Analysis on Packet Resequencing for Reliable Network Protocols

Ye Xia David Tse

Department of Electrical Engineering and Computer Science
University of California, Berkeley

***Abstract*—**

Protocols such as TCP requires packets to be accepted (i.e., delivered to the receiving application) in the order they are transmitted at the sender. Packets are sometimes mis-ordered in the network. In order to deliver the arrived packets to the application in sequence, the receiver's transport layer needs to temporarily buffer out-of-order packets and re-sequence them as more packets arrive. Even when the application can consume the packets infinitely fast, the packets may still be delayed for resequencing. In this paper, we model packet mis-ordering by adding an 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.

I. Introduction

Protocols such as TCP requires packets to be accepted (i.e., delivered to the receiving application) in the order they are transmitted at the sender. Packets are sometimes mis-ordered in the network. For instance, since every packet contains the destination address, the network can deliberately route packets via different paths to the destination, possibly for load balancing purpose. Certain packets may be dropped by the network and retransmitted by the sender, causing the packets to arrive out-of-order at the receiver. In order to deliver the arrived packets to the application in sequence, the receiver's transport layer needs to temporarily buffer out-of-order packets and re-sequence them as more packets arrive. Even when the application can consume the packets infinitely fast, the packets may still be delayed for resequencing. We are interested in how large the resequencing buffer must be and how much the resequencing delay is for an average packet. We'd like to know how these two quantities scale with the network bandwidth. The results will enable us to examine one consequence of a fundamental principle of packet networks: each packet contains its destination address and can be routed independently.

A. Network Model

We will examine a model shown in figure 1, where the sender and the receiver are separated by a network that causes a random variable delay on each data packet. 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 $C_r = \infty$. The receiver can send perfect feedback information to the sender about the reception status of each packet, subject to a fixed delay, T. The sender transmits new packets in increasing order of the packet IDs. Each packet experiences a fixed propagation delay T, and a variable delay X_i. We assume that the $\{X_i\}$'s are independently and identically distributed (IID) random variables. The receiver needs to accept packets IN ORDER, and can temporarily store out-of-order packets in its resequencing buffer of size b. When an arrival packet finds the buffer full, *the packet with the largest ID among all stored and incoming packets is dropped.* The sender will retransmit the dropped packet at a later time.

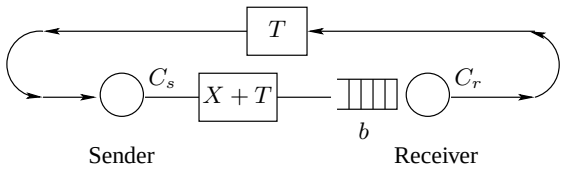


Fig. 1. Our network model

The causes for packet mis-ordering in the network can be many and are not precisely known at the present. As stated previously, retransmission of lost packets and multi-path routing, whether deliberate or not, may be among them in today's or future networks. Many previous studies in fact treat these two causes separately. Mis-ordering caused by packet retransmission is studied under the name of automatic repeat request (ARQ) protocol [16] [10] [12] [15] [1] [13] [14]. It is typically assumed that

ARQ is a link layer protocol between a sender-receiver pair over a link with constant propagation delay. When the communication channel from the sender to the receiver is noisy, the sender needs to retransmit corrupted packets based on the feedback information it gets from the receiver. Researchers have mainly been concerned with the throughput of ARQ protocols. Several papers deal with the resequencing delay at the receiver caused by packet retransmissions [13] [14]. Note that the end-to-end reliable transport protocols, such as TCP, resemble 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 mis-ordering other than packet retransmission.

The studies that deal with packet mis-ordering due to multi-path routing (also including parallel processing or load balancing, etc.) typically analyze an open queueing network, with no feedback and no retransmission. Figure 2 shows a generic model, where packets (or customers) numbered sequentially arrive at the system following some stochastic process, get mis-ordered by the mis-ordering network and resequenced at the resequencing buffer. In some studies, a FIFO queue follows the resequencing buffer. The mis-ordering network is also modeled as a queueing system, whose type typically distinguishes different studies. For instance, the mis-ordering network is an M/M/∞ queue in [9], an M/GI/∞ queue in [7], a GI/GI/∞ queue in [2], an M/M/2 queue in [11], an M/M/K queue in [17], an M/H_2/K queue in [4], an M/M/2 queue with a threshold-type server assignment policy in [8], two parallel M/M/1 queues with additional fixed propagation delays in [6], and K parallel M/GI/1 queues in [6]. A survey is given in [3]. Most of these studies are concerned mostly with finding the distribution and/or mean of the resequencing delay or end-to-end delay. Several also give results about the number of packets in the resequencing queue.

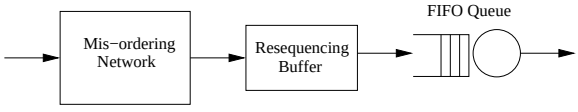


Fig. 2. Resequencing network model followed by a GI/GI/1 queue

Even though we talk about packet retransmission in the paper, our analysis never handle that aspect. Therefore, from analytical point of view, our model falls in the class of models shown in figure 2. We choose to model the causes for packet mis-ordering by adding IID random delays to each packet for both simplicity and for generality. This is equivalent to say the packet mis-ordering network is a D/GI/∞ queue. In reality, the variable packet delays are most likely correlated. For instance, multi-path routing is probably better modeled as K parallel $\cdot$/GI/1 queues. However, we do not know the value of K, the dispatching policy to each of the queues, and the server rates of the queues that give a realistic model.

A natural question is what distribution we should choose for the random delay. We choose the exponential and the Pareto distributions, whose tail probabilities have very different decaying behaviors. We will see that the tail probability is very important to the resequencing buffer requirement.

B. Summary of Results

We analyze two different variations of the model shown in figure 1. In the first case, discussed in section II, we assume the sender's capacity, C_s, is very large so that it can dump many packets onto the network almost instantaneously. Here, it makes sense to consider the Stop-and-Wait-n ARQ (automatic repeat request) protocol [1], where, at 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 introduce the Selective-Repeat ARQ, as discussed in section III. In this case, the sender transmits one packet on each time slot. When a packet is rejected at the receiver due to buffer overflow, this information is fed back to the sender. The sender always transmits the reject packet with the smallest packet sequence number on each time slot and only transmits new packets 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

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

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

where $\bar{Q}(t)$ is an upper bound on the resequencing queue size and τ is the packet transmission time. For the Pareto delay distribution with CDF $F(x) = 1 - K^\alpha x^{-\alpha}$, where $x \geq K$, we have

$$\begin{aligned} & P\{\bar{Q}(t) > m\} \\ \approx \quad & \frac{K^\alpha}{(\alpha-1)\tau}((k^* - k_o + m + 2)\tau)^{-\alpha+1} \end{aligned} \tag{2}$$

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

Our analysis in section III 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 III-C.

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.

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

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

$$\mathbf{E}[W] \approx \frac{1}{\lambda}(\log(\frac{C_s}{2\lambda U}) + 0.5) \tag{4}$$

where U is the packet size in bits. We see that the mean waiting time of a packet scales logrithmically 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 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 packet propagation delay is Pareto. With simulation, we can show that the expected waiting time can be very large in this case. To reduce the resequencing waiting time, it is very helpful to add a bound, say d, to the Pareto delay by retransmitting the packets that have not arrived at the receiver after d seconds.

A feature of our 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 are in complicated terms and are hard to construe.

II. 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 it is ready to be accepted by the receiver, it can be accepted instantaneously. The propagation time, T, is zero.

Data packets originated at the sender are numbered by a packet ID, 1, 2, 3, ..., so on, and are transmitted in that order. We require that the receiver consumes those packets in increasing order of the packet IDs. Suppose we use a *Stop-and-Wait-n* ARQ protocol with block size n, where $1 \leq b \leq n$. More specifically, let the time be divided into intervals of identical length. At the beginning of each interval, the sender sends a block of n packets simultaneously. These packets experience IID random network delays. When they reach the receiver buffer, they can be misordered. The receiver re-orders the received packets in the receive 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. Let us also assume the random delay X is bounded by the length of the interval, so that all n packets in the block will arrive at the receiver before the end of the interval. At the end of the interval, the receiver drops any packet that might be in the buffer but cannot be accepted due missing packets with smaller packet IDs. The receiver sends perfect feedback to the sender during the interval. By the end of the interval, the sender knows which packets have been received successfully and which ones have been dropped. In the next interval, the sender sends n more packets, contiguous in sequence number, starting from the next packet expected by the receiver.

Take the example of $n = 3$. Suppose $b = 1$ and suppose the order of packet arrival at the receiver is 2, 1 and 3. Packet 2 will be dropped when packet 1 arrives because it is out-of-order and there is no additional buffer space to hold it. Packet 1 will be accepted immediately when it arrives. Packet 3 cannot 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 packet 2, 3 and 4.

Let the number of packets accepted by the receiver in interval i be N_i, $i = 1, 2, ...$ 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 1: Let integer $n \geq 1$ be the block size and let integer b be the buffer size. Then, for $1 \leq b \leq n$,

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

Proof: Let J_k be the set $\{1,2,...,k\}$, 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 they 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 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 $b < k \leq n$, let $a_1, a_2, ..., a_k$ be the sequence of arrivals for packets $1,2,...,k$. If $a_1 = 1$, packet 1 will be immediately accepted and the buffer will become empty. We only need to count the number of b-acceptable permutations of $\{2,3,...,k\}$. This number is identical to A_{k-1}.

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

Suppose $a_1 = 2$. In order not to drop any packets from J_k, it is necessary that packet 1 arrives before or in position b. When packet 1 arrives, both 1 and 2 will be accepted and the buffer will become empty. From then on, we only need to count the number of b-acceptable permutations for $\{3,4,...,k\}$. That number is identical to A_{k-2}. So the number of b-acceptable permutations for $\{1,2,...,k\}$ in which $a_1 = 2$ is $(b-1)A_{k-2}$.

In general, suppose $a_1 = i$, for some $1 \leq i \leq k-b+2$, In order not to drop any packets from J_k, packet 1 should arrive before or in position b, packet 2 should arrive before or in position $b+1$, ..., and packet $i-1$ should arrive before or in position $b+i-2$ [3]. Immediately after all packets from 1 to i have arrived, packet 1, 2, ..., i will all be accepted and the buffer will become empty. After that, we need to count the number of b-acceptable permutations of $\{i+1, i+2, ..., k\}$. That number is A_{k-i}. So the total number of b-acceptable permutations of the set J_k starting with packet i is $(b-1)^{k-i+1}A_{k-i}$.

When $k-b+2 < i \leq k$, packet 1 should arrive before or in position b, packet 2 should arrive before or in position $b+1$, ..., packet $k-b+1$ should arrive before or in position k, and packet $k-b+2$ to i should all arrive before or in position k [4]. After that, we still need to count the total number of b-acceptable permutations of the set $\{i+1, i+2, ..., k\}$, which is A_{k-i}. Here, we define $A_0 = 1$. So the total number of b-acceptable permutations of the set J_k starting with packet i is $(b-1)^{k-b+1}(b-2)(b-3)...(b-(k-i)+1)A_{k-i}$.

Hence, we get

$$\begin{aligned}
& A_k \\
= \; & A_{k-1} + (b-1)A_{k-2} + (b-1)^2 A_{k-3} + ... \\
& +(b-1)^{k-b-1}A_b + (b-1)^{k-b}A_{b-1} \\
& +(b-1)^{k-b+1}A_{b-2} \\
& +(b-1)^{k-b+1}(b-2)A_{b-3} \\
& +(b-1)^{k-b+1}(b-2)(b-3)A_{b-4} + ... \\
& +(b-1)^{k-b+1}(b-2)(b-3)...(2)A_1 \\
& +(b-1)^{k-b+1}(b-2)!A_0 \\
= \; & b!b^{k-b-1} + (b-1)b!b^{k-b-2} + (b-1)^2 b!b^{k-b-3} \\
& +... + (b-1)^{k-b-1}b! + (b-1)^{k-b}(b-1)! \\
& +(b-1)^{k-b+1}(b-2)!(b-1) \\
= \; & b!b^{k-b}(\frac{1}{b} + \frac{b-1}{b^2} + \frac{(b-1)^2}{b^3} + ... \\
& +\frac{(b-1)^{k-b}}{b^{k-b+1}} + \frac{(b-1)^{k-b+1}}{b^{k-b+1}})
\end{aligned}$$

[3]Note that $b+i-2 \leq k$.

[4]Of course, all k packets will arrive before position k.

$$= b!b^{k-b}(\frac{1}{b}\frac{1-(\frac{b-1}{b})^{k-b+1}}{1-\frac{b-1}{b}}+\frac{(b-1)^{k-b+1}}{b^{k-b+1}})$$
$$= b!b^{k-b}$$

In the above derivation, we used the induction hypothesis that (6) is true for all A_j, where $1 \le j < k$. Since all permutations of the set J_k are equally likely to be the packet arrival orders, we have

$$\begin{aligned}\mathbf{E}N &= \sum_{k=1}^{n} P\{N \ge k\} = \sum_{k=1}^{n} \frac{A_k}{k!} \\ &= \sum_{k=b}^{n} \frac{b!\, b^{k-b}}{k!} + \sum_{k=1}^{b-1} \frac{k!}{k!} \\ &= \sum_{k=b}^{n} \frac{b!\, b^{k-b}}{k!} + b - 1\end{aligned}$$

■

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

Theorem 2: For any $0 \le \alpha \le 1$, let $b = \lfloor \alpha n \rfloor$ [5]. Then, $\rho(n, b) \to \alpha$, as $n \to \infty$.

Proof: The proof uses a standard convergence argument and is omitted for brevity. ■

Theorem 2 says, in order to achieve reasonable acceptance ratio, the buffer size has to 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 has to be and what happens when n is not so large.

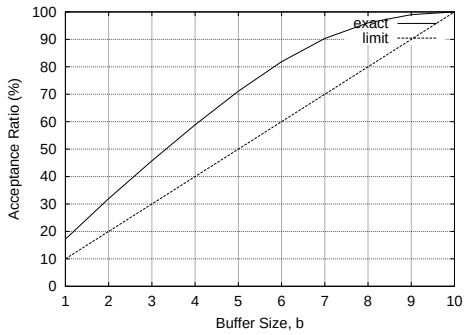
Figure 3 shows the acceptance ratio ρ versus the buffer size for block size $n = 10$ and 100, respectively. In these plots, the label "limit" refers to the asymptotic limit of the ρ as in Theorem 2, and the label "exact" refers to the exact value of ρ. We see that the asymptotic result becomes good approximation for $n > 100$ at all buffer sizes. Even at very small values of n, say, $n \le 10$, the asymptotic result is not too far from the exact value.

In figure 4, we show the convergence of ρ to the limit as n increases to infinity while $b = 0.5n$. In this case, the limit is 0.5.

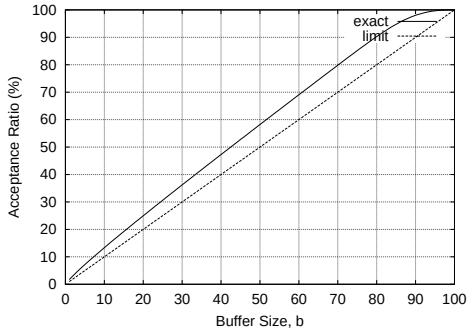
III. Selective-Repeat ARQ

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 sends 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 typical transport and link-layer protocols.

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



(a)



(b)

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

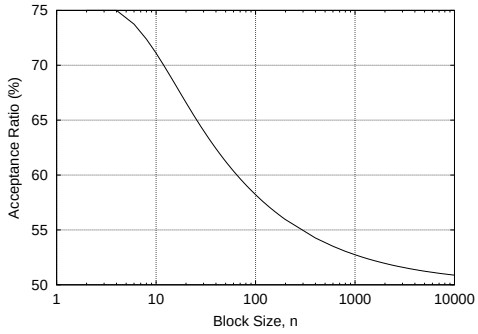


Fig. 4. Acceptance ratio converges to 50%: buffer size $b = 0.5n$

A. Buffer Size to Achieve Near 100% Throughput

Suppose the buffer size is infinite, we wish to approximate the required buffer size for near 100% throughput

by the queue size m for which Prob$\{Q \le m\} \approx 1$. Note 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. Suppose the sender has been transmitting forever and suppose the sender starts transmitting packet 1 at time 0 [6]. Then packet k leaves the sender at $t = k\tau$, for $k \ge 1$. 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 $k = ..., 1, 2, ...$

At time t, let $S(t)$ be the set of packets that have arrived at the 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. They can be expressed as

$$S(t) = \{i : A_i \le t\}$$

$$M(t) = \max\{i : A_i \le t\}$$

$$L(t) = \max\{i : \max\{..., A_{i-1}, Ai\} \le t\}$$

The queue size at time t is $Q(t) = |S(t) - \{..., L(t) - 2, L(t)-1, L(t)\}|$. 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 $\bar{Q}(t) = (M(t)-L(t)-1)^+$. 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, large queue size is typically due to a large delay of a very early packet, say j, which will arrive after packet $j+1, j+2, ..., M(t)$. For $m = 0, 1, 2, ...,$

$$P\{\bar{Q}(t) \le m\} = P\{L(t) \ge M(t) - m - 1\}$$

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 possible arrive before time t.

$$\begin{aligned} & P\{\bar{Q}(t) \le m\} \\ = & \sum_{k \le k^*} P\{..., A_{k-m-2} \le t, A_{k-m-1} \le t, M(t) = k\} \\ = & \sum_{k \le k^*} P\{..., A_{k-m-2} \le t, A_{k-m-1} \le t, A_k \le t, \\ & \qquad A_{k+1} > t, ..., A_{k^*} > t\} \\ = & \sum_{k \le k^*} P\{..., A_{k-m-2} \le t, A_{k-m-1} \le t\} \\ & \cdot P\{M(t) = k\} \end{aligned} \quad (7)$$

where

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

[6] Packet ID numbers can be negative.

1) Computation of $P\{M(t) = k\}$: For exponential delay with mean $1/\lambda$,

$$\begin{aligned} P\{M(t) = k\} &= (1 - e^{-\lambda(t-k\tau-T)}) \\ & \quad \cdot e^{-(k^*-k)\lambda(t-(k+1+k^*)/2-T)} \end{aligned} \quad (8)$$

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

$$\begin{aligned} & P\{M(t) = k\} \\ = & (1 - \frac{K^\alpha}{(t-k\tau-T)^\alpha}) \frac{K^\alpha}{(t-(k+1)\tau-T)^\alpha} ... \\ & \frac{K^\alpha}{(t-k^*\tau-T)^\alpha} \end{aligned} \quad (9)$$

Since this number decays to 0 geometrically fast as k decreases, we can find a $k_o < k^*$ so that $P\{M(t) = k\}$ is negligible for $k < k_o$. For practical purpose, the sum in (7) involves a small number of terms. Suppose we set $P\{M(t) = k\} \le \epsilon$ for some $0 < \epsilon < 1$. In the case of exponential distribution, let us ignore the factor $(1 - e^{-\lambda(t-k\tau-T)})$ in (8) since it is no greater than 1. Using $k^* = \lfloor (t-T)/\tau \rfloor$, we get,

$$k^* - k \ge \sqrt{\frac{-2\log\epsilon}{\lambda\tau}} + 1 \quad (10)$$

Table I shows the lower bounds on $k^* - k$ obtained by using (10) (labeled "Analysis") and by using (8) (labeled "Exact"). It shows that the values of the lower bound on the right hand side of (10) are not very large for very small ϵ's. Expression (10) shows that these values grow very slowly as ϵ decreases or as $\lambda\tau$ increases.

TABLE I

LOWER BOUND ON $k^* - k$ TO ACHIEVE $P\{M(t) = k\} \le \epsilon$ FOR THE EXPONENTIAL DISTRIBUTION: $1/\lambda = 20$ MS

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

We can do similar analysis for the Pareto case. We do not have a more compact expression for (9). Hence, we will show at what value k the factor $\frac{K^\alpha}{(t-(k+1)\tau-T)^\alpha}$ in (9) becomes small, say 0.1. From that point on, as k continues to decrease, the value of $P\{M(t) = k\}$ will rapidly decrease to nearly zero. By setting $\frac{K^\alpha}{(t-(k+1)\tau-T)^\alpha} \le \epsilon$, we get,

$$k^* - k \approx \frac{K}{\tau\epsilon^{1/\alpha}} \quad (11)$$

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 II. With exact numerical analysis on (9), 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 III.

TABLE II

LOWER BOUND ON $k^* - k$ TO ACHIEVE $\frac{K^\alpha}{(t-(k+1)\tau-T)^\alpha} \leq \epsilon$ FOR THE PARETO DISTRIBUTION: MEAN DELAY = 20 MS

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

TABLE III

LOWER BOUND ON $k^* - k$ TO ACHIEVE $P\{M(t) = k\} \leq \epsilon$ FOR THE PARETO DISTRIBUTION: MEAN DELAY= 20 MS

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

2) Asymptotic behavior of $P\{\bar{Q}(t) \leq m\}$: In order to compute $P\{\bar{Q}(t) \leq m\}$ as in (7), we next turn to the calculation of $a(k,m) := P\{..., A_{k-m-2} \leq t, A_{k-m-1} \leq t\}$. 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 $P\{\bar{Q}(t) \leq m\}$ for large values of m. From the previous analysis, we can assume the summation in (7) is over a small number of terms, $k_o, k_o + 1, ..., k^*$. Since $a(k,m)$ increases as k decreases, we get,

$$a(k^*, m) \leq P\{\bar{Q}(t) \leq m\} \leq a(k_o, m) \qquad (12)$$

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

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

Lemma 3: For $x \geq 2\log 2$,

$$\log(1 - e^{-x}) \geq -e^{-x/2} \qquad (13)$$

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

$$\begin{aligned} g'(x) &= \frac{e^{-x} + e^{-3x/2} - e^{-x/2}}{1 - e^{-x}} \\ &\leq \frac{e^{-x}(2 - e^{x/2})}{1 - e^{-x}} \end{aligned}$$

When $2 - e^{x/2} \leq 0$, or equivalently, when $x \geq 2\log 2$, $g'(x) \leq 0$. Since $g(x)$ decreases to 0 on $[2\log 2, \infty)$, it must be true that $g(x) \geq 0$ on $[2\log 2, \infty)$. ■

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

$$\begin{aligned} \log a(k^*, m) &= \sum_{j<k^*} \log(1 - e^{-\lambda(t-(j-m)\tau-T)}) \\ &\geq -\sum_{j<k^*} e^{-\lambda(t-(j-m)\tau-T)/2} \\ &= -\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1 - e^{-\lambda\tau/2}} \qquad (14) \end{aligned}$$

The condition for the above inequality is $\lambda(t-(j-m)\tau - T) \geq 2\log 2$ for $j < k^*$. This is satisfied if $(k^* - j + m) \geq 2\log 2/(\lambda\tau)$ for $j < k^*$. A weaker condition is simply $m \geq 2\log 2/(\lambda\tau)$. When $1/\lambda = 20$ ms, $m \geq 3$, 24 and 232 for the link speed $C_s = 1$, 10 and 100 Mbps, respectively. We'd like to point out that these are very loose bound. In practice, we expect the inequality (14) to hold for much smaller m. Combining (12) and (14), we get

$$P\{\bar{Q}(t) \leq m\} \geq \exp(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1 - e^{-\lambda\tau/2}}) \qquad (15)$$

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

$$\begin{aligned} P\{\bar{Q}(t) > m\} &\leq 1 - \exp(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1 - e^{-\lambda\tau/2}}) \\ &\leq \frac{e^{-\lambda(t-(k^*-m-1)\tau-T)/2}}{1 - e^{-\lambda\tau/2}} \\ &\leq \frac{e^{-(m+2)\lambda\tau/2}}{1 - e^{-\lambda\tau/2}} \qquad (16) \end{aligned}$$

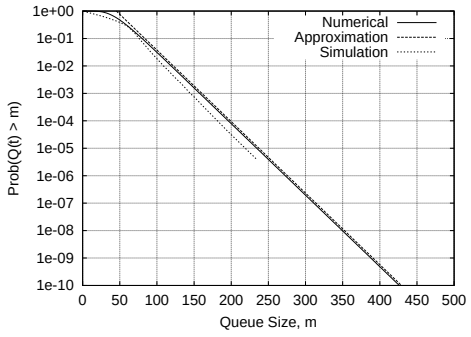
Thus, for large enough m, $P\{\bar{Q}(t) > m\}$ converges to zero very rapidly, at a rate no slower than exponential in m. In the above analysis, if we suppose $\lambda(t - (j-m)\tau - T)$ is large enough, we can write

$$\begin{aligned} \log a(k^*, m) &= \sum_{j<k^*} \log(1 - e^{-\lambda(t-(j-m)\tau-T)}) \\ &\approx -\sum_{j<k^*} e^{-\lambda(t-(j-m)\tau-T)} \\ &= -\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1 - e^{-\lambda\tau}} \qquad (17) \end{aligned}$$

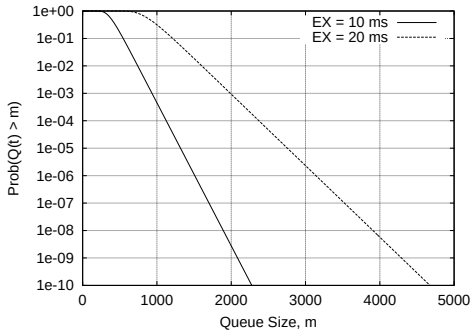
Furthermore, since $e^x \approx 1 + x$, for x near 0,

$$\begin{aligned} P\{\bar{Q}(t) > m\} &\leq 1 - \exp(-\frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1 - e^{-\lambda\tau}}) \\ &\approx \frac{e^{-\lambda(t-(k^*-m-1)\tau-T)}}{1 - e^{-\lambda\tau}} \\ &\approx \frac{e^{-(m+2)\lambda\tau}}{1 - e^{-\lambda\tau}} \qquad (18) \end{aligned}$$

The approximation in (18) turns out to be very good. Figure 5 (a) compare values of $P\{\bar{Q}(t) > m\}$ obtained with (18) (labeled as "Approximation") with numerical results of (7) (labeled as "Numerical") [7]. The mean delay, $1/\lambda$, is 20 ms for figure 5 (a). The approximation agrees extremely well with the numerical results. In figure 5 (a), we also compare the distribution of $\bar{Q}(t)$ with that of $Q(t)$, the real queue length, obtained by simulation (The curves are labeled as "Simulation"). We see that the two distributions agree very well when the queue size is large enough. In figure 5 (b), we show a comparison between two different mean delays: $1/\lambda = 10$ ms and 20 ms. The link speed for figure 5 (b) is $C_s = 100$ Mbps.



(a)



(b)

Fig. 5. Tail probability for queue size for the exponential distribution (a) $C_s = 10$ Mbps; (b) comparison of different mean delays, $C_s = 100$ Mbps.

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

$$\begin{aligned} & \log a(k,m) \\ = & \sum_{j<k} \log(1 - \frac{K^\alpha}{(t-(j-m)\tau - T)^\alpha}) \\ \leq & -\sum_{j<k} \frac{K^\alpha}{(t-(j-m)\tau - T)^\alpha} \\ \leq & \int_{-\infty}^{k-1} -\frac{K^\alpha}{(t-(x-m-1)\tau - T)^\alpha} dx \\ = & \frac{-K^\alpha}{(\alpha-1)\tau}(t-(k-m-2)\tau - T)^{-\alpha+1} \quad (19) \end{aligned}$$

By $P\{\bar{Q}(t) \leq m\} \leq a(k_o, m)$,

$$\begin{aligned} & P\{\bar{Q}(t) \leq m\} \\ \leq & \exp\{\frac{-K^\alpha}{(\alpha-1)\tau}(t - k_o\tau - T + (m+2)\tau)^{-\alpha+1}\} \\ \leq & \exp\{\frac{-K^\alpha}{(\alpha-1)\tau}((k^* - k_o + m + 2)\tau)^{-\alpha+1}\} \quad (20) \end{aligned}$$

$$\begin{aligned} & P\{\bar{Q}(t) > m\} \\ \geq & 1 - \exp\{\frac{-K^\alpha}{(\alpha-1)\tau}((k^* - k_o + m + 2)\tau)^{-\alpha+1}\} \\ \approx & \frac{K^\alpha}{(\alpha-1)\tau}((k^* - k_o + m + 2)\tau)^{-\alpha+1} \quad (21) \end{aligned}$$

Formula (21) shows that the tail probability decays as a power tail. In figure 6, we show an example of the tail probability, $P\{\bar{Q}(t) > m\}$, based on the approximation in (21), and compare the result with those based on numerical analysis. As will be explained in section III-A.3, numerical analysis of the queue length distribution for the Pareto case is difficult. The numerical results in figure 6 are for Pareto distributions "truncated" at large values, denoted by d. In these plots, we see extraordinary 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.

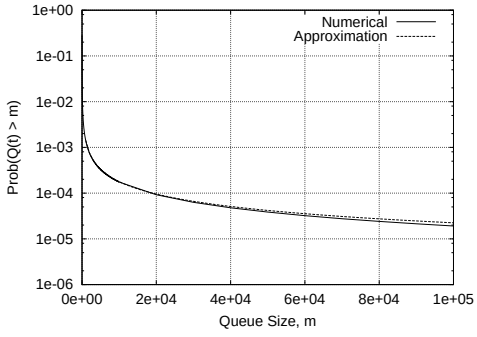


Fig. 6. Tail probability for queue size for the Pareto distribution: $C_s = 10$ Mbps

In figure 7, we compare the distribution of $\bar{Q}(t)$ with that of $Q(t)$, obtained through simulation. The Pareto distribution is truncated at $d = 1000$ seconds. We see that they are very close to each other for large queue size.

After we establish enough confidence on the "goodness" of the approximation, we use the approximation to

[7] Throughout section III, all simulation results are for the real queue size, Q. All analytical and numerical results are for $\bar{Q}$.

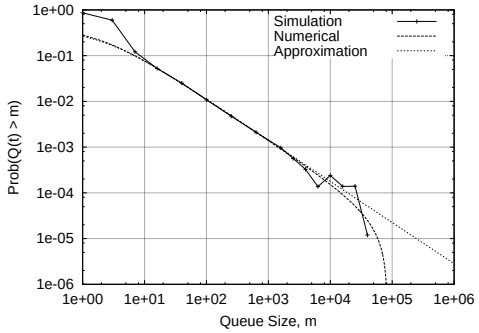


Fig. 7. Tail Probability for Queue Size for the Pareto Distribution: Comparison of $\bar{Q}(t)$ with $Q(t)$. $C_s = 1$ Mbps, $\alpha = 1.9$ and $d = 1000$ seconds

investigate dependency of the tail probability of the queue size on various parameters. The results are shown in figure 8 in log-log scale. We use the tail probability of the queue size, $P\{\bar{Q}(t) > m\}$, as an approximation of the packet loss ratio when the buffer size is m. From figure 8 (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, $m > 10^2, 10^4$, and 10^6 for $\alpha = 1.9$, for $C_s = 1$, 10 and 100 Mbps, respectively. As illustrated in figure 8 (b) and (c), the loss ratio depends crucially on the parameter α, but less crucially on the mean variable delay.

3) Numerical computation of $P\{\bar{Q}(t) \leq m\}$: Because we do not know the value of each $a(k,m)$, it is not easy to say how many A_i's we need in order to compute each $a(k,m)$ with enough accuracy. The fact that the variable delays are always bounded in practice can help us. Specifically, in the case of Pareto distribution, we consider the corresponding "truncated" Pareto distribution.

$$P\{X \leq x\} = \begin{cases} 1 - (\frac{K}{x})^\alpha & \text{for } K \leq x < d \\ 1 & \text{for } x >= d \end{cases}$$

When the variable delay is bounded by d, any packet $i \leq k(d) := \lceil (t - T - d)/\tau \rceil$ must have arrived by time t. Hence, the computation of $a(k,m)$ involves only a finite number of A_i's.

We show the tail probability of the queue size for "truncated" Pareto distribution in figure 9. We also compare the "truncated" case with the "not-truncated" case (labeled as "Pareto"). We see that truncating the Pareto random variable significantly alters the resulting queue size distribution, leading to a much faster decay. We also see that the delay bound, d, dictates the buffer requirement for low loss probability. That is, the buffer size must be as large as the worst case delay, d, multiplied by the sending capacity, C_s.

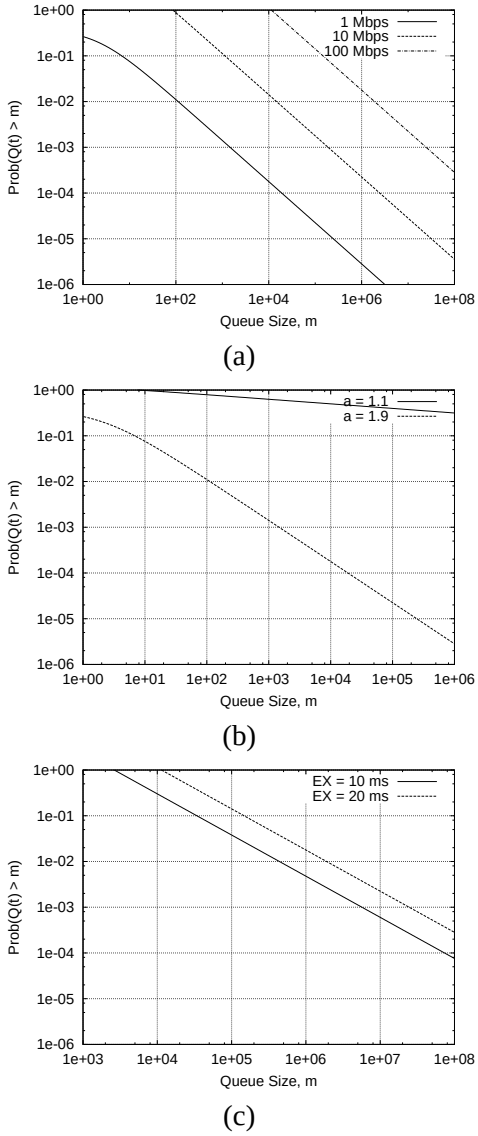


Fig. 8. Tail Probability for Queue Size for the Pareto Distribution: Log-Log Plot (a) $\alpha = 1.9$, Mean Delay = 20 ms; (b) $Cs = 1$ Mbps, Mean Delay = 20 ms, and (c) $C_s = 100$ Mbps, $\alpha = 1.9$, Mean Delays = 10 and 20 ms

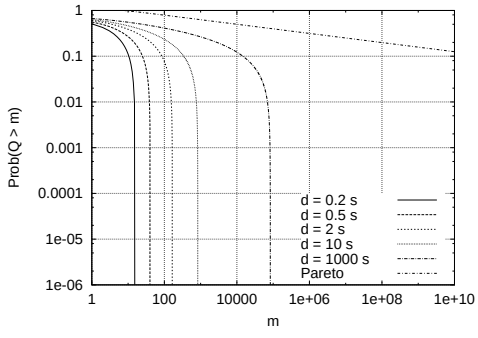


Fig. 9. Tail probability for queue size for the "truncated" Pareto distribution: log-log Plot, mean delay = 10 ms. $\alpha = 1.1$, $C_s = 1$ Mbps.

B. Waiting Time in the Queue

Packets are often delayed at the receiver queue for them to be accepted in order. In this section, we analyze the waiting time distribution. 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 $j \leq i$ arrive at the queue. Therefore, packet i's waiting time is,

$$\begin{aligned} W_i &= \max_{j \leq i} A_j - A_i \\ &= \max_{j \leq i} \{A_j - A_i\} \end{aligned}$$

Hence, for $t \geq 0$,

$$\begin{aligned} & P\{W_i \leq t\} \\ = \; & P\{A_j - A_i \leq t, \text{ for all } j < i\} \\ = \; & \prod_{j<i} P\{A_j - A_i \leq t\} \\ = \; & \int \prod_{j<i} F(x + (i-j)\tau + t) dF(x) \end{aligned} \tag{22}$$

When the variable delay is bounded by d,

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

The mean waiting time of a packet i is

$$\mathbf{E}W = \mathbf{E} \max_{j \leq i} A_j - \mathbf{E}A_i \tag{23}$$

1) Waiting time for exponential variable delay: In the case of exponential variable delay, we can find an expression that can approximate 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,

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

where the Y_j's are IID exponential random variables representing packets' remaining times in the network. We know from [5] (page 49) that the k^{th} order statistics for a collection of n IID exponential random variables has the expectation

$$\mathbf{E}X_{(k)} = \frac{1}{\lambda} \sum_{i=1}^{k} \frac{1}{n-i+1} \tag{24}$$

where $1/\lambda$ is the mean of the exponential distribution and $k = 1, 2, ..., n$. Conditional on G_i and using (24), 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.

$$\mathbf{E}[W_i] \approx \frac{1}{\lambda}(\log \mathbf{E}G_i + 0.5) \tag{25}$$

The expected value of G_i can be computed as follows.

$$\begin{aligned} \mathbf{E}[G_i] &= \mathbf{E}[\sum_{j<i} \mathbf{1}_{(A_j > A_i)}] \\ &= \sum_{j<i} P\{A_j > A_i\} \\ &= \sum_{j<i} P\{X_j - X_i > (i-j)\tau\} \\ &= \sum_{j<i} \frac{1}{2} e^{-\lambda(i-j)\tau} \\ &= \frac{1 - e^{-\lambda i \tau}}{2(1 - e^{-\lambda\tau})} \end{aligned}$$

Substituting the result for $\mathbf{E}[G_i]$ into (25) and let i goes to infinity, we get the stationary mean waiting time.

$$\mathbf{E}[W] \approx \frac{1}{\lambda}(\log(\frac{1}{2(1 - e^{-\lambda\tau})}) + 0.5) \tag{26}$$

In table IV, 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 (26) 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 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,

$$\mathbf{E}[W] \approx \frac{1}{\lambda}(\log(\frac{C_s}{2\lambda U}) + 0.5) \tag{27}$$

where U is the packet size in bits. We see that the mean waiting time of a packet scales logrithmically 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.

TABLE IV
MEAN WAITING TIME FOR 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

2) Waiting time for Pareto variable delay: Table V 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 seconds. We see that $\mathbf{E}W$ depends d, α and C_s in significant ways. In many cases, this resequencing delay is non-trivial. When the Pareto distribution has "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 time-out mechanism similar to the one used in TCP, then it is typical d ranges from 0.5 to 2 seconds. The resequencing delay ranges from 232 ms to 1.8 seconds 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 seconds can greatly reduce the expected delay. In the case of TCP, there is incentive to reduce the time-out value, which depends on the round-trip time of the transmission path. Therefore, it pays to accurately estimate the round-trip time.

C. Throughput for finite buffer sizes

This section shows some simulation results of the throughput under finite buffer sizes. Figure 10 is for the exponential delays. Figure 11 shows similar result for the "truncated" Pareto distributions on semi-log scale. We see that the delay bound, d, severely affects the throughput-buffer characteristics.

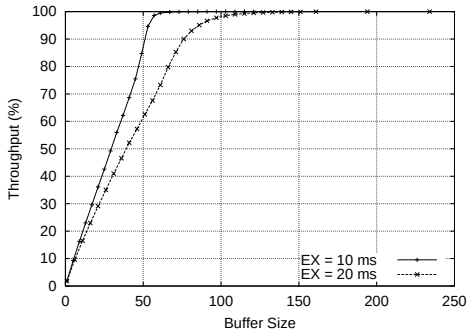


Fig. 10. Throughput versus buffer size for the exponential distribution. $C_s = 10$ Mbps, $T = 30$ ms.

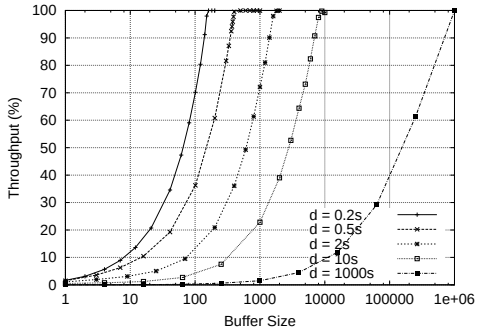


Fig. 11. Throughput versus buffer size for the Pareto distribution. $C_s = 10$ Mbps, semi-log scale.

IV. CONCLUSION

This paper studies the required buffer size and extra delay for resequencing mis-ordered packets at the receiver. The results are summarized in section I-B. We have shown that both quantities can be significant. Besides requiring extra resource, the delay caused by packet resequencing negatively affects the performance of delay-sensitive applications. A network that causes severe packet mis-ordering, such as one that allows multi-path routing, must employ mechanisms to reduce the resequencing delay. Fast detection and feedback about packet that are delayed by the network and retransmission of these packets can be very helpful or even necessary. The implication of our results on network engineering, besides the cautionary notes on multi-path routing, is suggested by

TABLE V
MEAN WAITING TIME FOR PARETO DELAY: SIMULATION RESULTS

		$\mathbf{E}[W]$: Simulation (ms)				
	C_s(Mbps)	$d = 0.2$ s	$d = 0.5$ s	$d = 2$ s	$d = 10$ s	$d = 1000$ s
$\mathbf{E}X = 20$ ms	1	33.12	81.82	279.15	1093.49	51403.21
$\alpha = 1.1$	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	1	11.56	20.02	34.52	59.00	71.81
$\alpha = 1.9$	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	1	3.73	6.08	9.92	16.89	20.02
$\alpha = 1.9$	10	37.50	58.27	94.60	158.44	211.46
	100	121.30	232.90	519.41	1079.91	1205.94

fact that the mean resequencing delay is proportional to $\log(C_s/U)$ (See equation (4).) for the exponential propagation delay. As the sources become faster, the resequencing delay increases. Moreover, making the packet size smaller also increases the resequencing delay.

REFERENCES

[1] M. E. Anagnostou and E. N. Protonotarios. Performance analysis of the selective repeat ARQ protocol. *IEEE Transactions on Communications, Vol. COM-34, No. 2*, pages 127–135, February 1986.

[2] F. Baccelli, E. Glenbe, and B. Plateau. An end-to-end approach to the resequencing problem. *Journal of the Association for Computing Machinery, Vol. 31, No. 3*, pages 474–485, July 1984.

[3] F. Baccelli and A. M. Makowski. Queueing models for systems with synchronization constraints. *Proceedings of the IEEE, Vol. 77, No. 1*, pages 138–161, January 1989.

[4] S. Chowdhury. An analysis of virtual circuits with parallel links. *IEEE Transactions on Communications, Vol. 39, No. 8*, pages 1184–1188, August 1991.

[5] H.R. David. *Order Statistics*. John Wiley & Sons, 2nd edition, 1981.

[6] N. Gogate and S. S. Panwar. On a resequencing model for high speed networks. In *Proceedings of INFOCOM '94*, pages 40–47, Toronto, Canada, June 1994.

[7] G. Harrus and B. Plateau. Queueing analysis of a reordering issue. *IEEE Transaction on Software Engineering, Vol. SE-8, No. 2*, pages 113–123, March 1982.

[8] I. Iliadis and L. Y.-C. Lien. Resequencing delay for a queueing system with two heterogeneous servers under a threshold-type scheduling. *IEEE Transactions on Communications, Vol. 36, No. 6*, pages 692–702, June 1988.

[9] F. Kamoun, L. Kleinrock, and R. Muntz. Queueing analysis of the reordering issue in a distributed database concurrency control mechanism. In *Proceedings of the 2nd International Conference on Distributed Computing Systems*, pages 13–23, Versailles, France, April 1981.

[10] A. G. Konheim. A queueing analysis of two ARQ protocols. *IEEE Transactions on Communications, Vol. COM-28, No. 7*, pages 1004–1014, July 1980.

[11] 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*, pages 189–197, Tokyo, Japan, September 1985.

[12] M. J. Miller and S.-L. Lin. The analysis of some selective-repeat ARQ schemes with finite receiver buffer. *IEEE Transactions on Communications, Vol. COM-29, No. 9*, pages 1307–1315, September 1981.

[13] Z. Rosberg and N. Shacham. Resequencing delay and buffer occupancy under the selective-repeat ARQ. *IEEE Transactions on Information Theory, Vol. 35, No. 1*, pages 166–172, January 1989.

[14] Z. Rosberg and M. Sidi. Selective-repeat ARQ: the joint distribution of the transmitter and the receiver resequencing buffer occupancies. *IEEE Transactions on Communications, Vol. 38, No. 9*, pages 1430–1438, September 1990.

[15] B. H. Saeki and I. R. Rubin. An analysis of a TDMA channel using stop-and-wait, block, and selective-and-repeat ARQ error control. *IEEE Transactions on Communications, Vol. COM-30, No. 5*, pages 1162–1173, May 1982.

[16] D. Towsley and J. K. Wolf. On the statistical analysis of queue lengths and waiting times for statistical multiplexers with ARQ retransmission schemes. *IEEE Transactions on Communications, Vol. COM-27, No. 4*, pages 693–702, April 1979.

[17] T.-S. P. Yum and T.-Y. Ngai. Resequencing of messages in communication networks. *IEEE Transactions on Communications, Vol. COM-34, No. 2*, pages 143–149, February 1986.