OPTIMIZING PARTIALLY ORDERED TRANSPORT SERVICES FOR MULTIMEDIA APPLICATIONS [*]

RAHMI MARASLI   PAUL D. AMER   PHILLIP T. CONRAD
Computer and Information Science Department
University of Delaware, Newark, DE 19716 USA
Email: {marasli,amer,pconrad}@cis.udel.edu

References [1, 5] introduce a transport protocol that offers partially ordered service for multimedia applications. This paper investigates how much the selection of a linear extension affects system performance in a partially ordered service. We first show how to identify better linear extensions of a partial order, and then determine the performance gains by using such linear extensions at the time of transmission. To quantify linear extensions of a partial order, we propose a new metric ($pBuf$-metric) that is derived from buffering probabilities. Since $pBuf$-metric is complex to calculate, a simplified version called α-metric is also investigated. An OPNET simulation shows that for certain partial orders, a linear extension optimized according to these metrics provides some delay, and significant buffer utilization improvements over a non-optimal linear extension. Thus, prudent transmission order selection in a partially ordered service does improve system performance. Results also show that, in general, α-metric is as effective as $pBuf$-metric in identifying better linear extensions of a partial order.

1 Introduction

Computer networks traditionally offer either ordered (e.g., TCP) or unordered (e.g., UDP) transport service. Some applications such as multimedia do not need an ordered service since they can tolerate some reordering in the delivery of the objects. The degree of reordering should be within the specific limits of the applications; otherwise problems result at the application layer such as increased complexity, increased buffering, and loss of synchronization. For such applications, neither ordered nor unordered service is a perfect fit. Ordered service insists on delivering all data in sequence even if it results in higher delays and buffer utilization. Unordered service, on the other hand, minimizes delay and buffer utilization, but provides no order guarantees. If an application with some order constraints uses an unordered transport service, the application programmer is burdened with the task of implementing mechanisms for object ordering.

To achieve better tradeoffs between order and other quality-of-service (QoS) parameters, and to satisfy the minimal order requirements of applications, partially

[*]This work supported, in part, by the National Science Foundation (NCR-9314056), the US Army Communication Electronics Command (CECOM), Ft. Monmouth, the US Army Research Office (DAAH04-94-G-0093, DAAL03-92-G-0070), and the US Department of the Army, Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002 Federated Laboratory ATIRP Consortium.

ordered transport service has been proposed [1, 3, 5]. Partially ordered service fills the gap between ordered and unordered service by allowing multimedia applications to specify the delivery order of objects in the form of a partial order. Since partially ordered service does not insist on delivering all objects in sequence, it can provide lower delays and buffer utilization than ordered service, while, at the same time, guaranteeing a multimedia application's partial order requirements.

Analytic results from [8] show that, under particular network conditions, a partially ordered service provides delay, buffer utilization, and buffering time improvements over an ordered service. In a partially ordered service, for a given partial order, any of potentially many valid orderings of the objects (i.e., any linear extension) is permitted as a transmission order. Results also show that the actual choice of which linear extension to use at transmission time affects the degree of improvement of different performance statistics (e.g., delay, buffer utilization, buffering probabilities). In general, performance improves as the distance between dependent objects in the sender's transmission order increases. These results suggest that for a given set of network conditions, there exists an *optimal* (or a set of equally optimal) linear extension(s) that will result in the lowest buffer utilization, or lowest delay, etc.

This paper investigates (1) ways of identifying optimal or near-optimal linear extensions, and (2) the effects on overall system performance of using such linear extensions as a transmission order. Unfortunately, with the current state-of-the-art, determining the optimal linear extension requires performing a simulation experiment for every possible linear extension of the partial order being considered. This is clearly impractical. Thus, we first investigate how to identify near-optimal linear extensions in a practical way. For this, we propose a new metric (*pBuf*-metric) as a means of quantifying a linear extension's goodness. This metric is based on buffering probabilities and is derived from analytic results [8]. Since *pBuf*-metric has a complex expression, we also propose another metric (α-metric) which is a simplified version of *pBuf*-metric and easier to compute.

By optimizing linear extensions according to these two metrics, we investigate by way of simulation the significance of the performance improvements obtained by using near-optimal linear extensions over suboptimal ones. These results are helpful to the users of partially ordered services in deciding whether it is even worthwhile to seek near-optimal linear extensions as transmission order. We also compare the performance gains that occur from using a linear extension optimized by *pBuf*-metric with that of α-metric. Such results are helpful in deciding which metric to use in practice in finding good linear extensions.

The paper is organized as follows: Section 2 motivates a partially ordered service through example applications. *pBuf*-metric and α-metric are introduced in Section 3, and the simulation study is presented in Section 4. Section 5 summarizes the main results and discusses future work.

2 Why Use a Partially Ordered Service?

References [1, 5] introduce the development and motivation for a partially ordered protocol/service including several examples. For completeness, a summary of these findings is provided here.

Essentially, a partially ordered service can be employed and is motivated whenever a total order on the delivery of objects is not mandatory. When two objects can be delivered to a transport service user in either order, there is no need to use an ordered service that delays delivery of the second one transmitted until the first arrives. In general, the order requirements of objects in a partially ordered service can be represented by using a partial order PO over the set $[N] = \{1, 2, \ldots, N\}$, where N is the total number of objects to be communicated, and $x \prec y$ in PO signifies that object x must be delivered to the receiving application prior to object y.

2.1 A Simple Application for Partially Ordered Service: Screen Refresh

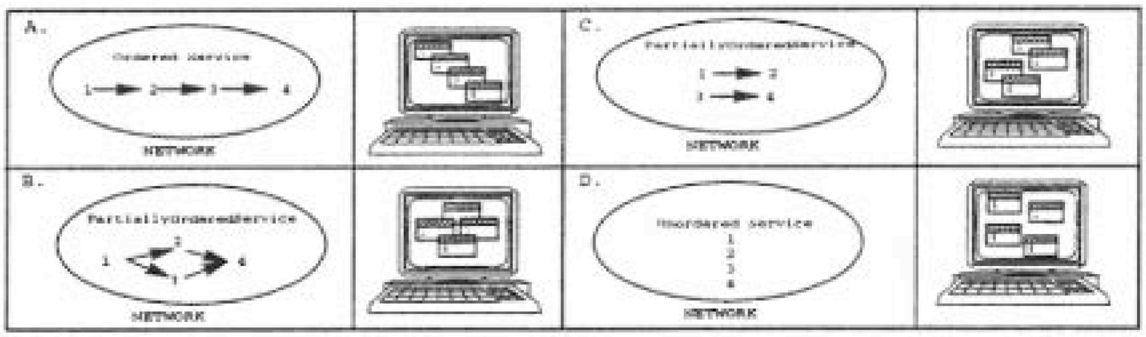


Figure 1: Screen Refresh

Consider an application that must do a "screen refresh" on a workstation screen/display containing multiple windows (see Figure 1). In refreshing the screen from a remote source, objects (icons, still or video images) that overlap one another should be refreshed from bottom to top for optimal redisplay efficiency. Objects that do not overlap may be refreshed in any order. Therefore, the way in which the windows overlap induces a partial order.

Consider the four cases in Figure 1. A sender wishes to refresh a remote display that contains four active windows (objects) named {1 2 3 4}. Assume that the windows are transmitted in numerical order and that the receiving application refreshes windows as soon as the transport layer delivers them. If the windows are configured as seen in Figure 1.A, an ordered service (sometimes referred as a FIFO channel) is required. In this case, only one ordering is permitted at the destination. If window 2 is received before window 1, the transport layer must buffer window 2 and deliver it only after window 1 arrives and is delivered.

At the other extreme, if the windows are configured as in Figure 1.D, an unordered service would suffice. Here any of 4! delivery orderings would satisfy the application since the four windows can be refreshed in any order. Each of these orderings represents a linear extension (LE) of the partial order (PO). As notation, four ordered objects are written $1 \prec 2 \prec 3 \prec 4$, and unordered objects are written using a parallel operator: $1||2||3||4$ ($x||y$ means there is no dependency relation between objects x and y). Figures 1.B and 1.C demonstrate two (of many) window configurations that call for a partial order delivery service. In these cases, two and six linear extensions, respectively, are permitted at the destination.

2.2 Partially Ordered Service for Remote Document Retrieval

Reference [4] describes a prototype system for the retrieval and display of multimedia documents from a remote server using Partial Order Connection version 2 (POCv2), a partially ordered and partially reliable[a] transport protocol providing coarse-grained synchronization support. In this system, multimedia documents with temporal characteristics are described using a Prototype Multimedia Specification Language (PMSL). This language gives the author the ability to express the synchronization, order, and reliability requirements of the objects that make up a temporal multimedia document. The application serving these documents can extract the order, reliability and synchronization requirements from such a specification and communicate them to the transport layer, which then provides the necessary support.

This simplifies application development, since the document display client need not contain complex mechanisms for object synchronization and reordering. It also allows for graceful degradation, since the document can be presented “perfectly” when network conditions allow, and in a less than perfect but nevertheless acceptable manner when network service degrades. Finally, the use of partial order and partial reliability rather than ordered/reliable or unordered/unreliable service allows better QoS tradeoffs between order/reliability and other parameters such as delay, buffer utilization and throughput.

The software that parses and encodes a PMSL document for transmission chooses a linear extension of the partial order as the transmission order. In choosing this linear extension various factors must be considered, including the duration of the individual multimedia objects, their synchronization relationships, and the impact on performance. Therefore, the development of techniques for determining the relative performance of various linear extension alternatives is useful to the development of such systems.

3 Does the Choice of Linear Extension Matter?

In a partially ordered service, the transport sender is permitted to transmit objects in any order that does not violate the partial order [3]. Results from [8] show that the choice of which linear extension (LE) is used by the sender can have significant impact on expected performance. In general, as the distance between dependent objects increases, expected performance improves. Intuitively, this result can be

[a]Partial reliability refers to the notion that individual objects may have different QoS requirements with respect to loss; some may require guaranteed no-loss transport service, while for others, best-effort transport service may suffice. Partially reliable transport service provides a middle ground between these two in which the loss tolerance of each object can be specified individually. References [1, 3, 4] consider partial order and partial reliability in juxtaposition, while [8] and this paper focus solely on partial order.

explained as follows: given objects a and b such that $a \prec b$ in partial order PO, by increasing the separation of a from b, the expected time that b will be buffered due to the network's loss of a will decrease. This is illustrated in Figure 2 which depicts a scenario for five objects where the first transmission fails. In this example scenario, the LE_1 (i.e., "$a\ b\ c\ d\ e$") of $PO = ((a \prec b)||(c \prec d)) \prec e$ buffers objects longer than the LE_2 (i.e., "$a\ c\ b\ d\ e$") of PO. Notice that the LE_2's improvement is provided by increasing the distance between objects a and b. For the simple scenario of Figure 2 (i.e., the scenario where only the first transmission fails), it is easy to find the better LEs of PO. On the other hand, when the possibility of losing any of the objects is considered, identifying good linear extensions or more importantly, optimal linear extensions, is more difficult.

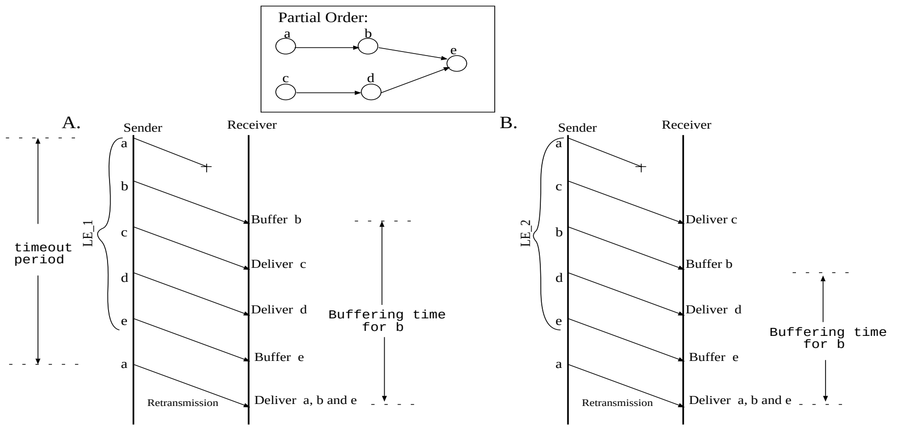


Figure 2: Different linear extensions result in different buffering times

How can we find the optimal LE that maximizes system performance? One possible way is simulating every linear extension, and choosing the one with the best performance. Obviously, this is impractical because of the time needed to simulate every LE of PO. Even a small PO with just 10 objects can have up to $3,648,800$ linear extensions.

To find a good LE in a reasonable time, we propose two new metrics ($pBuf$-metric and α-metric) that are designed to predict the expected performance of different linear extensions of POs. Given two LEs of a PO, the better one can be determined by comparing their $pBuf$-metric (or α-metric) values, instead of their simulation result values. Thus, these predictors will be extremely useful by the transport sender in selecting a better transmission order.

Both of these metrics are based on buffering probabilities and derived from analytic results [8]. Through these metrics, we hope to identify the LE of a PO that results in improved, if not optimal, system performance. These metrics consider the effects of different system parameters (e.g., loss rate, buffer sizes) while quantifying the goodness of a linear extension as a transmission order. In the simulation study of Section 4, we use linear extensions optimized according to these two metrics.

3.1 A metric for quantifying linear extensions: *pBuf*-metric

An ideal metric for finding the LE that minimizes buffering probabilities would be a function of buffering probabilities. Unfortunately since we do not have an exact expression for buffering probabilities, the first metric proposed, entitled *pBuf*-metric, is based on an approximation to the average of buffering probabilities.

Let $pBuf_a(PO, LE)$ be the buffering probability for object a. Then, $pBuf_a(PO, LE)$ can be approximated as follows [8]:

$$pBuf_a(PO, LE) \quad \simeq \quad \sum_{i \prec a \text{ in } PO} pBuf_{i,a}(LE) \tag{1}$$

where $pBuf_{i,a}(LE)$ is the probability that object i is received after object a. For constant network delays, constant object sizes, random (i.e., Bernoulli) packet and ack losses, and relatively large receiver buffer size, $pBuf_{a,b}(LE)$ is computed as:

$$\begin{aligned}
& pSB_{a,b}(LE) \\
= \quad & p^{\lfloor \frac{Dist_{a,b}(LE)-1}{Buf_S-1} \rfloor + 1} \sum_{i=\text{remainder}(\frac{Dist_{a,b}(LE)-1}{Buf_S-1})+1}^{Buf_S-1} \binom{\lfloor \frac{Dist_{a,b}(LE)-1}{Buf_S-1} \rfloor * (Buf_S-1)+i-1}{Dist_{a,b}(LE)-1} * \\
& \left((p_{succ})^{Dist_{a,b}(LE)} * (1-p_{succ})^{\lfloor \frac{Dist_{a,b}(LE)-1}{Buf_S-1} \rfloor * (Buf_S-1)+i-Dist_{a,b}(LE)} \right) \\
+ \quad & \sum_{t=\lfloor \frac{Dist_{a,b}(LE)-1}{Buf_S-1} \rfloor + 1}^{\infty} p^{t+1} \sum_{i=1}^{Buf_S-1} \binom{t*(Buf_S-1)+i-1}{Dist_{a,b}(LE)-1} * \\
& \left((p_{succ})^{Dist_{a,b}(LE)} * (1-p_{succ})^{t*(Buf_S-1)+i-Dist_{a,b}(LE)} \right)
\end{aligned} \tag{2}$$

$$\begin{aligned}
& pBuf_{a,b}(LE) \\
= \quad & \frac{1}{1+p} * pSB_{a,b}(LE)
\end{aligned} \tag{3}$$

where $pSB_{a,b}(LE)$ is the probability that all transmissions of object a preceding the first transmission of object b fail, $Dist_{a,b}(LE)$ is the distance[b] between objects a and b in linear extension LE, p and q are packet and ack loss rates, respectively, $p_{succ} = (1-p)*(1-q)$, and Buf_S is the sender buffer size. Let N be the total number of objects in PO. Our proposed *pBuf*-metric is then defined as follows:

$$pBuf\text{-metric}(PO, LE) \quad = \quad \frac{\sum_{a \prec b \text{ in } PO} pBuf_{a,b}(LE)}{N} \simeq \frac{\sum_{i=1}^{N} pBuf_i(PO, LE)}{N} \tag{4}$$

The *pBuf*-metric approximates the average of the buffering probabilities when

[b] $Dist_{a,b}(LE)$, the distance between objects a and b in linear extension LE, is defined as "seq(b)-seq(a)" where seq(x) returns the assigned sequence number for object x in linear extension LE [8].

the linear extension LE of PO is used as transmission order. Because of this characteristic of *pBuf*-metric, it can be used to discover the LE (or the set of LEs) of a PO that achieves low buffering probabilities in the system. That is, in deciding between two linear extensions of a PO, if we choose the one with smaller *pBuf*-metric value, then we expect smaller buffering probabilities and possibly other performance advantages. Based on *pBuf*-metric, we can define the following two extreme LEs for a PO:

- ***pBuf*-Best_LE(PO,p,q,Buf_S):** Linear extension(s) of PO that has the best (i.e., minimal) *pBuf*-metric value for the given system parameters.
- ***pBuf*-Worst_LE(PO,p,q,Buf_S):** Linear extension(s) of PO that has the worst (i.e., maximal) *pBuf*-metric value for the given system parameters.

Theoretically, *pBuf*-Best_LE and *pBuf*-Worst_LE can be considered as the LEs expected to result in the lowest and the highest buffering probabilities in the system, respectively. However, in actual practice, this may not be the case because (1) *pBuf*-metric is only an approximation to the buffering probabilities, and (2) there are some limiting assumptions in the derivation of the analytic model results (e.g., constant network layer delay) which may limit the effectiveness of this metric. On the other hand, since *pBuf*-metric is derived directly from analytic results, we expect it to be a good indicator of expected performance. That is, *pBuf*-Best_LE is expected to be one of the near-optimal LEs of PO, if not the optimal, and the performance gain by using this LE over *pBuf*-Worst_LE is expected to be close to the maximal performance gain possible by using one LE over another.

Theorem 1 *For $PO = A_1 \oplus ... \oplus A_M$ where each A_i is a partial order, pBuf-Best_LE of PO is the concatenation of pBuf-Best_LEs of A_is.*

Theorem 1 is useful[c] for determining *pBuf*-Best_LEs of certain types of POs. This theorem shows that if a partial order PO is composed of two or more component partial orders that are chained together, then we can find the *pBuf*-Best_LE of PO by concatenating *pBuf*-Best_LEs of each component partial order. The proof of this theorem can be found in [9].

In general, the more linear extensions a partial order has, the more difficult it is to find *pBuf*-Best_LE of that PO. The number of linear extensions of a PO increases, in general, exponentially with the size of PO. Thus, it is much harder to find *pBuf*-Best_LE of a larger partial order. Theorem 1 significantly reduces the time to find *pBuf*-Best_LE of a PO if that PO is composed of smaller partial orders chained together.

In particular, Theorem 1 is useful in finding *pBuf*-Best_LEs of *periodic* partial orders. A periodic PO is defined as a partial order repeating itself some number of times. Periodic POs can be represented as $P \oplus .. \oplus P$ where P is the base partial order concatenated together one or more times. Notice that a periodic partial order can be optimized by finding *pBuf*-Best_LE of just one period. In the simulation study of Section 4, we use periodic partial orders that are optimized by using the result of Theorem 1.

[c] "$\oplus$" is the linear sum or concatenation operator for POs defined [6] as $x \prec y$ in $P \oplus Q$ if and only if $x, y \epsilon P$ and $x \prec y$ in P, or $x, y \epsilon Q$ and $x \prec y$ in Q, or $x \epsilon P$ and $y \epsilon Q$.

3.2 A simplified version of $pBuf$-metric: α-metric

While $pBuf$-metric is a strong indicator of expected performance, using it in linear extension optimization is problematic since its expression is complex. First of all, it is difficult to investigate the algorithmic aspects of LE optimization through $pBuf$-metric since we do not have a closed-form expression for it. Secondly, the time needed for the computation of $pBuf$-metric may be significant since there is an infinite summation in its expression. Thus, we consider an approximation (i.e., simplification) of $pBuf$-metric that is easier to compute. The new metric, entitled α-metric, is defined as follows:

$$\alpha\text{−metric}(PO, LE) \quad = \quad \frac{1}{N} * \sum_{a \prec b \text{ in } PO} \alpha^{Dist_{a,b}(LE)} \tag{5}$$

where $0 \leq \alpha \leq 0.5$. α-metric is derived from $pBuf$-metric as follows: When $Buf_S = 2$, expression (3) reduces to $pBuf_{a,b}(LE) = \frac{1}{1+p} * \left(\frac{p*(1-q)}{1+p*(1-q)}\right)^{Dist_{a,b}(LE)}$ (see [8] for computational details). Let $\alpha = \frac{p*(1-q)}{1+p*(1-q)}$. Notice that since the network loss rate will always be between 0 and 1, $0 \leq \alpha \leq 0.5$. Then, when $Buf_S = 2$, we have $pBuf\text{−metric} = \frac{1}{1+p} * \left(\frac{1}{N} * \sum_{a \prec b \text{ in } PO} (\alpha^{Dist_{a,b}(LE)})\right) = \frac{1}{1+p} * (\alpha\text{−metric})$. Since $1/(1+p)$ term of this expression does not depend on linear extensions, given two LEs of a PO, it does not make any difference whether or not to include it in the computation while deciding which LE is better. Thus, when $Buf_S = 2$ and $\alpha = \frac{p*(1-q)}{1+p*(1-q)}$, both α-metric and $pBuf$-metric result in the same best LEs.

α-metric provides us a simpler expression to work with in investigating better linear extensions of a PO. Based on this metric, we can define the following two types of LEs for a PO:

- **α-Best_LE(PO,α):** Linear extension of PO that has the minimal α-metric value for a given α value.
- **α-Worst_LE(PO,α):** Linear extension of PO that has the maximal α-metric value for a given α value.

In the simulation study of Section 4, we determine how effective α-metric is in finding good linear extensions of a partial order. We will carry out this study by comparing the performance of α-Best_LE with that of $pBuf$-Best_LE.

4 Simulation Study

From analytic study [8], we expect under certain circumstances, using partially ordered service over ordered service can provide valuable performance improvements. Additionally, under these circumstances, using an optimal LE as the transmission order can maximize the expected improvements. In Section 3, we present the ways of identifying near-optimal LEs by proposing $pBuf$-metric for quantifying the goodness of a linear extension. In the same section, α-metric is also introduced as an alternative to complex $pBuf$-metric. At this point, there are two important questions to be answered: (1) Assuming there is an advantage to using a partially ordered service, just how much additional performance improvement can be gained by using an optimal or near-optimal LE vs a suboptimal LE? (2) As a metric for predicting the goodness of a linear extension, is α-metric as effective as $pBuf$-metric? In this section, we try to answer these questions by way of simulation. But first, we introduce the performance statistics and the partial orders used in the simulation study.

4.1 Performance Statistics of a Partially Ordered Service

Table 1: Definition of Performance Statistics

Throughput (λ)	Rate at which packets are delivered to receiving application
End-to-end Packet Delay ($\overline{T_{end}}$)	Average end-to-end packet delay
$\sigma_{T_{end}}$	Standard deviation of end-to-end packet delay
Buffers Used at Receiver($\overline{R\_Buff}$)	Average number of buffers used at receiver

Table 1 defines a set of performance statistics for a partially ordered service. Throughput, λ, is the rate at which a receiving application gets data packets. End-to-end packet delay, $\overline{T_{end}}$, is the average time for a packet to reach to the receiving application once it is given to the sending transport entity. For many applications such as real time audio and video, lower delay is more important than higher throughput. $\sigma_{T_{end}}$ is the standard deviation of the end-to-end packet delay. Multimedia applications generally consist of different streams such as video and audio, and objects that need to be synchronized with each other. Generally, if the variation on the delays (i.e., $\sigma_{T_{end}}$) is smaller, then finer synchronization among different streams and objects can be achieved. Hence, $\sigma_{T_{end}}$ helps quantify a system's jitter. Finally, expected buffers used at the receiver, $\overline{R\_Buff}$, indicates the average memory resources utilized at receiver. In general, it is desirable to have higher λ, and lower $\overline{T_{end}}$, $\sigma_{T_{end}}$ and $\overline{R\_Buff}$.

4.2 Partial Orders Used In Experiments

There exists a large number of partial orders from which to choose for our experiments. The partial orders chosen can be classified into three classes: chain-singleton, parallel-streams, and random partial orders. The first two classes are derived from multimedia applications and they help to evaluate the question of choosing good linear extensions in practical situations. Random partial orders are not suggested by any real application; we use them solely for gaining mathematical insight into linear extension selection. The random POs are generated by methods discussed in [7].

Figures 3 and 4 present all of the chain-singleton and parallel-streams POs used in the simulation experiments. Each PO is characterized by its $m(PO)$ value and density (denoted as "*Dens(PO)*" in the figures). Density and $m(PO)$ are two metrics previously proposed [1, 9] as complexity measures of different partially ordered services. Let $e(PO)$ be the number of linear extensions of PO. $m(PO)$ is then defined [1] as $\frac{\log e(PO)}{\log N!}$. Similarly, let D be the total number of restrictions in PO (e.g., the total number of edges in the transitively closed precedence graph). Using [7], we then define density$= 2D/[N(N-1)]$. In general, one can imagine that a PO with

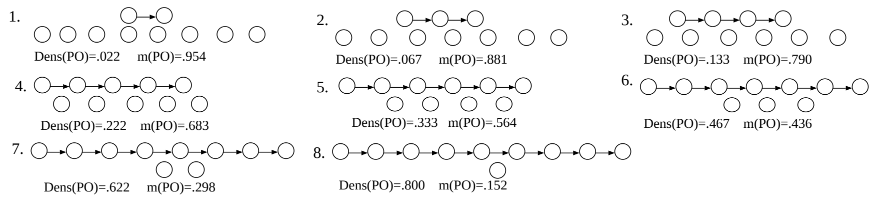


Figure 3: Chain-Singleton POs

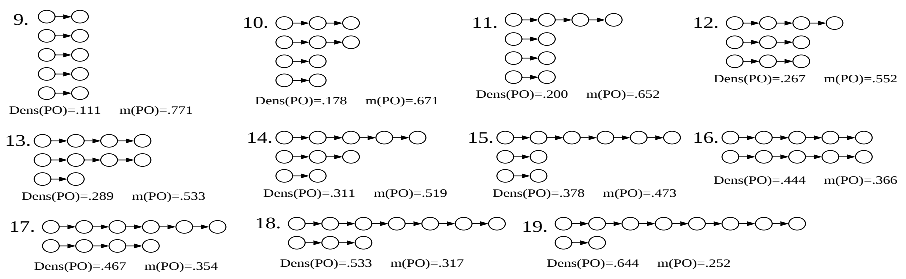


Figure 4: Parallel-Streams POs

lower density provides better performance than a PO with higher density. Similarly, a PO with larger $m(PO)$ value can be expected to result in better performance than the one with lower $m(PO)$ value. Reference [9] shows that these two metrics are almost equally effective in quantifying partially ordered services. Thus, in the simulation study of Section 4.3, we investigate the performance improvements using optimal LEs of POs with different densities.

1. Chain-Singleton POs: These partial orders contain one chain and a set of singletons (see Figure 3). Such POs can be represented as two components composed in parallel: $C||S$ where $C = c_1 \prec .. \prec c_m$ and $S = s_1||..||s_l$ are the chain and the singleton components, respectively. Consider a multimedia application that opens with a welcome message and concurrently paints the screen with non-overlapping objects. The welcome message can be represented by a chain where each word (or sentence) is a separate object. The objects put on the screen as they arrive from the network can be identified as singletons (i.e., antichain). In general, any application that contains an audio or video stream in parallel with some independent objects to be displayed can be represented by this partial order class.

2. Parallel-Streams POs: These are partial orders composed of multiple streams in parallel (see Figure 4). Such POs can be represented as $S_1||..||S_n$ where each $S_i = s_1 \prec .. \prec s_m$ is a stream. Multimedia applications that contain independent streams in parallel can be represented by this class of partial orders.

In addition to chain-singleton and parallel-streams POs, reference [9] introduces two more classes of partial orders that are motivated by multimedia applications: (1) *chain-of-antichain* and (2) *antichain-chain* POs. It turns out that all LEs of a chain-of-antichain or antichain-chain partial order result in identical system performance. Thus, for these POs, there is no best or worst LE. Because of this reason, no experiments are performed for these partial orders.

4.3 Simulation Experiments

We use simulation to answer the following two questions:

1. Under conditions where partially ordered service is useful, does $pBuf$-Best_LE provide significant performance improvement over $pBuf$-Worst_LE? If the answer is *yes*, then we will conclude that, whenever possible, near-optimal LEs of a partial order should be used as the transmission order. If the answer is *no*, then, in terms of system performance, it does not matter which LE a transmitter chooses to use. Section 4.4 addresses this question.

2. Assuming the answer to first question is yes, how close is the system performance obtained using α-Best_LE to that of $pBuf$-Best_LE? If the system performance is close, then we will conclude that α-metric (which is simpler to compute) is as effective as $pBuf$-metric in identifying near-optimal LEs of a PO. Section 4.5 addresses this question.

At the University of Delaware's Protocol Engineering Lab, we built an OPNET-based simulation model to investigate these questions. OPNET (OPtimize Network Engineering Tools) is a comprehensive engineering system capable of simulating large communication networks with detailed protocol modeling and performance analysis [2]. The simulation model was verified by (i) detailed code-inspection and debugging, (ii) comparing the results against those of the analytic model (whenever possible), and (iii) designing a set of 22 experiments, hypothesizing their expected results, running the experiments, and verifying the results as expected [9]. In the simulation model's verification phase, results for λ, $\overline{R\_Buff}$, and $\overline{T_{end}}$ were generally within 1% of the analytic model results when each experiment was repeated **three** times with $30,000$ objects. For the current study, each simulation experiment is repeated **five** times with $30,000$ objects.

There existed a large number of independent system parameters (e.g., loss rates, buffer sizes, etc.) to study in our experiments. It was impractical to exhaustively simulate millions of possible system configurations. Thus, in our study, we focused on four important parameters: partial orders, network layer delays, loss rates and buffer sizes. For each of these parameters, we tried to simulate a reasonable range of values. For example, while studying the effects of network lossiness, we simulated loss levels ranging from 1% up to 40%, the higher being well over the loss rate of most practical networks.

Our simulation study involves four sets of experiments. The base values used in all experiments are given in Table 2. In the experiments, a sender or receiver buffer

Table 2: Base Values Used In Experiments

Partial Order	Chain-Singleton PO #4 in Figure 3
One-Way Network Layer Delay	Normal($\mu = 4, \sigma = 1$)
Loss Rate	0.1
Number of Sender and Receiver Buffers	5 each

size of 1 refers to a buffer space to store one packet. Additionally, network layer delays are generated by a normal distribution in "unit times." Similarly, packet and ack transmission times are normalized as one unit time. Timeouts for retransmissions are set to mean roundtrip delay (RT) plus twice the standard deviation of RT. Timeout values larger than mean RT are used to avoid unnecessary retransmissions of packets due to late-returning acks.

In each of the four experiments, we change only one parameter from Table 2, and study its effects on the performance improvements by $pBuf$-Best_LE and α-Best_LE. The parameters studied in the experiments are:

- **Experiment 1:** Partial Orders = (19 POs in Figures 3 and 4) + (20 random POs)
- **Experiment 2:** One-way network layer delay = Distribution: Normal(μ, σ); μ: 4, 3.5, 3, 2.5, 2, 1.5; σ: $0.25 * \mu$
- **Experiment 3:** Loss rates = 0.01, 0.05, 0.1, 0.2, 0.3, 0.4
- **Experiment 4:** Number of sender buffers = 2, 3, 4, 5, 6, 7, 8, 9, 10

In all experiments, we use periodic POs with 10 objects and $3,000$ periods. The lossiness of the network layer in all experiments was modeled by a Bernoulli process. Additionally, constant object sizes are used. In general, given a PO with variable object sizes, we can obtain an equivalent PO with constant object size by breaking large objects into smaller ones that are chained to each other. Thus, using fixed object sizes for these experiments should not limit the effectiveness of our results.

These four sets of experiments show that throughput is unaffected by the choice of linear extension unless the sender has many more buffers than the receiver. Since in Experiments 1-3, equal buffer sizes at sender and receiver are used, in these experiments, we only focus on the improvements in other performance statistics (i.e., end-to-end packet delay, standard deviation of packet delay and buffer utilization at receiver). Throughput results are presented only for Experiment 4.

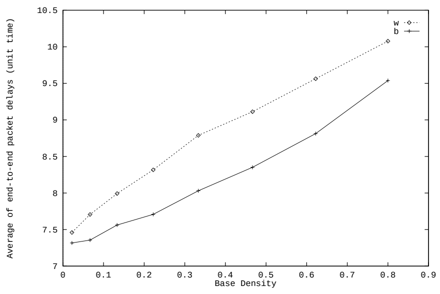
4.4 How Important It is To Use Near-Optimal LEs: $pBuf$-Best_LE vs $pBuf$-Worst_LE?

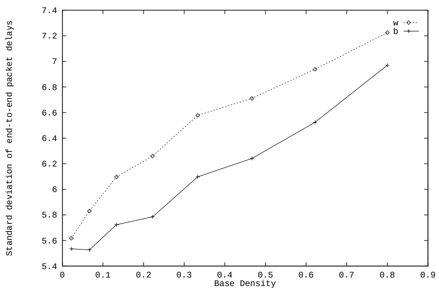
In this section, we determine the significance of the performance improvements that can be obtained by choosing near-optimal LEs. We will carry out this study by comparing the performance of $pBuf$-Best_LE with that of $pBuf$-Worst_LE in Experiments 1-4. While $pBuf$-Best_LE may not be optimal, by comparing $pBuf$-Best_LE with $pBuf$-Worst_LE, we do get a lower bound on the potential improvements of one LE over another one.

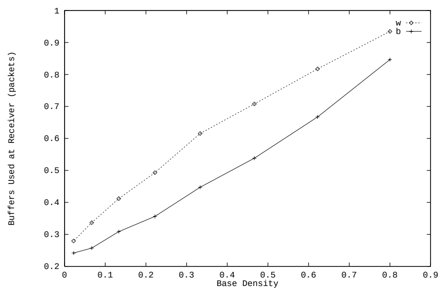
The simulation results are presented in the graphs of Figures 5-9. These graphs illustrate $\overline{T_{end}}$, $\sigma_{T_{end}}$ and $\overline{R\_Buff}$ versus system parameters (i.e., densities of POs, network layer delays, loss rates and sender buffer sizes) for *pBuf*-Best_LE and *pBuf*-Worst_LE. In the graphs, the corresponding curves for *pBuf*-Best_LE and *pBuf*-Worst_LE are labeled as "**b**" and "**w**", respectively.

For each set of graphs, there is a corresponding table in Figures 5-9 that summarizes the percentage improvements in performance statistics. Each table is vertically divided into two parts. The left column(s) introduce the system parameter(s) studied in the corresponding experiment. The right columns present the percentage improvements in various performance statistics by using *pBuf*-Best_LE over *pBuf*-Worst_LE when the corresponding system parameter value is simulated. As an example, consider the table in Figure 6. This table introduces the percentage improvements in parallel-streams POs from Experiment 1. The entry *Density=0.111* and $\overline{T_{end}}$*=6.19* can be interpreted as follows: for an application whose order requirements can be represented by PO #9 in Figure 4, under the conditions given in Table 2, end-to-end packet delay can be reduced by 6.19% if *pBuf*-Best_LE is used as the transmission order instead of *pBuf*-Worst_LE.

4.4.1 Experiment 1: Different POs



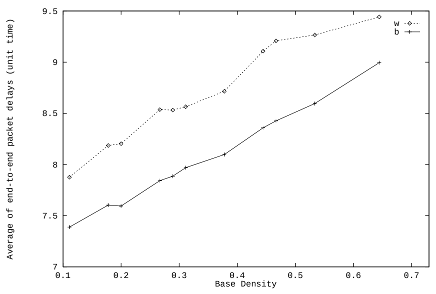


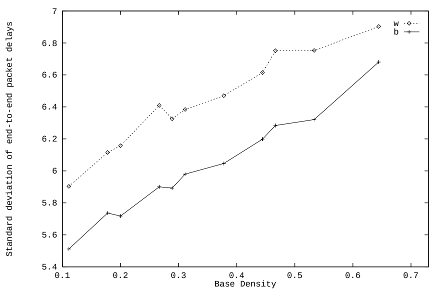


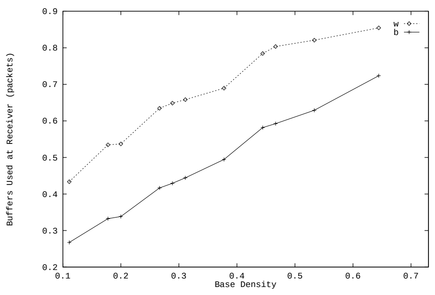
PO	Base Dens	$\overline{T_{end}}$	$\sigma_{T_{end}}$	$\overline{R\_Buff}$
1	.022	1.92	1.49	13.50
2	.067	4.54	5.20	23.59
3	.133	5.39	6.13	25.06
4	.222	7.32	7.58	27.75
5	.333	8.62	7.31	27.31
6	.467	8.35	7.00	23.97
7	.622	7.84	5.97	18.38
8	.800	5.36	3.53	9.42

Figure 5: Experiment 1: Chain-Singleton POs

The first set of experiments simulate chain-singleton POs of Figure 3, parallel-streams POs of Figure 4, and 20 random POs. These experiments quantify the effects of POs on performance gains by optimal LEs.







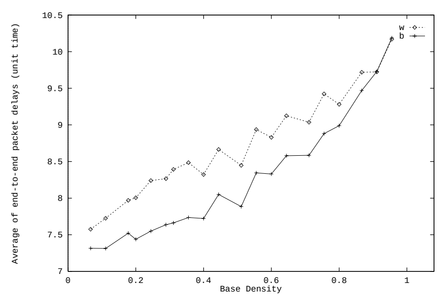
PO	Base Dens	$\overline{T_{end}}$	$\sigma_{T_{end}}$	$\overline{R\_Buff}$
9	.111	6.19	6.64	38.31
10	.178	7.12	6.20	37.78
11	.200	7.42	7.14	37.00
12	.267	8.13	7.94	34.34
13	.289	7.57	6.85	33.80
14	.311	6.95	6.34	32.51
15	.378	7.09	6.55	28.29
16	.444	8.22	6.29	25.84
17	.467	8.50	6.93	26.25
18	.533	7.23	6.40	23.36
19	.644	4.74	3.22	15.32

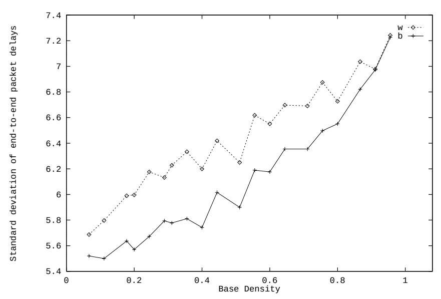
Figure 6: Experiment 1 : Parallel-Streams POs

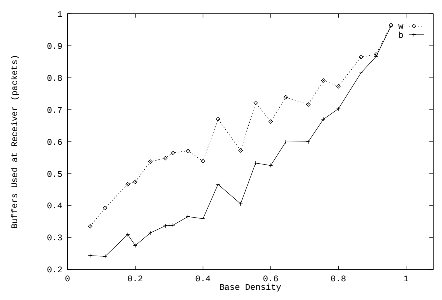
The graphs in Figures 5-7 introduce the "$\overline{T_{end}}$, $\sigma_{T_{end}}$ and $\overline{R\_Buff}$ vs base density" curves for chain-singleton, parallel-streams and random POs, respectively.[d] In general, as order constraints (i.e., density) increase, the system performs worse. End-to-end delay increases and becomes more variable, and buffer requirements increase. However, the performance using $pBuf$-Best_LE is almost always better than that using $pBuf$-Worst_LE, and sometimes significantly better. The tables in Figures 5-7 highlight the corresponding percentage gains in using $pBuf$-Best_LE over $pBuf$-Worst_LE. Well chosen LEs (i.e., small $pBuf$-metric values) almost always provide significant buffer utilization improvements over poor ones. The $\overline{R\_Buff}$ improvements are roughly $10\% - 40\%$ when $Base\ Density(PO) \leq 0.8$. Additionally, $pBuf$-Best_LE provides about $3\% - 8\%$ $\overline{T_{end}}$ and $\sigma_{T_{end}}$ improvements over $pBuf$-Worst_LE for POs having base densities between 0.05 and 0.8. In general, near-optimal LEs provide smaller delay improvements than buffer utilization improvements. Intuitively, this is because the dominant factors in packet delay and its standard deviation such as network layer delay and retransmissions due to packet and ack losses cannot be reduced

[d] *Base density* is defined as the density of just one period of a periodic partial order.

by using a better transmission order. Nevertheless, there is still some improvement obtainable in $\overline{T_{end}}$ and $\sigma_{T_{end}}$ by judicious choice of LE.







PO	Base Dens	$\overline{T_{end}}$	$\sigma_{T_{end}}$	$\overline{R\_Buff}$
20	.067	3.43	2.94	27.26
21	.111	5.35	5.13	38.54
22	.178	5.63	5.90	33.69
23	.200	7.08	7.10	41.97
24	.244	8.38	8.16	41.43
25	.289	7.61	5.52	38.54
26	.311	8.70	7.23	40.07
27	.356	8.83	8.25	36.00
28	.400	7.21	7.37	33.33
29	.444	7.09	6.28	30.40
30	.511	6.66	5.60	29.13
31	.556	6.62	6.48	26.07
32	.600	5.66	5.72	20.69
33	.644	5.97	5.12	18.96
34	.711	5.01	5.02	16.26
35	.756	5.77	5.49	15.34
36	.800	3.14	2.61	9.11
37	.867	2.56	3.06	5.71
38	.911	-.05	.08	.78
39	.956	-.17	.22	.37

Figure 7: Experiment 1: Random POs

Generally, when $0.11 \leq \textit{Base Density}(PO) \leq 0.33$, we obtain the highest buffer utilization gains by 25%, 35% and 40% for chain-singleton, parallel-streams and random POs, respectively. Similarly, when $0.11 \leq \textit{Base Density}(PO) \leq 0.55$, *pBuf*-Best_LE provides the highest $\overline{T_{end}}$ and $\sigma_{T_{end}}$ improvements over *pBuf*-Worst_LE by about $7\% - 8\%$. Thus, we can say that, when the base density of PO is between 0.11 and 0.33, near-optimal LEs provide the largest improvements in all three performance statistics studied.

As the base density increases from 0.33 to 1 (i.e., PO converges to a chain), or as the base density decreases from 0.11 to 0 (i.e., PO converges to an antichain), near-optimal LEs provide decreasingly less performance improvement. This conclusion is supported by the graphs in Figures 5 and 7 for chain-singleton and random

*PO*s, respectively.[e] In general, as base density increases, the performance difference between *pBuf*-Best_LE and *pBuf*-Worst_LE first increases. Then, after some point (e.g., base density=0.3 or 0.5), these two performances start converging. Finally, when *Base Density*$(PO) \geq 0.85$ in Figure 7, the two performances overlap. The small negative values for *PO*s 38 and 39 are due either to statistical error in the simulation or the approximate nature of *pBuf*-metric.

Based on these experiments, we conclude that *pBuf*-Best_LE provides considerable buffer utilization, and some delay improvements over *pBuf*-Worst_LE when *PO* is not an antichain or not close to a chain. Thus, in general, the choice of *LE* is important for certain classes of *PO*s.

4.4.2 Experiment 2: Different Network Layer Delays

These experiments study how network layer delay impacts the performance gains of using near-optimal *LE*s. As the network layer delay between the sender and the receiver increases, does it become more important to wisely select a transmission order? We investigate this question by simulating six different network layer delay values in the interval $[1.5, 4]$. When network layer delay is 1.5, the sender buffer size will roughly be equal to the pipesize (i.e., delay-bandwidth product of the system). At a network layer delay of 4, the sender buffer size will be about half the pipesize.

These experiments[f] show that at all network layer delays simulated, *pBuf*-Best_LE provides roughly 6.5%, 6.5% and 26% improvements over *pBuf*-Worst_LE in average end-to-end packet delay, standard deviation end-to-end packet delay and receiver buffer utilization, respectively. Hence, regardless of the network delay between the sender and the receiver, the improvements by near-optimal *LE*s remain significant for buffer utilization, and minor for packet delay and its standard deviation. Thus, the choice of *LE* is important at all network layer delays.

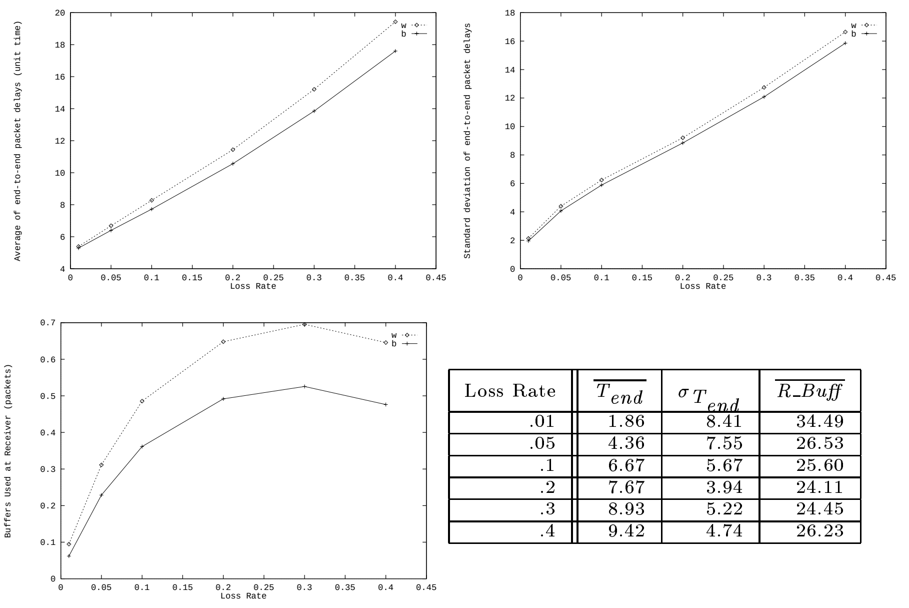
4.4.3 Experiment 3: Different Loss Rates

These experiments study the effects of different loss levels on *LE* selection by simulating loss rates in the interval [0.05, 0.4]. The table in Figure 8 introduces the percentage improvements in three performance statistics by using *pBuf*-Best_LE over *pBuf*-Worst_LE, while the graphs illustrate the absolute gains.

Judging solely by the percentage improvements, we would conclude that the choice of *LE* is important at all loss levels for $\sigma_{T_{end}}$ and $\overline{R\_Buff}$, and at loss levels $\geq$ 0.05 for $\overline{T_{end}}$. However, at low loss rates, the absolute gains for all three performance statistics appear insignificant. This is consistent with the fact that partially ordered service is generally no better than ordered service at low loss rates. We therefore conclude that only at loss rates greater than 0.1, the choice of *LE* really matters, and the greatest impact is on receiver buffer utilization.

[e] Notice that since we only simulated base densities between 0.1 and 0.65 for parallel-streams *PO*s, Figure 6 does not illustrate this behavior.

[f] Due to space limitations, no table and graphs are provided for the second set of experiments.



Loss Rate	$\overline{T_{end}}$	$\sigma_{T_{end}}$	$\overline{R\_Buff}$
.01	1.86	8.41	34.49
.05	4.36	7.55	26.53
.1	6.67	5.67	25.60
.2	7.67	3.94	24.11
.3	8.93	5.22	24.45
.4	9.42	4.74	26.23

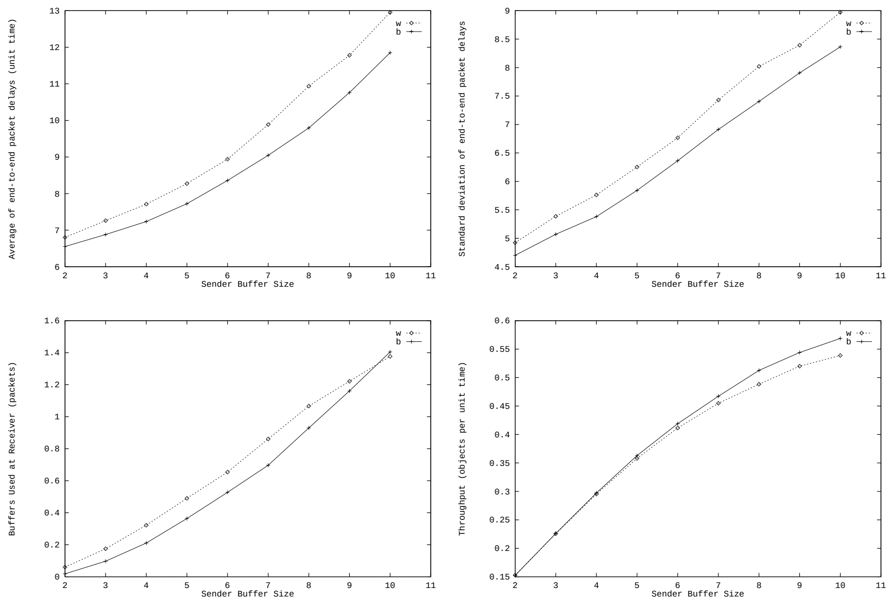
Figure 8: Experiment 3

4.4.4 Experiment 4: Different Buffer Sizes

These experiments study the effects of sender buffer size and $\frac{\text{sender buffer size}}{\text{receiver buffer size}}$ ratios on the performance improvements by near-optimal *LE*s. In the experiments, a receiver buffer size (Buf_R) of 5 and sender buffer sizes (Buf_S) of 2 through 10 are simulated. Notice that with these choices of buffer sizes, we investigate a range of values from two interesting cases: (1) $Buf_S \leq Buf_R$ and (2) $Buf_S \geq Buf_R$. Unlike in other experiments, we present the throughput results in Experiment 4 since relatively larger sender buffer sizes are simulated.

Figure 9 shows that, in general, near-optimal *LE*s provide roughly $5-10\%$ $\overline{T_{end}}$ and $\sigma_{T_{end}}$ improvements over sub-optimal *LE*s at all sender buffer sizes simulated. Results also show that the highest buffer utilization improvement by *pBuf*-Best_LE is obtained when Buf_S is small. For example, for $Buf_S = 2$, $\overline{R\_Buff}$ can be reduced by almost 70% with near-optimal *LE*s. The percentage $\overline{R\_Buff}$ improvement decreases as Buf_S increases. It is noteworthy that when sender has many buffers (i.e., when $Buf_S = 10$), *pBuf*-Best_LE performs slightly worse than *pBuf*-Worst_LE in terms of buffer utilization. We conclude that the choice of *LE* does not influence receiver buffer utilization at large sender buffer sizes.

This experiment also shows that near-optimal *LE*s do not provide any throughput improvement over nonoptimal ones when $\frac{Buf_S}{Buf_R} \leq 1$. As $\frac{Buf_S}{Buf_R}$ increases beyond



Buf_S	$\overline{T_{end}}$	$\sigma_{T_{end}}$	$\overline{R\_Buff}$	λ
2	3.66	4.51	69.79	-0.41
3	5.24	5.88	44.52	0.21
4	6.15	6.62	34.58	0.71
5	6.64	6.57	25.71	1.38
6	6.53	5.95	19.31	1.82
7	8.49	6.96	19.08	2.73
8	10.44	7.67	12.81	4.95
9	8.66	5.78	4.91	4.61
10	8.51	6.77	-2.03	5.57

Figure 9: Experiment 4

one, near-optimal LEs start providing increasingly greater throughput than bad ones. For example, when $\frac{Buf_S}{Buf_R} = 2$, the throughput with $pBuf$-Best_LE is roughly 5.5% greater than that with $pBuf$-Worst_LE. Thus, near-optimal LEs can provide throughput improvement when sender buffer size is much larger than receiver buffer size.

Generally, equal sender and receiver buffer sizes are employed in most transport layer protocols (e.g., TCP). Thus, for most practical cases, the choice of LE is important with respect to delay and receiver buffer utilization, but not with respect to throughput.

4.5 Which Metric To Use: *pBuf*-metric vs α-metric?

Section 4.4 shows that under certain conditions, *pBuf*-Best_LE provides some delay and significant buffer utilization improvements over *pBuf*-Worst_LE. Thus, whenever possible, a sender should use a near-optimal LE as transmission order. In this section, we determine how effective α-metric is in distinguishing near-optimal LEs from poor ones. We do so by comparing the performance of *pBuf*-Best_LE with that of α-Best_LE using the same system parameters as those of Experiments 1-4.

The results of Experiments 1-3 show that λ, $\overline{T_{end}}$, $\sigma_{T_{end}}$ and $\overline{R\_Buff}$ with α-Best_LE are respectively within roughly 0.5%, 1%, 2% and 4% of those with *pBuf*-Best_LE. In general, these experiments do not show any tendency that either of these LEs outperform the other. That is, in some simulations, α-Best_LE performs slightly better than *pBuf*-Best_LE, while in some others, just the opposite. Hence, based on the first three experiments, when $Buf_S = Buf_R$, *pBuf*-metric is no better a metric than α-metric in quantifying the goodness of a LE.

The results of Experiment 4, however, slightly favor *pBuf*-metric at larger sender buffer sizes. α-Best_LE performs as well as *pBuf*-Best_LE when $\frac{Buf_S}{Buf_R} \leq 1$. On the other hand, when $\frac{Buf_S}{Buf_R}$ increases beyond one, *pBuf*-Best_LE provides some minor performance improvements.

Since, in practice, equal buffer sizes both at sender and receiver are generally used, we can conclude that for most practical purposes, one can use α-metric instead of the complex *pBuf*-metric in evaluating LEs.

5 Summary and Future Work

Previous work demonstrates that under certain network conditions, a partially ordered service provides better performance than an ordered service [9]. Given a partially ordered service, this paper investigates whether or not the careful selection of a linear extension can further improve system performance, and if so, how much the performance is improved. We first show how to identify near-optimal linear extensions of a partial order, and then simulate the performance gains by using such linear extensions at the time of transmission.

To quantify linear extensions, we propose a new metric (*pBuf*-metric) that is based on buffering probabilities and derived from analytic results [8]. Since *pBuf*-metric has a complex expression, we propose a simplified version of it, called α-metric, as an alternative metric. Both metrics act as accurate predictors of expected performance of different linear extensions of POs and can be used by a transport sender in selecting a better transmission order.

Simulation study shows that, for certain POs, the choice of a near-optimal LE does provide some delay, and considerable buffer utilization improvements over a non-optimal linear extension. By prudent transmission order selection, system performance can be improved. Results also show that, as a transmission order, α-Best_LE (LE of PO optimized by α-metric) performs almost as well as *pBuf*-Best_LE (LE of PO optimized by *pBuf*-metric). Thus, in general, we can say that α-metric is as effective as $pBuf$-metric in determining near-optimal linear extensions of a partial order.

Given two LEs of a PO, we can easily decide on which one is a better transmission order by comparing their $pBuf$-metric or α-metric values. On the other hand, for a given PO, finding $pBuf$-Best_LE or α-Best_LE is difficult. Currently, we have no polynomial time algorithm to find these LEs. It is unlikely to investigate algorithmic aspects of LE optimization based on $pBuf$-metric because of its complex expression. The authors are pursuing algorithms and heuristics that can be used in finding α-Best_LE(s).

In this paper, we study the better transmission orders in a partially ordered service based on expected system behavior. That is, by generalizing the possible uncertainties such as loss of an object, a near-optimal LE is determined statically before the communication starts, and that LE is used throughout the communication as the transmission order. But what happens as system conditions change? An open problem is how to dynamically determine an optimal transmission order based on the actual system behavior while a communication is in progress. That is, as system changes, can the sending transport entity dynamically choose the next object to be transmitted in a way that optimizes the system performance? In such a method, the next object to be transmitted is unknown until just before it is transmitted. The authors are pursuing dynamic LE optimization.

References

[1] Paul D. Amer, C. Chassot, Thomas J. Connolly, Phillip T. Conrad, and M. Diaz. Partial order transport service for multimedia and other applications. *IEEE/ACM Trans on Networking*, 2(5), 440–456, Oct 1994.

[2] Nate Baxter, Herman Chien, Andy Loreen, Kathryn Marshall, and Steven Baraniuk. *OPNET Manual.* MIL 3, Inc, 1993.

[3] Thomas J. Connolly, Paul D. Amer, and Phillip T. Conrad. RFC-1693, An Extension to TCP: Partial Order Service.

[4] Phillip T. Conrad, Edward Golden, Paul. D. Amer, and Rahmi Marasli. A Multimedia Document Retrieval System Using Partially-Ordered/Partially-Reliable Transport Service. In *Multimedia Computing and Networking 1996 (MMCN96; sponsored by SPIE/IS&T)*, San Jose, CA, USA, Jan 1996.

[5] M. Diaz, A. Lozes, C. Chassot, and P. Amer. Partial order connections: a new concept for high speed and multimedia services and protocols. *Annals of Telecommunications*, 49(5-6), 270–281, May 1994.

[6] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[7] William V. Gehrlein. On Methods for Generating Random Partial Orders. *Operations Research Letters*, 5(6), 285–291, December 1986.

[8] Rahmi Marasli, Paul D. Amer, and Phillip T. Conrad. An Analytic Study of Partially Ordered Transport Services. (Submitted for publication).

[9] Rahmi Marasli. *Partially Ordered and Partially Reliable Transport Protocols: Performance Analysis.* PhD thesis, University of Delaware, (In progress).