is always pressed exactly one second after it appears.[62]

5.7.2 Experiment R5: flow control, and the use of R3 and T3 mechanisms

Experiment **R5** uses the UTL mechanisms R3 and T3, rather than R2E and T2E, which were used in Experiments **R1** through **R4**. R3 and T3 correspond to PO/R and O/R service, respectively, just as R2E and T2E did. From the perspective of their UTL implementation, there are two main differences between {R3,T3} and {R2E,T2E}:

(1) R3 and T3 are based on the KX3 layer rather than the KX2 layer.

(2) R3 and T3 have the TCP-friendly mechanisms (slow start, etc.) enabled.

However, of more interest are the operational and qualitative differences between {R3,T3} and {R2E,T2E}:

- R2E and T2E use only selective acks. R3 and T3 supplement the use of selective acks with a cumulative ack value (based on sending order sequence numbers) on each TPDU.
- R3 and T3 provide a *fast-retransmit* feature, which is based on receiving three duplicate cumulative acks, as in TCP. By contrast, R2E and T2E provide retransmission *only* when an RTO timer expires. Note that this change might be expected to actually reduce the benefit of out-of-sequence delivery. In spite of this expectation,

[62] The ReMDoR browser and experiment scripts allow the experimenter to simulate the pressing of continue buttons by a human user after a fixed, specified delay.

> our results show that PO/R service still provides an advantage over O/R service.

While the differences between {R3,T3} and {R2E,T2E} cited above are useful improvements to the transport protocol, they are not the chief reasons for choosing {R3,T3} for Experiment **R5**. Rather, the reason it was necessary to use {R3,T3} for Experiment **R5** has to do with *flow control*.

The role of flow control in out-of-sequence delivery

Some initial test runs carried out while preparing for Experiment **R5** made it clear that flow control is an essential factor in evaluating out-of-sequence delivery. Early test runs using R2E vs. T2E (the results of which are omitted for sake of space) showed no significant difference between R2E and T2E performance for any set of parameters where performance was adequate at 0% loss. Closer inspection showed the lack of end-to-end flow control in R2E and T2E was the key factor in this outcome. Specifically, R2E and T2E lack any means of providing feedback to the transport sender about the level of occupancy in the buffer that lies between the transport receiver and the receiving application. Without such a feedback mechanism, the only way to prevent the application from underflowing is to overprovision: that is, to make the sender send *faster* than the rate of playout at the receiver. Over the long run, this overprovisioning results in the occupancy of the receiver's buffer growing without bound.

For small documents (as in Experiments **R1**-**R4**) this unchecked growth in the receiver buffer occupancy is not an issue, since the occupancy can be made to grow slowly, if a proper transmission speed is chosen. However for larger documents, eventually the occupancy of the transport receiver's buffer provided a playout buffer such that even at relatively high loss rates, all retransmissions were completed before a

packet reached the front of the delivery queue. The browser was downloading most of the document in advance; hence, the document was being pre-fetched rather than streamed.

The dilemma regarding flow control and the benefits of out-of-sequence delivery can be summarized as follows:

> *If the receiving transport entity builds up a receive buffer that, in terms of document playout delay, is larger than the time required to do a retransmission, than out-of-sequence delivery cannot possibly be of any benefit to the application.*
>
> *On the other hand, if the receiving transport entity's buffer is too small, or the transmission speed is too slow to keep the buffer occupancy strictly greater than zero, then the application may frequently underflow regardless of whether ordered or partially-ordered service is used.*

KX3 allows mechanisms built on top of it (such as R3 and T3) to provide full application-transport end-to-end flow control. The application can specify a strict upper bound (in bytes) on the amount of data that may be buffered at the receiver waiting to be delivered to the application. The sending transport entity maintains a conservative estimate of the available buffer space at the receiver, and sends packets only when there is buffer space available in the window. Thus, the sending rate at the transport sender is regulated by the occupancy of the transport receiver's buffer. This scenario is more realistic for a usable transport protocol.

As the remainder of this section shows, by using R3 and T3, and experimenting with the receiver window size, we were able to find two sets of parameters where PO/R service provides an advantage over O/R service for a larger document.

Table 5.10 Parameters for Experiment R5

	Experiment Number	
Parameter	***R5.1***	***R5.2***
Mechanisms	**R3,T3**	
Loss Rates (%)	**0,5,10**	
Network	reflector	
Bit rate (kbps)	512	
One-way delay (ms)	250	
Document	`parisScene1.pmsl`	
Sender Window Size (pkts)	256	
Receiver Window Size (bytes)	**4096**	**8192**

Bold indicates the parameters that are the focus of the experiment

5.7.3 Experiment R5: parameters and hypotheses

Table 5.10 presents our parameters for the experiments that make up Experiment **R5.** This experiment focusses on whether the gains that were illustrated in smaller scale experiments can be realized in the context of a larger document. Overall, our goal in Experiment **R5** is not to do a *complete* study of the performance of PO/R vs. O/R service for larger documents, but rather to provide a starting point for such a study by an example document, and a set of parameters meeting two criteria:

(1) the performance gains for PO/R service over O/R service illustrated in the earlier experiments in this dissertation can be extended to a larger document at these parameter values, and,

(2) the parameter values represent a realistic scenario for multimedia document retrieval.

Together with the results of the previous experiments, such a set of parameters provides a framework for future, more detailed study of the benefits of PO/R protocols, perhaps within the framework of emerging multimedia document standards such as SMIL, and emerging partial order transport protocols such as SCTP.

Our hypotheses for Experiment **R5** are as follows:[63]

Hypothesis 5.7.1 No difference at 0% loss: There will be no significant gain or penalty for using PO/R service vs. O/R service at 0% loss from the standpoint of (a) progressive display of bytes/pixels, (b) or in any of the audio metrics.

Hypothesis 5.7.2 Better graceful degradation of progressive display of bytes and pixels: At loss rates greater than zero, progressive display of bytes and pixels will be better when using PO/R service rather than O/R service.

Hypothesis 5.7.3 Gain increases with loss rate: In terms of the graceful degradation of the progressive display of bytes and pixels, there will be increasing gains from using PO/R service vs. O/R service at 10% loss vs 5% loss.

Hypothesis 5.7.4 Better graceful degradation of audio: At loss rates greater than zero, all three audio metrics will degrade more slowly when PO/R service is used rather than O/R service.

Hypothesis 5.7.5 Throughput will improve with larger receive window sizes: As the window size is increased from 4096 bytes to 8192 bytes, the throughput will improve for both PO/R and O/R service.

5.7.4 Experiment R5: observations and conclusions for pixels and bytes

Figures 5.59 through 5.61 show performance graphs for bytes and pixels

We make the following observations concerning these graphs:

(1) At 0% loss, there is little difference between the bytes and pixels graphs for PO/R service vs. O/R service. This offers support for Hypothesis 5.7.1(a).

Observation (1) can be seen in the top rows of Figures 5.59 and 5.60.

These figures plot the progressive display of bytes and pixels, respectively, vs. time.

[63] The titles in bold such as "No difference at 0%loss" refer to the benefits of PO/R service vs. O/R service captured in each hypothesis.

We see that the green and blue lines representing PO/R and O/R service, (R3 and T3, respectively) are directly on top of one another. In fact, the blue line for T3 obscures the green line for R3 almost entirely.

Observation (1) can also be seen in the green lines of the graphs in Figure 5.61. Each point in Figure 5.61 shows the difference between the average performance of R3 and the average performance of T3 at each point in time. These points are plotted for all three loss rates, for both R3 and T3. We observe that the green line in each graph, representing the performance at 0% loss, remains close to the *x*-axis throughput the entire presentation of the document, showing that the performance of R3 and T3 are nearly identical.

(2) The first derivative of the pixels graphs for the 0% loss case varies over time, while the byte graph for the 0% loss case is close to linear.

The near linear shape of the bytes graph reflects the fact that the flow control is effectively regulating the throughput; the application is consuming data at a steady rate, thereby opening up space for new packets to be submitted at a steady rate.

The curves in the pixels graph represent the fact that at different points in the linear extension, the fraction of the byte stream devoted to pixels vs. other data, most notably audio, changes over time. When present, audio is given preferential treatment in the linear extension selection algorithm (see Chapter 6). The linear extension used in this case was tuned so that audio would receive, on average, 50% of the bandwidth during periods where an audio element was available for transmission.

(3) As compared to O/R service, PO/R service offers significant gains in both the progressive display of bytes and pixels at two different window sizes, for both 5% and 10% loss, offering support for Hypothesis 5.7.2.

(4) The fact that there is a gain for PO/R service, and that the gain increases for 10% vs. 5% loss provides evidence to support Hypotheses 5.7.3

(5) Throughput increases with increased window size, regardless of the loss rate, providing evidence to support Hypothesis 5.7.5

(6) The advantage of PO/R over O/R service is reduced at the window size is increased from 4096 to 8192.

Observations (3) through (6) can be seen in the 2nd and 3rd rows of Figures 5.59 and 5.60, but are more evident in Figure 5.61, In Figures 5.59 and 5.60, the gap between the green and blue lines representing PO/R and O/R service shows the performance gain. The gain, relative to the entire size of the document, may appear small on these graphs. Figure 5.61 is more useful in putting the absolute gain into perspective. We see that the blue and red lines, representing 5% and 10% loss, respectively, show a gain that for bytes, starts at zero and increases in a near linear fashion, until near the end of the document. For the 8192 byte receive window, the gain tops out at 30–35KB, while for the 4096 byte receive window, the gain is even larger: a gain of 50–70KB. The drop in gain near the end can be explained by the fact that with out-of-sequence delivery, the end of the transmission is marked by a dramatic decrease in throughput, while the transport protocol retransmits the last few remaining packets. The decrease in throughput due to packet losses for the average performance of an ordered protocol is more evenly distributed over the entire transmission.

For pixels, the gain rises and falls with an interesting shape with three smaller peaks followed by a fourth larger peak. This shape is consistent across all four combinations for loss rate (5% or 10%) and window size (4096, 8192). This shape is an artifact of the proportion of data in the document devoted to pixels vs. other data, and can be easily understood via an analogy. Consider a race between two runners, A

and B, where A is faster than B on average, but both runners slow down and speed up from time to time. During periods where A is speeding up and B is slowing down, the distance between them will increase. During the periods where A is slowing down and B is increasing in speed, the distance between them will decrease.

The comparison between the progressive display of pixels for PO/R and O/R service is analogous to the distance between the runners. The "speeding up" and "slowing down" of the runners corresponds to the fact that the proportion of the bandwidth available to pixels is larger at certain parts of the document, and smaller at other parts. The user accessing a document via PO/R service arrives *earlier* at each of the points in the document where pixels are displayed rapidly, on average, than the user accessing the same document via O/R service. The gain for PO/R service "shoots up" when the PO/R user arrives at each of these points. The gain for PO/R service then falls when the O/R user "catches up" to the point where pixels are displayed more rapidly.

The exact shape of the curve is tied to the particular document content; other documents would have different curved shapes, as would the same document, if audio were scheduled with a different priority with respect to non-audio data.

Note that for 5% and 10% loss, the average gain over time is strictly positive, and increases steadily almost to the end of the document, and once established, for the bulk of the document, never falls below:

30,000 pixels in the case of the receive window of 4096, and

15,000 pixels in the case of the receive window of 8192.

Overall conclusions related to bytes/pixels

Overall, we conclude that we have found a set of parameters and a document where partial order delivery offers user-perceivable performance benefits in terms of progressive display of pixels and bytes. These results can provide a starting point for future investigations aimed at establishing the limits of the parameter space in which PO/R service can offer such perceptible improvements. Based on the observations above, along with those of all previous experiments, we conclude that this parameter space should be explored further along the dimensions of

- document size and structure
- round-trip delay
- bitrate
- sender and receiver window size
- loss rate

Of particular interest would be to investigate what happens to the absolute and relative gain for PO/R service when all the parameters of **R5.1** and **R5.2** are repeated, and the size of the document is increased.

Experiment R5: observations and conclusions for audio metrics[64]

Figures 5.62 through 5.67 show performance graphs for audio.

We make the following observations concerning these graphs:

(7) At 0% loss, there are some minor audio performance problems at a window size of 4096, but virtually no problems at a window size of 8192.

---

[64] See also the discussion labeled "Interpreting the graphs for the Audio Metrics" in Section 5.6.3.

(8) At 0% loss, the performance of PO/R vs. O/R service was virtually identical, even down to the distribution of interruptions occurring for the receiver window 4096 case, offering support for Hypothesis 5.7.1(b).

Observations.(7) and (8) pertain to the top row of each of the Figures 5.62, 5.63 and 5.64 for Experiment **R5.1** (receive window 4096) and Figures 5.65, 5.66, and 5.67 for Experiment **R5.2**. For **R5.1**, we see that the small window size resulted in at least one audio interruption in every run, and in two audio interruptions 20% of the time. As we would expect, the distribution of audio interruptions for R3 and T3 is virtually identical. For Experiment **R5.2** (receive window 8192), we observe that at 0% loss, not more than 1 out of 40 experiments experienced anything less than perfect audio performance (defined as the absence of any interruptions in playout). Again, the distribution of the (now rare) defects is virtually identical for the two transport services.

(9) For a receive window size of 4096, the CDFs of all three metrics indicate that a user can expect better audio quality from PO/R service than O/R service at both 5% and 10% loss.

(10) For a receive window size of 4096, the performance advantage of PO/R over O/R service is higher for 10% loss than for 5% loss.

Figures 5.62, 5.63 and 5.64 show that there is a measurable advantage to PO/R vs O/R service for audio performance, when the receiver window is limited. The advantage is somewhat modest at 5% loss: as Figure 5.62 shows, PO/R service nearly always experiences only 2 interruptions, while this is only true of O/R service about 3/4 of the time. However, at 10% loss the advantage is clearer. The average number of interruptions is only 2.46 for PO/R service, vs. 3.41 for O/R service. However, a more telling statistic is that for PO/R service, the number of interruptions is 3 or less, 97% of the time. For O/R service, the number of interruptions is 3 or less only 56% of the

time. The other metrics (FRACPLAY and FRACPLAY^{INT}) show similar trends: a slight advantage for PO/R vs. O/R at 5% loss, and a larger advantage at 10% loss. As indicated earlier in this chapter, future work is needed to correlate these metrics with subjective opinions from human listeners.

(11) For a receive window size of 8192, the difference between PO/R and O/R service ranges from practically nothing, to only a slight advantage for PO/R service.

As with our results for bytes and pixels, the gains for PO/R service were reduced when the window size was increased from 4096 to 8192. As stated before, as the receiver window size increases, this effectively increases the playout delay that is available for retransmission of missing audio packets. When playout delay in increased, out-of-sequence delivery is less helpful in reducing audio interruptions.

Overall conclusions related to audio

Overall, with respect to audio, we conclude that PO/R service certainly does no harm with respect to audio, and may offer some help. As we suggested in our analysis of Experiment **R4**, it would be interesting to pursue further objective and human factors studies to explore the parameter space further with respect to Experiment **R5**. In particular, it would be interesting to investigate what subjective score human subjects give to the audio performance at 4096 for both 5% and 10% loss, for both PO/R and O/R service. This would be useful in determining whether the gains seen in the metrics are perceived to be useful by end users

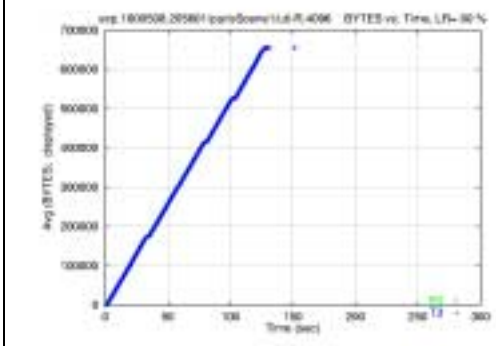
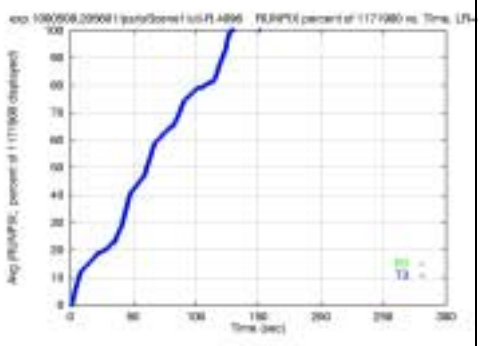
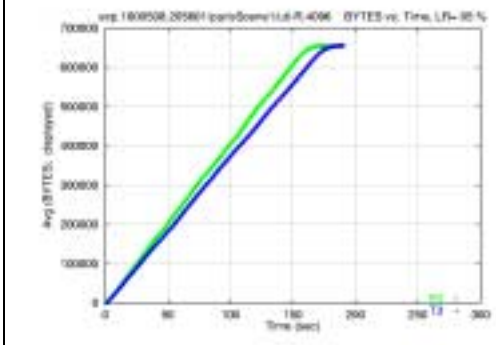
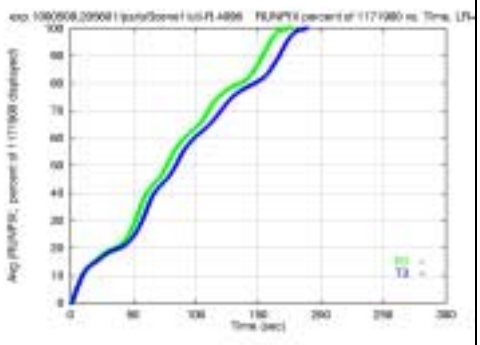
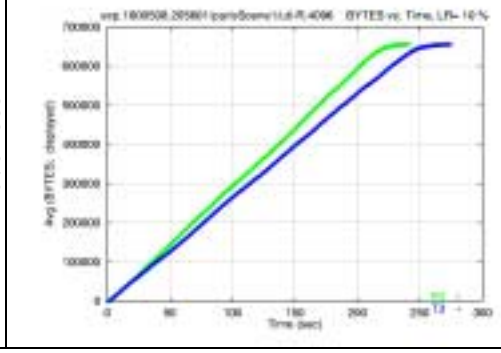
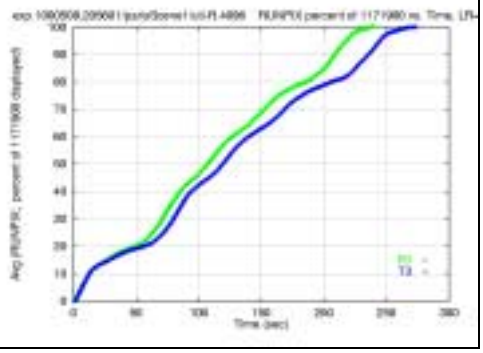
R5.1, Receive Window=4096 bytes		
	bytes	pixels
0%loss		
5%loss		
10% loss		

Figure 5.59 Exp. R5.1: bytes, pixels perf. graphs

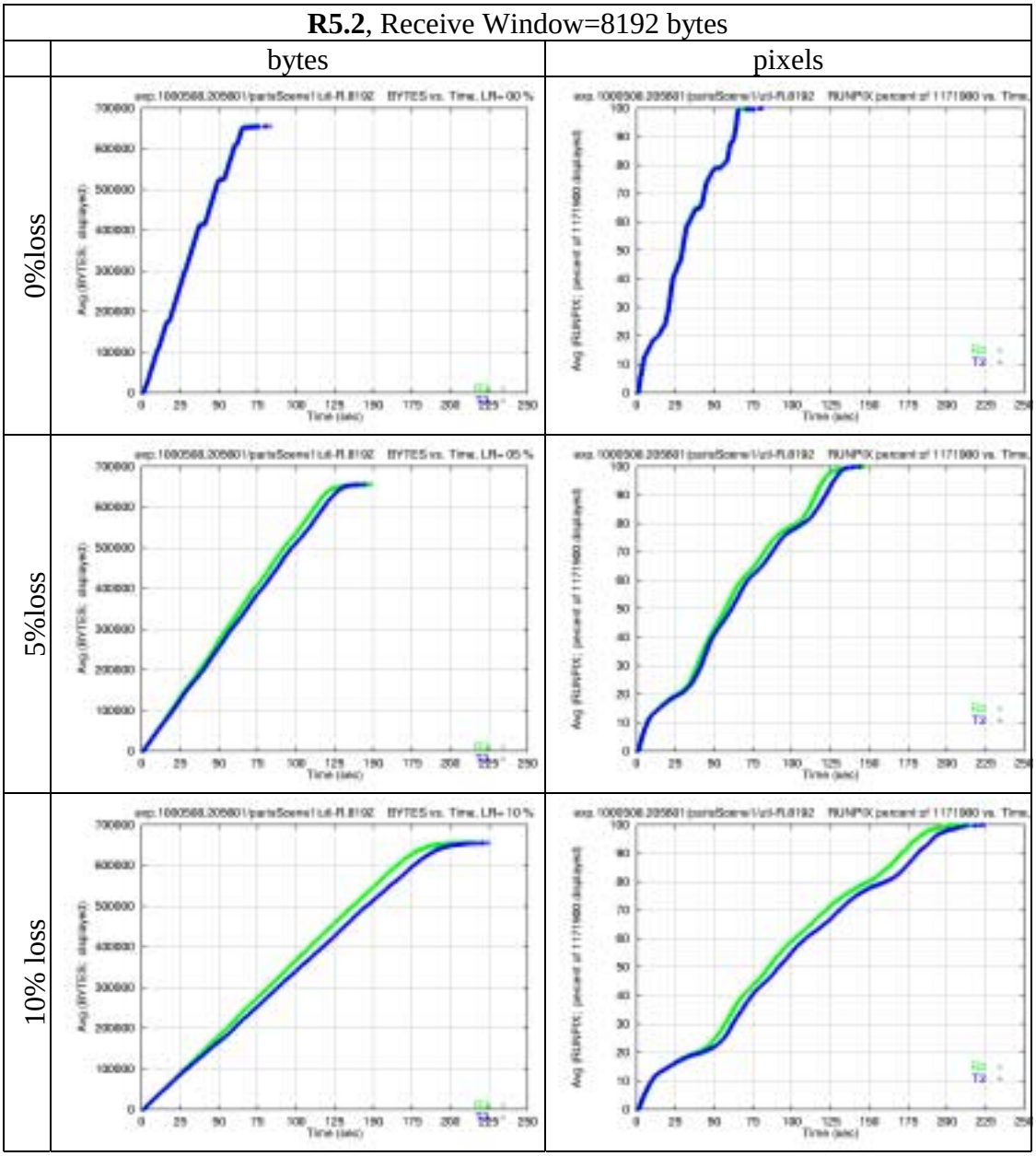


Figure 5.60 Exp. R5.2: bytes, pixels perf. graphs

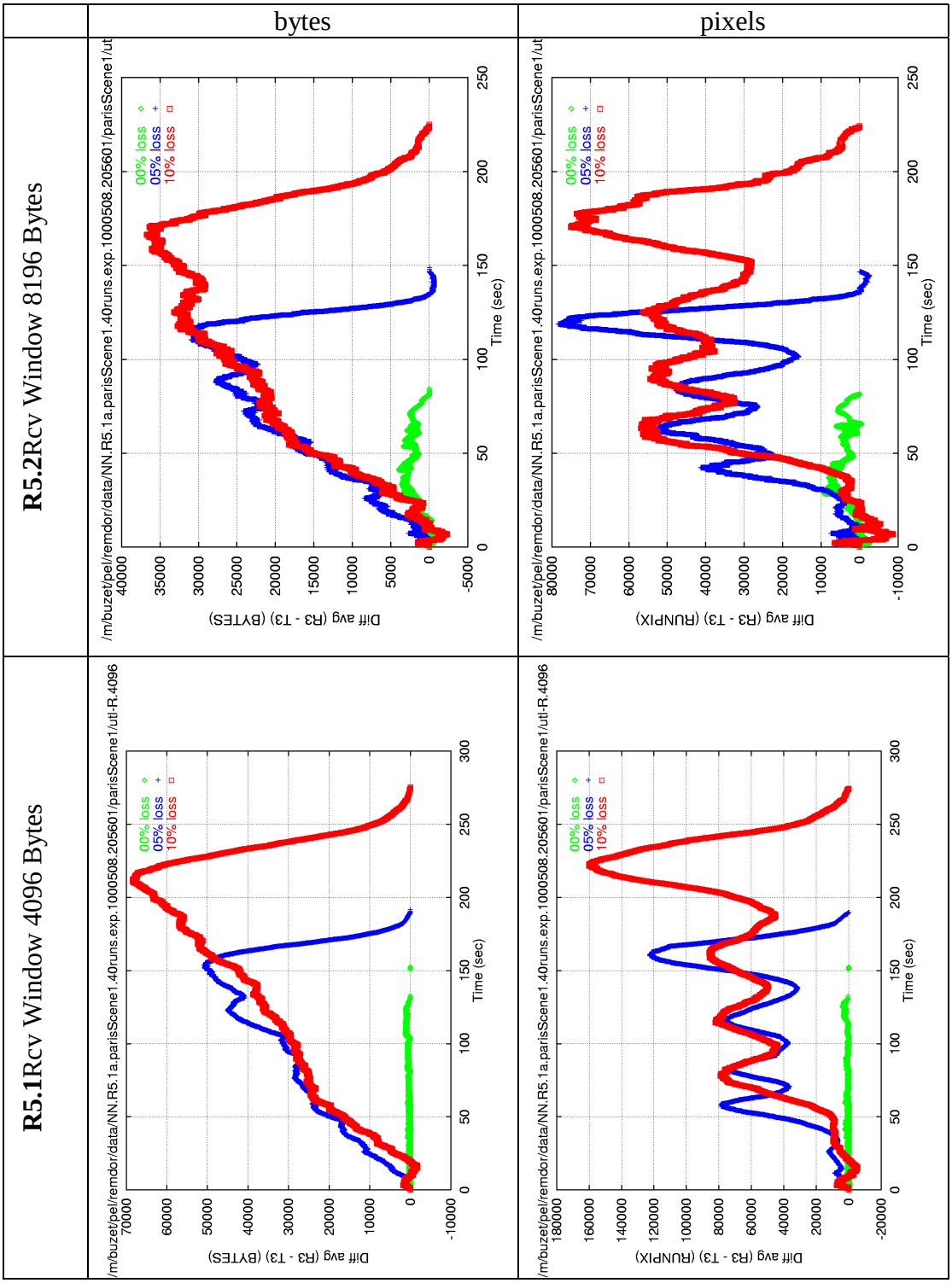


Figure 5.61 Exp 5.1 and 5.2, plotting the advantage of R3 over T3

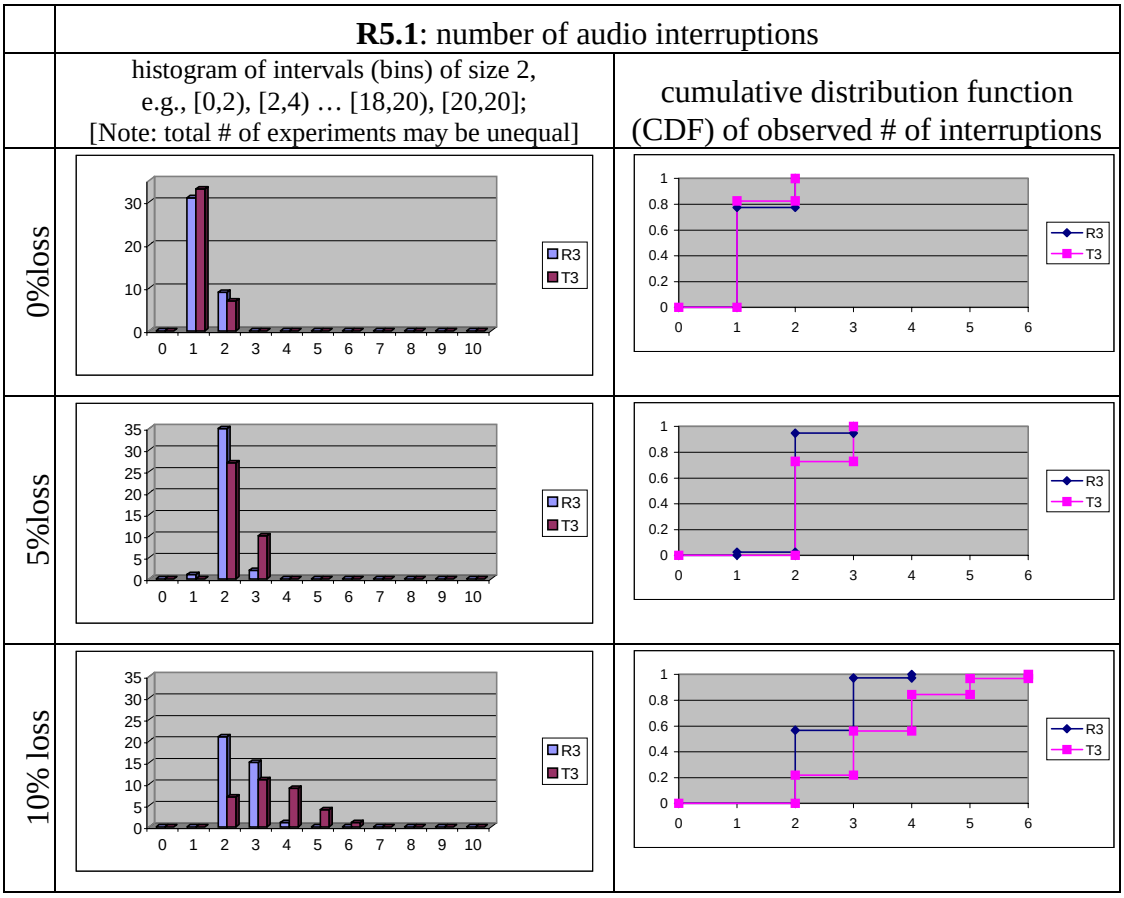


Figure 5.62 Experiment R5.1: audio interruptions

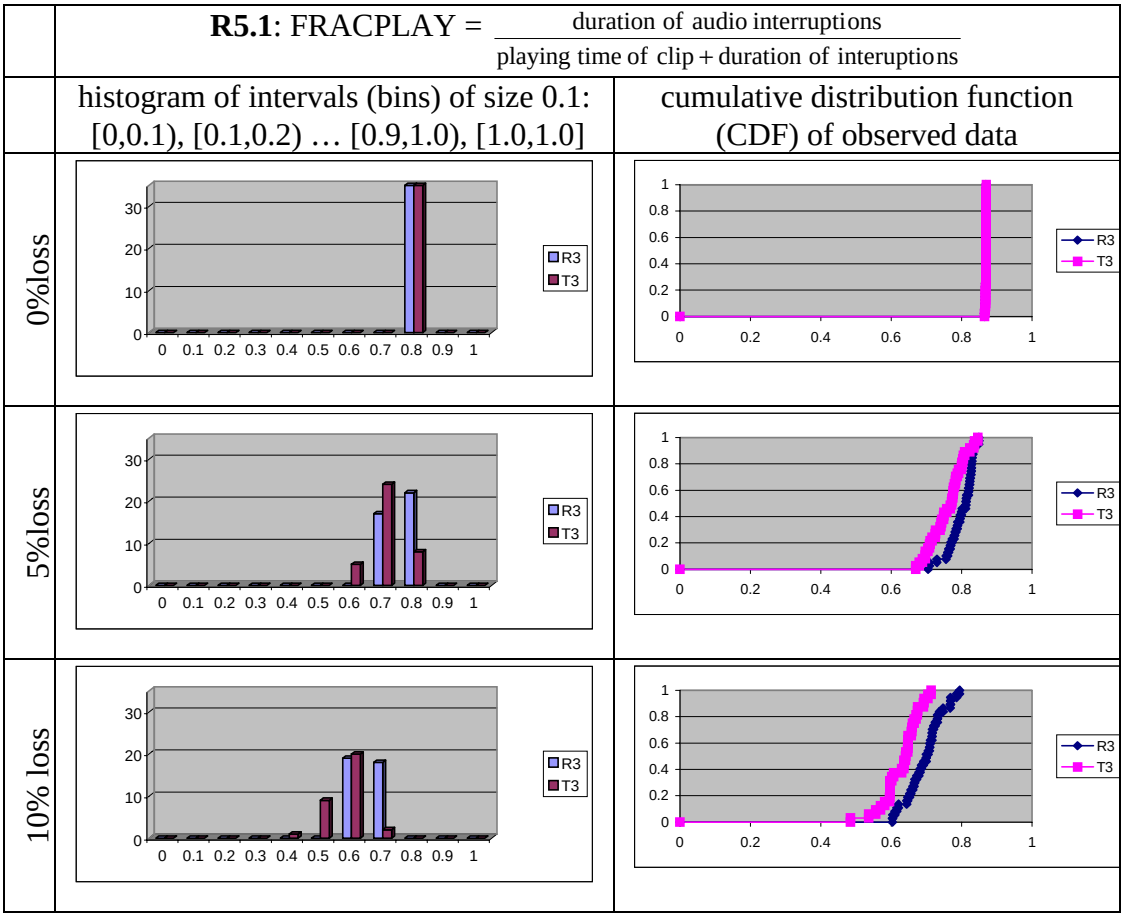


Figure 5.63 Experiment R5.1: FRACPLAY metric

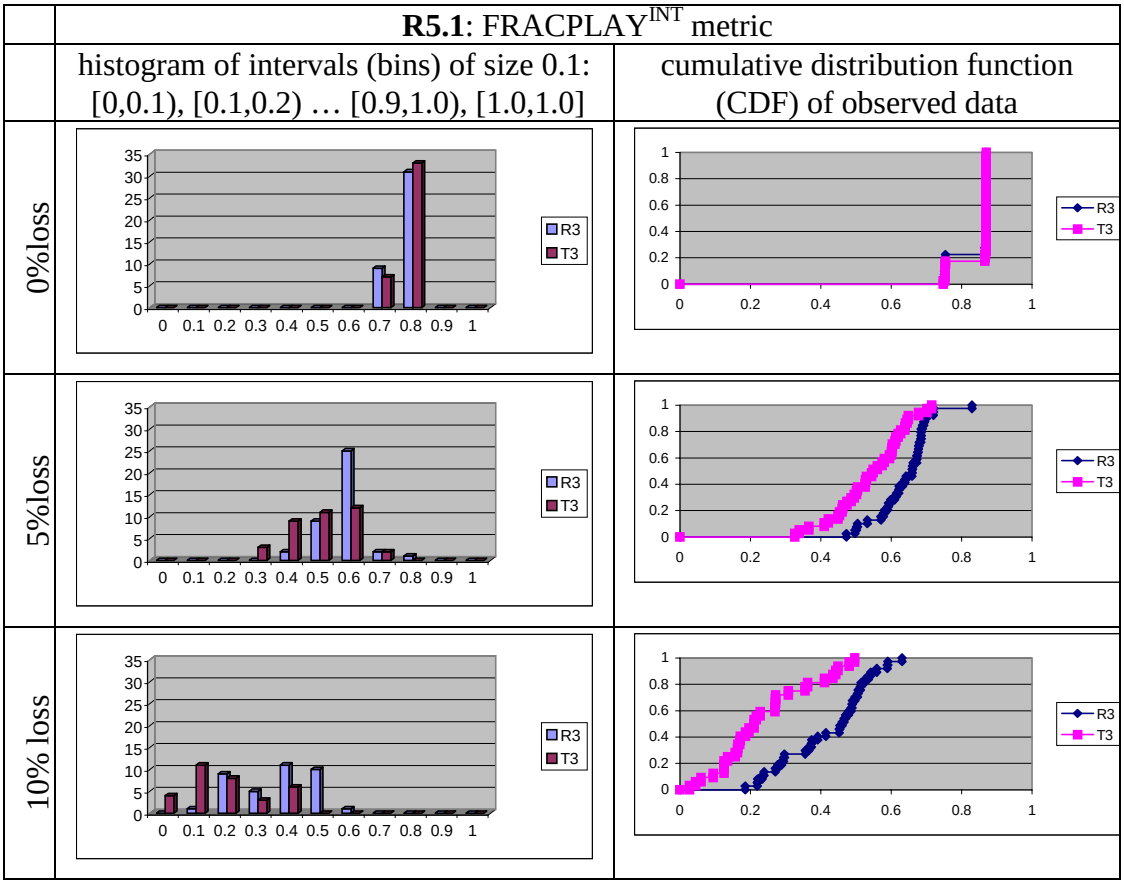


Figure 5.64 Experiment R5.1: FRACPLAY^{INT} metric

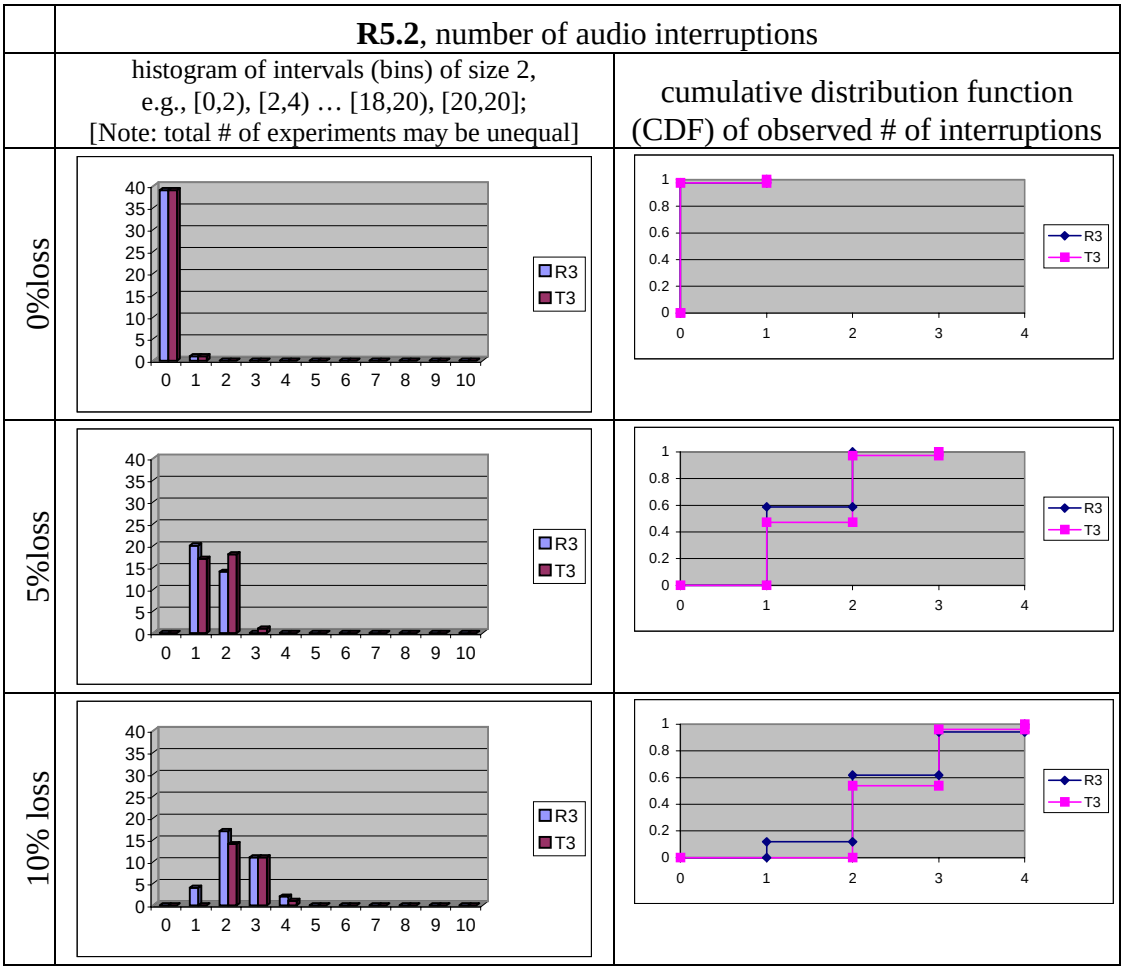


Figure 5.65 Experiment R5.2: audio interruptions

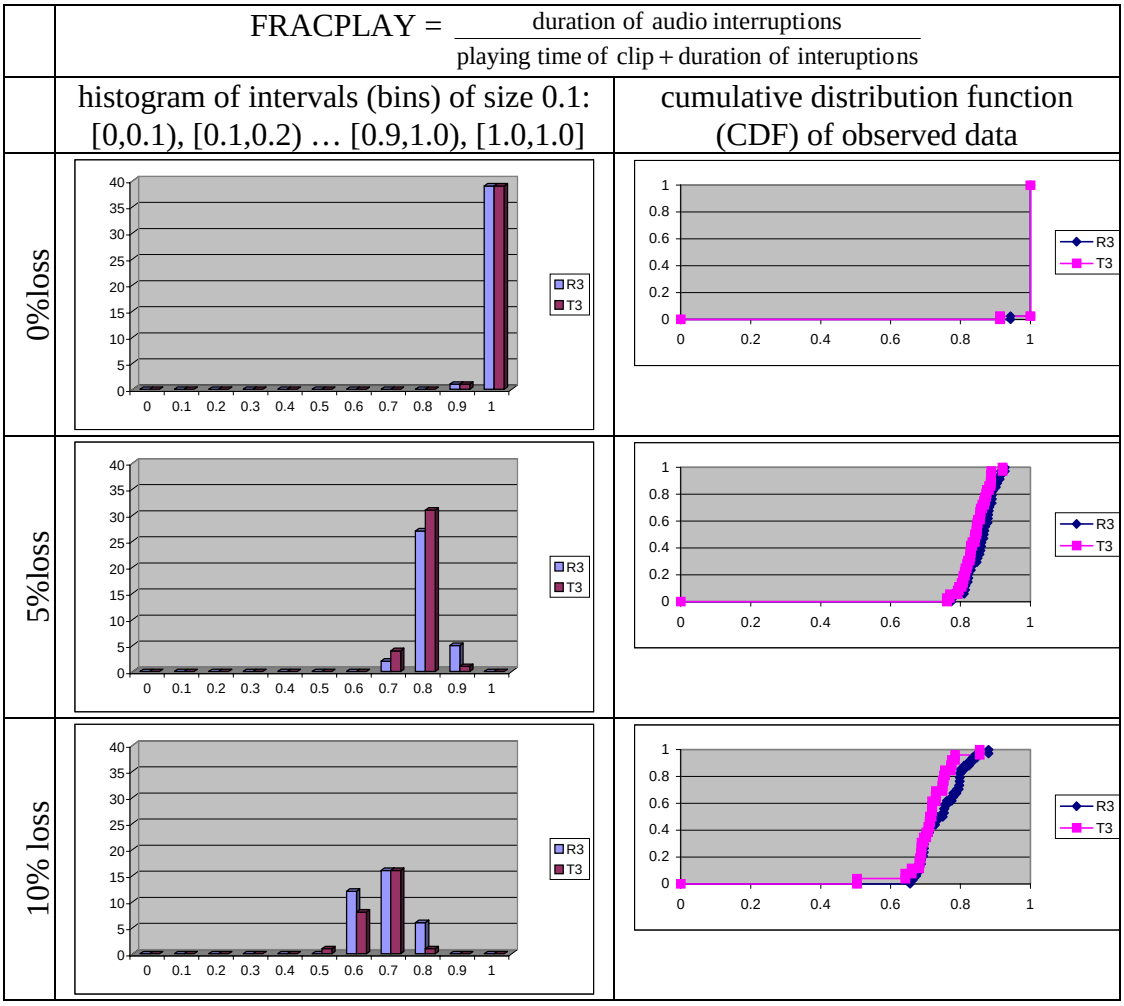


Figure 5.66 Experiment R5.2: FRACPLAY metric

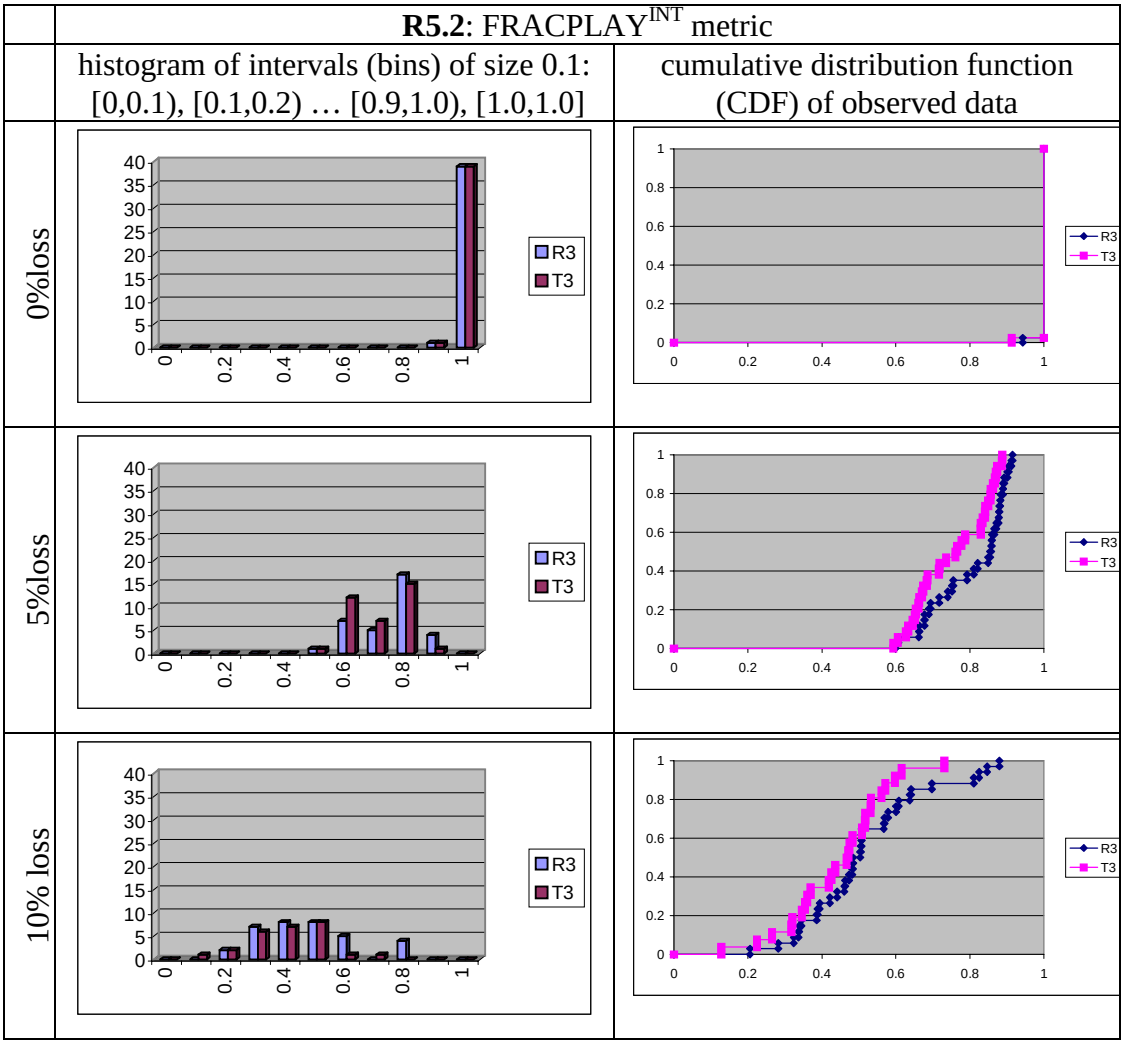


Figure 5.67 Experiment R5.2: FRACPLAY^{INT} metric

5.8 Problems in performance measurement

Many difficulties present themselves in measuring computer system performance in general, and the performance of distributed applications in particular. Classic texts in this area of computer system performance measurement (in general) include (Ferrari et al., 1983) and (Jain, 1991). There is also a helpful chapter by Mogul in (Lynch and Rose, 1993), providing advice more specific to performance measurements in the Internet. A useful summary of the main issues raised in these

primary sources is provided in (Tanenbaum, 1996), in the form of a list of seven pitfalls that experimenters should avoid. We organize our discussion around this list, describing each of these pitfalls, and the steps we took to avoid them.

5.8.1 Tannenbaum's Pitfall #1: Insufficient sample size

Tanenbaum advises: "Make sure that the sample size is large enough". For each of the performance statistics we report, we took many independent measurements, and calculated the mean and std. deviation across these multiple runs. The number of runs varies with each experiment, and is reported along with the experimental data. In each case, we performed at least 30 runs.

In some cases we chose to discard some of the runs. Specifically, we discarded runs in which there was packet loss during the initial connection establishment. Our reason for doing is that given the initial RTO values used in most TCP implementations (which values we also use in UTL) cause a delay of several seconds if there is a packet timeout before the true RTT has been measured. In experiments where the entire document retrieval lasts only a few seconds, inclusion of these runs can distort the results in ways that are entirely unrelated to the use of ordered vs. unordered or partially ordered transport. We claim that discarding such runs:

(1) introduces no bias, since the initial connection establishment is equivalent regardless of the ordering used, and we applied the same discard criteria to all experiments regardless of the ordering being used, and

(2) actually provides a more accurate comparison between or among protocols, since it reduces variance that is unrelated to the aspect of protocol performance being studied.

Table 5.11 shows the number of observations on which each experiment is based. In most cases, our results are based on at least 40 repetitions. In no case do we report any numbers based on fewer than 15 repetitions.

5.8.2 Tannenbaum's Pitfall #2: Non-representative samples

Tanenbaum advises: "Make sure that the samples are representative," observing that network conditions may vary with time of day, day of week, etc., due to fluctuations in system or network load. We performed the experiments reported in this dissertation during a periods of time (June 1999 to May 2000) when the Protocol Engineering Lab was used almost exclusively for the experiments in this dissertation thus we would expect the effects of time of day and day of week to be minimal. Nevertheless, in case there were such effects, we designed the repetition of experiments in such a way that all observations for a given run would be uniformly distributed over the entire time range of the experiment.

Table 5.11 Number of repetitions for each experiment

Experiment N1

Parameter	X2E	R2E	S2E	T2E
LR00	30	30	26	30
LR10	29	29	27	30
LR20	25	26	24	25

Exp.	Parameters	R2E	T2E
R1.1	LR00	29	28
	LR10	22	22
	LR20	22	22
R1.2	LR00	24	25
	LR10	21	21
	LR20	15	18
R1.3	win8	29	31
	win16	27	34
	win32	25	32
R1.4	win8	29	24
	win16	26	17
	win32	29	26
R2.1	LR00	61	61
	LR10	56	53
	LR20	45	45
	LR30	34	39
R2.2	LR00	61	61
	LR10	52	50
	LR20	51	52
	LR30	45	47
R2.3	prop 0, LR00	59	59
	prop 0, LR20	51	48
	prop 125, LR00	59	59
	prop 125, LR00	53	50
	prop 250, LR00	59	59
	prop 250, LR20	51	46
	prop 500, LR00	59	59
	prop 500, LR20	53	42
R2.4	win 4	49	49
	win 8	51	53
	win 16	50	52
	win 32	51	53
	win64	52	53
	win128	50	54

Exp.	Parameters	R2E	T2E
R3	bitrate 2.4, LR00	56	57
	bitrate2.4, LR20	49	44
	bitrate 9.6, LR00	57	57
	bitrate 9.6, LR20	44	49
	bitrate 33.6, LR00	57	57
	bitrate 33.6, LR20	43	48
	bitrate 128, LR00	57	57
	bitrate 128, LR20	45	47
R4.1	LR00	50	50
	LR10	46	50
	LR20	42	50
R4.2	prop 0, LR00	35	35
	prop 0, LR20	32	28
	prop 125, LR00	35	35
	prop 125, LR00	29	26
	prop 250, LR00	35	35
	prop 250, LR20	25	27
	prop 500, LR00	35	35
	prop 500, LR20	27	26
R4.3	bitrate 80, LR00	31	31
	bitrate80, LR20	25	25
	bitrate 96, LR00	30	30
	bitrate 96, LR20	27	22
	bitrate 128, LR00	30	30
	bitrate 128, LR20	23	25
	bitrate 256, LR00	30	30
	bitrate 256, LR20	25	25
R4.4	win 64	34	31
	win 128	33	38
	win 256	32	35

Exp.	Parameters	R3	T3
R5.1	LR00	40	40
	LR05	39	37
	LR10	37	32
R5.2	LR00	40	40
	LR05	34	36
	LR10	34	26

To ensure this, we designed the loop to automate experiments in a particular way. Our experiments were designed test the effect on the performance of two more transport protocols/services (i.e., UTL mechanisms) at one or more loss rates. In some experiments, we also varied one or more other parameters such as bitrate, round-trip delay, or window size. The scripts to repeat these experiments followed the pseudocode shown in Figure 5.68.

```
for ( i = 1; i < numberOfExperiments; i++) do
    foreach lossrate (listOfLossRates) do
        foreach value (listOfValuesForTheVaryingParameter) do
            foreach utlMechanism (listOfUtlMechanisms) do
                { perform the i'th iteration of the experiment for
                    (lossrate, value, utlMechanism); }
```

Figure 5.68 Pseudocode for Experiment Loop

To understand why this design is advantageous, consider that the pitfall to be avoided is to compare, for example, results gathered for partial order in the morning, with results for total order gathered in the afternoon. Thus, the more importance or significance we intend to attach to a particular comparison of two experiments, the more deeply nested the varying of that parameter should be.

For example, since the main purpose of the experiment is to find the performance advantage or disadvantage of one protocol with respect to another, we should place experiments with two different protocols at the same set of parameters as close to each other in time as possible. Similarly, for experiments where we vary a parameter *other than* loss rate—say, window size—we will be comparing experiments

that have the same loss rate, but two different window sizes. Therefore, we want to vary the window size more quickly than the loss rate, so that two experiments that will be compared are close to each other in time.

The outermost loop is the experiment iteration. Because of this structure, our experiment framework is robust against changes in background network conditions. For example, suppose, for sake of argument, that background network conditions change dramatically over the period of some experiment. Provided the change is gradual enough that on average, an entire period of the outermost loop is affected in roughly equal proportion, the comparisons of different protocols, different parameter settings, or different loss rates should still be a fair comparison.

In addition to this design, we also plotted the statistics of each individual run, in addition to the averages and standard deviations of the values shown in this chapter. We do not show all of these individual runs for reasons of space, however we did examine them for each experiment. In each experiment, we ran a case at 0% loss, even when we do not report results for 0% loss; this allowed us to detect anomalies more easily. In rare cases we found outliers; we were able to correlate these with periods where the operator was backing up the hard disks of our experimental systems over the network. We therefore added lines to our scripts to detect when backups were running, and we threw out all such runs. Other than the period backups, we observed no obvious artifacts of time-of-day or day–or-week on system performance across individual runs at the same loss rate.

5.8.3 Tannenbaum's Pitfall #3: Inaccurate time measurements

Tanenbaum advises: "Be careful when using a coarse-grained clock." The timer measurements in this dissertation are taken using the `gettimeofday()` Unix

system call. While this system call reports results at a microsecond level, it is noted in the documentation that only the millisecond level is considered accurate. Fortunately, most of our experiments are generally concerned with the impact of performance on the human user of the system, so we are generally not concerned with differences in times of less than 50-100ms. Thus the millisecond accuracy of the `gettimeofday()` call should be sufficient for our purposes.

There are three other concerns related to time measurement. The first is the cost of the `gettimeofday()` call itself: how much time does it take to call `gettimeofday()` and take a time measurement? We found through repeating the `gettimeofday()` system call in a loop, that the call itself takes, on average, one microsecond or less to complete and thus does not add significantly to the overall time for the experiment.

The second concern pertains to context switches: while the effect of a single `gettimeofday()` system call may be at the microsecond level, it is possible that a context switch may occur either just before, or just after a `gettimeofday()` system call. This can result in over or underreporting the actual elapsed time for some sequence of instructions by tens of milliseconds. In practice, this event does occur, but rarely enough that we can consider it as noise in the system, the effect of which is eliminated by taking the average of repeated measurements.

The third concern pertains to taking time measurements in a distributed system, where clocks may not be synchronized among hosts. Our solution to this problem was simple: we never compare time measurements taken on different systems. Since our focus is on performance improvements as perceived by the end user, we make all time measurements on the end users system (the client) relative to

the instant at which the document is requested. Thus, "time zero" is always the instant at which a user of the ReMDoR system clicks a button or hits the enter key to indicate a request for a particular document; all subsequent times are recorded only on the client host.

5.8.4 Tanenbaum's Pitfall #4: Unexpected interference

During early testing with ReMDoR, we occasionally noticed that performance problems we had solved the week before would suddenly seem to reappear when we had, we thought, made no changes to the source code (it turns out we were correct.) After some investigation (and loss of hair) we discovered that a particular version of a popular web browser had a bad habit of leaving runaway background processes in a CPU bound loop, eating up all available CPU cycles on a given machine. After this discovery, we added instrumentation to our experiment scripts to record statistics on the top CPU time processes between each iteration of the experiment loop. This instrumentation allowed us to check for any rogue processes that might interfere with the experiments. To the extent possible, we also kept other use of the machines involved in the experiments to a minimum while the experiments were being conducted.

5.8.5 Tanenbaum's Pitfall #5: Artifacts of Caching

Tanenbaum also mentions the artifacts of caching as a pitfall of performance experimentation. Caching is a concern, for example, in measuring the performance of the World Wide Web, where repeated retrieval of a particular web page may result in only one transfer, and many subsequent consistency checks on a cached copy that complete much more quickly. Other caching concerns might include

loading of files from an NFS server, loading of instructions into memory, Domain Name Service lookups, or Address Resolution Protocol (ARP) lookups.

In some experiments, we noted that the very first run of set of experiments experienced extra delays, ranging from 200ms to 4s. We were unable to determine the exact cause of this phenomenon, but it is reasonable to suggest that it may be cache-related. In some cases we compensated for this by, in some cases, throwing out the entire first iteration (the outermost loop referred to in Section 5.8.2.) In other cases, based on the hypothesis that the artifact in the first run was cache-related, we tried doing a single "priming run" at 0% loss for an arbitrarily chosen parameter set prior to the running the actual experiment. In practice, this eliminated the artifact in question.

5.8.6 Tanenbaum's Pitfall #6: Misunderstanding what is being measured

Tanenbaum notes that it is important to understand that performance is affected by many factors. When one wishes to compare the performance of one or more transport protocols, it is crucial that there not be any unrelated bottlenecks in, for example, inefficient application level code, poorly designed Ethernet or PPP drivers, etc., that would obscure the main subject of study. In early tests with ReMDoR, we found that performance was noticeably affected by several such unrelated factors. Here is a partial list of these, and the steps we took to work around them:

- To plot performance graphs of progressive display, it is necessary to record the delivery time and number of pixels or audio samples in each packet. Writing this out to disk while the experiment is in progress could significantly affect the performance of the system. Therefore, in the ReMDoR browser, we pre-allocate a large static memory buffer into which all statistics are written while the experiment is in progress; only after it is complete is this buffer dumped to a file on disk.

- Originally the client and server software was stored on a single file server and retrieved via the Sun Network File System (RFC1094, RFC1813). This retrieval sometimes created a noticeable artifact in system performance. For the performance experiments reported in this chapter, we took steps to avoid NFS related interference. All software was placed on the local disks of the client and server, respectively, as were all log files for recording data.

- Our local X-Windows environment is setup, by default, to do encryption of remote X-Windows sessions via the Secure Shell protocol (Metzger et al., 1999). We found that this extra layer of data copying significantly slowed down the performance of the ReMDoR client. Therefore, for the experiments, we used "setenv DISPLAY unix:0.0" to avoid this extra overhead.

5.8.7 Tanenbaum's Pitfall #7: Unwarranted extrapolation

Tanenbaum warns against the dangers of extrapolating performance data for one range to input parameters to values outside this range. In the interpretation of the experiments in this chapter, we are careful to limit our claims only to the particular ranges of parameters that we have studied.

5.9 Overall conclusions from our experiments

Our goal in conducting the experiments presented in this chapter was to investigate the extent to which PO/R service could provide tangible performance benefits for some application: in particular, multimedia document retrieval, and to show some parameter values for which this gain is possible. We have accomplished this goal, and have determined that, indeed, PO/R service can provide measurable performance benefits over O/R service. We will not reiterate the specific numbers have been presented in the chapter already, but will instead provide our interpretation of the larger significance of these results, and what future directions they may suggest for work in transport protocol and multimedia system development.

Let us recognize that the analysis of PO/R service presented in this dissertation represents a starting point, rather than a destination. We have investigated only PO/R service; the incorporation of partial reliability with partial order, PO/PR service, remains to be studied empirically. We have shown benefits for PO/R for a handful of documents that we believe are representative of typical multimedia documents. Further study will be needed to validate this claim in some reasonable scientific manner. We have shown a range of parameter values over which there is measurable benefit. Future study is needed both to determine over what range these performance benefits are perceptible, and to assess (and continually reassess, as the Internet and other network infrastructures undergo constant change) the extent to which these parameter ranges are realistic for various environments. Thus, there remains much work to be done.

Nevertheless, the results of this chapter are significant, and timely for at least three reasons. First, the protocol SCTP (discussed further in Chapter 7) is currently under review by the IETF, and represents an important step in the direction of PO/PR transport protocols within the official Internet standards track. While SCTP is primarily intended as a protocol for signaling of infrastructure devices within the Public Switched Telephone Network (PSTN), the authors of the related Internet Drafts recognize that SCTP may have other uses. The results of the Experiments **N1**, **N2**, and **R1** through **R3** suggest that SCTP may indeed be useful for transport of Web documents (static in time) containing multiple GIF or JPEG images over a modified HTTP protocol.

Second, the W3C consortium has standardized a multimedia document specification language known as SMIL (W3C 1998), which contains many (though not

all) of the same capabilities as ReMDoR. Again, the results of this chapter suggest that it is worthwhile to incorporate features into SMIL that would allow SMIL document retreival systems to take advantage of PO/PR protocols such as POCv2, or perhaps SCTP.

Third, the study of TCP-friendly congestion control is currently a topic of considerable interest and importance to the Internet community. While flow control and the related topic of TCP-friendly congestion control can be isolated from the study of out-of-sequence processing, our results clearly indicate that the converse is not true. We have seen that the details of particular flow/congestion control mechanisms, and indeed, other related details such as retransmission and acknowledgment schemes, RTO estimation, etc., can have a significant impact on the measured benefits of out-of-sequence delivery (i.e., partial order or unordered delivery.) We can conclude that it is unwise to extrapolate either positive or negative results about the benefits of out-of-sequence delivery from one flow-control and retransmission scenario to another. This suggests a need to reevaluate results such as those reported in (Diot and Gagnon, 1999) in the context of more precise simulations of TCP-friendly flow and congestion control, to either scientifically reproduce and confirm them, or report the impact of TCP-friendly mechanism on their conclusions, if any.

Chapter 7 provides an overall discussion of the six chief contributions of this dissertation, however to end Chapter 5, we recap three of these, which can be summarized as *means*, *methods*, and *results*:

- We have provided a *means* to carry out PO/PR experiments, namely the Universal Transport Library, taking PO/PR service from the conceptual, theoretical and simulation realm into the experimental realm.

- We have provided *methods* by which to carry out experiments with PO/PR transport protocols, by providing an experimental framework, including an application for PO/PR service, and a set of experimental tools, parameters and metrics to evaluate PO/PR service.

- We have provided the *results* of enough experiments to (1) demonstrate that PO/PR service is useful and worthy of further study and (2) provide a starting point for a number of future investigations.

Suggestions made throughout the chapter concerning possible directions for these future investigations will be summarized in the "Future Work" section of Chapter 7.