Available online at www.sciencedirect.com
SCIENCE DIRECT®
Performance Evaluation 61 (2005) 299–328
PERFORMANCE EVALUATION
An International Journal
www.elsevier.com/locate/peva

Analysis on packet resequencing for reliable network protocols

Ye Xia[a,*], David Tse[b]

[a] *Computer and Information Science and Engineering Department, University of Florida, Room 301, CSE Building, P.O. Box 116120, Gainesville, FL 32611-6120, USA*
[b] *Electrical Engineering and Computer Science Department, University of California, 261M Cory Hall, Berkeley, CA 94720-1770, USA*

Received 3 April 2004; received in revised form 22 September 2004
Available online 11 November 2004

Abstract

Packets are sometimes disordered in the network. Reliable protocols such as TCP require packets to be accepted, i.e., delivered to the receiving application, in the order they are transmitted at the sender. In order to do so, the receiver's transport layer must resequence the packets with the help of a resequencing buffer. Even if the application can consume the packets infinitely fast, the packets may still be delayed for resequencing. In this paper, we model packet disordering by adding an independently and identically distributed (IID) random propagation delay to each packet and analyze the required buffer size for packet resequencing and the resequencing delay for an average packet. We demonstrate that these two quantities can be significant and show how they scale with the network bandwidth.

Keywords: Resequencing queue; Packet disordering; Transmission control protocol

1. Introduction

Packets can be disordered by the communication networks for various reasons [17]. For instance, with the help of the destination address contained in every packet, the network can deliberately route packets via different paths to the destination, possibly for load balancing or for reducing transfer delay. Some packets may be dropped when the network is congested or when the packet is corrupted. For reliable

* Corresponding author. Tel.: +1 352 392 2714; fax: +1 352 392 1220.
E-mail address: yx1@cise.ufl.edu (Y. Xia).

0166-5316/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2004.09.002

communication, the sender must retransmit the dropped packet, possibly causing it to arrive out-of-order at the receiver.

Most applications can only accept packets (which contain application-level data) in the same order they are transmitted at the sender. They typically rely on reliable transport protocols, such as the transmission control protocol (TCP), to temporarily buffer out-of-order packets and to resequence them as new packets arrive. The study of packet disordering and resequencing is important because of the following performance implications:

- Insufficient buffer size causes packet losses and reduced throughput.
- Even when the application can consume the packets infinitely fast, the packets may still suffer resequencing delay, which increases the response time of the application.
- The large number of queued packets create bursty load to the processor. Long queue length is typically the result of one or a few very late packets. During the time of queue build up, the processor stays idle most of the time. When the late packets finally arrive, all queued packets are suddenly eligible for processing.
- The out-of-order packets that have arrived at the receiver must wait at the transport layer, consuming precious system resources such as memory and computation cycles. Since they are shared resources, an unusually large amount of out-of-order packets can negatively affect all applications in the same system.
- For network queues, many performance measures typically improve when link bandwidth scales up. For example, aggregate traffic may become smoother; router throughput may improve; given the same traffic load, the delay at the router queue may be reduced. We will see that this is not the case for the resequencing delay and queue size.

In this paper, we model packet disordering by adding an independently and identically distributed (IID) random propagation delay to each packet and analyze the required buffer size for packet resequencing and the resequencing delay for an average packet. We demonstrate that these two quantities can be significant and show how they scale with the network bandwidth. The models formulated in this paper are fairly general and may have applications in other areas, such as the distributed database or link-layer automatic repeat request (ARQ). The results of this paper caution us to carefully consider the consequence of a fundamental principle of the packet network, that is, each packet contains its destination address and can be routed independently.

This paper is organized as follows. In the reminder of this section, we describe the resequencing model analyzed in this paper, discuss related models that have been studied previously, and give an overview of the results. We then give two variations of the resequencing model. In Section 2, we describe the results for the first variation, called *stop-and-wait ARQ*. The focus of the paper is on the second variation, called *selective-repeat ARQ*, which is presented in Section 3. The conclusion is presented in Section 4.

1.1. Network model

We will examine the model shown in Fig. 1, where the sender and the receiver are separated by a network that causes a random variable delay on each data packet, in addition to a fixed propagation delay. The transmission capacity of the sender is denoted by C_s. When packets are ready to be accepted, the receiver can consume them at the capacity, C_r. Typically, we assume $C_r = \infty$. Each transmitted packet

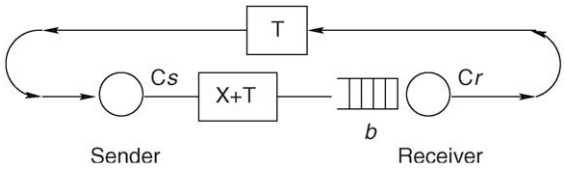

Fig. 1. The network model.

contains a sequence number, known also as the packet identification (ID). The sender transmits new packets in increasing order of the packet ID. The fixed delay experienced by each packet is denoted by T, and the variable delay experienced by packet i is denoted by X_i. We assume that the $\{X_i\}$'s are independently and identically distributed random variables. The receiver needs to accept packets *in order*. It may temporarily store out-of-order packets in its resequencing buffer of size b. When a packet that has just arrived finds the buffer full, *the packet with the largest ID among all stored and incoming packets is dropped*. The receiver sends perfect feedback information to the sender about the reception status of each packet, subject to a fixed delay, T. The sender must retransmit the dropped packet at a later time.

1.2. Related previous studies

1.2.1. ARQ models

There can be many causes for packet disordering in the network, many of which are not precisely known at the present. As stated previously, the retransmission of dropped packets and multi-path routing are among them in today's and possibly future networks. Many previous studies in fact treat these two causes separately. Disordering caused by packet retransmission is studied within the context of automatic repeat request protocol [23,13,15,20,1,18,19]. In these studies, ARQ is typically considered as a link-layer protocol running between a sender–receiver pair over a noisy link with constant propagation delay. The sender must retransmit corrupted or dropped packets based on the feedback information it gets from the receiver. Models in this family can not be easily combined into a generic model. Their details and analytical techniques involved differ greatly. Their strength lies in that they typically can model the feedback from the receiver to the sender. We will call the models in this family *ARQ models*.

Note that the end-to-end reliable transport protocols, such as TCP, resemble the link-layer ARQ protocol in the areas of packet retransmission at the sender and resequencing at the receiver. In this paper, we borrow the term ARQ even though what we have is not the same as the link-layer ARQ. In particular, we allow causes for packet disordering other than packet retransmission.

Many previous studies on ARQ models focused on the throughput of the ARQ protocol, or the delay and queue size at the sender side. For instance, Miller and Lin [15] analyzed the throughput for certain selective-repeat ARQ schemes. Towsley and Wolf analyzed the queue size and delay at the sender side for the stop-and-wait ARQ and the go-back-N ARQ in [23], and mean queue length for the stutter-go-back-N ARQ in [24]. Konheim [13] analyzed a go-back-N ARQ and a selective-repeat ARQ. Anagnostou and Protonotarios [1] analyzed the queue size and delay at the sender side in a selective-repeat ARQ model.

There are also several studies on the resequencing delay and queue size at the receiver in the ARQ literature. Rosberg and Shacham [18] analyzed a specific selective-repeat ARQ protocol over a noisy forward channel from the sender to the receiver and a perfect feedback channel. The time is divided into intervals, each with length equal to N packet transmission time slots. Suppose on interval i and slot j,

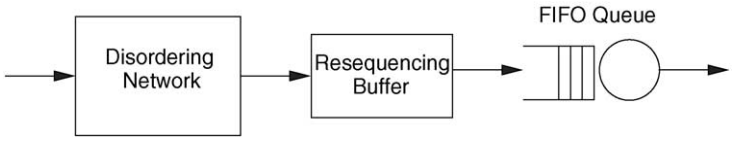

Fig. 2. Resequencing network model followed by a GI/GI/1 queue.

packet k is transmitted. On interval $i+1$ and slot j, if packet k is successfully acknowledged (ACKed), the sender transmits the new packet with the smallest sequence number. If packet k is negatively acknowledged (NAKed) or no feedback information has been received for it (i.e., a timeout has occurred), the sender retransmits packet k. The distributions of the buffer occupancy and the resequencing delay at the receiver were derived. Rosberg and Sidi [19] extended the above model to allow non-greedy source.

We briefly list several other studies of the resequencing problem in the ARQ literature. Shacham and Towsley [22] considered the resequencing problem for a multicast selective-repeat ARQ. Shacham and Shin [21] analyzed the resequencing problem of a selective-repeat ARQ with parallel channels, using a discrete-time model. The delays on the channels are all equal to a constant and the packet losses are independent from time slot to slot and among different channels. Varma [25] and Ayoun and Rosberg [2] considered optimal control problems in a queue with two servers of different service rates. The question is how to assign the customers to the servers so as to minimize the end-to-end delay [25] or the long-run average holding costs of the customers [2]. Packets get disordered at the server-assignment stage and are required to be resequenced after leaving the two-server queue.

1.2.2. Open queueing models

The studies on packet disordering due to multi-path routing (also including parallel processing or load balancing) typically analyze an open queueing network, with no feedback and no retransmission. Fig. 2 shows a generic model, where packets (or customers) numbered sequentially arrive at the system following some stochastic process, get disordered by the disordering network and are resequenced at the resequencing buffer. In some studies, a FIFO queue follows the resequencing buffer. The disordering network is also modelled as a queueing system, whose type typically distinguishes different studies. For instance, the disordering network is an M/M/∞ queue in [12], an M/GI/∞ queue in [9], a GI/GI/∞ queue in [3], an M/M/2 queue in [14], an M/M/K queue in [27], an M/H_2/K queue in [5], an M/M/2 queue with a threshold-type server-assignment policy in [10], two parallel M/M/1 queues with additional fixed propagation delays in [8], and K parallel M/GI/1 queues in [11]. A survey is given in [4]. Most of these studies are concerned mostly with finding the distribution and/or the mean of the resequencing delay or the end-to-end delay. Several also give results on the number of customers in the resequencing queue.

1.3. Summary of results

We analyze two different variations of the model shown in Fig. 1. In the first case, discussed in Section 2, we assume the sender's capacity, C_s, is infinitely large so that it can dump a block of packets onto the network instantaneously. Here, it makes sense to consider the stop-and-wait-n ARQ protocol,[1] where, at

[1] In this paper, we borrow the term ARQ from the commonly known link-layer ARQ. Stop-and-wait-n ARQ is a variation of the link-layer stop-and-wait ARQ.

the beginning of each fixed time interval, the sender transmits a block of n packets simultaneously. The receiving status of the packets reaches the sender at the end of the interval. The main result is, for large n, if we want to accept a fraction α in each block of n packets, the resequencing buffer size must be αn. As a result, for any fixed buffer size, the fraction of accepted packets in each block becomes vanishingly small when the block size n approaches infinity.

When the sending capacity is limited, we consider the selective-repeat ARQ, as discussed in Section 3. In this case, the sender transmits one packet in each time slot. When a packet is rejected at the receiver due to buffer overflow, this information is fed back to the sender. In each time slot, the sender always retransmits the rejected packet with the smallest packet sequence number and only transmits a new packet when there are no rejected packets. For the selective-repeat ARQ, we have results for the buffer requirement to achieve small packet rejection ratio, or equivalently, near 100% throughput. When the variable packet delay is exponential with mean $1/\lambda$, we show that

$$P\{\bar{Q}(t) > m\} \approx \frac{\mathrm{e}^{-(m+2)\lambda\tau}}{1-\mathrm{e}^{-\lambda\tau}} \tag{1}$$

where $\bar{Q}(t)$ is an upper bound on the resequencing queue size and τ is the packet transmission time at the sender. Here, $\tau = 1/C_\mathrm{s}$ if the unit of C_s is packet per second, or $\tau = U/C_\mathrm{s}$ if C_s is in bits per second and U is the packet size in bits. For the Pareto delay distribution with CDF $F(x) = 1 - K^\alpha x^{-\alpha}$, where $x \geq K$, we have

$$P\{\bar{Q}(t) > m\} \approx \frac{K^\alpha}{(\alpha-1)\tau}((k^* - k_0 + m + 2)\tau)^{-\alpha+1}, \tag{2}$$

where $k^* - k_0$ is a small number compared with the queue size m for which the above approximation holds.

Our analysis in Section 3 assumes the buffer size is infinite. This simplifies the analysis since no packets ever get rejected at the resequencing queue. We use the probability that the queue size exceeds a threshold, b, as the approximation for the packet rejection (or loss) ratio when the buffer size is b. This approximation can be expected to be accurate only when the buffer size is large, and hence, the probabilities involved are very small. As a result, we do not have results for the packet loss ratio for small buffer sizes. We supplement this deficiency with a set of simulation results for the finite buffer case in Section 3.3.

Again assuming the resequencing buffer is infinite, the packet's waiting time in the queue before it is accepted, also called the resequencing delay, can be computed in principle. We do not have concise expressions for it in either the exponential or the Pareto case. For the exponential case, we do have a simple approximation for the expected waiting time at the resequencing buffer

$$\mathbf{E}[W] \approx \frac{1}{\lambda}\left(\log\left(\frac{1}{2(1-\mathrm{e}^{-\lambda\tau})}\right) + 0.5\right), \tag{3}$$

where W is the waiting time. When $C_\mathrm{s} \geq 10$ Mbps, we can further approximate $\mathbf{E}[W]$ and get

$$\mathbf{E}[W] \approx \frac{1}{\lambda}\left(\log\left(\frac{C_\mathrm{s}}{2\lambda U}\right) + 0.5\right), \tag{4}$$

where U is the packet size in bits. We see that the mean waiting time of a packet scales logarithmically with the link capacity. We also see that, at a given link speed, reducing the packet size, and hence, the packet transmission time, increases the mean waiting time.

The above analysis for the expected waiting time uses the memoryless property of the exponential distribution. The technique does not apply to the case where the network delay of the packet is Pareto. With simulation, we can show that the expected waiting time can be very large in this case. To reduce the resequencing delay, it is very helpful to add a bound, say d, to the Pareto delay. In practice, this can be approximated by retransmitting the unacknowledged packets at appropriate times.

A feature of this paper is, when possible, we rely on approximations and bounds to get simple and easily interpretable results. This is complementary to many previous studies whose solutions are exact but complicated and are hard to construe.

1.4. Discussions

Our models can be compared with the model in [18]. In [18], retransmissions of the same packet are always separated by N time slots. Furthermore, the packet delay in the network is due to the time it takes to retransmit an erroneous or dropped packet. Its statistics is completely specified by the ARQ protocol. Our models allow different delay statistics, and therefore, are more versatile in modelling different causes of delays. In our paper, we give results for the exponential and Pareto network delay distributions.

Our simulation experiments handle packet retransmissions faithfully, as specified by each of the ARQ protocols. Our analysis never handles that aspect. Therefore, from analytical point of view, our model falls in the class of models shown in Fig. 2. We choose to model the causes for packet disordering by adding IID random delays to the packets for both simplicity and for generality. This is equivalent to saying that the packet disordering network is a D/M/∞ queue in the exponential delay case, and is a D/P/∞ queue in the Pareto delay case, where "P" stands for Pareto. In reality, the variable packet delays are most likely correlated. For instance, multi-path routing is probably better modelled as K-parallel $\cdot$/GI/1 queues. The IID delay model is still valuable because we typically do not know the parameters that give a realistic model, such as the value of K, the dispatching policy to each of the queues, and the server rates of the queues.

At this point, a natural question is what distribution we should choose for the random delay. We choose the exponential distribution whose tail probability decays exponentially and the Pareto distribution whose tail probability has a power-law decay and is said to have a heavy tail. We will see that the tail probability is a very important factor to the resequencing delay and the buffer requirement. In some sense, the exponential distribution represents a "nice" case and the Pareto distribution represents a "bad" case. To be more precise, the exponential distribution has CDF

$$F(x) = 1 - \mathrm{e}^{-\lambda x},$$

where $1/\lambda$ is the mean. As stated earlier, the Pareto distribution has the CDF, for $x > K$,

$$F(x) = 1 - \left(\frac{K}{x}\right)^{\alpha}.$$

Here, α governs the rate of decay of the tail probability and is the important parameter. We require $\alpha > 1$ in order for the mean to exist, which is $\alpha K/(\alpha - 1)$. In our simulation and numerical examples, we typically fix α and the mean, which together determine K.

2. Stop-and-wait-*n* ARQ

In this section, we assume that the sender's capacity and the receiver capacity are both infinite. That is, a packet can be transmitted instantaneously at the sender, and when all preceding packets have arrived at the receiver, it can be accepted instantaneously. We also assume that the fixed propagation time, T, is zero. The resequencing buffer has a finite size b.

Data packets transmitted at the sender are assigned sequential packet IDs, starting from 1. We require that they must be presented to the receiving application in increasing order of their IDs. In this section, we assume the stop-and-wait-n ARQ protocol is used, where n is the block size and $1 \leq b \leq n$. More specifically, let the time be divided into intervals of identical length. At the beginning of each interval, the sender transmits a block of n packets simultaneously. These packets experience IID random network delays, denoted by X_i's. In this section, we assume the network delays are bounded by the interval length. Hence, by the end of the interval, all n packets in the block have arrived at the receiver and the arrival order is a random permutation with the uniform distribution on the set of permutations. The receiver re-orders the packets in the resequencing buffer, and immediately accepts all packets that do not leave sequencing gaps in the accepted packets. When an incoming packet finds the resequencing buffer full, the receiver drops the packet with the largest sequence number. At the end of the interval, the receiver also drops all remaining packets in the buffer, i.e., the ones that cannot be accepted due to missing packets with smaller packet IDs. The receiver sends perfect and instantaneous feedback to the sender during the interval. By the end of the interval, the sender knows which packets have been accepted and which have been dropped. In the following interval, the sender transmits n more packets, sequentially numbered, starting from the next packet ID expected by the receiving application.

Take the example of $n = 3$. Suppose the buffer size $b = 1$ and suppose the order of packet arrival at the receiver is 2, 1 and 3. When packet 2 arrives at the receiver, it will be stored in the resequencing buffer. When packet 1 arrives, packet 2 will be evicted from the buffer and dropped because there is no additional buffer space to hold it. Packet 1 enters the buffer and is immediately accepted by the application. Packet 3 will not be accepted because 2 is missing. In our case, we will drop it at the end of the current interval. In the next interval, the sender will transmit packets 2–4.

Let the number of packets accepted by the receiver in interval i be N_i, $i = 1, 2, \ldots$. The above algorithm makes $\{N_i\}$ an IID random sequence. Let the interval length be L. Then, the long time throughput of the communication system is simply $\mathbf{E}N_i/L$. To improve the throughput, one can increase $\mathbf{E}N_i$ or reduce L. We investigate what quantities $\mathbf{E}N_i$ depends on. Define the packet acceptance ratio $\rho(n, b) = \mathbf{E}N_i/n$. The main result is the following theorem.

Theorem 2.1. *Let integer $n \geq 1$ be the block size and let integer b be the buffer size. Then, for $1 \leq b \leq n$,*

$$\rho(n, b) = \frac{1}{n}\left(\sum_{k=b}^{n} \frac{b!\, b^{k-b}}{k!} + b - 1\right) \tag{5}$$

Proof. Let J_k be the set $\{1, 2, \ldots, k\}$, where k can vary from 1 to n. First, note that the order of packet arrival at the receiver during each interval is the random permutation of the set J_n with a uniform probability

distribution. To compute $\mathbf{E}N$,[2] we will use

$$\mathbf{E}N = \sum_{k=1}^{n} P\{N \geq k\}.$$

The event $\{N \geq k\}$ is the same as the event $\{\text{all } i \in J_k \text{ are accepted}\}$, denoted by E_k. Given an arrival sequence of the n packets, denoted by Π, we only need to focus on the sub-sequence of Π generated by restricting Π to the set J_k, when we consider the event E_k. Denote this sub-sequence Π_k. It is easy to see that the event E_k occurs in Π if and only if it occurs in Π_k, due to the rule of rejecting the packet with the largest ID when the buffer is full.

For any set $S \subseteq J_n$ of contiguous packets, a permutation of S is said to be b-acceptable if it can be arranged in sequence with the help of a resequencing buffer of size b. For instance, the permutation (3, 2, 4) of $S = \{2, 3, 4\}$ is 2-acceptable, but not 1-acceptable. It is b-acceptable for any $b \geq 2$. We will count the number b-acceptable permutations of J_k. Denote this number by A_k. Note that a permutation on J_k is b-acceptable if and only if none of the packets in J_k is dropped. We claim

$$A_k = \begin{cases} k! & \text{for } 1 \leq k \leq b, \\ b!\, b^{k-b} & \text{for } b < k \leq n. \end{cases} \tag{6}$$

For $1 \leq k \leq b$, it is obvious that, any of the $k!$ permutations of J_k is b-acceptable, since the buffer size is large enough to hold all of them.

For $b < k \leq n$, a permutation Π_k on J_k can be accepted if and only if (1) packet 1 is in one of the first b positions in Π_k and (2) $\Pi_k \backslash \{1\}$ is b-acceptable. Here, $\Pi_k \backslash \{1\}$ denotes the permutation Π_k with 1 removed, which is an permutation on the set $\{2, 3, \ldots, k\}$. Since the number of b-acceptable permutations on $\{2, 3, \ldots, k\}$ is A_{k-1} and packet 1 can be in any of the first b positions, we have, for $b < k \leq n$,

$$A_k = bA_{k-1}.$$

By iterating the index k from b, we get (6).

Since all permutations of the set J_k are equally likely to be the packet arrival orders, we have

$$\mathbf{E}N = \sum_{k=1}^{n} P\{N \geq k\} = \sum_{k=1}^{n} \frac{A_k}{k!} = \sum_{k=b}^{n} \frac{b!\, b^{k-b}}{k!} + \sum_{k=1}^{b-1} \frac{k!}{k!} = \sum_{k=b}^{n} \frac{b!\, b^{k-b}}{k!} + b - 1.$$

We next study some features of the function $\rho(n, b)$. □

Theorem 2.2. *For any $0 \leq \alpha \leq 1$, let $b = \lfloor \alpha n \rfloor$.*[3] *Then, $\rho(n, b) \to \alpha$, as $n \to \infty$.*

Proof. The proof uses a standard convergence argument. It is sufficient to show, for $b = \lfloor \alpha n \rfloor$, $(1/n) \sum_{k=b}^{n} b!\, b^{k-b}/k! \to 0$ as $n \to \infty$. We drop the floor notation for simplicity, with the understanding that the relevant quantities are integers. Let $m = n/l$, for some positive number $l > 1$. Split

[2] We have omitted the index to the interval, i.

[3] $\lfloor x \rfloor$ stands for the floor of x, i.e., the largest integer less than or equal to x.

$\sum_{k=b}^{n} b!\, b^{k-b}/k!$ into two parts: $\sum_{k=b}^{m-1} b!\, b^{k-b}/k!$ and $\sum_{k=m}^{n} b! b^{k-b}/k!$. Notice that the ratio between the $(i+1)$th term and ith term in the sum is $\alpha n(\alpha n + i)$. For $i \geq m$,

$$\frac{\alpha n}{\alpha n + i} \leq \frac{\alpha n}{\alpha n + n/l} = r < 1$$

where $r = \alpha/(\alpha + 1/l)$ by definition. For $b = \alpha n$,

$$\begin{aligned}\frac{1}{n}\sum_{k=m}^{n}\frac{b!\, b^{k-b}}{k!} &\leq \frac{1}{n}\frac{(\alpha n)^m}{(\alpha n+1)(\alpha n+2)\cdots(\alpha n+m)}(1 + r + r^2 + \cdots + r^{n-\alpha n-m+1}) \\ &= \frac{1}{n}\frac{(\alpha n)^m}{(\alpha n+1)(\alpha n+2)\cdots(\alpha n+m)}\frac{1-r^{n-\alpha n-m+2}}{1-r} \\ &\leq \frac{1}{n}\frac{1-r^{n-\alpha n-m+2}}{1-r} \longrightarrow 0 \quad \text{as } n \longrightarrow \infty, \end{aligned} \tag{7}$$

$$\begin{aligned}\frac{1}{n}\sum_{k=b}^{m-1}\frac{b!\, b^{k-b}}{k!} &= \frac{1}{n}\left(1 + \frac{\alpha n}{\alpha n+1} + \frac{(\alpha n)^2}{(\alpha n+1)(\alpha n+2)} + \cdots + \frac{(\alpha n)^{m-1}}{(\alpha n+1)(\alpha n+2)\cdots(\alpha n+m-1)}\right) \\ &\leq \frac{1}{n}m \longrightarrow \frac{1}{l} \quad \text{as } n \longrightarrow \infty. \end{aligned} \tag{8}$$

Combining (7) and (8), for $b = \alpha n$, and for any positive number $l > 1$, we get

$$\limsup_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}\frac{b!\, b^{k-b}}{k!} \leq \frac{1}{l}.$$

Let l goes to infinity, we get, for $b = \alpha n$,

$$\limsup_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}\frac{b!\, b^{k-b}}{k!} = 0. \qquad \square$$

Theorem 2.2 says, in order to achieve reasonable acceptance ratio, the buffer size must scale linearly with the block size, n. As an easy corollary, for any fixed buffer size b, $\lim_{n\to\infty} \rho(n, b) = 0$. These asymptotic results can be good approximations for large n. We will use numerical examples to show how large the block size n must be and what happens when n is not so large.

Fig. 3 shows the acceptance ratio ρ versus the buffer size for block size $n = 10$ and 100, respectively. In these plots, the label "limit" refers to the asymptotic limit of ρ as in Theorem 2.2, and the label "exact" refers to the exact value of ρ. We see that the asymptotic result becomes good approximation for $n > 100$ at all buffer sizes. Even at very small values of n, say, $n = 10$, the asymptotic result is not too far from the exact value.

In Fig. 4, we show the convergence of ρ to the limit as n increases while the buffer size is kept at $b = 0.5n$. In this case, the limit is 0.5.

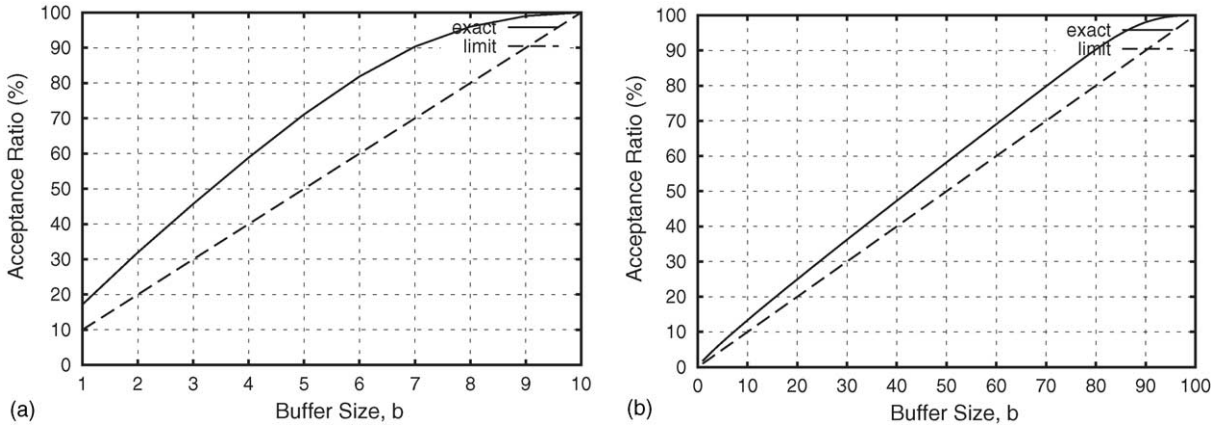

Fig. 3. Acceptance ratio vs. buffer size for block sizes: (a) $n = 10$; (b) $n = 100$.

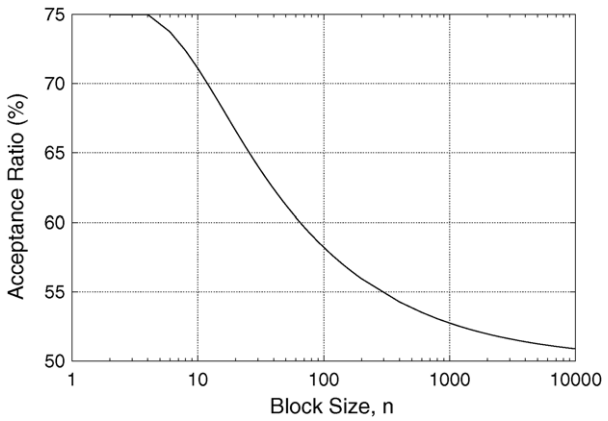

Fig. 4. Acceptance ratio converges to 50%: buffer size $b = 0.5n$.

The convergence in Theorem 2.2 can be strengthened to an almost sure convergence. Let $N(n, b)$ be the number of packets accepted by the receiver in one interval. The notation makes the dependency on the block size n and the buffer size b explicit.

Theorem 2.3. *For any* $0 \leq \alpha \leq 1$, *let* $b = \lfloor \alpha n \rfloor$. *Then,* $N(n, b)/n \to \alpha$ *almost surely as* $n \to \infty$.

Proof. For convenience, let us drop the floor notation in the buffer size b and assume $b = \alpha n$ is an integer. We wish to show

$$\lim_{n\to\infty} \frac{N(n, b) - b}{n} = 0 \quad \text{almost surely} \tag{9}$$

Fix $\epsilon > 0$. For every positive integer n, define the event

$$E_n = \left\{ \frac{N(n,b) - b}{n} \geq \epsilon \right\}.$$

Hence,

$$P\{E_n\} = P\{N(n,b) \geq b + n\epsilon\}. \tag{10}$$

From the proof of Theorem 2.2, we know that, for $b < k \leq n$,

$$P\{N(n,b) \geq k\} = \frac{A_k}{k!} = \frac{b!\, b^{k-b}}{k!}. \tag{11}$$

Hence, assuming $n\epsilon$ is rounded to an integer

$$P\{N(n,b) \geq b + n\epsilon\} = \frac{b!\, b^{n\epsilon}}{(b+n\epsilon)!} = \frac{(\alpha n)!(\alpha n)^{n\epsilon}}{((\alpha+\epsilon)n)!}. \tag{12}$$

We will use the following Stirling-type of bounds for $n!$ (see page 184 of [16]):

$$\sqrt{2\pi n} n^n \, \mathrm{e}^{-n} \, \exp\frac{1}{12n+1} < n! < \sqrt{2\pi n} n^n \, \mathrm{e}^{-n} \, \exp\frac{1}{12n}. \tag{13}$$

We get

$$\begin{aligned} \frac{(\alpha n)!(\alpha n)^{n\epsilon}}{((\alpha+\epsilon)n)!} &\leq \frac{\sqrt{2\pi\alpha n}(\alpha n)^{\alpha n} \, \mathrm{e}^{-\alpha n}(\alpha n)^{\epsilon n}}{\sqrt{2\pi(\alpha+\epsilon)n}((\alpha+\epsilon)n)^{(\alpha+\epsilon)n} \, \mathrm{e}^{-(\alpha+\epsilon)n}} \frac{\exp(1/12\alpha n)}{\exp(1/(12(\alpha+\epsilon)n+1))} \\ &= \sqrt{\frac{\alpha}{\alpha+\epsilon}} \left(\left(\frac{\alpha}{\alpha+\epsilon} \right)^{\alpha+\epsilon} \mathrm{e}^{\epsilon} \right)^n \frac{\exp(1/12\alpha n)}{\exp(1/(12(\alpha+\epsilon)n+1))}. \end{aligned} \tag{14}$$

Define

$$f(\epsilon) = \left(\frac{\alpha}{\alpha+\epsilon} \right)^{\alpha+\epsilon} \mathrm{e}^{\epsilon}.$$

We claim, $\epsilon > 0$

$$f(\epsilon) < 1.$$

To see this, note that $f(0) = 1$, and that

$$(\log f(\epsilon))' = \log \frac{\alpha}{\alpha+\epsilon} < 0$$

for $\epsilon > 0$. Hence, $f(\epsilon)$ strictly increases to 1 as ϵ decreases to 0. Next,

$$\frac{\exp(1/12\alpha n)}{\exp(1/(12(\alpha+\epsilon)n+1))} \to 1 \quad \text{as } n \to \infty.$$

Hence, the right-hand side of (14) decreases to 0 geometrically fast. Hence,

$$\sum_{n=0}^{\infty} P\{E_n\} = \sum_{n=0}^{\infty} P\left\{\frac{N(n,b)-b}{n} \geq \epsilon\right\} < \infty.$$

By the Borel–Cantelli lemma [7], the probability that E_n occurs infinitely often is 0. That is,

$$P\left\{\frac{N(n,b)-b}{n} \geq \epsilon \text{ infinitely often}\right\} = 0$$

which implies

$$P\left\{\limsup_{n\to\infty} \frac{N(n,b)-b}{n} > \epsilon\right\} = 0.$$

Or,

$$P\left\{\limsup_{n\to\infty} \frac{N(n,b)-b}{n} \leq \epsilon\right\} = 1.$$

Let ϵ decreases to 0, we see that

$$\frac{N(n,b)-b}{n} \to 0 \quad \text{almost surely as } n \to \infty. \qquad \square$$

We now know that, for large n, the fraction of packets that are not accepted is $1-\rho$. An interesting question is, out of these packets, how many are rejected due to buffer overflow and how many are discarded at the end of the interval. It can be shown that the number of rejected packets is at least $\frac{(1-\alpha)^2}{2}n$. Hence, at least a constant fraction of the packets are rejected. This result implies that, to achieve near 100% acceptance ratio, we must ensure that the size of the resequencing buffer is close to n, regardless whether we keep the remaining packets or not at the end of each interval.

3. Selective-repeat ARQ

In this section, we will study a more realistic refinement to the stop-and-wait-n ARQ, called *selective-repeat ARQ*, and find its throughput-buffer relationship. We assume that C_s is finite and the propagation delay, T, can be non-zero. In every packet time slot, the sender either transmits a new packet or retransmits a previously rejected packet. More specifically, the sender maintains a list of packets rejected by the receiver, and retransmits them in increasing order of the packet IDs. When this list is empty, it sends the next new packet. The receiver behavior and the packet-dropping rule are similar to those for the stop-and-wait-n ARQ.

Selective-repeat ARQ resembles the retransmission and resequencing behaviors of TCP and many link-layer protocols. In particular, it captures the resequencing behavior at the TCP receiver. The feedback is similar to TCP's selective acknowledgement. We will discuss in more details the relevance of our model to TCP in Section 4. We emphasize that, by assuming C_r is infinite, the receiver's processing capacity is not a

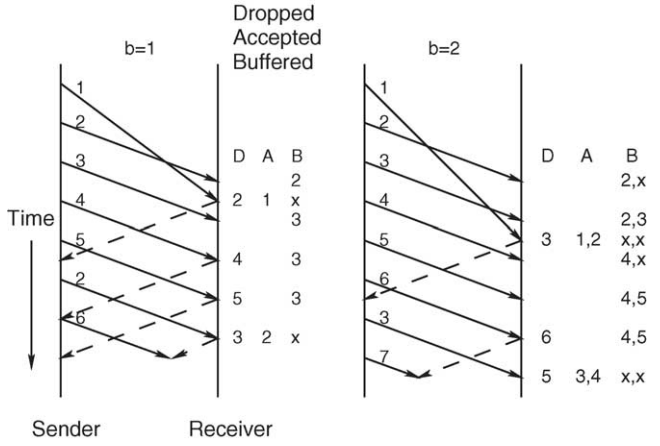

Fig. 5. Selective-repeat ARQ examples. Two cases for the buffer size: $b = 1$ (left) and $b = 2$ (right).

limitation. This models the situation that the transmission bottleneck is at the sender or the network, not at the receiver. If packets were not required to be resequenced, they would have been accepted immediately and the buffer would have been empty. The receiver's buffer is for resequencing only.

Fig. 5 shows two examples about the operation of the selective-repeat ARQ. The left-hand side is for the case of receiver buffer size $b = 1$, and the right-hand side is for the case of $b = 2$. Let us focus on the $b = 2$ case. In the figure, time progresses downward. The labels "D", "A" and "B" stand for "dropped", "accepted" and "buffered", respectively. The "x" represents an available buffer space that can store one packet. The sender transmits packets 1–3. Packet 1 is delayed in the network, and packet 2 arrives at the receiver first. Since packet 2 is not in sequence, it is stored in the receiver buffer, i.e., it is buffered. When packet 3 arrives, it is also buffered and the buffer becomes full. Next, packet 1 arrives. The packet with the largest sequence number is dropped, in this case, 3. At this point, packets 1 and 2 can be accepted in sequence, leaving an empty buffer. The receiver sends a negative feedback to the sender about the dropped packet. The feedback packet will reach the sender after the transmission of packet 6, and packet 3 will be retransmitted at that point. The next packet that arrives at the receiver is 4. Since packet 3 has not been accepted, packet 4 must be stored in the buffer. Then, packets 5 and 6 arrive, in that order. Packet 5 is stored and packet 6 is dropped. When the retransmitted packet 3 arrives next, packet 5 is evicted from the buffer, and packets 3 and 4 are accepted.

The retransmission strategy in the model is intuitively a good one. Suppose, at a fixed time, i is the smallest packet ID of all rejected packets known to the sender. Then, in terms of achievable throughput, sending i at the next packet time slot is at least as good as sending any packet greater than i. For, without i, the receiver cannot accept any packets greater than i. However, this strategy may be worse than a strategy that allows retransmission of packets whose rejection–acceptance status is unknown to the sender. We do not pursue a strategy of the latter type at this point.

3.1. Buffer size to achieve near 100% throughput

In this section, we wish to approximate the required buffer size for achieving near 100% throughput by considering an infinite buffer queue and finding the queue size m for which $\text{Prob}\{Q \leq m\} \approx 1$. Note

Fig. 6. A snapshot of the resequecing buffer.

that when the buffer size is infinite, no packets will be retransmitted. The sender simply transmits new packets one after another. Then, the selective-repeat ARQ is the same as the stop-and-wait-n ARQ with infinite block size and with finite sending capacity, C_s. Suppose the sender has been transmitting forever and suppose it starts transmitting packet 1 at time 0.[4] Then, packet k leaves the sender at $t = k\tau$, for $k \geq 1$, where τ is the packet transmission time at the sender. Let the random delay for packet k be X_k, where the $\{X_k\}$'s are IID random variables. Packet k arrives at the receiver at time $A_k = k\tau + T + X_k$, for $k = \ldots, 1, 2, \ldots$.

At time t, let $S(t)$ be the set of packets that have arrived at the receiving queue, let $M(t)$ be the largest packet in $S(t)$, and let $L(t)$ be the largest packet that has been accepted by the receiver.[5] They can be expressed as

$$S(t) = \{i : A_i \leq t\}$$
$$M(t) = \max\{i : A_i \leq t\}$$
$$L(t) = \max\{i : \max\{\ldots, A_{i-1}, A_i\} \leq t\}$$

Fig. 6 illustrates various quantities about the resequencing buffer. The stripe represents packet sequence numbers, increasing from the left to the right. To the left of the sequence number L are packets that have arrived and have been accepted. The unshaded areas are the packets that have not arrived. By definition, none of the packets greater than M have arrived yet. The darker-shaded areas between L and M represent the packets that have arrived but have not been accepted. They constitute the content of the current queue.

The queue size at time t is $Q(t) = |S(t) - \{\ldots, L(t) - 2, L(t) - 1, L(t)\}|$. It does not seem easy to keep track the set $S(t)$. Instead of computing the distribution of $Q(t)$, we will compute the distribution of an upper bound for $Q(t)$, denoted by $\bar{Q}(t) = (M(t) - L(t) - 1)^+$. Note that if there is exactly one gap between $L(t)$ and $M(t)$, that is, exactly one packet between $L(t)$ and $M(t)$ has not arrived at the receiver, then $\bar{Q}(t) = Q(t)$. In this case, the missing packet must be $L(t) + 1$. In order for $\bar{Q}(t)$ be a tight upper bound, there should be no or few gaps from $L(t)$ to $M(t)$. When the queue size is large, we have good reason to believe this is the case. For, the large queue size is typically caused by the exceptionally long delay of a very early packet that has not arrived after most other packets between $L(t)$ and $M(t)$ have arrived.

For $m = 0, 1, 2, \ldots$, we have

$$P\{\bar{Q}(t) \leq m\} = P\{L(t) \geq M(t) - m - 1\}.$$

[4] Packet ID numbers can be negative.

[5] The terms "largest" or "smallest" packet refer to the packet ID. All packets are identical in size.

We can partition the above probability with the event $\{M(t) = k\}$, for $k \le k^*$, where $k^* = \lfloor (t - T)/\tau \rfloor$ is the largest packet that can possibly arrive before time t

$$\begin{aligned} P\{\bar{Q}(t) \le m\} &= \sum_{k \le k^*} P\{\ldots, A_{k-m-2} \le t, A_{k-m-1} \le t, M(t) = k\} \\ &= \sum_{k \le k^*} P\{\ldots, A_{k-m-2} \le t, A_{k-m-1} \le t, A_k \le t, A_{k+1} > t, \ldots, A_{k^*} > t\} \\ &= \sum_{k \le k^*} P\{\ldots, A_{k-m-2} \le t, A_{k-m-1} \le t\} P\{M(t) = k\}. \end{aligned} \tag{15}$$

where we have used the fact that the A_i's are independent random variables. Due to the same independence property, it is easy to write an expression for the probability

$$P\{M(t) = k\} = P\{A_k \le t, A_{k+1} > t, \ldots, A_{k^*} > t\}.$$

The right hand of (15) is a key quantity (or 1 minus of it) we wish to compute. In Section 3.1.1, we will first compute $P\{M(t) = k\}$ and will argue that the infinite sum in (15) can be approximated by a finite sum. Then, in Section 3.1.2, we will compute $P\{\ldots, A_{k-m-2} \le t, A_{k-m-1} \le t\}$ and give the final answer to $P\{\bar{Q}(t) \le m\}$.

3.1.1. Computation of $P\{M(t) = k\}$

For exponential delay with mean $1/\lambda$,

$$P\{M(t) = k\} = (1 - \mathrm{e}^{-\lambda(t-k\tau-T)})\, \mathrm{e}^{-(k^*-k)\lambda(t-(k+1+k^*)\tau/2-T)} \tag{16}$$

For Pareto delay with parameter $K > 0$ and $\alpha > 1$,

$$P\{M(t) = k\} = \left(1 - \frac{K^\alpha}{(t - k\tau - T)^\alpha}\right) \frac{K^\alpha}{(t - (k+1)\tau - T)^\alpha} \cdots \frac{K^\alpha}{(t - k^*\tau - T)^\alpha}. \tag{17}$$

Since this number decays to 0 geometrically fast as k decreases, we can find a $k_0 < k^*$ so that $P\{M(t) = k\}$ is negligible for $k < k_0$. For practical purpose, the sum in (15) involves a small number of terms. Suppose k satisfies $P\{M(t) = k\} \le \epsilon$ for some $0 < \epsilon < 1$. In the case of exponential distribution, let us ignore the factor $(1 - \mathrm{e}^{-\lambda(t-k\tau-T)})$ in (16) since it is no greater than 1. Using $k^* = \lfloor (t - T)/\tau \rfloor$, we get

$$k^* - k \ge \sqrt{\frac{-2 \log \epsilon}{\lambda\tau}} + 1. \tag{18}$$

Hence, we can choose

$$k_0 = k^* - \left(\sqrt{\frac{-2 \log \epsilon}{\lambda\tau}} + 1\right).$$

Table 1 shows the lower bounds on $k^* - k$ obtained by using (18) (labelled "analysis") and by using (16) (labelled "exact"). The purpose of the comparison is to make sure that the simpler expression on the right hand of (18) does not lose much accuracy. This is indeed confirmed by the table of results. We also see

Table 1
Lower bound on $k^* - k$ to achieve $P\{M(t) = k\} \leq \epsilon$ for the exponential distribution

C_s (Mbps)	Lower bound on $k^* - k$: analysis/exact		
	$\epsilon = 10^{-5}$	$\epsilon = 10^{-10}$	$\epsilon = 10^{-20}$
1	8/6	10/9	14/13
10	21/20	29/28	41/40
100	63/58	89/87	125/124

$1/\lambda = 20$ ms.

Table 2
Lower bound on $k^* - k$ to achieve $K^\alpha/(t - (k+1)\tau - T)^\alpha \leq \epsilon$ for the Pareto distribution

C_s (Mbps)	Lower bound on $k^* - k$
1	2
10	13
100	123

Mean delay = 20 ms, $\alpha = 1.1$, $\epsilon = 0.1$.

Table 3
Lower bound on $k^* - k$ to achieve $P\{M(t) = k\} \leq \epsilon$ for the Pareto distribution

C_s (Mbps)	Lower bound on $k^* - k$		
	$\epsilon = 10^{-5}$	$\epsilon = 10^{-10}$	$\epsilon = 10^{-20}$
1	6	9	14
10	12	17	25
100	37	48	65

Mean delay = 20 ms, $\alpha = 1.1$.

that the values of the lower bound are not very large for very small ϵ's. Expression (18) shows that these values grow very slowly as ϵ decreases or as $\lambda\tau$ increases.

We can do similar analysis for the Pareto case. We do not have a more compact expression for (17). Hence, we will show at what value k the factor $K^\alpha/(t - (k+1)\tau - T)^\alpha$ in (17) becomes small, say 0.1. From that point on, as k continues to decrease, the value of $P\{M(t) = k\}$ will rapidly decrease toward zero. By setting $K^\alpha/(t - (k+1)\tau - T)^\alpha \leq \epsilon$, we get

$$k^* - k \approx \frac{K}{\tau \epsilon^{1/\alpha}}. \tag{19}$$

The approximation above is due to rounding real numbers to integers. For the case where $\alpha = 1.1$, $\mathbf{E}X = 20$ ms, and $\epsilon = 0.1$, the results are shown in Table 2. With exact numerical analysis on (17), the lower bound on $k^* - k$ to achieve $P\{M(t) = k\} \leq \epsilon$ is in fact very small. The results are shown in Table 3.

3.1.2. Asymptotic behavior of $P\{\bar{Q}(t) \leq m\}$

In order to compute $P\{\bar{Q}(t) \leq m\}$ as in (15), we next turn to the calculation of $a(k, m) := P\{\ldots, A_{k-m-2} \leq t, A_{k-m-1} \leq t\}$. For each m, as k decreases, $a(k, m)$ increases to 1 for the delay

distributions we are considering. We will first study the asymptotic behavior of $P\{\bar{Q}(t) \le m\}$ for large values of m. From the previous analysis, we can *assume* the summation in (15) is over a small number of terms, $k_0, k_0+1, \ldots, k^*$. That is, let us write[6]

$$P\{\bar{Q}(t) \le m\} = \sum_{k=k_0}^{k^*} a(k,m) P\{M(t)=k\}.$$

Similarly, let us assume

$$\sum_{k=k_0}^{k^*} P\{M(t)=k\} = 1.$$

Since $a(k,m)$ increases as k decreases, we get

$$a(k^*,m) \sum_{k=k_0}^{k^*} P\{M(t)=k\} \le P\{\bar{Q}(t) \le m\} \le a(k_0,m) \sum_{k=k_0}^{k^*} P\{M(t)=k\}.$$

Hence,

$$a(k^*,m) \le P\{\bar{Q}(t) \le m\} \le a(k_0,m). \tag{20}$$

The key is to compute $a(k,m)$.

3.1.2.1. Exponential case. First, let us consider the exponential case. We will use the following result.

Lemma 3.1. *For* $x \ge 2\log(1+\sqrt{2})$,

$$\log(1-\mathrm{e}^{-x}) \ge -\mathrm{e}^{-x/2}. \tag{21}$$

Proof. Form a function $g(x) := \log(1-\mathrm{e}^{-x}) + \mathrm{e}^{-x/2}$ on $(0,\infty)$. We see that $\lim_{x\to\infty} g(x) = 0$. Next,

$$g'(x) = \frac{\mathrm{e}^{-x}(2+\mathrm{e}^{-x/2}-\mathrm{e}^{x/2})}{2(1-\mathrm{e}^{-x})}.$$

[6] This can be made more rigorous by

$$\sum_{k=k_0}^{k^*} a(k,m) P\{M(t)=k\} \le P\{\bar{Q}(t) \le m\} \le C \sum_{k=k_0}^{k^*} a(k,m) P\{M(t)=k\}$$

for some small constant $C > 1$.

When $2 + e^{-x/2} - e^{x/2} \le 0$, or equivalently, when $x \ge 2\log(1+\sqrt{2})$, $g'(x) \le 0$. Since $g(x)$ decreases to 0 on $[2\log(1+\sqrt{2}), \infty)$, it must be true that $g(x) \ge 0$ on $[2\log(1+\sqrt{2}), \infty)$. □

We use this lemma in the following derivation with $x = \lambda(t - (j-m)\tau - T)$.

$$\log a(k^*, m) = \sum_{j<k^*} \log(1 - e^{-\lambda(t-(j-m)\tau-T)}) \ge -\sum_{j<k^*} e^{-\lambda(t-(j-m)\tau-T)/2} = -\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1-e^{-\lambda\tau/2}}. \tag{22}$$

The last equality above is derived from summing the geometric series in the previous expression. The condition for the above inequality is $\lambda(t-(j-m)\tau-T) \ge 2\log(1+\sqrt{2})$ for $j < k^*$. This is satisfied if $(k^* - j + m) \ge 2\log(1+\sqrt{2})/(\lambda\tau)$. A weaker condition is simply $m \ge 2\log(1+\sqrt{2})/(\lambda\tau)$. When $1/\lambda = 20$ ms and when the packet size is 1500 bytes, $m \ge 3$, 30 and 294 for the link speed $C_s = 1$, 10 and 100 Mbps, respectively. We would like to point out that these are very loose bound. In practice, we expect the inequality (22) to hold for much smaller m. Combining (20) and (22), we get the lower bound

$$P\{\bar{Q}(t) \le m\} \ge \exp\left(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1-e^{-\lambda\tau/2}}\right). \tag{23}$$

Or, using the fact $e^x \ge 1 + x$ for all x, we get the upper bound

$$P\{\bar{Q}(t) > m\} \le 1 - \exp(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1-e^{-\lambda\tau/2}}) \le \frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1-e^{-\lambda\tau/2}} \le \frac{e^{-(m+2)\lambda\tau/2}}{1-e^{-\lambda\tau/2}}. \tag{24}$$

Thus, for large enough m, $P\{\bar{Q}(t) > m\}$ converges to zero very rapidly, at a rate no slower than exponential in m. In the above analysis, if we suppose $\lambda(t-(j-m)\tau-T)$ is large enough, we can write

$$\log a(k^*, m) = \sum_{j<k^*} \log(1 - e^{-\lambda(t-(j-m)\tau-T)}) \approx -\sum_{j<k^*} e^{-\lambda(t-(j-m)\tau-T)} = -\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1-e^{-\lambda\tau}}. \tag{25}$$

Furthermore, since $e^x \approx 1 + x$, for x near 0,

$$P\{\bar{Q}(t) > m\} \le 1 - \exp\left(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1-e^{-\lambda\tau}}\right) \approx \frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1-e^{-\lambda\tau}} \approx \frac{e^{-(m+2)\lambda\tau}}{1-e^{-\lambda\tau}}. \tag{26}$$

The approximation in (26) turns out to be very good. Fig. 7 compares the values of $P\{\bar{Q}(t) > m\}$ obtained with (26) (labelled as "approximation") with numerical results of (15) (labelled as "numerical").[7] The mean delay, $1/\lambda$, is 20 ms. The approximation agrees extremely well with the numerical results. In Fig. 7, we also compare the tail distribution of $\bar{Q}(t)$ with the loss probability under finite buffer sizes, obtained by simulation (the curves are labelled as "simulation"). We see that the two distributions agree very well when the queue size is large enough. We may consider the tail probability of the queue size upper bound, $P\{\bar{Q}(t) > m\}$, as an approximation of the packet loss ratio when the buffer size is m. Fig. 7(a) and (b) differ in the transmission speed, where $C_s = 1$ and 100 Mbps, respectively.[8] We see that the queue size

[7] Throughout Section 3, all simulation results are for finite buffer sizes, and the queue size should be interpreted as the buffer size. All analytical and numerical results are for $\bar{Q}$ under the infinite buffer assumption.

[8] In all our simulation results, the chosen packet size is 1500 bytes, which is the size of Ethernet's maximum transfer unit (MTU).

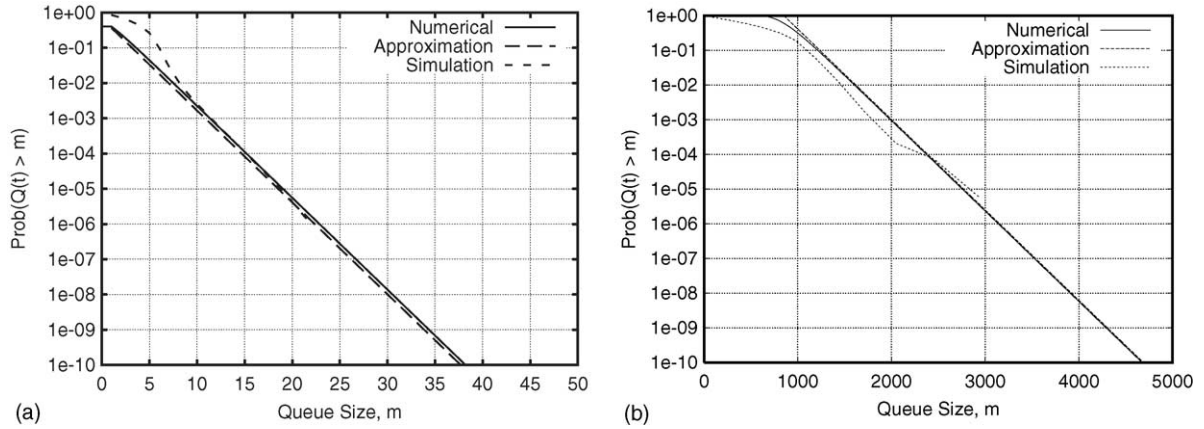

Fig. 7. Tail probabilities for $\bar{Q}$ ("numerical" and "approximation") and loss probabilities under finite buffer size ("simulation"), for the exponential distribution ($1/\lambda = 20$ ms): (a) $C_s = 1$ Mbps; (b) $C_s = 100$ Mbps.

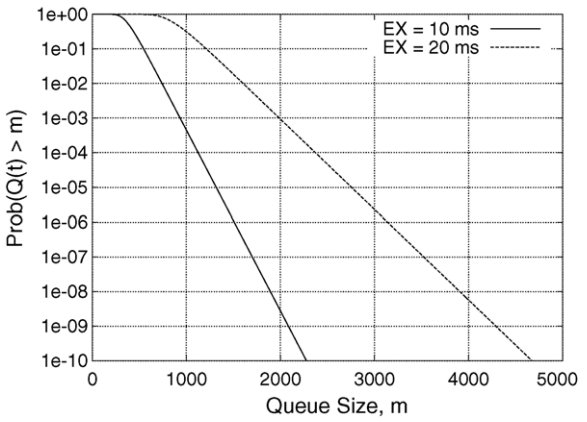

Fig. 8. Tail probabilities for $\bar{Q}$ for the exponential distribution. Comparison of different mean delays. $C_s = 100$ Mbps.

depends the transmission speed quite sensitively. In Fig. 8, we show the comparison between two different mean delays: $1/\lambda = 10$ and 20 ms. The sender's link speed in this case is $C_s = 100$ Mbps.

3.1.2.2. Pareto case. Next, we will show a similar analysis for the Pareto case. For $k \le k^*$,

$$\log a(k,m) = \sum_{j<k} \log\left(1 - \frac{K^\alpha}{(t-(j-m)\tau - T)^\alpha}\right) \le -\sum_{j<k} \frac{K^\alpha}{(t-(j-m)\tau - T)^\alpha}$$

$$\le \int_{-\infty}^{k-1} -\frac{K^\alpha}{(t-(x-m-1)\tau - T)^\alpha} \mathrm{d}x = \frac{-K^\alpha}{(\alpha-1)\tau}(t-(k-m-2)\tau - T)^{-\alpha+1}. \quad (27)$$

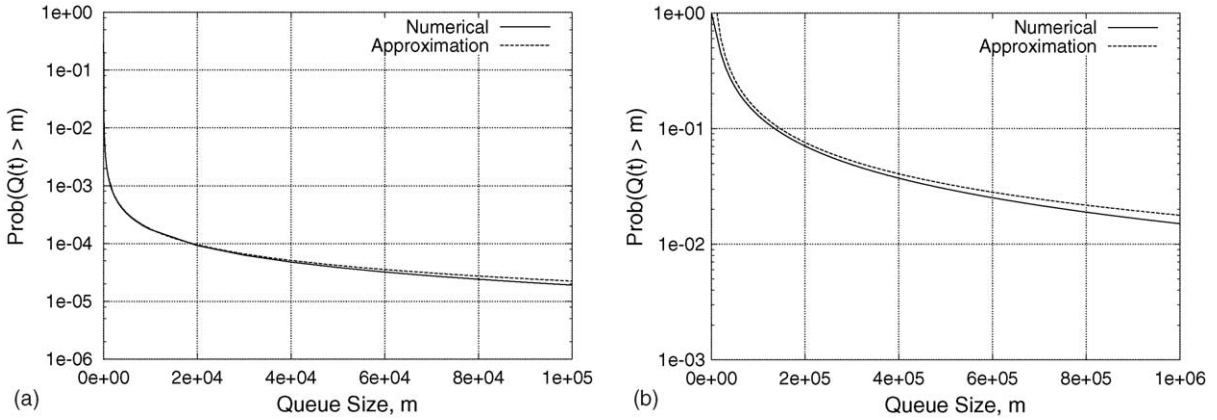

Fig. 9. Tail probabilities for $\bar{Q}$ for the Pareto distribution ($\alpha = 1.9$): (a) $C_s = 10$ Mbps; (b) $C_s = 100$ Mbps.

By $P\{\bar{Q}(t) \le m\} \le a(k_0, m)$,

$$P\{\bar{Q}(t) \le m\} \le \exp\left\{\frac{-K^\alpha}{(\alpha-1)\tau}(t - k_0\tau - T + (m+2)\tau)^{-\alpha+1}\right\}$$
$$\le \exp\left\{\frac{-K^\alpha}{(\alpha-1)\tau}((k^* - k_0 + m + 2)\tau)^{-\alpha+1}\right\}, \tag{28}$$

$$P\{\bar{Q}(t) > m\} \ge 1 - \exp\left\{\frac{-K^\alpha}{(\alpha-1)\tau}((k^* - k_0 + m + 2)\tau)^{-\alpha+1}\right\}$$
$$\approx \frac{K^\alpha}{(\alpha-1)\tau}((k^* - k_0 + m + 2)\tau)^{-\alpha+1}. \tag{29}$$

Eq. (29) shows that the tail probability decays as a power function. In Fig. 9, we show an example of the tail probability, $P\{\bar{Q}(t) > m\}$, based on the approximation in (29), and compare the result with those based on numerical analysis. Numerical analysis of the queue length distribution for the Pareto case is difficult. The numerical results in Fig. 9 are for Pareto distributions "truncated" at large values, denoted by d. In these plots, we see extraordinarily good match between the approximation and the numerical results. We believe that the slight discrepancy between the two is because the delay bounds, d, are not large enough.

In Fig. 10, we compare the tail distribution of $\bar{Q}(t)$ with the loss probabilities under finite buffer sizes, obtained through simulation. The Pareto distribution is truncated at $d = 1000$ s. We see that they are very close to each other for large queue sizes. Unlike the semi-log scale in Fig. 9, Fig. 10 uses log–log scale.

After we establish enough confidence on the "goodness" of the approximation, we use the approximation to investigate dependency of the tail probability of the queue size on various parameters. The results are shown in Fig. 11 in log–log scale. Again, we may consider the tail probability of the queue size upper bound, $P\{\bar{Q}(t) > m\}$, as an approximation of the packet loss ratio when the buffer size is m. From Fig. 11(a) and (b), we notice that to achieve low packet loss probability, the buffer size must be large. For instance, to achieve less than 1% packet loss ratio or equivalently 99% of throughput at $\alpha = 1.9$, we must

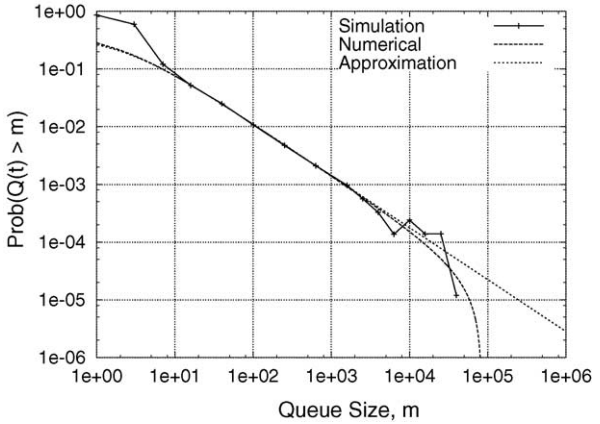

Fig. 10. Tail probabilities for $\bar{Q}$ for the Pareto distribution ("numerical" and "approximation"): comparison with actual loss probabilities under finite buffer sizes ("simulation"). $C_s = 1$ Mbps; $\alpha = 1.9$; $d = 1000$ s.

have $m > 10^2, 10^4$ and 10^6 for $C_s = 1$, 10 and 100 Mbps, respectively. As illustrated in Fig. 11(b) and (c), the loss ratio depends crucially on the parameter α, but less crucially on the mean variable delay.

3.2. Waiting time in the queue

In this section, we analyze the distribution of the resequencing delay, i.e., the waiting time at the receiver queue. Again, suppose the buffer size b is infinite. Let the waiting time for packet i be W_i. Packet i will be accepted immediately after all packets $j \le i$ arrive at the queue. Therefore, packet i's waiting time is,

$$W_i = \max_{j \le i} A_j - A_i = \max_{j \le i} \{A_j - A_i\}.$$

Hence, for $t \ge 0$,

$$P\{W_i \le t\} = P\{A_j - A_i \le t, \text{ for all } j < i\} = \prod_{j<i} P\{A_j - A_i \le t\}$$

$$= \int \prod_{j<i} F(x + (i-j)\tau + t)\, \mathrm{d}F(x). \tag{30}$$

Note that $A_k = k\tau + T + X_k$. The last equality is obtained by conditioning on $X_i = x$.

When the variable delay is bounded by d,

$$P\{W_i \le t\} = \int_0^d \prod_{i - \lceil (d-t-x)/\tau \rceil}^{i-1} F(x + (i-j)\tau + t)\, \mathrm{d}F(x).$$

The mean waiting time of a packet i is

$$\mathbf{E}W = \mathbf{E} \max_{j \le i} A_j - \mathbf{E}A_i. \tag{31}$$

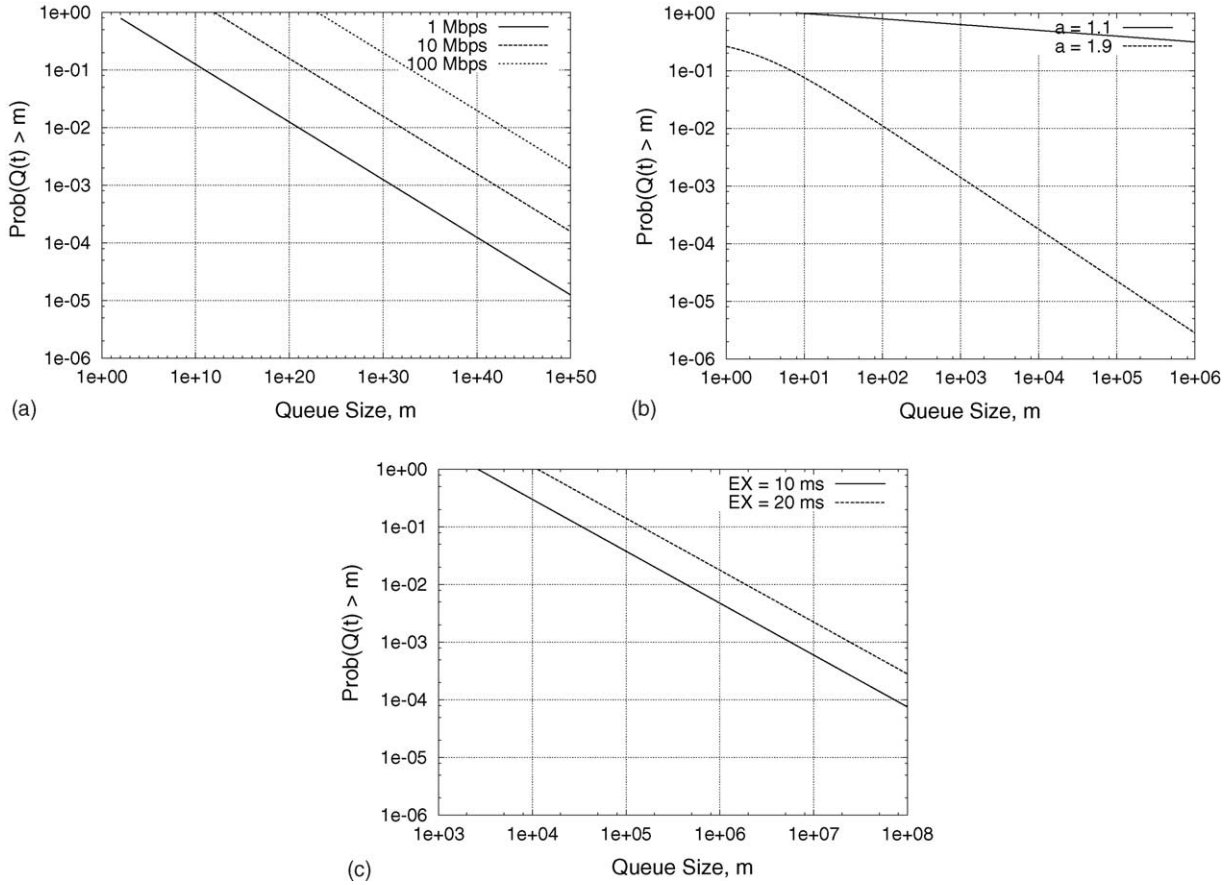

Fig. 11. Tail probabilities for $\bar{Q}$ for the Pareto distribution (log–log plot): (a) $\alpha = 1.9$, mean delay = 20 ms; (b) $C_s = 1$ Mbps, mean delay = 20 ms; (c) $C_s = 100$ Mbps, $\alpha = 1.9$, mean delays = 10 and 20 ms.

3.2.1. Waiting time for exponential variable delay

In the case of the exponential variable delay, we can find the expression for an approximation of the expected waiting time. When packet i arrives at the queue, let G_i be the number of sequence gaps in the received packets prior to i. In other words, G_i is the number of packets that are transmitted before i but have not arrived at the receiver

$$G_i = |\{j < i : A_j > A_i\}|.$$

Packet i needs to stay in the queue until all these G_i packets arrive. Due to the memoryless property of the exponential distribution, the remaining time in the network of each packet is still exponentially distributed. Hence,

$$W_i = \max\{Y_j : j = 1, 2, \ldots, G_i\},$$

where the Y_j's are IID exponential random variables representing the packets' remaining times in the network. We know from ([6], p. 49) that the kth order statistics for a collection of n IID exponential random variables has the expectation

$$\mathbf{E}X_{(k)} = \frac{1}{\lambda}\sum_{i=1}^{k}\frac{1}{n-i+1}, \tag{32}$$

where $1/\lambda$ is the mean of the exponential distribution and $k = 1, 2, \ldots, n$. Conditional on G_i and using (32), we get

$$\mathbf{E}[W_i|G_i] = \frac{1}{\lambda}\sum_{j=1}^{G_i}\frac{1}{j}.$$

For $G_i > 1$, the sum can be approximated by

$$\mathbf{E}[W_i|G_i] \approx \frac{1}{\lambda}(\log G_i + 0.5).$$

Then,

$$\mathbf{E}[W_i] \approx \frac{1}{\lambda}(\mathbf{E}\,\log G_i + 0.5) \leq \frac{1}{\lambda}(\log \mathbf{E}G_i + 0.5).$$

Since the log function is fairly flat over the range of values of practical interest, we will use the following approximation

$$\mathbf{E}[W_i] \approx \frac{1}{\lambda}(\log \mathbf{E}G_i + 0.5). \tag{33}$$

The expected value of G_i can be computed as follows:

$$\mathbf{E}[G_i] = \mathbf{E}\left[\sum_{j<i}\mathbf{1}_{(A_j>A_i)}\right] = \sum_{j<i}P\{A_j > A_i\} = \sum_{j<i}P\{X_j - X_i > (i-j)\tau\}$$
$$= \sum_{j<i}\frac{1}{2}\mathrm{e}^{-\lambda(i-j)\tau} = \frac{1-\mathrm{e}^{-\lambda i\tau}}{2(1-\mathrm{e}^{-\lambda\tau})}.$$

Substituting the result for $\mathbf{E}[G_i]$ into (33) and let i go to infinity, we get the stationary mean waiting time.

$$\mathbf{E}[W] \approx \frac{1}{\lambda}\left(\log\left(\frac{1}{2(1-\mathrm{e}^{-\lambda\tau})}\right) + 0.5\right). \tag{34}$$

In Table 4, we compare the mean waiting times derived from the above analysis with those from simulation. The analytic approximation becomes quite good when the link speed exceeds 10 Mbps. When the link speed is smaller, it appears that the approximation is not accurate. The reason is that, at 1 Mbps, G is very close to 1 on average, making the two approximations we use in deriving (34) less appropriate. However, at this link speed, the packet transmission time, τ, is 12 ms. So the inaccuracy in the approximation of $\mathbf{E}[W]$ at 1 Mbps is no greater than one packet transmission time. In any case, we are more interested in

Table 4
Mean waiting time for the exponential delay: analysis vs. simulation

C_s (Mbps)	$\mathbf{E}[W]$: analysis/simulation (ms)	
	$1/\lambda = 10$	$1/\lambda = 20$
1	1.65/0.2	12.1/0.2
10	19.9/17.5	53.0/48.5
100	42.3/40.0	98.5/94.0

situations where the link speed, and hence, the waiting time, are large. At link speed $C_s = 10$ Mbps and $1/\lambda = 10$ ms, we have $\mathbf{E}[G] = 4.4$, which is not a very large number. We see that the approximation of $\mathbf{E}[W]$ is already quite good. When $C_s \geq 10$ Mbps, we can further approximate $\mathbf{E}[W]$ by noticing that $e^{-\lambda\tau} \approx 1 - \lambda\tau$ when $\lambda\tau$ is small. Then,

$$\mathbf{E}[W] \approx \frac{1}{\lambda}\left(\log\left(\frac{C_s}{2\lambda U}\right) + 0.5\right), \tag{35}$$

where U is the packet size in bits. We see that the mean waiting time of a packet scales logarithmically with the link capacity. We also see that, at a given link speed, reducing the packet size, hence, the packet transmission time, increases the mean waiting time. The observation that small packet size is undesirable from the point of view of resequencing delay cautions us that a design to improve the performance in one area may negatively affect the performance in another area. For instance, in the ATM network, the small packet (cell) size, 53 bytes, is considered appropriate for packetizing voice traffic. But it may result in worse resequencing delay than the 1500-byte Ethernet packet.

Table 4 shows that, at the moderately high link speed of 100 Mbps and at the mean variable delay of 20 ms, the mean resequencing delay is about 100 ms. This is fairly large considering that (i) the maximum tolerable delay for interactive voice is 200 ms, (ii) the total end-to-end delay also includes the propagation

Table 5
Mean waiting time for Pareto delay: simulation results

C_s (Mbps)	$\mathbf{E}[W]$: simulation (ms)				
	$d = 0.2$ s	$d = 0.5$ s	$d = 2$ s	$d = 10$ s	$d = 1000$ s
$\mathbf{E}X = 20$ ms, $\alpha = 1.1$					
1	33.12	81.82	279.15	1093.49	51403.21
10	129.39	327.84	1223.43	5434.29	342886.45
100	172.83	458.22	1859.36	9183.65	826939.37
$\mathbf{E}X = 20$ ms, $\alpha = 1.9$					
1	11.56	20.02	34.52	59.00	71.81
10	74.67	135.29	257.73	431.85	677.99
100	152.28	351.43	988.85	2433.43	4416.03
$\mathbf{E}X = 10$ ms, $\alpha = 1.9$					
1	3.73	6.08	9.92	16.89	20.02
10	37.50	58.27	94.60	158.44	211.46
100	121.30	232.90	519.41	1079.91	1205.94

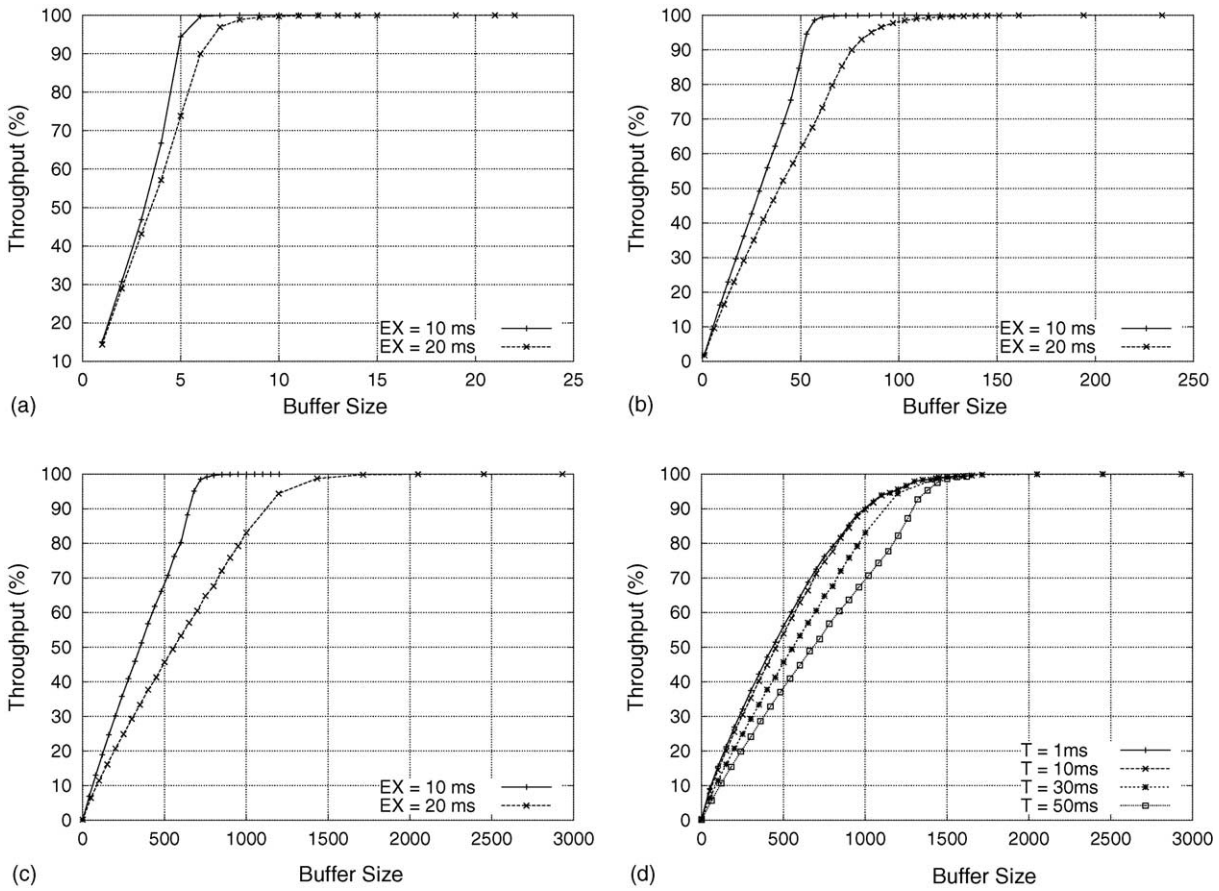

Fig. 12. Throughput vs. buffer size for exponential distributions: (a) $C_s = 1$ Mbps, $T = 30$ ms; (b) $C_s = 10$ Mbps, $T = 30$ ms; $C_s = 100$ Mbps, $T = 30$ ms; (d) $\mathbf{E}X = 20$ ms, $C_s = 100$ Mbps.

delay and the queueing delay in the network, (iii) the packet size here is quite large (1500 bytes), and (iv) the exponential distribution for the random delay is a "nice" case, compared to heavy-tail distributions.

3.2.2. Waiting time for Pareto variable delay

Table 5 shows the simulation results for the expected waiting time, $\mathbf{E}W$, for "truncated" Pareto delays. The delay bounds are $d = 0.2, 0.5, 2, 10$ and 1000 s. We see that $\mathbf{E}W$ depends d, α and C_s in significant ways. In many cases, the resequencing delay is non-trivial. When the Pareto distribution has a "heavy" tail, e.g., $\alpha = 1.1$, and when the sending rate is fairly large, e.g., $C_s = 100$ Mbps, $\mathbf{E}W$ is close to the delay bound, d. If the delay bound is determined by a timeout mechanism similar to the one used in TCP, then it is typical d ranges from 0.5 to 2 s. The resequencing delay ranges from 232 ms to 1.8 s for $C_s = 100$ Mbps, and is expected to be higher for larger C_s. Luckily, the delay increases much more slowly than C_s increases. Also notice that reducing d to 0.2 s can greatly reduce the expected delay. In the case of TCP, there is incentive to reduce the transmission timeout value, which depends on the round-trip time of the transmission path. Therefore, it pays to accurately estimate the round-trip time.

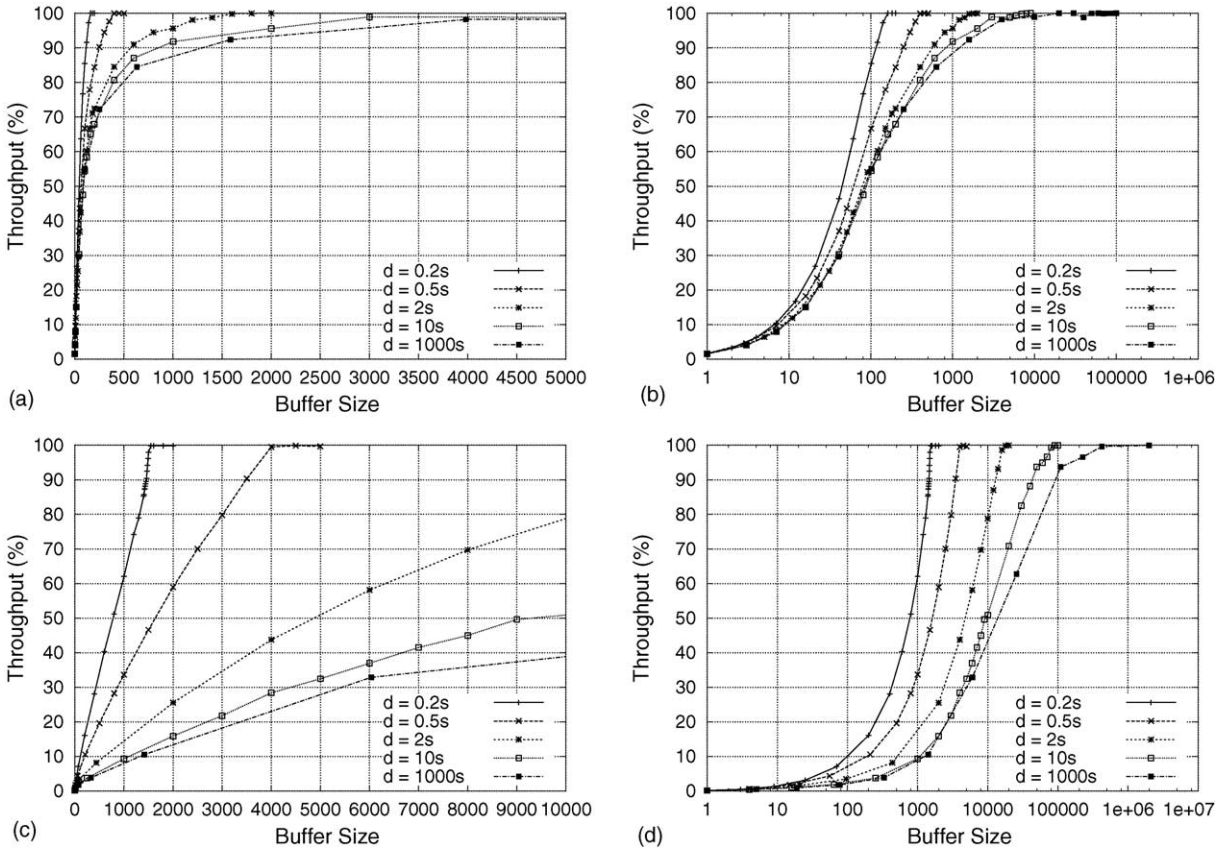

Fig. 13. Throughput vs. buffer size for "truncated" Pareto distributions ($\alpha = 1.9$, $T = 30$ ms): (a) $C_s = 10$ Mbps, linear scale; (b) $C_s = 10$ Mbps, semi-log scale; (c) $C_s = 100$ Mbps, linear scale; (d) $C_s = 100$ Mbps, semi-log scale.

3.3. Throughput for finite buffer sizes

This section shows some simulation results on the throughput under finite buffer sizes. We also vary other parameters, such as the link speed, the propagation delay and the parameters for the distribution functions. Fig. 12 is for the exponential delays. The link speed is 1, 10 and 100 Mbps for plots (a)–(c), respectively. The two curves in each of the above plots correspond to the mean variable delay 10 and 20 ms, respectively. The buffer requirement depends on the link speed in fairly regular fashion for a large range of the throughput values, say between 10 and 90%. For instance, for the case where the mean delay, $\mathbf{E}X$, is 20 ms and throughput is 50%, the buffer size is 4, 40 and 550 when the link speed is 1, 10 and 100 Mbps, respectively. In plot (d), we vary the fixed propagation delay T.

Figs. 13 and 14 are for the "truncated" Pareto distributions with $\alpha = 1.9$ and 1.1, respectively. We see that the delay bound, d, severely affects the throughput-buffer size characteristics in the Pareto case. So do the sending speed, C_s, and the speed of decay of the Pareto complimentary distribution function, represented by α. To demonstrate the large buffer requirement under large values of d, the axis for the buffer size is shown in both linear scale and log scale.

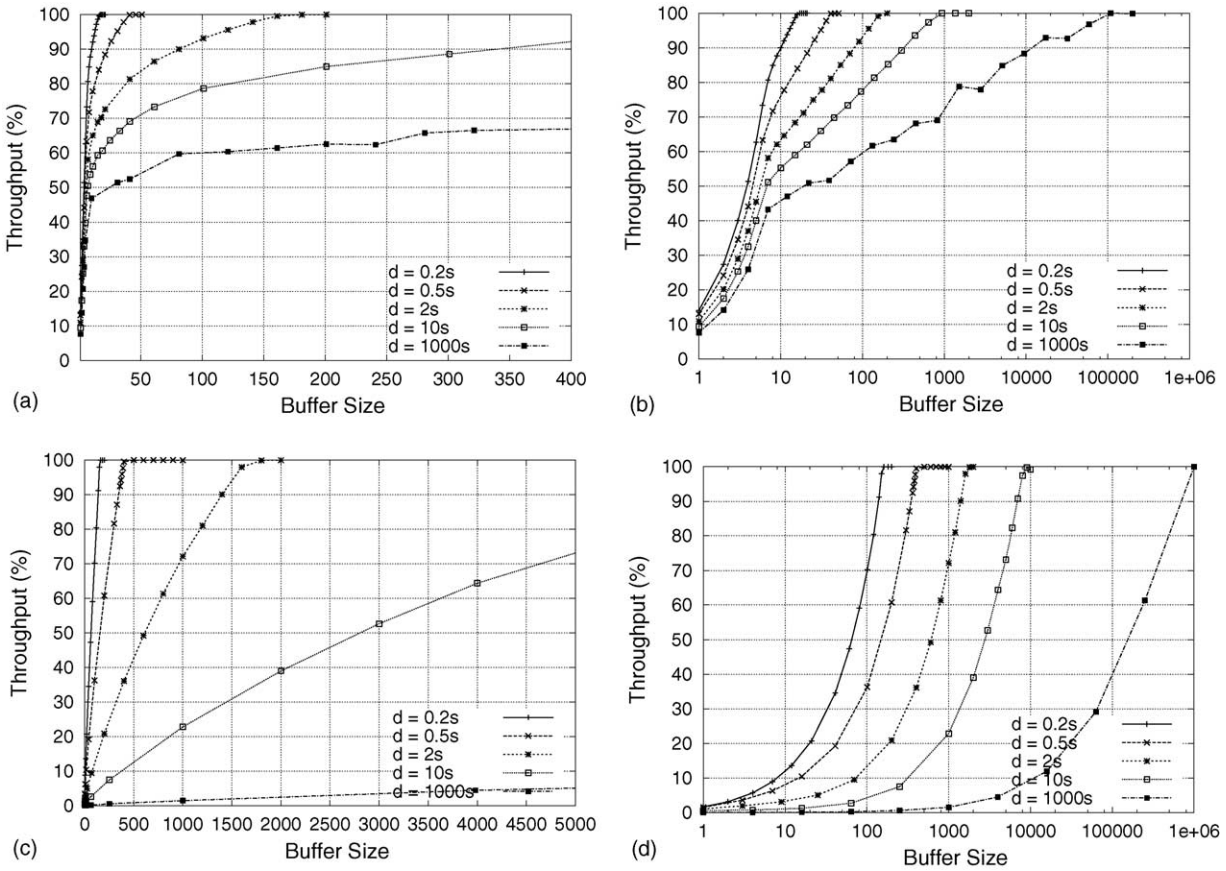

Fig. 14. Throughput vs. buffer size for "truncated" Pareto distributions ($\alpha = 1.1$, $T = 30$ ms): (a) $C_s = 1$ Mbps, linear scale; (b) $C_s = 1$ Mbps, semi-log scale; (c) $C_s = 10$ Mbps, linear scale; (d) $C_s = 10$ Mbps, semi-log scale.

4. Conclusion

4.1. General comments

For reliable communication, the receiver's transport layer must resequence packets that are disordered by the network. This paper studies the delay and the required buffer size for packet resequencing. The results are summarized in Section 1.3. We have shown that both quantities can be significant, especially when the variable packet delay in the network has a heavy-tail distribution. Besides requiring extra resource, the delay caused by packet resequencing negatively affects the performance of delay-sensitive applications. A network that causes severe packet disordering, such as one that allows multi-path routing, must employ mechanisms to reduce the resequencing delay. Fast detection and retransmission of delayed or dropped packets can be very helpful or even necessary. The results show how the loss probability and the resequecing delay scale with technology, i.e., the link speed, the packet size, and the distributions and parameters for the variable delay. The key message is: the resequencing problem becomes worse as the link speed increases. This is in contrast with many other performance measures. For instance, assuming

the characteristics of the data sources do not change, as the link speed of the backbone increases, more connections and traffic can be aggregated together. The combined traffic is typically smoother and its queueing delay in the network is smaller under the same link utilization. Another message is that making the packet size smaller also increases the resequencing delay.

In terms of modelling, this paper presents two models, the stop-and-wait ARQ and the selective-repeat ARQ. The two models are increasingly more complex, but are also increasingly more suitable for reality. The analysis of the stop-and-wait ARQ is for the whole range of buffer sizes. The analysis of the selective-repeat ARQ is for large buffer sizes only. Many results of this paper are in terms of simple expressions, which can be easily interpreted. We frequently use approximations in their derivation. In this paper, the random packet delays that cause the disordering of the packets are assumed be IID. In our other work [26], we analyze a model with correlated delays where the disordering is caused by multi-path routing.

4.2. Selective-repeat ARQ: relevance to TCP

How are the models relevant to the resequencing problem in TCP, the dominant transport protocol? We will focus our discussion on the selective-repeat ARQ model. The model captures the resequencing behavior at the TCP receiver. The feedback is similar to TCP's selective acknowledgement. However, TCP is more complicated. TCP has flow control and congestion control. With the flow control, packets are never really dropped at the receiver. Packet can be dropped in the network due to congestion. TCP uses cumulative acknowledgement by default, and optionally selective acknowledgement, to carry feedback information. There is uncertainty in the feedback information because the reverse channel is unreliable. There is also variation in the timing of packet retransmissions. In a real TCP connection, the bandwidth bottleneck can be at the sender, the network or the receiver.

Given the many interacting factors of TCP, a fully accurate TCP model with resequencing will be hard to analyze, or even to write down. Our model represents a piece-meal approach for understanding the resequencing problem under TCP. We get rid of many details of TCP and concentrate on a single isolated phenomenon, the disordering/resequencing of packets. In essence, we ask: if everything else goes well, even better than what TCP allows, how bad is the resequencing problem?

In our model, we assume that (i) the receiver has infinite capacity, i.e., $C_r = \infty$ (see Fig. 1 for notation). We also assume that (ii) the connection is bandwidth-bottlenecked at the sender. Assumption (ii) leads to the fact that no packets are dropped in the network. TCP has a flow control window, whose size is the amount of packets that the sender can transmit without overflowing the receiver's buffer. It is equal to the buffer space available at the receiver minus the number of packets on the flight to the receiver. Under assumptions (i) and (ii), if all packets are in sequence, there will never be packets waiting at the receiver buffer, because they get immediately accepted (to the upper layer) when they arrive at the receiver. Hence, the buffer at the receiver is used solely for packet resequencing.

Let us imagine what happens in actual TCP under the assumptions (i) and (ii). Note that the missing packets in the receiver's buffer are due to packet disordering in the network, not due to packet losses. With TCP's flow control, packets are never dropped at the receiver, because the TCP sender has an estimate of the receiver's buffer size based on the flow control window size contained in the feedback packets. The TCP sender will receive feedback about the missing packets, say, with the selective acknowledgement, and will retransmit the missing packets. From the feedback information, the TCP sender can not deduce whether the packets are lost in the network or merely delayed, and it might reduce the congestion control window, depending on the exact sequence of events. The TCP sender will stop sending new packets if either

the congestion control window or the flow control window is used up. Hence, insufficient resequencing buffer size will lead to reduced throughput of TCP, instead of packet losses at the receiver as in our model.

With all these differences, what can be said about TCP based on our mode? In a sense, the results from our model tell us how to size the receiver's buffer so that TCP throughput does not suffer from packet disordering. For instance, if in our model, a particular buffer size leads to 10^{-4} packet loss ratio, we know that will be the buffer size at which TCP throughput won't be affected much by packet disordering. Our model-based results are, on one hand, optimistic because, first, we assume C_r is infinite, and second, TCP's congestion control does not kick in to further reduce the throughput. The buffer size would have to be even larger with a finite C_r and with TCP's congestion control activated. On the other hand, our mechanism for packet disordering (i.e., due to IID delays on packets) is likely to be pessimistic for normal situations.

References

[1] M.E. Anagnostou, E.N. Protonotarios, Performance analysis of the selective repeat ARQ protocol, IEEE Trans. Commun. COM-34 (2) (1986) 127–135.

[2] S. Ayoun, Z. Rosberg, Optimal routing to two parallel heterogeneous servers with resequencing, IEEE Trans. Automat. Control 36 (12) (1991) 1436–1449.

[3] F. Baccelli, E. Gelenbe, B. Plateau, An end-to-end approach to the resequencing problem, J. Assoc. Comput. Machinery 31 (3) (1984) 474–485.

[4] F. Baccelli, A.M. Makowski, Queueing models for systems with synchronization constraints, Proc. IEEE 77 (1) (1989) 138–161.

[5] S. Chowdhury, An analysis of virtual circuits with parallel links, IEEE Trans. Commun. 39 (8) (1991) 1184–1188.

[6] H.R. David, Order Statistics, 2nd ed., Wiley, New York, 1981.

[7] R. Durrett, Probability—Theory and Examples, 2nd ed., Duxbury Press, Florence, KY, 1996.

[8] N. Gogate, S.S. Panwar, On a resequencing model for high speed networks, in: Proceedings of INFOCOM '94, Toronto, Canada, June 1994, pp. 40–47.

[9] G. Harrus, B. Plateau, Queueing analysis of a reordering issue, IEEE Trans. Software Eng. SE-8 (2) (1982) 113–123.

[10] I. Iliadis, L.Y.-C. Lien, Resequencing delay for a queueing system with two heterogeneous servers under a threshold-type scheduling, IEEE Trans. Commun. 36 (6) (1988) 692–702.

[11] A. Jean-Marie, L. Gün, Parallel queues with resequencing, J. Assoc. Comput. Machinery 40 (5) (1993) 1188–1208.

[12] F. Kamoun, L. Kleinrock, R. Muntz, Queueing analysis of the reordering issue in a distributed database concurrency control mechanism, in: Proceedings of the Second International Conference on Distributed Computing Systems, Versailles, France, April 1981, pp. 13–23.

[13] A.G. Konheim, A queueing analysis of two ARQ protocols, IEEE Trans. Commun. COM-28 (7) (1980) 1004–1014.

[14] Y.-C. Lien, Evaluation of the resequence delay in a Poisson queueing system with two heterogeneous servers, in: Proceedings of the International Workshop on Computer Performance Evaluation, Tokyo, Japan, September 1985, pp. 189–197.

[15] M.J. Miller, S.-L. Lin, The analysis of some selective-repeat ARQ schemes with finite receiver buffer, IEEE Trans. Commun. COM-29 (9) (1981) 1307–1315.

[16] D.S. Mitrinovic, Analytic Inequalities, Springer, Berlin, 1970.

[17] V. Paxson, End-to-end Internet packet dynamics, IEEE/ACM Trans. Netw. 7 (3) (1999) 277–292.

[18] Z. Rosberg, N. Shacham, Resequencing delay and buffer occupancy under the selective-repeat ARQ, IEEE Trans. Inform. Theory 35 (1) (1989) 166–172.

[19] Z. Rosberg, M. Sidi, Selective-repeat ARQ: the joint distribution of the transmitter and the receiver resequencing buffer occupancies, IEEE Trans. Commun. 38 (9) (1990) 1430–1438.

[20] B.H. Saeki, I.R. Rubin, An analysis of a TDMA channel using stop-and-wait, block, and selective-and-repeat ARQ error control, IEEE Trans. Commun. COM-30 (5) (1982) 1162–1173.

[21] N. Shacham, B.C. Shin, A selective-repeat-ARQ protocol for parallel channels and its resequencing analysis, IEEE Trans. Commun. 40 (4) (1992) 773–782.
[22] N. Shacham, D. Towsley, Resequencing delay and buffer occupancy in selective-repeat ARQ with multiple receivers, IEEE Trans. Commun. COM-39 (6) (1991) 928–937.
[23] D. Towsley, J.K. Wolf, On the statistical analysis of queue lengths and waiting times for statistical multiplexers with ARQ retransmission schemes, IEEE Trans. Commun. COM-27 (4) (1979) 693–702.
[24] D. Towsley, The stutter go back-*N* ARQ protocol, IEEE Trans. Commun. COM-27 (6) (1979) 869–875.
[25] S. Varma, Optimal allocation of customers in a two server queue with resequencing, IEEE Trans. Autom. Control 36 (11) (1991) 1288–1293.
[26] Y. Xia, D. Tse, On the large deviation of resequencing queue size: 2-M/M/1 case, in: Proceedings of the IEEE Infocom 2004, Hong Kong, March 2004.
[27] T.-S.P. Yum, T.-Y. Ngai, Resequencing of messages in communication networks, IEEE Trans. Commun. COM-34 (2) (1986) 143–149.

Ye Xia is an assistant professor at the Computer and Information Science and Engineering department at the University of Florida, starting in August 2003. He has a PhD degree from the University of California, Berkeley, in 2003, an MS degree in 1995 from Columbia University, and a BA degree in 1993 from Harvard University, all in Electrical Engineering. Between June 1994 and August 1996, he was a member of the technical staff at Bell Laboratories, Lucent Technologies in New Jersey. His research interests are in computer networking area, including performance evaluation of network protocols and algorithms, congestion control, resource allocation, and load balancing on peer-to-peer networks. He is also interested in probability theory, stochastic processes and queueing theory.

David Tse received the BASc degree in systems design engineering from University of Waterloo, Canada in 1989, and the MS and PhD degrees in electrical engineering from Massachusetts Institute of Technology in 1991 and 1994 respectively. From 1994 to 1995, he was a postdoctoral member of technical staff at A.T. & T. Bell Laboratories. Since 1995, he has been at the Department of Electrical Engineering and Computer Sciences in the University of California at Berkeley, where he is currently a Professor. He received a 1967 NSERC 4-year graduate fellowship from the government of Canada in 1989, a NSF CAREER award in 1998, the Best Paper Awards at the Infocom 1998 and Infocom 2001 conferences, the Erlang Prize in 2000 from the INFORMS Applied Probability Society, the IEEE Communications and Information Theory Society Joint Paper Award in 2001, and the Information Theory Society Paper Award in 2003. His research interests are in information theory, wireless communications and networking.