Efficient Computation of Queueing Delay at a Network Port from Output Link Packet Traces

M. Farhan Habib
Department of Computer Science and Engineering
University of California, Riverside
Email: habib@cs.ucr.edu

Mart Molle
Department of Computer Science and Engineering
University of California, Riverside
Email: mart@cs.ucr.edu

***Abstract*—Current Internet core routers provide enough buffer capacity at each output port to keep the link busy for 250 msec. to avoid disrupting TCP flows because of dropped packets. Since link speeds are rising much more quickly than the availability and cost-effectiveness of large high-speed memories, there is now significant interest in reducing these buffers. Queue inferencing (QI) — a passive, external method for calculating the time-dependent queue lengths and waiting times using start/end service event timestamps — is an ideal tool for studying the effects of such buffer size reductions because it can be applied *in situ* to packet traces collected from existing equipment carrying live traffic without any service disruptions. Although existing QI algorithms are too computationally expensive for this purpose, we observe that packet-sizes in a typical network trace are skewed towards a few "favored" sizes, and introduce two different methods for exploiting this property. First, since the output of a QI algorithm is a deterministic function of the service time sequence in a busy period, and our traces contain many repetitions of common packet-size sequences, we cache the output of the QI algorithm for later reuse. Second, we develop a new incremental QI algorithm, which can start from cached results for any prefix of the current busy period. Combining both methods leads us to structure the cache as a decision tree. Using network traces from WIDE project backbone to evaluate our method, we found that both the frequency of occurrence for particular busy periods and for busy-period lengths follow a decreasing power law, where busy periods lengths greater than 20 were very rare and none were greater than 70. Moreover, although we found that the size of the *complete* decision tree grows linearly with the length of the trace, we can restrict the tree to a *finite number* of "active" nodes (i.e., those nodes for which the probability of a visit is above some threshold) and use *simple constant-time bounds* to handle the rare exceptional cases.**

I. Introduction

Consider a single-server FIFO queueing system, where customer i arrives at time a_i, begins service at time b_i, and departs at time d_i (after receiving $x_i \equiv d_i - b_i$ units of service, assuming the system is work conservative). Given access to "event logs" with this data, we can easily find the waiting $w_i \equiv b_i - a_i$ or sojourn time $t_i \equiv d_i - a_i$ for the ith customer, or evaluate any operational value for their respective distributions (i.e., means, percentiles, etc).

Unfortunately, for a number of important application domains where such event logs are readily available, a_i is not directly measureable. For example, the motivation for Larson's pioneering work [1] was to estimate customer waiting times at an automated banking machine in a remote location without actually visiting it, using only the sequence of card-insertion and card-removal events from the machine's transaction log to represent $\{b_i\}$ and $\{d_i\}$. The methods for estimating $\{a_i\}$ from such event logs are called *queue inferencing algorithms*.

The goal of our work is to develop a queue inferencing algorithm that is fast enough to process real-time packet traces generated by a passive "network sniffer" attached to a backbone link in the Internet core. In this way, we can provide a mechanism for viewing the buffer occupancy at an output port of a high-performance Internet backbone router, *non-obtrusively and during its normal operation*. To see this, consider that the ith record in the packet trace contains (at least) the packet length l_i and an end-of-packet timestamp τ_i. Without loss of generality, we can ignore the fixed propagation delay between the physical locations of the output port and "sniffer" and assume that $d_i \equiv \tau_i$. Furthermore, given the physical and data-link layer parameters for the link, we can see immediately that the service at the output port for transmitting the ith packet over this link is $x_i \equiv (l_i + f)/C$, where f is the fixed framing overhead added to each packet and C is the capacity of the link.

The significance of our work is to provide a simple method for collecting operational data from existing equipment to help guide the current proposals for reducing the output buffer capacities in commercial routers. A well-known "rule of thumb" for router design [2], [3] states that output buffers should be sized to handle the combined bandwidth-delay product for all flows traversing the link, or about 250 ms of packet data. A backbone router generally requires at least 10 Gbits of buffer memory [4] which is a major hardware cost. High power consumption and higher cost of SRAM and slow access time of DRAM makes it more difficult to build buffers for backbone router. Under this scenario, some recent works proposed a considerable reduction in backbone router buffer sizes, still maintaining the performance. Appenzeller el al. [4] proposed that a buffer of size $RTT \times C/\sqrt{N}$ is sufficient for maintaining the throughput, where C is the link capacity, N is the number of flows and RTT is the average round-trip propagation delay. Later Beheshti et al. [5] performed several simple experiments on operational backbone networks to evaluate the effect of reducing output buffer sizes. However, it cannot be duplicated without the full cooperation of a network operator. Moreover, artificially reducing buffer sizes

to see how it degrades performance is not going to please paying customers. An accurate model to infer the queueing behavior can solve this problem.

The rest of the paper is organized as follows. In section II, we describe the queue inferencing problem and define our notation. In section III, we present the new incremental QI algorithm in detail and derive its complexity. We validate our model through some experimental results in section IV-A. In section IV-B, we show some experimental results from some actual network traces giving the relative frequencies of busy periods with different sequences of packet sizes in a weighted search tree representation, and estimate the optimal speedup for our incremental QI algorithm if it organized the cached partial results into the same structure. In section V, we find the lower and upper bound for the arrival time which can be computed in linear time. We show how we can bound the size of the decision tree using that bound of the arrival time. And finally, in section VI, we conclude the paper.

II. The Queue Inferencing Problem

Queue inferencing algorithms estimate customer waiting times and/or queue lengths within a single busy period (BP) from its sequence of start-service and end-service event times. Thus, we introduce the following notation for describing the current (active) busy period:

$$
\begin{aligned}
n &= \text{Number of customers in this busy period} \\
a_i &= \text{Arrival time for } i\text{th customer} \\
b_i &= \text{Begin-service time for } i\text{th customer} \\
d_i &= \text{Departure time for } i\text{th customer} \\
x_i &= d_i - b_i;\ \text{Service time for } i\text{th customer} \\
m_i &= \text{Number of arrivals during the } i\text{th service time}
\end{aligned}
$$

We then extend the notation to include aggregates of each basic element type. Using m_i, the number of arrivals in a service time as an example, we have:

$$
\begin{aligned}
\vec{m}(i) &= [m_1, \ldots, m_i];\ \text{vector of arrivals during the first } i \text{ service times of the busy period, } 1 \le i \le n \\
M_i &= m_1 + \ldots + m_i;\ \text{Scalar sum over } \vec{m}(i)
\end{aligned}
$$

Finally, we define the following sets of arrival vectors:

$$
\begin{aligned}
\mathcal{M}_{n,k} &= \{\vec{m}(n) \mid 0 \le m_i\ \forall i \le n, M_n = k\};\ \text{The set of all possible partitions of } k \text{ arrivals into } n \text{ groups} \\
\mathcal{M}^*_{n,k} &\subseteq \mathcal{M}_{n,k};\ \text{Those partitions of } k \text{ into } n \text{ groups that form a single busy period, i.e., } M_i \ge i\ \forall i < n.
\end{aligned}
$$

Using these definitions, we see that $\mathcal{M}^*_{n,n-1}$ is the set of all arrival vectors that represent a *complete* busy period of length n, in which case we know that $m_n \equiv 0$ must hold.

Larson [1] introduced the queue inferencing problem by describing an $O(n^5)$ "batch" algorithm, which he called the Queue Inference Engine (QIE), for finding the mean waiting time for each customer, the time-dependent mean queue length, and the probability distribution function for the queue size as seen by a randomly-chosen arrival over a single n-customer busy period, given only the sequence of start-service and end-service events within that busy period. The results generated by the QIE are exact if the interarrival time distribution is exponential (with any constant, but possibly unknown, rate) and approximately correct if the interarrival time distribution has a time-dependent exponential distribution whose changes slowly over the length of each busy period. Bertsimas and Servi [6] subsequently developed an $O(n^3)$ algorithm for calculating transient queue lengths and customer waiting times in an n-customer busy period with Poisson arrivals, while Larson [7] discovered a modification of his original $O(n^5)$ algorithm to reduce its complexity $O(n^3)$.

The starting point for both algorithms is the recognition that the unconditional joint density function for any arrival pattern $[a_2, \ldots a_n]$ over the interval $(0, t]$ is simply $(n-1)!/t^{n-1}$ under the Poisson arrivals model. However, if these arrivals form an n-customer busy period then we must impose the constraint that $a_1 = 0,\ a_i \le d_{i-1},\ 1 < i \le n$. For example, by integrating ($n-1$ times) the expression for the number in system at d_i given $\vec{a}(n)$ over the constraint region, they obtained an $O(n^3)$ expression for the mean queue size at the ith departure instant. Repeating this $O(n^3)$ calculation at every departure instant within the busy period *multiplies* this complexity by a factor of n. Since the average number in system is a linear function between departure instants, the overall complexity of finding the overall average number in system, and hence the average waiting time using Little's law, using their queue inferencing method is $O(n^4)$.

Later on, Manjunath and Molle [8] generalized the method to shared-medium Local Area Networks, where they could infer information about individual nodes connected to the LAN from its individual contribution to the combined packet trace and knowledge about the Medium Access Control protocol.

Bertsimas and Servi [6] also extended the method to allow the interarrival time distribution to be a general renewal process, such as the Erlang-k and hyper-exponential distributions. Unfortunately, non-exponential interarrival time distributions require (computationally expensive) numerical methods for evaluating multi-dimensional integrals.

Daley and Servi [9] proposed a different method for solving the queue inferencing problem. They observed that the distribution of the number in system at departure instants, $\{d_i\}$, forms an embedded Markov chain under the Poisson arrival assumption, and developed an exact $O(n^3)$ algorithm for calculating this distribution at each departure point, together with an $O(n^2 log(n))$ algorithm for approximating this distribution to a specified accuracy by assuming a choice of arrival rate for which the numerical values for terms far from the mean can be neglected. They also described how to apply their method to k-Erlang and hyper-exponential interarrival time distributions, by extending the Markov chain to include the number of "exponential stages" present at a departure instant.

Hohn et. al. [10] developed an analytical model for the packet forwarding delay in a router, from the input-port to the output-port. In their method, packet traces are collected

simultaneously from all router ports in a tier-1 operational access router, and the arrival of each packet packet (at port i, say) is matched with its subsequent departure (from port j, say). They figured out that the forwarding delay has two major components: queueing delay at the output buffer, and processing delay due to path selection, segmentation of a packet into fixed-length cells, transmission of cells through switching fabric, and reassembly upon arrival at the output buffer. They observed that, except for the first packet of each output-port busy period, the total forwarding delay for a packet is independent of its processing delay. Thus, the processing delay is estimated as the minimum forwarding delay from their measurement data for the first packet of each busy period, and ignored for all other packets. The accuracy of the predicted delay depends on that pre-computed minimum value. On the other hand, this minimum delay is variable in reality and the effect of the inaccurate delay value could be cumulative on the following packets within a busy period.

In general, these QI algorithms impose several key assumptions to make the problem tractable. Most importantly, the systems must be *work conservative*, so $b_i > d_{i-1}$ implies that the system was completely empty between those two event times. In addition, $\{a_i\}$ should be generated by a Poisson process whose rate is assumed constant over each busy period.

Under the Poisson assumption the system has no memory across idle periods, so each busy period can be studied independently. Therefore, without loss of generality, we can assume that the "tagged" busy period begins at time zero with the arrival of customer 1 and ends after n customers have been served where $d_n < b_{n+1}$ for the first time. Furthermore, we can exploit the property that Poisson arrivals are uniformly distributed over any given interval.

Although several authors [6], [9] studied the generalization to renewal-type interarrival times (i.e., k-Erlang, hyper-exponential, or similar application of the "exponential stages"), the complexity of the algorithms is greatly increased by this modification. Indeed, by implementing the Bertsimas and Servi algorithm, Gawlick [11] found the non-exponential version method to be prohibitively expensive:

> "For example, for the k-Erlang arrival distribution, the evaluation of $E[Q(t)]$ using integral III has a complexity of $O(n^2 2^n)$ even at the lowest possible degree of accuracy. On a 68030 at 24Mhz with a f80882 coprocessor it took two days to calculate expected queue lengths with the assumption of a 2-Erlang arrival distribution for a busy period with 11 packets."

Furthermore, we recognize that there is a long history of work showing that the measured behavior of network traffic is not consistent with the Poisson assumption because of long-range dependence in the interarrival time process and heavy tailed distributions for the aggregate number of arrivals over large intervals. However, our primary goal in this paper is to study queueing behavior at a high-speed output port for Internet core router. This exact situation was recently studied by Karagiannis et al. [12], who found that network backbone traffic can be viewed as a piecewise-constant Poisson process punctuated by distinct "rate-change events" that occur at a time scale of seconds – which is orders of magnitude greater than the duration of a busy period – and (only) the sequence of rate-change events demonstrates long-range dependence. Similarly, Appenzeller et al. [4] observed that with sufficiently large number of flows (which is true for core routers), in-phase synchronization is very rare and so bursts are smoothed out which indicates the Poisson behavior of the arrival process.

In the remainder of this paper, we develop a new "cache-assisted" QI algorithm that is specifically designed for processing network traces. First, since the output of a QI algorithm is a *deterministic* function of the service-time sequence $\vec{x}(n)$ in a busy period – and packet sizes in a network trace are *highly concentrated* onto a small number of "popular" values, determined by the dynamics of the higher layer protocols – we can substantially reduce the total execution time for any QI algorithm by *caching* the answer for each service-time sequence the first time it is encountered, rather than recomputing it from scratch every time. Second, we develop a new *incremental* QI algorithm, which greatly extends the concept of caching by showing how to efficiently compute the average delay for the BP pattern $\vec{x}(n)$ from previously-cached partial results for BP patterns $\vec{x}(1), \vec{x}(2), \ldots, \vec{x}(n-1)$.

III. Cache-Assisted Convolution Algorithm

A. Derivation

Let $a_1 = b_1 \equiv 0$ be the start of a new busy period, and let $\vec{m}(i) = [m_1, \ldots, m_i]$, with scalar sum $M_i = m_1 + \ldots + m_i$, be a partition of the subsequent arrivals at end-of-service events $(0, d_1], \ \ldots, (d_{i-1}, d_i]$. Clearly, if the system is work conserving, then arrivals $1, \ldots, i$ will all be served in a single busy period if $a_j \le d_{j-1}$ for all $j \le i$. In other words, if the server always finds at least one customer in the queue at each end-of-service event, it will never go idle. Recognizing that a_1 does not get counted in M_j and there is no arrival in the last slot (i^{th} slot), we see that $(a_j \le d_{j-1}) \equiv (1 + M_{j-1} \ge j)$, or

$$M_j \ge j \quad \forall j < i. \tag{1}$$

More formally, we say that customers $1, \ldots, i$ are served in a single busy period if and only if $\vec{m}(i) \in \mathcal{M}^*_{i,M_i}$, where:

$$\mathcal{M}^*_{n,k} \equiv \{\vec{m}(n) \mid m_j \ge 0 \ \forall j; \ M_j \ge j \ \forall j < n; \ M_n = k\} \tag{2}$$

represents the set of all partitions of k elements into n groups, subject to the constraint of Eq.(1).

LEMMA 1 For all $n > 0$ and all $k \ge n-1$, the set of ways to distribute k arrivals over n service times so as to generate a single n-customer busy-period can be found from the recurrence relation:

$$\mathcal{M}^*_{n,k} = \bigcup_{j=0}^{k-n+1} \left\{\vec{m}(n-1) \,\|\, [j] \mid \vec{m}(n-1) \in \mathcal{M}^*_{n-1,k-j})\right\} \tag{3}$$

starting from the base case $\mathcal{M}^*_{1,k} = \{[k]\}$.

Proof: If the arrival pattern $\vec{m}(n)$ generates a single busy period that covers (at least) n customers, then the busy period did not end with the $(n-1)$st customer and thus the arrival pattern $\vec{m}(n-1)$ must have generated a single busy period that covers (at least) $(n-1)$ customers. Thus, combining Eq.(1) with the observation that every arrival in $\vec{m}(n-1)$ is also in $\vec{m}(n)$, we must have $(n-1) \leq M_{n-1} \leq M_n \equiv k$, and hence the number of arrivals during the final interval satisfies $0 \leq m_n \leq k-(n-1)$.

LEMMA 2

$$P(\vec{m}^*(i) \mid \vec{x}(i), \mathcal{M}^*_{i,k}) = \frac{1}{G^*(\vec{x}(i),k)} \prod_{j=1}^{i} \frac{x_j^{m_j}}{m_j!} \tag{4}$$

where we define the following normalization constant

$$G^*(\vec{x}(i),k) = \sum_{\vec{\mu}(i) \in \mathcal{M}^*_{i,k}} \prod_{j=1}^{i} \frac{x_j^{\mu_j}}{\mu_j!} \tag{5}$$

Proof: Under the Poisson assumption, the k arrivals have an i.i.d. uniform distribution over $(0, X_i)$. Thus, the multinomial distribution gives us the unconditional probability of forming the partition $\vec{m}(i) \rightarrow \vec{x}(i)$:

$$P(\vec{m}(i) \mid \vec{x}(i)) = \frac{k!}{\prod_{j=1}^{i} m_j!} \prod_{j=1}^{i} \left(\frac{x_j}{X_i}\right)^{m_j} \tag{6}$$

Since the operational log tells us that the partition of arrivals created a single busy period, we must convert Eq. (6) into a conditional probability, given $\vec{m}^*(i) \in \mathcal{M}^*_{i,k}$:

$$\begin{aligned} P(\vec{m}^*(i) \mid \vec{x}(i), \mathcal{M}^*_{i,k}) &= \frac{P(\vec{m}^*(i) \mid \vec{x}(i))}{\sum_{\vec{\mu}(i) \in \mathcal{M}^*_{i,k}} P(\vec{\mu}^*(i) \mid \vec{x}(i))} \\ &= \frac{\frac{k!}{(X_i)^k} \prod_{j=1}^{i} \frac{(x_j)^{m_j}}{m_j!}}{\frac{k!}{(X_i)^k} \sum_{\vec{\mu}(i) \in \mathcal{M}^*_{i,k}} \prod_{j=1}^{i} \frac{(x_j)^{\mu_j}}{\mu_j!}} \end{aligned}$$

which gives Eqs. (4–5) after canceling the common factors.

LEMMA 3: For all $i > 0$ and $k \geq i-1$, the normalization constant in Eq.5 can be found from the recurrence relation:

$$G^*(\vec{x}(i),k) = \sum_{m_i=0}^{k-i+1} \frac{x_i^{m_i}}{m_i!} \cdot G^*(\vec{x}(i-1), k-m_i) \tag{7}$$

starting from the base case $G^*(\vec{x}(1),k) = x_1^k/k!$.

Proof: Using Lemma 1, Eq.(5) can be expanded into the following double summation by conditioning on m_i:

$$G^*(\vec{x}(i),k) = \sum_{m_i=0}^{k-i+1} \sum_{\substack{\vec{\mu}(i-1) \in \\ \mathcal{M}^*_{i-1,k-m_i}}} \prod_{j=1}^{i-1} \frac{x_j^{\mu_j}}{\mu_j!} \cdot \frac{x_i^{m_i}}{m_i!} \tag{8}$$

By moving the common factor $x_i^{m_i}/m_i!$ ahead of the inner summation, we see that the inner sum can be reduced to by $G^*(\vec{x}(i-1), k-m_i)$ using Eq.(5), which proves the lemma.

Let us now turn our attention to the calculation of average waiting times. Given $\vec{a}(i) = [a_1, a_2, \ldots, a_i]$ as the vector of arrival times for the first i customers, and $A_i = a_1 + \ldots + a_i$ as its scalar sum, we can find the average waiting time for these customers as $(D_i - A_i)/i$, where D_i is the sum of departure times for the same customer sequence. Since $d_i \equiv X_i = x_1 + \ldots x_i$ is readily available from the event logs, the only challenge is finding a suitable estimate for A_i starting from the event-log data.

THEOREM 1: Let $\overline{A^*}_k(\vec{x}(i))$ be the expected value for the sum of the k arrival times that occur during the interval $(0, X_i]$, given that the associated (but unknown) arrival pattern $\vec{m}(i) \in \mathcal{M}^*_{i,k}$ and hence all of the arrivals fall into a single busy period. Then for all $i \geq 1$:

$$\overline{A^*}_k(\vec{x}(i)) = \frac{H^*(\vec{x}(i),k)}{G^*(\vec{x}(i),k)}, \tag{9}$$

where $H^*(\vec{x}(i),k)$ can be found from the recurrence relation:

$$\begin{aligned} H^*(\vec{x}(i),k) = \sum_{m_i=0}^{k-i+1} \frac{x_i^{m_i}}{m_i!} \Big(& H^*(\vec{x}(i-1), k-m_i) + \\ & G^*(\vec{x}(i-1), k-m_i) \cdot m_i \left(X_{i-1} + \frac{x_i}{2}\right)\Big) \end{aligned} \tag{10}$$

starting with the base case $H^*(\vec{x}(1),k) = \dfrac{x_1^{k+1}}{2(k-1)!}$ and hence $\overline{A^*}_k(\vec{x}(1)) = kx_1/2$.

Proof: Under the Poisson assumption, if m_i arrivals fall within the ith service time of the busy period, those arrival times will be i.i.d. uniform $(X_{i-1}, X_i]$ variables with mean $X_{i-1} + x_i/2$. Thus for any given arrival pattern $\vec{m}(i) \in \mathcal{M}^*_{i,k}$, we have:

$$\overline{A^*}_k(\vec{x}(i), \vec{m}(i)) = \sum_{j=1}^{i} m_j \cdot \left(X_{j-1} + \frac{x_j}{2}\right) \tag{11}$$

Unconditioning over $\vec{m}(i)$, Eq.(11) yields

$$\overline{A^*}_k(\vec{x}(i)) = \sum_{\vec{m}(i) \in \mathcal{M}^*_{i,k}} P(\vec{m}(i), \mathcal{M}_{i,k}) \sum_{j=1}^{i} m_j \cdot \left(X_{j-1} + \frac{x_j}{2}\right)$$

$$\begin{aligned} H^*(\vec{x}(i),k) &\triangleq G^*(\vec{x}(i),k) \cdot \overline{A^*}_k(\vec{x}(i)) = \\ & \sum_{\vec{m}(i) \in \mathcal{M}^*_{i,k}} \left(\prod_{j=1}^{i} \frac{x_j{}^{m_j}}{m_j!}\right) \left(\sum_{j=1}^{i} m_j \cdot \left(X_{j-1} + \frac{x_j}{2}\right)\right) \end{aligned} \tag{12}$$

where we have introduced the compound function $H^*(\cdot)$ to reduce the notational clutter below. Substituting $i = 1$ into Eq.(12) gives us the base case for evaluating $H^*(\cdot)$. Similar to Lemma 3, we can expand the outer sum in Eq.(12) using Lemma 1 by conditioning on m_i, to obtain:

$$\begin{aligned} & H^*(\vec{x}(i),k) = \\ & \sum_{m_i=0}^{k-i+1} \frac{x_i^{m_i}}{m_i!} \sum_{\substack{\vec{m}(i-1) \in \\ \mathcal{M}^*_{i-1,k-m_i}}} \left(\prod_{j=1}^{i-1} \frac{x_j^{m_j}}{m_j!}\right) \left(\sum_{j=1}^{i} m_j \left(X_{j-1} + \frac{x_j}{2}\right)\right) \end{aligned} \tag{13}$$

By partitioning the inner sum, we can apply Eq.(13) to the first $i-1$ terms, and Eq.(5) to the ith term, to obtain Eq.(10).

Notice that Eqs.(7) and (10) include all $k \geq (i-1) \geq 0$. However, if $k = i-1$ then the busy period must end after the i-th packet, in which case these expressions reduce to $G^*(\vec{x}(i), i-1) \equiv G^*(\vec{x}(i-1), i-1)$, and $H^*(\vec{x}(i), i-1) \equiv H^*(\vec{x}(i-1), i-1)$. Thus, only the terms $k \geq i \geq 1$ in Eqs.(7) and (10) are needed to solve the recurrence relations.

B. Cache Structure

Consider the execution of our convolution algorithm across a four-packet busy period. Starting from the root of the decision tree, the algorithm traverses the tree, one "step" per packet, to visit node $\vec{x}(1)$ after seeing x_1, $\vec{x}(2)$ after seeing x_2, and so on until it ends the busy period from node $\vec{x}(4)$ when it sees the first idle period.

G*(x(1),0)	G*(x(1),1)	G*(x(1),2)	G*(x(1),3)	G*(x(1),4)	G*(x(1),5)
	G*(x(2),1)	G*(x(2),2)	G*(x(2),3)	G*(x(2),4)	G*(x(2),5)
		G*(x(3),2)	G*(x(3),3)		
			G*(x(4),3)		

Fig. 1. Triangular cache structure for $G^*(\vec{x}(i), k)$. Columns show increasing busy-period lengths while rows give total number of arrivals in them.

This path-traversal of the decision tree is shown in Figure 1, where the jth row represents node $\vec{x}(j)$, and the columns show partial results for G^* cached at the corresponding node. (For simplicity, we omit the partial results for H^*, which use the same structure.) When the algorithm visits some node $\vec{x}(j)$ along this path (after possibly creating the node $\vec{x}(j)$ if this is the first occurrence of this particular packet-length sequence), it checks the cache at this node to see if it already contains $G^*(\vec{x}(j), j-1)$, which is the desired value of G^* if the busy period ends after this packet. If so, then no further calculations are required and the algorithm simply waits for the next packet arrival. Otherwise, it goes back up to node $\vec{x}(j-1)$ and compute $G^*(\vec{x}(j-1), j-1)$ (if it's not already there) as a weighted sum of $G^*(\vec{x}(j-2), j-1)$ and $G^*(\vec{x}(j-2), j-2)$ using Lemma 3 and then adds $G^*(\vec{x}(j), j-1) \equiv G^*(\vec{x}(j-1), j-1)$ into the cache for node $\vec{x}(j)$.

Assume the shaded entries in Figure 1 were added to the cache in a previous 6-packet busy period in which the first two packets (x_1 and x_2) are identical to those in the current busy period, but x_3 is different. Starting from the beginning of the new BP, the algorithm visits the existing cache nodes $\vec{x}(1)$ and $\vec{x}(2)$ represented by rows 1 and 2 in the Figure. Since cache node $\vec{x}(i)[i = 1, 2]$, already contains $G^*(\vec{x}(i), i-1)$, it just waits for the next packet arrival. In this case, since x_3 follows x_2 without an idle period, and is not already in the cache, then it expands the tree by adding node $\vec{x}(3)$ as another child of node $\vec{x}(2)$ and add $G^*(\vec{x}(3), 2)$ == $G^*(\vec{x}(2), 2)$ to its cache. Similarly, since x_4 follows x_3 without an idle period, it adds $\vec{x}(4)$ to the tree. This time, however, it first goes back up to node $\vec{x}(3)$ and apply Lemma 3 to calculate $G^*(\vec{x}(3), 3)$ as a weighted sum of $G^*(\vec{x}(2), 3)$ and $G^*(\vec{x}(2), 2)$ — the entries in the next-higher row above and/or to the left of $G^*(\vec{x}(3), 3)$ in the Figure, before adding $G^*(\vec{x}(4), 3) = G^*(\vec{x}(3), 3)$ into the cache for node $\vec{x}(4)$.

In the worst case, to calculate $G^*(\vec{x}(j), j-1)$ we must add one new entry to the cache for every node along the path $\vec{x}(j)$. These new entries form the jth column in Figure 1. Since each entry is a weighted sum of the all entries above and/or to its left in the previous row, cost per entry grows linearly as we move up the column, giving a total computational complexity of $O(j^2)$ for initializing a node at depth j in the decision tree. Therefore, the worst-case computational complexity for our algorithm to process an entire n-packet busy period (i.e., starting from an empty cache) is $O(1^2) + \ldots O(n^2) \equiv O(n^3)$.

However, this caching structure lets us avoid this worst-case computation in almost all cases. Clearly, if (any prefix of) the same packet-arrival sequence $\vec{x}(n)$ appears later in the trace, the algorithm requires $O(1)$ time per packet to "step" along the path through the decision tree to the last node and return the cached result. Moreover, suppose the cache contains the data for some previous packet-arrival sequence $\vec{x}'(m)$ for which $x_j = x'_j$ for all $j \leq p$ but $x_{p+1} \neq x'_{p+1}$. In this case, we can reuse the first p rows of the cache (up to the branch node between the two paths), so that only the last $n-p$ columns need to be evaluated. Furthermore, the first p rows have already been extended to m columns, so only the last $max\{n-m, 0\}$ columns of these existing rows must be evaluated.

IV. Experimental Results

A. Model Validation

To demonstrate the accuracy of our QI algorithm, we carried out the following experiment. First, we chose a simple test system for which our analytical formulas should give exact results, namely a single-server queueing system with Poisson arrivals, where the arrival-rate does not change during a busy period. Note that our QI algorithm makes *no assumptions* about the service-time sequence $\vec{x}(n)$ over the busy period: every sequence is equally acceptable, and generates its own deterministic result. Therefore, we restricted the experiment to use the same packet-length sequence for every busy period: two "large packets" (i.e., $l_1 = l_2 = 1514$ bytes) followed by an arbitrary number of "small packets" (i.e., $l_3 = l_4 = \ldots 60$ bytes). When this "front heavy" packet-length sequence is combined with a moderate arrival rate, few busy periods will end before the 3rd packet or continue beyond the 20th packet.

We then used CSIM-19 [13] to simulate approximately 0.5 million independent busy periods for the test system, as shown in Table I. The results were grouped according to busy-period length and then each group was split to form five independent "batches" of approximately equal size. Finally, we calculated a point estimate and 95% confidence interval for the mean packet delay at each busy period length. In all cases, the analytical result produced by our QI algorithm was well within the 95% confidence interval from the simulation, and

Busy Period Info		Mean Delay per Batch					Grand	95% Conf.	QI
length	count	batch 1	batch 2	batch 3	batch 4	batch 5	mean	Interval	Method
3	11232	145.28	147.30	145.35	146.55	146.55	146.206	145.24-147.17	145.33
4	25082	134.29	134.83	133.95	134.97	134.21	134.45	133.96-134.93	134.52
5	39347	130.05	130.39	129.96	130.28	129.93	130.12	129.90-130.35	130.01
6	50052	127.67	127.89	128.10	128.27	127.97	127.98	127.73-128.23	128.01
7	54533	127.11	127.07	126.99	126.78	126.07	126.80	126.33-127.28	126.95
8	52891	126.48	126.24	126.30	126.21	126.11	126.27	126.12-126.42	126.40
9	46415	126.58	126.26	126.32	126.28	126.28	126.34	126.20-126.49	126.11
10	37877	125.69	126.05	126.66	126.14	125.32	125.97	125.41-126.53	125.97
11	28366	125.92	125.50	125.59	125.66	125.70	125.67	125.50-125.85	125.90
12	20311	126.12	125.91	125.37	126.21	125.63	125.85	125.46-126.24	125.89
13	13571	125.53	126.24	125.90	125.91	125.43	125.80	125.44-126.16	125.91
14	9120	125.71	126.28	126.25	125.39	125.95	125.92	125.50-126.33	125.94
15	15010	126.16	126.20	126.51	125.99	125.83	126.14	125.85-126.42	125.99
16	11581	125.96	125.59	126.43	126.75	126.56	126.26	125.73-126.78	126.04
17	10024	125.91	127.10	126.18	125.27	126.45	126.18	125.43-126.93	126.10
18	14951	125.68	126.50	127.19	126.58	126.31	126.45	125.85-127.06	126.17
19	10933	126.42	126.64	125.71	126.52	125.37	126.13	125.51-126.75	126.23
20	10774	126.98	126.71	126.42	125.92	126.70	126.55	126.10-126.99	126.30

TABLE I
VALIDATION OF OUR ALGORITHM AGAINST THE MEASURED RESULTS FROM SIMULATING THE SAME SYSTEM.

indeed the widths of the confidence intervals themselves were generally quite small (about 1% of the mean). We also tried repeating the experiment after changing the Poisson arrival rate and/or the link speed. As expected, these changes had no effect on the mean delay at each busy period length, although they did change the distribution of busy period lengths.

B. Busy Periods in a Network Trace

Previous authors of QI algorithms did not consider caching of results because they assumed that service times are drawn from some arbitrary continuous distribution. In the case of network traffic, however, service times at the router/ switch port are a deterministic function of the respective packet length, and a small number of specific lengths — such as a maximum size Ethernet packet, minimum size TCP/IP segment (i.e., an ACK), etc. — cover a significant share of the traffic. Thus, we have selected two traces from Wide project backbone traffic data repository to study the busy periods at a core router output port. Trace 1 [14] has 2138973 packets and 518190 busy periods collected over 844.17 seconds, giving an average rate of 14.24 Mbps. Trace 2 [15] has 1998060 packets and 323829 busy periods collected over 725.82 seconds, giving an average rate of 12.73 Mbps.

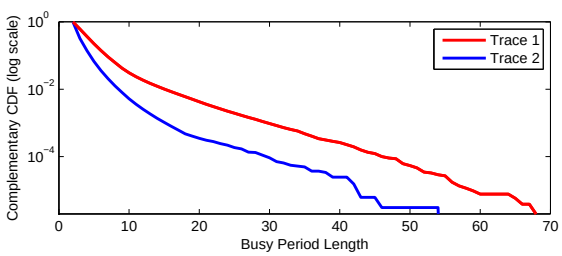

Fig. 2. Complementary CDF for busy period lengths.

Figure 2 shows the complementary cumulative distribution function of the number of packets served in a busy period. We observe that busy periods serving large numbers of packets are very rare: approximately 90% have fewer than 10 packets, and 99% have fewer than 20 packets. Furthermore, the tail of the distribution exhibits an exponential decrease, which is qualitatively similar to the known formula for the number of customers served in an M/M/1 busy period [16].

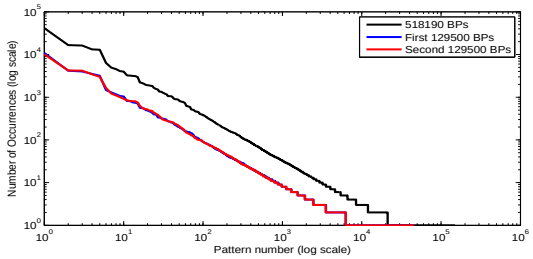

Fig. 3. Number of occurrences of busy period patterns (log-log graph).

We found that about 2/3 of the busy periods duplicate the exact packet sequence from a previous busy period. Thus, we counted the total number of occurrences for each unique sequence and then ordered the sequences from most-frequent to least-frequent. The black (upper) curve in figure 3 plots the number of occurrences as a function of rank for all the unique packet sequences found in trace 1 on a log-log scale. (Trace 2 gives similar results, which are omitted to save space.) We also partitioned the trace into four equal quartiles and repeated the process on each quartile. The two (lower) colored curves in figure 3 cover the first and second quartiles. It is interesting to note the linearity of curves in figure 3, indicating a power-law relationship between frequency and rank. Furthermore, the similarity among curves obtained from the complete trace and

individual quartiles shows that the distribution of busy period frequencies is stationary over time.

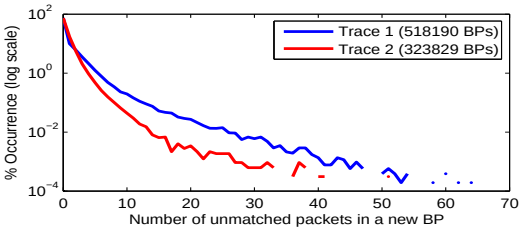

Fig. 4. Percentage of BP with $n-p$ unmatched packets in the cache.

Recall from section III-B that if the first p packets of an n-packet busy period were seen in an earlier busy period, then (at least) the first p columns of the cache must already be filled. Thus, in figure 4, we plot the distribution for the number of unmatched packets, $n-p$, over all busy periods in both trace 1 and trace 2. Notice that about 90% of the busy periods have 0 unmatched packets, so no new calculations are required. (The proportion of busy periods that are *completely matched in the cache* is significantly higher than the proportion (about 2/3) of *duplicate busy periods* described previously because every proper prefix of sequence $\vec{x}(n)$ is a complete match but not a duplicate sequence.) Furthermore, most of the remaining busy periods have significant partial matches in the decision tree. For example, the 99.99th percentile (i.e., 4 decades below the maximum value on the y-axis) is $n \approx 45$ for the busy period length (figure 2), but only $(n-p) \approx 25$ for the number of unmatched packets (figure 4). Therefore, a cache of reasonable size would provide a significant speed-up for our convolution algorithm when applied to typical network trace data.

V. Asymptotically Linear Computation per Packet

The yellow (upper) curve in figure 5 plots the growth in size of the decision tree as a function of the number of packets processed from the trace 1 on a log-log scale. (Trace 2 gave similar results, and are omitted to save space.) For an arbitrarily long trace, we see no indication that the size of the decision tree will converge to some limiting value, which is an obvious concern for any practical implementation of this algorithm. Moreover, since the incremental cost of adding one new node to the tree at depth d is $O(d^2)$, the average processing time per packet is likely to increase over time unless we can modify the algorithm to achieve $O(1)$ processing time per packet in the large trace-size limit.

A. Bounding the Total Size of the Decision Tree

We begin by restricting the decision tree to include a bounded number of "popular" nodes, i.e., those nodes for which the visit frequency is above a threshold $v << 1$. Although the cost of *adding* a node at depth d is $O(d^2)$, the cost of *visiting* an existing node is always $O(1)$ independent of d. Thus, the processing time per packet will converge to $O(1)$ after all the "popular" nodes are in the decision tree.

The visit frequency, c/T, for a node is its visit count c divided by the current trace length T. The frequency increases every time the algorithm visits this node, and decreases at every other "step". Thus, we define a new intermediate status *discovered* for leaf node $\vec{x}(j)$ in the decision tree, such that its parent is a *visited* node $\vec{x}(i)$ with a fully initialized cache, but node $\vec{x}(j)$ only has a visit count, c, and an uninitialized cache. Suppose the algorithm reaches this node for the cth time. If $c/T \geq v$, a parameter of the algorithm, then we initialize the cache and upgrade its status to *visited*. Otherwise, the node retains its *discovered* status and the algorithm "rests" here for the remainder of the busy period.

The other six curves in figure 5 plot the number of nodes (instantaneous: black, magenta, green and cumulative: blue, cyan, red) for which the minimum visit frequency, i.e., c/T, is at least 10^{-3}, 10^{-4} and 10^{-5} respectively. In each case, the curves run parallel to the upper curve until $T \approx 1/v$, which shows that at any time only about 10% of the nodes in the full decision tree have been visited more than once. Thereafter, the curves quickly level off to approach a constant asymptotic size that is significantly less than $1/v$. Therefore, we can use this method to identify those nodes at which we can "prune" the decision tree to a bounded size by truncating the "tails" of rarely-occurring packet-arrival sequences. It remains to find an $O(1)$ algorithm for bounding the expected waiting times for *all* packets served in one of those truncated busy periods.

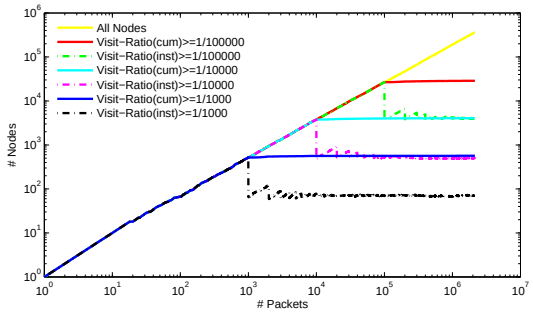

Fig. 5. Number of nodes in the decision tree vs packets seen for trace 1.

B. Linear Cost Bound for the Expected Arrival Sum

Suppose the algorithm enters the *discovered* node $\vec{x}(n)$ upon seeing the last packet of a busy period, and we wish to find simple upper and lower bounds for $\overline{A^*}_{n-1}(\vec{x}(n))$ while avoiding the high cost of initializing the cache at this node. Clearly, the *arrival time sum can only increase* if we force the last packet arrival to occur within the minimum possible range of $(X_{n-2}, X_{n-1}]$, and the summations in Eqs.(7, 10) collapse to a single term with $m_{n-1} = 1$, so that Eq. (9) reduces to:

$$\overline{A^*}_{n-1}(\vec{x}(n)) \leq \overline{A^*}_{n-2}(\vec{x}(n-1)) + X_{n-2} + \frac{x_{n-1}}{2} \quad (14)$$

Conversely, the *arrival time sum can only decrease* if we allow the last packet to arrive anywhere in the range $(0, X_{n-1}]$ without changing the busy-period forming requirement on the earlier packet arrivals imposed by Eq. (1). Thus:

$$\overline{A^*}_{n-1}(\vec{x}(n)) \geq \overline{A^*}_{n-2}(\vec{x}(n-1)) + \frac{X_{n-1}}{2} \quad (15)$$

Figure 6 compares the average delays computed for the unbounded tree and the bounded tree with visit frequency 10^{-5}. The average of the lower and upper bounds almost overlaps with the delay computed from unbounded tree.

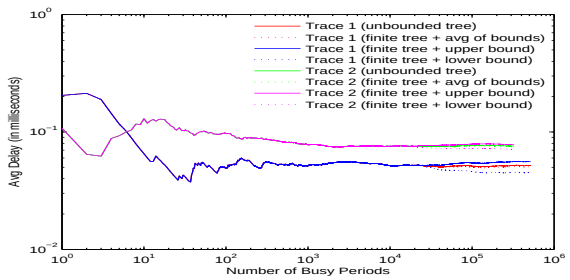

Fig. 6. Average packet delay over all the busy periods.

VI. Conclusions

We have developed a new incremental queue inferencing algorithm and associated decision-tree based caching scheme that provides an efficient method for collecting waiting time information from existing networking equipment during normal operation. Indeed, once the cache has been expanded to include all packet-length sequences whose frequency of occurrence within the trace is above a selectable minimum threshold (say, a probability of 10^{-6}) then the algorithm can continue running indefinitely, while processing the remainder of the trace in approximately linear time and producing tight bounds on waiting times.

Recently Vishwanath et al. [17] surveyed recent work related to the reduction output-port buffers in Internet core routers and its likely effects on TCP performance. Many of the results they described were based on simulations and other "offline" experiments, with few examples of actual measurement data. We believe that the queue inferencing algorithm developed in this paper will help to advance this discussion by making experimental data for real systems readily available.

So far, we have only tested our method on a few traces from the WIDE project, so more experimentation is clearly needed. However, our results so far are very encouraging. As expected, we found that the distribution of packet sizes was highly concentrated on a small number of packet values, resulting in 65% to 70% duplicate busy periods. We also found that most of the busy periods were short and their length distribution decreases in the form of a power law — which is again consistent with known asymptotic results for G/G/1 queues (see [18], section 2.4).

One reason for the relative simplicity of our queue inferencing algorithm is that we avoid calculating the distributions for number in system at departure points and proceed directly to the sum of expected arrival times. Those who are familiar with the analysis of closed product-form queueing networks may recall the dramatic speedup from using Mean Value Analysis (MVA) in place of early solution algorithms based on the calculation of a normalization constant by summing a simple product-form expression over an irregular state space. Although the queue inferencing problem seems to have a similar structure — namely the sum/integral of a simple expression over an irregular state space — the performance increase from our average-based method is far more modest. We attribute the relatively small performance benefit for our method to the lack of any analog to the "arrival instant theorem" for queue inferencing.

Acknowledgments

This work was supported in part by the Regents of the University of California through a Graduate Fellowship for Mr. Habib. The authors would also like to thank the Measurement and Analysis on the WIDE Internet (MAWI) Working Group Traffic Archive for publishing the network trace files.

References

[1] R. Larson, "The queue inference engine: Deducing queue statistics from transactional data," *Management Science*, vol. 36, no. 5, pp. 586–601, May 1990.

[2] C. Villamizar and C. Song, "High performance TCP in ANSNET," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[3] R. Bush and D. Meyer, "Some internet architectural guidelines and philosophy," RFC 3439, Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3439.txt

[4] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *SIGCOMM '04*. New York, NY, USA: ACM Press, 2004, pp. 281–292.

[5] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, and G. Salmon, "Experimental study of router buffer sizing," in *IMC '08*. Vouliagmeni, Greece: ACM Press, 2008.

[6] D. Bertsimas and L. D. Servi, "Deducing queueing from transactional data: The queue inference engine revisited," *Operations Research*, vol. 40, no. 2, pp. 217–228, May-June 1992.

[7] R. Larson, "The queue inference engine: Addendum," *Management Science*, vol. 37, no. 8, p. 1062, Aug. 1991.

[8] D. Manjunath and M. L. Molle, "Passive estimation algorithms for queueing delays in lans and other polling systems," in *IEEE INFOCOM '96*, 1996, pp. 240–247.

[9] D. A. Daley and L. D. Servi, "Exploiting markov chains to infer queue length from transactional data," *Journal of Applied Probability*, vol. 29, pp. 713–732, 1992.

[10] N. Hohn, K. Papagiannaki, and D. Veitech, "Capturing router congestion and delay," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 3, pp. 789–802, June 2009.

[11] R. Gawlick, "Estimating disperse network queues: The queue inference engine," *ACM Computer Communication Review*, vol. 20, no. 5, pp. 111–118, Oct. 1990.

[12] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary poisson view of internet traffic," in *INFOCOM '04*, Mar. 2004, pp. 1558–1569.

[13] H. Schwetman, "CSIM19: A powerful tool for building system models," in *WSC '01: Proceedings of the 33nd conference on Winter simulation*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 250–255.

[14] "Wide project, mawi working group, network trace, sample point b, june 30, 2006," http://tracer.csl.sony.co.jp/mawi/samplepoint-B/2006/200606301815.html.

[15] "Wide project, mawi working group network trace, sample point b, dec 28, 2000," http://tracer.csl.sony.co.jp/mawi/samplepoint-B/2000/200012281400.html.

[16] B. Bunday and R. Scraton, "The number of customers served during a busy period for an m/m/1 queue: an elementary treatment," *Int. J. Math. Educ. Sci. Technol.*, vol. 11, no. 1, pp. 25–27, 1980.

[17] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on router buffer sizing: recent results and open problems," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 34–39, 2009.

[18] L. Kleinrock, *Queueing Systems, Volume 2: Computer Applications*. Wiley, 1976.