Computation of the Packet Delay in Massey's Standard and Modified Tree Conflict Resolution Algorithms with Gated Access

Mart L. Molle and Alvin C. Shih

Technical Report CSRI-264
February 1992

Computer Systems Research Institute
University of Toronto
Toronto, Canada
M5S 1A1

Computation of the Packet Delay in Massey's Standard and Modified Tree Conflict Resolution Algorithms with Gated Access

Mart L. Molle and Alvin C. Shih

Computer Systems Research Institute
University of Toronto
Toronto, Ontario M5S 1A1

Recently, Polyzos derived methods for calculating conflict resolution interval lengths when the number of contenders in the initial set is Poisson distributed with a given mean. From these formulae, Polyzos computed the mean and standard deviation of the packet delay under window access. In this report, we show how to extend his methods to handle gated access.

1. Overview

The goal in this study is to find the mean and standard deviation of the packet delay in Massey's Standard and Modified Tree Algorithms [2], which we denote by the STA and the MTA respectively. These algorithms use gated access to select packets for transmission in a given conflict resolution interval (CRI), and thereafter use a recursive binary preorder tree traversal algorithm (optionally including the level-skipping modification) to resolve the resulting conflicts, if any.

The approach is to examine the delay from the point of view of a randomly chosen "tagged" packet. Under gated access, the delay for the "tagged" packet can be partitioned into the following two components:

t_0 the residual life of the CRI that is ongoing at the moment of arrival. This covers the time until the "gate" opens, admitting the "tagged" packet (and possibly some others) into the conflict resolution process.

t_2 the age of the next CRI at the moment where the tagged packet leaves the system at the end of its successful transmission. This covers the rest of the time from the first transmission of the "tagged" packet until it is successful.

The notation t_0 and t_2 to describe these two delay components is chosen for compatibility with Polyzos[1]. However, our work differs substantially from that of Polyzos, because t_0 and t_2 depend on the solution to an embedded Markov chain, describing the sequence of CRI lengths in the execution of the protocol. This is quite different from Polyzos' study, where t_0 and t_2 are i.i.d. and the most important part of the delay is the lag of the window, t_1, which does not appear in gated access.

Our approach is as follows: first, we solve the Markov chain to obtain the distribution of CRI lengths in steady state, represented by:

$$\vec{\pi} = (\pi_1, \pi_2, \pi_3, \cdots)$$

where π_n is the probability that a randomly chosen CRI will be n slots long. Next, we use the fact that we have Poisson packet arrivals to apply standard renewal-theoretic results to obtain $\tilde{\pi}_n$, the probability that our "tagged" arrival joins the system during a CRI of length n:

$$\tilde{\pi}_n = \frac{n\pi_n}{\sum_k k\pi_k} = \frac{n\pi_n}{\bar{N}} \qquad (1)$$

Intuitively, we must weight CRI's of length n in proportion to the product of their relative length and relative frequency of occurrence, since Poisson arrivals take a random look at the time axis. So, $\tilde{\pi}_n$ is the probability that a random packet arrives during a CRI of length n.

If we condition on the event that our "tagged" packet arrives during a CRI of length n, then t_0 has a uniform distribution over the interval $[0, n)$, and the distribution of t_2 should be conditioned on the fact that the "tagged" packet will be competing with a group of other packets having a Poisson distribution with mean $\lambda \cdot n$. Thus, assuming we use random addressing (i.e., splitting via random "coin tosses" rather than arrival times or station addresses), these two conditional distributions are independent of one another and the distribution of their sum factors into the product of their marginal distributions. Therefore, the Laplace transform for the packet delay can be written immediately in the form:

$$D^*(s) = \sum_{n=0}^{\infty} \tilde{\pi}_n \cdot U^*(n,s) \cdot G(\lambda\, n, e^{-s}) \tag{2}$$

where

$$U^*(n,s) = \frac{1 - e^{-sn}}{ns}$$

is the Laplace transform of the uniform distribution over $[0,n]$, with mean $\bar{u}_n = n/2$, and variance $\sigma_{u\,|n} = n^2/12$. Also, $G(x,z)$ is Polyzos' expression for the z-transform of the conditional distribution of t_2, given that the "tagged" packet is competing with Poisson traffic with mean x, as described below. In particular, the distribution of t_2 has mean:

$$T_{2|n} = T_2(\lambda\, n) = \sum_{i=0}^{\infty} \theta_i \cdot (\lambda\, n)^i$$

and variance:

$$\sigma^2_{T_{2|n}} = \sum_{i=0}^{\infty} \phi_i \cdot (\lambda\, n)^i$$

Using standard techniques, we can obtain all the moments of the packet delay from Eq. (2). In particular, the mean packet delay is:

$$\bar{T} = \sum_{n=0}^{\infty} \tilde{\pi}_n \cdot [\bar{u}_n + T_{2|n}] \tag{3}$$

and the variance is:

$$\sigma^2_T = \sum_{n=0}^{\infty} \tilde{\pi}_n \left(\sigma^2_{u\,|n} + \sigma^2_{T_{2|n}} \right) + \sum_{n=1}^{\infty} \tilde{\pi}_n \sum_{m=0}^{n-1} \left(\bar{u}_n + T_{2|n} - \bar{u}_m - T_{2|m} \right)^2 \tilde{\pi}_m \tag{4}$$

It remains to find expressions for the distribution $\vec{\pi}$ and the delay components listed above.

2. Steady-State Distribution of CRI Lengths

Let

$$\vec{\pi} = (\pi_1,\ \pi_2,\ \pi_3,\ \cdots)$$

be the steady state distribution of CRI lengths. The vector, $\vec{\pi}$, has, properly speaking, infinitely many components. Each component represents the probability of finding the system in a state where a CRI of some particular length is ongoing. However, for the sake of computability, we truncate $\vec{\pi}$ at the mth component. This will turn out to be reasonable for stable systems.

Given this approximation, to find $\vec{\pi}$, we must solve the following matrix equation:

$$\vec{\pi} = \vec{\pi}\underline{P} \quad \text{where} \quad \sum_{i=1}^{m} \pi_i = 1 \quad \text{and} \quad \underline{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,m} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,m} \\ \vdots & \vdots & & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,m} \end{bmatrix}$$

To populate the matrix, pick any feasible throughput, λ. This value should be between 0 and about 0.34 for STA, and between 0 and about 0.37 for the MTA. (The curves start "flat" and then rise rapidly as they hit poles at 0.347 and 0.375, respectively, so a smooth curve requires more points at large λ than small λ.)

The entries $p_{i,j}(\lambda)$ correspond to the probability mass values in the distribution of CRI lengths. Using Polyzos' notation, we have $p_{i,j}(\lambda) = q_j(i \cdot \lambda)$, where $q_j(x)$ is the probability that a CRI initiated by a

Poisson set of contenders with mean x lasts for exactly j slots. Expressions for $q_j(x)$ for the STA and MTA have been found by Polyzos. In particular, for the STA we have [1, p.85]:

$$q_{2n}(x) = 0, \quad n = 0,\ 1,\ ,\ldots\ , \quad q_1(x) = (1{+}x)\, e^{-x}, \quad q_3(x) = \frac{x^2}{4} e^{-x},$$

$$q_{2n+1}(x) = \sum_{m=1}^{2n-1} q_m(x/2)\, q_{2n-m}(x/2), \ \ n = 2,\ 3, \ldots \tag{5}$$

The corresponding results for the MTA are[1] [1, p.88]:

$$q_0(x) = 0, \quad q_1(x) = (1{+}x)e^{-x}, \quad q_2(x) = 0, \quad q_3(x) = \pi\,\bar{\pi}\,x^2\, e^{-x}, \quad q_4(x) = \pi\,\bar{\pi}^3\, x^2\, e^{-x},$$

$$q_{n+1}(x) = \sum_{m=1}^{n-1} q_{n-m}(\pi\, x)\; q_m(\bar{\pi}\, x) + \left[q_n(\bar{\pi}\, x) - q_{n-1}(\bar{\pi}\, x)\right] e^{-\pi x}, \quad n = 4,\ 5,\ \cdots \tag{6}$$

In both cases, q_j (x) can be solved recursively, since the recurrence requires only lower order terms. (See section 4.1 for a computational method with running time of $O(m^3 \cdot \log m)$.)

Since $\vec{\pi} = \vec{\pi}\underline{P}$, it should be clear that $\vec{\pi}$ is a left eigenvector for $\underline{P}$ with eigenvalue 1. Thus, to solve for $\vec{\pi}$, we can solve the following equivalent system in standard form:

$$[\underline{P} - \underline{I}\,]^T\, \vec{\pi}^{\mathrm{T}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $\underline{I}$ is the $m{\times}m$ identity matrix. Since the solution, $\vec{\pi}$, is still subject to the boundary condition $\sum_i \pi_i = 1$, we replace the bottom rows of $[\underline{P} - \underline{I}\,]^T$ and $[0\ \cdots\ 0\,]^{\mathrm{T}}$ with 1's, which yields:

$$\begin{bmatrix} p_{1,1}-1 & p_{2,1} & \cdots & p_{m,1} \\ p_{1,2} & p_{2,2}-1 & \cdots & p_{m,2} \\ \vdots & \vdots & & \vdots \\ p_{1,m-1} & p_{2,m-1} & \cdots & p_{m,m-1} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_{m-1} \\ \pi_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{7}$$

Some care must be exercised in the interpretation of the solution to Eq. (7) because this problem formulation causes the weight of the truncated components of $\vec{\pi}$ to accumulate in the last component. As a result, the computation of the mean tends toward the "conservative" side. To avoid this, when the computation of an m-dimensional $\vec{\pi}$ is desired, a $(m{+}1)$-dimensional $\vec{\pi}$ is computed and the last component is discarded, and the remaining terms are rescaled. (See sections 4.2 and 4.3 for procedures for estimating the truncation delay in the calculation of the mean and standard deviation of the system time.)

Equation (7) can be solved using any one of a variety of techniques for solving systems of linear equations. In our case, we found $\vec{\pi}$ by performing LU factorization of the $[\underline{P} - \underline{I}\,]^T$ matrix through Gaussian elimination with row pivoting. The program solves for $\vec{\pi}$ and then tries some iterative refinement, using a published algorithm by Forsythe and Moler [3], which was translated into C by the authors. Empirically, it can be seen from Figure 1 that, for large n, $\tilde{\pi}_n$ is a geometrically decreasing function of n in both the STA and MTA.

[1] The expression for $q_{n+1}(x)$ corrects a typographical error in [1], where the last factor was incorrectly written as e^{-x} instead of $e^{-\pi x}$.

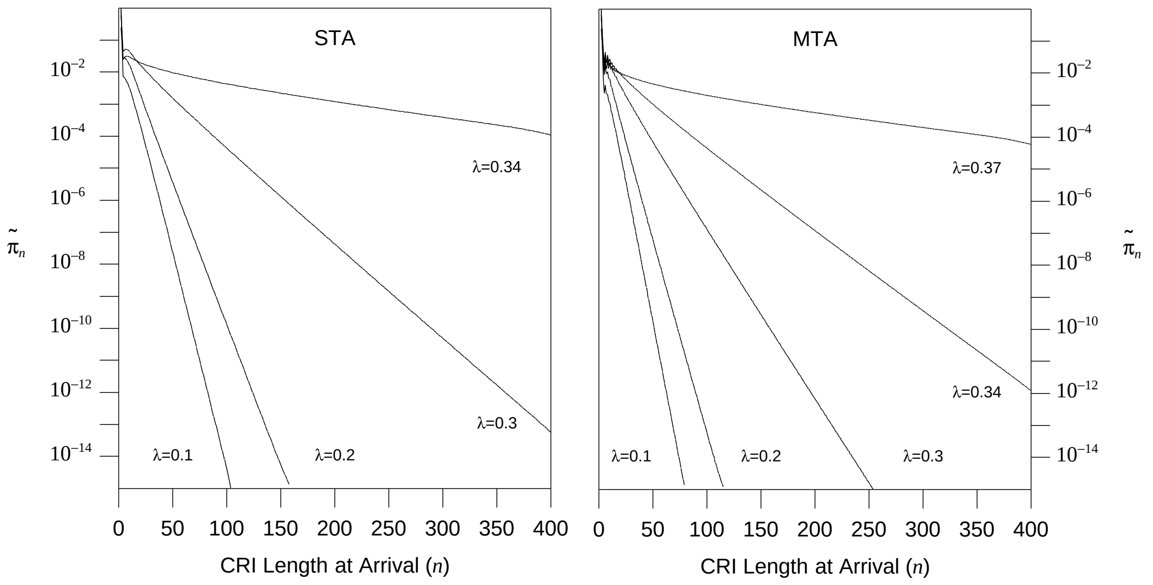

Figure 1: Geometric Progression for $\tilde{\pi}(n)$ when $n \gg 1$.

3. Expressions for the Delay Components

Once the steady-state solution $\vec{\pi}$ is found, it is clear from Eqs. (3-4) that the remaining step is to compute the conditional mean and variance of the distribution of t_2 for each state n, namely $T_{2|n}$ and $\sigma^2_{T_{2|n}}$ respectively.

3.1. Mean CRI Age at the Departure of the "Tagged" packet

Recall that t_2 is the age of the ongoing CRI at the point where the "tagged" packet departs (following its successful transmission). Its conditional mean, given a throughput of λ and the event that the "tagged" packet arrived during a CRI of length n, is given by:

$$T_{2|n} = T_2(\lambda \cdot n)$$

where $T_2(x)$ was found by Polyzos [1].

For the STA, Polyzos showed that $T_2(x)$ satisfies the following functional equation [1, p.99]:

$$T_2(x) = 1 + T_2(x/2) + \frac{1}{2} L(x/2) - \frac{3}{2} e^{-x}$$

Polyzos also showed how to solve this functional equation as a power series as:

$$T_2(x) = \sum_{n=0}^{\infty} \theta_n \, x^n$$

where [1, p.85]:

$$\theta_0 = 1 \text{ and } \theta_n = \frac{2^{-n}}{2(1-2^{-n})} \alpha_n - \frac{3}{2(1-2^{-n})} \frac{(-1)^n}{n\,!}, \;\; n=1,\, 2,\, \cdots$$

and [1, p.88]:

$$\alpha_0 = 1, \;\; \alpha_1 = 0, \text{ and } \alpha_n = (-1)^n \frac{2(n-1)}{n\,!(1-2^{1-n})}, \;\; n=2,\, 3,\, \cdots$$

For the MTA, Polyzos found the following functional equation for $T_2(x)$ [1, p.101]:

$$T_2(x) = 1 + \pi \, T_2(\pi \, x) + \bar{\pi} \, T_2(\bar{\pi} \, x) + \bar{\pi} \left[L(\bar{\pi} \, x) - e^{-\pi x} \right] - e^{-x}$$

and in power series form as:

$$T_2(x) = \sum_{n=0}^{\infty} \theta_n x^n$$

where

$$\theta_0 = 1 \text{ and } \theta_n = \frac{\bar{\pi}\,\pi^n\,\alpha_n - (-1)^n\,(1+\bar{\pi}\,\pi^n)\,/n!}{1-\pi^{n+1}-\bar{\pi}^{n+1}}, \quad n=1,\,2,\,\cdots$$

Expressions for α_n and β_n are not included in [1], so for completeness we rederive them below:

$$L(x) = 1 + 2L(x/2) - 2(1+x)e^{-x} + (1+x/2)e^{-x} - e^{-x/2}$$

$$\text{So, } \sum_{i=0}^{\infty}\alpha_i x^i = 1 + 2\sum_{i=0}^{\infty}\alpha_i (x/2)^i - (1+2x-x/2)\sum_{i=0}^{\infty}\frac{(-x)^i}{i!} - \sum_{i=0}^{\infty}\frac{(-x/2)^i}{i!}$$

$$\text{Giving: } \alpha_n = 2\alpha_n(1/2)^n - \frac{(-1)^n}{n!}\left(1+(1/2)^n\right) - (2-1/2)\frac{(-1)^{n-1}}{(n-1)!}$$
$$= \frac{-1}{1-(1/2)^{n-1}}\left[\frac{(-1)^n}{n!}\left(1 - 3/2\,n + (1/2)^n\right)\right]$$

$$H(x) = 1 + 2H(x/2) + 4L(x/2) + 2[L(x/2)]^2 - (3+11/2\,x)e^{-x} - (3+2L(x/2))e^{-x/2}$$

$$\therefore \sum_{i=0}^{\infty}\beta_i x^i = 1+2\sum_{i=0}^{\infty}\beta_i (x/2)^i + 4\sum_{i=0}^{\infty}\alpha_i (x/2)^i + \left(\sum_{i=0}^{\infty}\alpha_i (x/2)^i\right)\left(\sum_{j=0}^{\infty}\alpha_i (x/2)^i\right)$$
$$- (3 + 11/2\,x)\sum_{i=0}^{\infty}\frac{(-x)^i}{i!} - \left(3 + 2\sum_{i=0}^{\infty}\alpha_i (x/2)^i\right)\sum_{j=0}^{\infty}\frac{(-x/2)^j}{j!}$$

$$\beta_n = 2\beta_n(1/2)^n + 4\alpha_n(1/2)^n + 2\sum_{i=0}^{n}\alpha_i\alpha_{n-i}(1/2)^n - (3 - 11/2\,n)\frac{(-1)^n}{n!} - 3\frac{(-1/2)^n}{n!} - 2\sum_{i=0}^{n}\frac{\alpha_i}{2^i}\,\frac{(-1/2)^i}{(n-i)!}$$
$$= \frac{(1/2)^n}{1-(1/2)^{n-1}}\left[4\alpha n + 2\sum_{i=0}^{n}\alpha_i\alpha_{n-i} - 2\sum_{i=0}^{n}\alpha_i\frac{(-1)^{n-i}}{(n-i)!} - \frac{3(-1)^n}{n!} - 2^n(3-11/2\,n)\frac{(-1)^n}{n!}\right]$$
$$= \frac{1}{2^n-2}\left[4\alpha_n + 4\alpha_n + 2\sum_{i=i}^{n-2}\alpha_i\alpha_{n-i} - 2\alpha_n - 2\frac{(-1)^n}{n!} - 2\sum_{i=0}^{n-1}\alpha_i\frac{(-1)^{n-i}}{(n-i)!} - 3\frac{(-1)^n}{n!} - 2^n(3 - 11/2\,n)\frac{(-1)^n}{n!}\right]$$
$$= \frac{1}{2^n-2}\left[6\alpha n + 2\sum_{i=2}^{n-2}\alpha_i\left(\alpha_{n-i} - \frac{(-1)^{n-i}}{(n-i)!}\right) + 2\alpha_{n-1} - \frac{(-1)^n}{n!}\left(5 + (3 - 11/2\,n)\cdot 2^n\right)\right]$$

3.2. Conditional Variance of t_2 for the MTA.

The last component we need in order to compute the variance of the packet delay is the conditional variance of t_2. Starting from $G(x,z)$, Polyzos' expression for the z-transform of t_2 given Poisson traffic with mean x, an expression for $Var(t_2\,|\,x)$ can be obtained from its second derivative. We begin by finding $Var(t_2\,|\,x)$ for MTA:

$$G(x,z) = \pi z G(\pi x,z) + \bar{\pi} z G(\bar{\pi} x,z)\left[Q(\pi x,z) + (1-z)\,e^{-\pi x}\right] + (z-z^2)\,e^{-x}$$

$$\begin{aligned}
\therefore \quad G'(x,z) = {} & \pi\, G(\pi x,z) + \pi z\, G'(\pi x,z) \\
& + \bar{\pi}\, G(\bar{\pi} x,z) \left[Q(\pi x,z) + (1-z)\, e^{-\pi x}\right] + \bar{\pi} z\, G'(\bar{\pi} x,z) \left[Q(\pi x,z) + (1-z)\, e^{-\pi x}\right] \\
& + \bar{\pi} z\, G(\bar{\pi} x,z) \left[Q'(\pi x,z) - e^{-\pi x}\right] \\
& + (1-2z)\, e^{-x}
\end{aligned}$$

$$\begin{aligned}
\therefore \quad G''(x,z) = {} & \pi\, G'(\pi x,z) + \pi\, G'(\pi x,z) + \pi z\, G''(\pi x,z) \\
& + \bar{\pi} G'(\bar{\pi} x,z) \left[Q(\pi x,z) + (1-z)\, e^{-\pi x}\right] + \bar{\pi} G(\bar{\pi} x,z) \left[Q'(\pi x,z) - e^{-\pi x}\right] \\
& + \bar{\pi} G'(\bar{\pi} x,z) \left[Q(\pi x,z) + (1-z)\, e^{-\pi x}\right] + \bar{\pi} z G''(\bar{\pi} x,z) \left[Q(\pi x,z) + (1-z)\, e^{-\pi x}\right] + \bar{\pi} z G'(\bar{\pi} x,z) \left[Q'(\pi x,z) - e^{-\pi x}\right] \\
& + \bar{\pi} G(\bar{\pi} x,z) \left[Q'(\pi x,z) - e^{-\pi x}\right] + \bar{\pi} z G'(\bar{\pi} x,z) \left[Q'(\pi x,z) - e^{-\pi x}\right] + \bar{\pi} z G(\bar{\pi} x,z) \left[Q''(\pi x,z)\right] \\
& - 2\, e^{-x}
\end{aligned}$$

Let: $G''(x,z)\,|_{z=1} \equiv \overline{t_2^2} - \overline{t_2} = V(x) - T_2(x)$

$G'(x,z)\,|_{z=1} = T_2(x)$

$Q'(x,z)\,|_{z=1} = L(x)$

$Q''(x,z)\,|_{z=1} = H(x) - L(x)$

$$\begin{aligned}
\therefore \quad V(x) - T_2(x) = {} & \pi\, T_2(\pi x) + \pi\, T_2(\pi x) + \pi\left[V(\pi x) - T_2(\pi x)\right] \\
& + \bar{\pi}\, T_2(\bar{\pi} x) + \bar{\pi} \left[L(\pi x) - e^{-\pi x}\right] \\
& + \bar{\pi}\, T_2(\bar{\pi} x) + \bar{\pi} \left[V(\bar{\pi} x) - T_2(\bar{\pi} x)\right] + \bar{\pi}\, T_2(\bar{\pi} x)\left[L(\pi x) - e^{-\pi x}\right] \\
& + \bar{\pi} \left[L(\pi x) - e^{-\pi x}\right] + \bar{\pi}\, T_2(\bar{\pi} x)\left[L(\pi x) - e^{-\pi x}\right] + \bar{\pi} \left[H(\bar{\pi} x) - L(\pi x)\right] \\
& - 2\, e^{-x}
\end{aligned}$$

$$\begin{aligned}
= {} & T_2(\pi x)[\pi + \pi - \pi] + L(\pi x)[\bar{\pi} + \bar{\pi} - \bar{\pi}] + T_2(\bar{\pi} x)[\bar{\pi} + \bar{\pi} - \bar{\pi}] \\
& + V(\pi x)[\pi] + V(\bar{\pi} x)[\bar{\pi}] + H(\pi x)[\bar{\pi}] + L(\pi x)\; T_2(\bar{\pi} x)\; [\bar{\pi} + \bar{\pi}] \\
& - e^{-\pi x}\left[\bar{\pi} + \bar{\pi} T_2(\bar{\pi} x) + \bar{\pi} T_2(\bar{\pi} x) + \bar{\pi}\right] - 2e^{-x}
\end{aligned}$$

$$\begin{aligned}
\therefore \quad V(x) = {} & T_2(x) + \bar{\pi} L(\pi x) + \pi T_2(\pi x) + \bar{\pi} T_2(\bar{\pi} x)\left[1 - 2e^{-\pi x}\right] + \bar{\pi} H(\pi x) \\
& + 2\bar{\pi}\, L(\pi x)\; T_2(\bar{\pi} x) - 2e^{-x} - 2\bar{\pi} e^{-\pi x} + \pi\, V(\pi x) + \bar{\pi}\, V(\bar{\pi} x)
\end{aligned}$$

$$\text{But: } T_2(x) = 1 - e^{-x} + \pi T_2(\pi x) + \bar{\pi}\left[T_2(\bar{\pi}x) + L(\pi x) - e^{-\pi x}\right]$$

$$\therefore \quad V(x) = 1 - 3e^{-x} - 3\bar{\pi}e^{-\pi x} + 2\pi\, T_2(\pi x) + 2\bar{\pi}T_2(\bar{\pi}x)\left[1 - e^{-\pi x}\right]$$
$$+ 2\bar{\pi}\, L(\pi x) + \bar{\pi}\, H(\pi x) + 2\bar{\pi}\, L(\pi x)\; T_2(\bar{\pi}x) + \pi\, V(\pi x) + \bar{\pi}\, V(\bar{\pi}x)$$

Now, we assume that $\pi = \bar{\pi} \equiv ½$ (symmetric case):

$$V(x) = 1 - 3e^{-x} - \frac{3}{2}e^{-x/2} + T_2(x/2)\left[2 - e^{-x/2}\right] + L(x/2) + \frac{1}{2}H(x/2) + L(x/2)\cdot T_2(x/2) + V(x/2)$$

But: $Var(t_2\,|\,x) \equiv V(x) - [T_2(x)]^2$. Therefore, we have:

$$Var(t_2\,|\,x) = V(x) - \left[1 - e^{-x} + T_2(x/2) + \frac{1}{2}L(x/2) - \frac{1}{2}e^{-x/2}\right]^2$$
$$= V(x) - \left[\left(1 - e^{-x} - \frac{1}{2}e^{-x/2}\right)^2 + 2\left(1 - e^{-x} - \frac{1}{2}e^{-x/2}\right)\left[T_2(x/2) + \frac{1}{2}L(x/2)\right] + \left[T_2(x/2) + \frac{1}{2}L(x/2)\right]^2\right]$$
$$= V(x) - \left[1 - e^{-x} + \frac{1}{2}e^{-x/2} - e^{-x} + e^{-2x} + \frac{1}{2}e^{-3x/2} - \frac{1}{2}e^{-x/2} + \frac{1}{2}e^{-3x/2} + \frac{1}{4}e^{-x}\right.$$
$$+ 2\, T_2(x/2)\left[1 - e^{-x} - \frac{1}{2}e^{-x/2}\right] + L(x/2)\left[1 - e^{-x} - \frac{1}{2}e^{-x/2}\right]$$
$$\left. + [T_2(x/2)]^2 + T_2(x/2)\, L(x/2) + \frac{1}{4}[L(x/2)]^2\right]$$

Since:

$$V(x) = 1 - 3e^{-x} - \frac{3}{2}e^{-x/2} + T_2(x/2)\left[2 - e^{-x/2}\right] + L(x/2) + \frac{1}{2}H(x/2) + L(x/2)\cdot T_2(x/2) + V(x/2)$$

We get:

$$Var(t_2\,|\,x) = -e^{-x}\left[3 - 2 + \frac{1}{4}\right] - e^{-x/2}\left[\frac{3}{2} - 1\right] - e^{-2x} - e^{-3x/2} + T_2(x/2)\left[2 - 2 - e^{-x/2} + 2\,e^{-x} + e^{-x/2}\right]$$
$$+ L(x/2)\left[e^{-x} + \frac{1}{2}e^{-x/2}\right] + Var(t_2\,|\,x/2) + \frac{1}{2}H(x/2) - \frac{1}{4}[L(x/2)]^2$$

Therefore:

$$Var(t_2\,|\,x) = T_2(x/2)\cdot 2e^{-x} + L(x/2)\left[e^{-x} + \frac{1}{2}e^{-x/2}\right] + \frac{1}{2}H(x/2) - \frac{1}{4}[L(x/2)]^2 - \frac{5}{4}e^{-x} - \frac{1}{2}e^{-x/2}$$
$$- e^{-2x} - e^{-3x/2} + Var(t_2\,|\,x/2)$$

This recursive form is not terribly conducive to computation. Rather, we would prefer it in power series form. So, let

$$Var(t_2\,|\,x) = \sum_{i=0}^{\infty}\phi_i x^i$$

Also let:

$$L(x) = \sum_{i=0}^{\infty} \alpha_i x^i$$

$$T_2(x) = \sum_{i=0}^{\infty} \theta_i x^i$$

$$H(x) = \sum_{i=0}^{\infty} \beta_i x^i$$

$$\begin{aligned}\therefore \quad \sum_{i=0}^{\infty} \phi_i x^i = 2 &\left[\sum_{i=0}^{\infty} \theta_i (x/2)^i\right] \left[\sum_{j=0}^{\infty} \frac{(-x)^j}{j\,!}\right] + \left[\sum_{i=0}^{\infty} \alpha_i (x/2)^i\right] \left[\left[\sum_{j=0}^{\infty} \frac{(-x)^j}{j\,!}\right] + \frac{1}{2}\left[\sum_{j=0}^{\infty} \frac{(-x/2)^j}{j\,!}\right] - \frac{1}{4}\left[\sum_{j=0}^{\infty} \alpha_j (x/2)^j\right]\right] \\ &+ \frac{1}{2}\sum_{i=0}^{\infty} \beta_i (x/2)^i - \frac{5}{4}\left[\sum_{i=0}^{\infty} \frac{(-x)^i}{i\,!}\right] - \frac{1}{2}\left[\sum_{i=0}^{\infty} \frac{(-x/2)^i}{i\,!}\right] - \left[\sum_{i=0}^{\infty} \frac{(-2x)^i}{i\,!}\right] - \left[\sum_{i=0}^{\infty} \frac{(-3x/2)^i}{i\,!}\right] \\ &+ \sum_{i=0}^{\infty} \phi_i (x/2)^i\end{aligned}$$

Equating coefficients of x^i on both sides of this equation, we obtain, after some manipulation:

$$\phi_0 = 0$$

and

$$\begin{aligned}\phi_n = \frac{1}{1-2^{-(n+1)}} &\left[\frac{(-1)^n}{n\,!}\left(3 - \frac{9}{4}n\right) + \frac{3 - 2^{-(n-1)}}{2^{n-1}-1}\alpha_n - (1/2)^{n+1}\theta_n - \frac{3}{2^n}\left(\alpha_{n-1} + 2\theta_{n-1}\right)\right. \\ &\left. + \sum_{i=2}^{n-2}\left(\alpha_i + \theta_i\right)(1/2)^i \left(\frac{3}{2}\right)\frac{(-1)^{n-i}}{(n-i)!} + \sum_{i=2}^{n-2}\left(\alpha_i \alpha_{n-i}\left[\frac{3(1/2)^{n+1}-1}{2^n-2}\right] - \theta_i \theta_{n-i}(1/2)^n\right)\right]\end{aligned}$$

3.3. Conditional Variance of t_2 for the STA.

Through a similar derivation, we find that, for the STA:

$$Var(t_2\,|\,x) = L(x/2)\left(\frac{3}{2}\,e^{-x} - \frac{1}{4}\,L(x/2)\right) - T_2(x/2)\left(1 - \frac{3}{2}\,e^{-x} + \frac{1}{2}\,T_2(x/2)\right) + \frac{1}{2}H(x/2) + \frac{1}{2}Var(t_2\,|\,x/2)$$

Again, we want something in power-series form:

$$Var(t_2\,|\,x) = \sum_{i=0}^{\infty} \phi_i x^i$$

We have the following power series expansions for $L(x)$ and $T_2(x)$ [1, p.85] and $H(x)$ [1, p.99]:

$$L(x) = \sum_{i=0}^{\infty} \alpha_i x^i, \quad \text{where: } \alpha_0 = 1,\ \alpha_1 = 0,\ \alpha_n = \frac{(-1)^n\, 2\,(n-1)}{n\,!\,(1-2^{1-n})}$$

$$H(x) = \sum_{i=0}^{\infty} \beta_i x^i, \quad \text{where: } \beta_0 = 1,\ \beta_1 = 0,\ \beta_n = 4\,\alpha_n + \frac{2\,\alpha_n + \sum_{m=0}^{n} \alpha_m \alpha_{n-m}}{2^{n-1}-1}$$

$$T_2(x) = \sum_{i=0}^{\infty} \theta_i x^i, \quad \text{where: } \theta_0 = 1,\ \theta_1 = 3,\ \theta_n = \frac{2^{-n}}{2(1-2^{-n})}\alpha_n - \frac{3}{2(1-2^{-n})}\,\frac{(-1)^n}{n\,!}$$

By substituting, we get:

$$\sum_{i=0}^{\infty} \phi_i x^i = \sum_{i=0}^{\infty} \alpha_i (x/2)^i \left[\frac{3}{2} \sum_{j=0}^{\infty} \frac{(-x)^j}{j\,!} - \frac{1}{4} \sum_{j=0}^{\infty} \alpha_j (x/2)^j \right] + \sum_{i=0}^{\infty} \theta_i (x/2)^i \left[3 \sum_{j=0}^{\infty} \frac{(-x)^j}{j\,!} \right] + \frac{1}{2} \sum_{i=0}^{\infty} \beta_i (x/2)^i + \sum_{i=0}^{\infty} \phi_i (x/2)^i$$

Which can be simplified to obtain:

$$\phi_0 = 0$$

$$\phi_1 = 14$$

$$\phi_n = \frac{1}{1-2^{-n}} \left[-\frac{(-1)^n}{n\,!} \left(\frac{5}{2} + 9 \cdot 2^{n-2} \right) + \frac{\beta_n}{2^{n+1}} + \sum_{i=0}^{n} \left[\left(\alpha_i + 2\theta_i \right) \cdot 2^{-i} \cdot \frac{3}{2} \cdot \frac{(-1)^{n-i}}{(n-i)!} - \frac{\alpha_i\ \alpha_{n-i}}{2^{n+2}} \right] \right]$$

3.4. Second Moment of t_2.

Let $V(x) = E[t_2^2 \mid x]$ be the second moment of t_2. Since $Var(t_2 \mid x) \equiv V(x) - [T_2(x)]^2$, we could have solved for the power series representation of $V(x)$, instead of $Var(t_2 \mid x)$, as a computational method for finding $Var(t_2 \mid x)$. For example, to get $V(x)$ in the case of the STA, we have to go back to the MGF:

$$G(x,z) = \frac{z}{2} \left[Q(x/2,z) + 1 \right] G(x/2,z) + e^{-x} \left[z - \frac{z^2 + z^3}{2} \right]$$

Let

$$T_2(x) = G'(x,z)\,|_{z=1}$$

$$L(x) = Q'(x,z)\,|_{z=1}$$

Therefore:

$$G'(x,z) = lineup\ \frac{1}{2} \left[Q(x/2,z) + 1 \right] G(x/2,z) + \frac{z}{2} \left[Q'(x/2,z) \right] G(x/2,z) + \frac{z}{2} \left[Q(x/2,z) + 1 \right] G'(x/2,z) + e^{-x} \left[1 - \frac{2z + 3z^2}{2} \right]$$

$$\begin{aligned} G''(x,z)\,|_{z=1} &\equiv E[t_2(t_2 - 1)] \\ &= \frac{1}{2} \left[Q'(x/2,z) \right] G(x/2,z) + \frac{1}{2} \left[Q(x/2,z) + 1 \right] G'(x/2,z) \\ &\quad + \frac{1}{2} \left[Q'(x/2,z) \right] G(x/2,z) + \frac{z}{2} \left[Q''(x/2,z) \right] G(x/2,z) + \frac{z}{2} \left[Q'(x/2,z) \right] G'(x/2,z) \\ &\quad + \frac{1}{2} \left[Q(x/2,z) + 1 \right] G'(x/2,z) + \frac{z}{2} \left[Q'(x/2,z) \right] G'(x/2,z) + \frac{z}{2} \left[Q'(x/2,z) + 1 \right] G''(x/2,z) \\ &\quad - e^{-x} \left[\frac{2 + 6z}{2} \right] \end{aligned}$$

$$= [Q'(x/2,z)]G(x/2,z) + [Q(x/2,z)+1]G'(x/2,z)$$

$$+ \frac{z}{2}[Q''(x/2,z)]G(x/2,z) + z[Q'(x/2,z)]G'(x/2,z) + \frac{z}{2}[Q'(x/2,z)+1]G''(x/2,z)$$

$$- e^{-x}\left[\frac{2+6z}{2}\right]$$

Now since $Q''(x,z)|_{z=1} = H(x) - L(x)$, we get:

$$G''(x,z)|_{z=1} = V(x) - T_2(x)$$

$$= L(x/2) + 2\,T_2(x/2) + \frac{1}{2}\left[H(x/2) - L(x/2)\right] + L(x/2)\,T_2(x/2) + \left[V(x/2) - T_2(x/2)\right] - 4\,e^{-x}$$

$$= \frac{1}{2}\left[2\,V(x/2) + H(x/2) + L(x/2) + 2\,T_2(x/2) + 2\,L(x/2)\,T_2(x/2) - 8\,e^{-x}\right]$$

Therefore:

$$V(x) = T_2(x) + \frac{1}{2}\left[2\,V(x/2) + H(x/2) + L(x/2) + 2\,T_2(x/2) + 2\,L(x/2)\,T_2(x/2) - 8\,e^{-x}\right]$$

$$= 1 + L(x/2) + 2\,T_2(x/2) + L(x/2)\,T_2(x/2) - \frac{11}{2}e^{-x} + V(x/2) + \frac{1}{2}H(x/2)$$

From here, one could proceed, using similar methods as have already been demonstrated, to unwind the recursion for $V(x)$, giving $V(x)$ in terms of:

$$V(x) = \sum_{i=0}^{\infty}\gamma_i x^i$$

Where, in the case of the STA, we get:

$$\gamma_0 = 1$$

$$\gamma_1 = 20$$

$$\gamma_0 = \frac{1}{1-(1/2)^i}\left[(1/2)^i\left(\alpha_i + 2\,\theta_i + \beta_i/2 + \sum_{j=0}^{i}\alpha_j\theta_{i-j}\right) - \frac{11}{2}\,\frac{(-1)^i}{i\,!}\right]$$

and in the case of the MTA, we get:

$$\gamma_0 = 1$$

$$\gamma_1 = 13½$$

$$\gamma_n = \frac{(1/2)^n}{1-(1/2)^n}\left[2\,\alpha_n + \beta_n/2 + 2\,\theta_n - \frac{(-1)^n}{n\,!}\left[3\cdot 2^n + 5/2\right] + \sum_{i=1}^{n-1}\left[\alpha_i - \frac{(-1)^i}{i\,!}\right]\theta_{n-i}\right]$$

4. Computation Issues

4.1. Initialization of the $\underline{P}$ Matrix

Recall that the definitions of $q_{n+1}(x)$ for both the STA and MTA are expressed in terms of summations involving $q_1(x/2), \ldots, q_n(x/2)$. Thus, a naive implementation of this expression for $q_{n+1}(x)$ would lead to a doubly recursive function, and hence, running times exponential in the matrix size, m. Fortunately, a number of optimisations are available to speed up the calculations significantly.

First, we recognize that since $e^{-\pi x} \cdot e^{-\bar{\pi} x} \equiv e^{-x}$ for all π, $\bar{\pi} \equiv 1-\pi$, we can avoid repeated evaluation of the exponential factors by solving for $\hat{q}_n(x) \equiv q_n(x) \cdot e^x$, from which we get $p_{i,j}(\lambda) = \hat{q}_m(i \cdot \lambda) \cdot e^{-i \cdot \lambda}$. In this case, Eq. (5) for the STA reduces to:

$$\hat{q}_{2n}(x) = 0, \quad n = 0,\, 1,\, \cdots, \quad \hat{q}_1(x) = (1+x), \quad \hat{q}_3(x) = \frac{x^2}{4},$$
$$\hat{q}_{2n+1}(x) = \sum_{m=1}^{2n-1} \hat{q}_m(x/2)\, \hat{q}_{2n-m}(x/2), \;\; n = 2,\, 3, \ldots$$

Similarly, Eq. (6) for the symmetric MTA becomes:

$$\hat{q}_0(x) = 0, \quad \hat{q}_1(x) = (1+x), \quad \hat{q}_2(x) = 0, \quad \hat{q}_3(x) = 1/4\, x^2\,, \quad \hat{q}_4(x) = 1/16\, x^2\,,$$
$$\begin{aligned} \hat{q}_{n+1}(x) &= \sum_{m=1}^{n-1} \hat{q}_{n-m}(x/2)\, \hat{q}_m(x/2) + \left[\hat{q}_n(x/2) - \hat{q}_{n-1}(x/2) \right] \\ &= \hat{q}_n(x/2) + (1+x)\hat{q}_{n-1}(x/2) + \sum_{m=3}^{n-3} \hat{q}_{n-m}(x/2)\, \hat{q}_m(x/2)\,, \quad n = 4,\, 5,\, \cdots \end{aligned} \tag{8}$$

Next, we recognize that in the case of the STA, the dimensionality of the problem can be reduced by a factor of two since $\hat{q}_{2n}(x) \equiv 0$ for all $n \in \mathbf{N}$. By performing a change of variable, we can store only the components corresponding to recurrent states.

And finally, some additional savings can be gleaned by noticing the symmetry in the convolution, which allows us to write:

$$\hat{q}_{2n+1}(x) = \sum_{k=0}^{2n} \hat{q}_k(x/2)\, \hat{q}_{2n-k}(x/2) = 2 \cdot \sum_{k=0}^{n} \hat{q}_k(x/2)\, \hat{q}_{2n-k}(x/2), \;\; n = 2,\, 3,\, \cdots$$

Unfortunately, none of the above changes is enough to affect the exponential complexity of the matrix initialization step. Fortunately, however, we can make use of a dynamic programming strategy to obtain a polynomial time algorithm for matrix initialization. Since the expression for <u>each</u> $\hat{q}_j(x)$, $1 \le j \le 2n+1$ uses <u>all</u> the values of $\hat{q}_1(x/2), \ldots, \hat{q}_{j-1}(x/2)$, we can avoid repeated evaluations of the same terms by organizing the calculations to work row-by-row instead of element-by-element.[2] Thus, to find one row $\hat{q}_1(x), \ldots, \hat{q}_m(x)$ of the matrix, we first make one recursive call to find the row $\hat{q}_1(x/2), \ldots, \hat{q}_{m-1}(x/2)$ and then apply the summation in the definition m times to find the elements of the original row using $O(m^2)$ arithmetic operations. The row $\hat{q}_1(x/2), \ldots, \hat{q}_{m-1}(x/2)$, in turn, requires another recursive call to find the row $\hat{q}_1(x/4), \ldots, \hat{q}_{m-2}(x/4)$ and so on until we reach the base case after m levels of recursion. This dynamic programming approach reduces the time to initialize the matrix to a more manageable $O(m^4)$.

For $m \gg 1$, it is possible to limit the depth of the recursion to $O(\log m)$ based on the power series expansion of $\hat{q}_n(x)$. In particular, consider the case where $x \approx eps_{\text{mach}}$. Notice that $\hat{q}_1(x) = O(1)$, but $\hat{q}_n(x) = O(x^2)$ for all $n > 1$. (This result is easy to establish by induction, since it is obviously true for the first few terms, and it must also hold for the general case because each term in the summation contains at least

[2] In the case of the MTA, the procedure is not useful if biased splitting (i.e., $\pi \neq \bar{\pi} \neq ½$) is allowed. This is because each recursive call would require both $\hat{q}_1(\pi x), \ldots, \hat{q}_{m-1}(\pi x)$ and $\hat{q}_1(\bar{\pi} x), \ldots, \hat{q}_{m-1}(\bar{\pi} x)$ to be evaluated, leading, once more, to a combinatorial explosion. Thus, our computations for the MTA are only for *unbiased* splitting, allowing the use of the dynamic programming approach above.

one factor of the type $\hat{q}_n(x/2)$, for some $n>1$.)

In the case of the STA, it is clear that the coefficients of x^2 and x^3 in $\hat{q}_{2n+1}(x)$ are determined entirely by the term $\hat{q}_1(x/2) \cdot \hat{q}_{2n-1}(x/2)$ in Eq. (5). Thus, it is easy to establish that:

$$\hat{q}_{2n+1}(x) = \frac{x^2}{2^{n+1}} \left[1 + (1 - 1/2^{n-1}) \cdot x \right] + O(x^4) \qquad n=2, 3, \cdots$$

In the case of the MTA, we can use Eq. (8) to show that

$$\hat{q}_{n+1}(x) = \zeta_{n+1} \cdot x^2 + O(x^3)$$

where

$$\zeta_3 = 1/4 \,, \quad \zeta_4 = 1/16 \,, \quad \zeta_{n+1} = [\zeta_n + \zeta_{n-1}]/4 \,, \quad n=4, 5, \cdots$$

This change reduces the running time for the matrix initialization algorithm to $O(m^3 \cdot \log m)$. For large m, the difference between $O(m^4)$ and $O(m^3 \cdot \log m)$ can have a significant effect on execution times. For example, in the case of the MTA with $m=600$, this change reduced the execution time of the program from more than 1 million CPU seconds (approximately 2 weeks) to less than 12 thousand CPU seconds (approximately 3 hours) on a *Sun SPARCstation IPC* workstation.

4.2. Tail Estimation for the Mean

Recall that the formula for computing the mean system time for a packet involves the following infinite summation:

$$\overline{T} = \sum_{n=1}^{\infty} \tilde{\pi}_n \cdot \overline{T}_n$$

where $\overline{T}_n \equiv [\overline{u}_n + T_{2|n}]$ is the conditional mean system time, given that the "tagged" packet arrived during a CRI of length n. Obviously, in our numerical calculations, we must truncate the infinite state space at the mth term. Fortunately, the asymptotic behaviour of $\tilde{\pi}_n$ and of $\overline{T}_n$ are very easy to deal with. In particular, we have already demonstrated in Figure 1 that, for large n, $\tilde{\pi}_n$ is a geometrically decreasing function of n for both the STA and the MTA. Furthermore, it can be seen from Figure 2 that, for large n, $\overline{T}_n$ is a linearly increasing function of n. This combination of a geometric weighting of a linear sequence makes it possible to estimate the loss from truncating the series after the mth term in closed form.

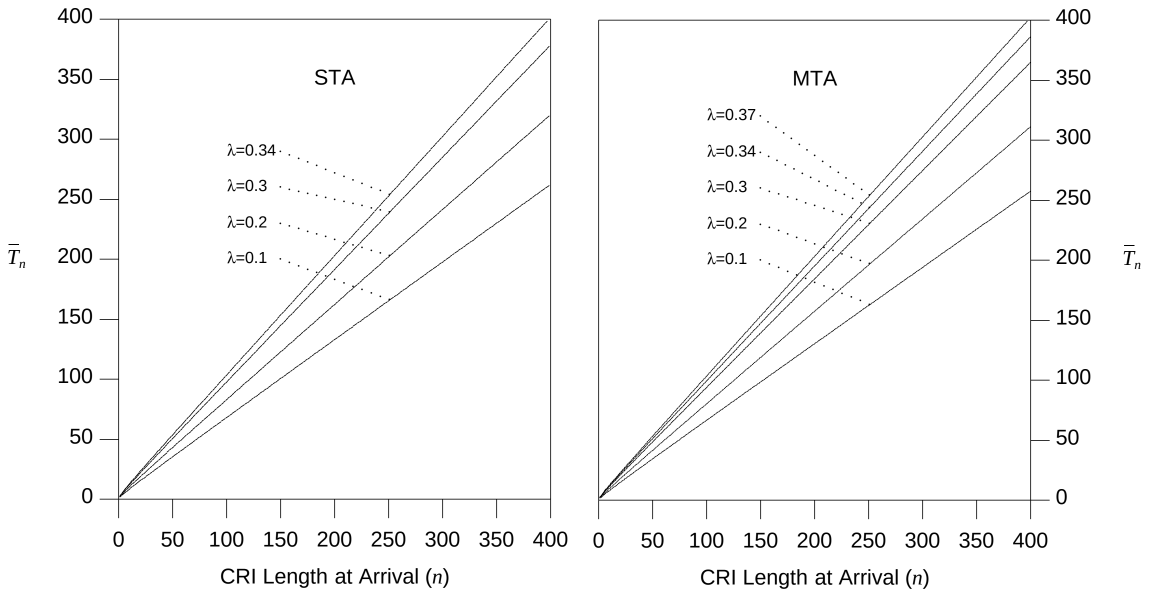

Figure 2: Linear Progression for $\overline{T}_n \equiv \overline{u}_n + T_{2|n}$ when $n \gg 1$.

The truncation error in this case is equal to the sum of the infinite tail of the series, starting from the $m+1$st term, namely:

$$R_1(m) = \sum_{i=m+1}^{\infty} \tilde{\pi}_i \, \bar{T}_i \tag{9}$$

where we know that $\tilde{\pi}_i$ is <u>decreasing</u> geometrically and $\bar{T}_i$ is <u>increasing</u> linearly. That is, $\tilde{\pi}_{i+1} \approx \alpha \, \tilde{\pi}_i$ and $\bar{T}_{i+1} \approx \bar{T}_i + \beta$.

Therefore, Eq. (9) is equivalent to:

$$R_1(m) \approx \tilde{\pi}_m \sum_{i=m+1}^{\infty} \alpha^{i-m} \bar{T}_i \approx \tilde{\pi}_m \left[\sum_{i=m+1}^{\infty} \alpha^{i-m} \left[\bar{T}_m + (i-m)\beta \right] \right] = \alpha \, \tilde{\pi}_m \left[\frac{\bar{T}_m}{1-\alpha} + \beta \sum_{j=1}^{\infty} j \, \alpha^j \right] \tag{10}$$

But $\sum_{j=1}^{\infty} j \, \alpha^j \equiv \dfrac{\alpha}{(1-\alpha)^2}$, so Eq. (10) reduces to:

$$R_1(m) \approx \frac{\alpha \, \tilde{\pi}_k}{1-\alpha} \left[\bar{T}_m + \frac{\alpha \, \beta}{1-\alpha} \right] \tag{11}$$

where $\tilde{\pi}_k$ is the last <u>weight</u> before truncation, T_m is the last <u>item</u> before truncation, α is the <u>ratio</u> between successive weights, and β is the <u>difference</u> between successive items. At present, these parameters are estimated by:

$$\alpha = \left(\frac{\tilde{\pi}_m}{\tilde{\pi}_{m-10}} \right)^{1/10}$$

and

$$\beta = \frac{1}{10} \left(\bar{T}_m - \bar{T}_{m-10} \right)$$

to smooth out any minor perturbations in the functions. This is rather inflexible, and should probably be a user-modifiable parameter.

Table 1 shows some sample data illustrating the use of the first order tail estimation procedure with the STA. For each truncation point, m, we show the first order tail estimate, $R_1(m)$, and the final value for the mean system time, $\bar{T}$, including the tail estimate. After numerous trial runs, the tail estimator seems reasonable. However, it is wise to take some runs with larger dimensionality to compare against runs which use tail estimation.

m	$\lambda=.25$		$\lambda=.3$	
	$R_1(m)$	$\bar{T}$	$R_1(m)$	$\bar{T}$
25	0.010841377	4.18212806	0.475691006	8.08852482
50	1.9223036e-05	4.18393434	0.026033635	8.23949245
75	2.8690251e-08	4.18393648	0.001218090	8.24584290
100	3.9893906e-11	4.18393648	5.345015e-05	8.24610187
200			1.523971e-10	8.24611247
300			1.228907e-13	8.24611247

Table 1: Use of the First Order Tail Estimator for the Mean System Time in the STA

4.3. Tail Estimation for the Variance

The variance calculation is another where a significant amount of truncation error can occur. Recall that by definition, the variance of the system time is given by:

$$\sigma_T^2 = \sum_{n=1}^{\infty} \tilde{\pi}_n \cdot \sigma_{T|n}^2$$

where

$$\sigma_{T|n}^2 \equiv \sigma_{u|n}^2 + \sigma_{T2|n}^2 + \sum_{k=1}^{n-1} \left(\bar{T}_n - \bar{T}_k\right)^2 \cdot \tilde{\pi}_k$$

is the sum of the conditional variance, given that the "tagged" packet arrived during a CRI of length *n*, and the covariance between CRIs of length *n* and *k*, for all $k < n$.

As with the mean system time, we need a formula for estimating:

$$R_2(m) = \sum_{i=m+1}^{\infty} \tilde{\pi}_i \cdot \sigma_{T|i}^2 \tag{12}$$

which is the truncation error in the variance calculation that results from stopping the summation at the *m*th term. Unfortunately, the tail estimation procedure for the mean, derived in the previous section, cannot be applied to the variance calculation, because $\sigma_{T|n}$ is *not* a linear function of *n* when $n \gg 1$. However, as shown in Figure 3, its square root, $\sigma_{T|n}$, is a linearly increasing function of *n*, and thus $\sigma_{T|n}^2$ is a <u>quadratic</u> function of *n* when $n \gg 1$.

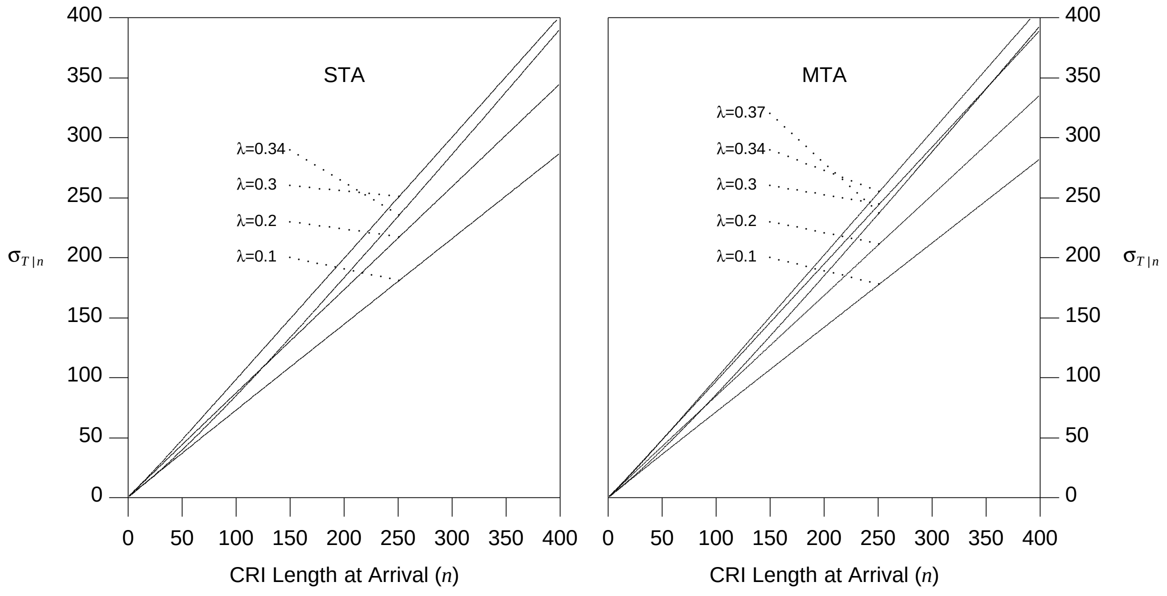

Figure 3: Linear Progression for $\sigma_{T|n}$ when $n \gg 1$.

In general, Figure 3 shows that for large *n*, $\sigma_{T|n}$ is a linear function of *n*, and its slope is an increasing function of λ. (The fact that the truncation error depends on $\sigma_{T|n}^2$, and $\sigma_{T|n}$ is *already* as large as $\bar{T}_n$, indicates that a proper tail estimation procedure is even more important in the variance calculation than it was for the mean.) For small λ, the function is remarkably close to a straight line passing through the origin. However, for large λ, the first part of the curve is a convex ∪ function and the tangent to the tail of the curve does not pass through the origin. Thus, we will use the approximation:

$$\sigma_{T|n}^2 \approx \beta \cdot (n - \delta)^2 \qquad n \gg 1$$

in the tail estimate. Its two parameters can be estimated from a pair of actual values, say $\sigma_{T|i}$ and $\sigma_{T|j}$, taken near the truncation point. We obtain:

$$\sqrt{\beta} = \frac{\sigma_{T|i} - \sigma_{T|j}}{i-j}$$

$$\delta = \frac{1}{2}\left[i + j - \frac{\sigma_{T|i} + \sigma_{T|j}}{\sqrt{\beta}} \right]$$

As before, the program uses $i = m$ and $j = m - 10$, but this should probably be a user-modifiable parameter.

Using this approximation in Eq. (12), we obtain the following second order tail estimate:

$$R_2(m) \approx \tilde{\pi}_m \sum_{i=m+1}^{\infty} \alpha^{i-m} \cdot \sigma^2_{T|i} \approx \beta \cdot \tilde{\pi}_m \sum_{i=m+1}^{\infty} \alpha^{i-m}(i-\delta)^2 = \alpha^{\delta-m} \cdot \beta \cdot \tilde{\pi}_m \sum_{j=m+1-\delta} j^2 \cdot \alpha^j \qquad (13)$$

But

$$\sum_{j=k}^{\infty} j^2 \cdot \alpha^j = \frac{\alpha^k}{1-\alpha}\left[k^2 + \frac{\alpha}{(1-\alpha)^2}\left[2k + 1 + \frac{2\alpha}{1-\alpha} \right] \right],$$

so Eq. (13) reduces to:

$$R_2(m) \approx \frac{\alpha \beta \tilde{\pi}_m}{1-\alpha}\left[(m+1-\delta)^2 + \frac{\alpha}{(1-\alpha)^2}\left[2(m-\delta) + 3 + \frac{2\alpha}{1-\alpha} \right] \right]. \qquad (14)$$

Table 2 shows some sample data illustrating the use of the second order tail estimation procedure with the STA. For each truncation point, m, we show the second order tail estimate, $R_2(m)$, and the final value for the system time variance, σ^2_T, including the tail estimate. Again, it is clear that the tail estimation procedure increases the speed of convergence significantly.

m	$\lambda = .25$		$\lambda = .3$	
	$R_2(m)$	σ^2_T	$R_2(m)$	σ^2_T
25	0.991020898	32.3210385	90.29885423	196.862538
50	0.002802641	32.0281122	7.267351932	150.750397
75	5.6690777e-06	32.0275502	0.424622369	147.465194
100	9.9224455e-10	32.0275493	0.022029354	147.301395
200			9.8609963e-08	147.293710
300			2.1090759e-10	147.293710

Table 2: Use of the Second Order Tail Estimator for the System Time Variance in the STA

4.4. Underflow vs. Overflow

In many cases, there are terms in many of the expressions of the form: $1/2^n$ or $k^n/n!$. For not unreasonable n, it is possible to overflow the precision available under the IEEE-754 format supported by most workstations, if "brute force" computation of the numerators and denominators is used. This results in propagation of the meaningless quantity, "inf".

Rather, it is preferable to tend towards underflow. Zero is a meaningful approximation to small quantities. Infinity is **not** a meaningful approximation to large quantities.

So, $1/2^n$ is computed as $(1/2)^n$, and $k^n/n!$ is computed by: $\prod_{i=0}^{n}(k/i)$. Eventually, i will exceed k and cause the product to decrease.

Various comparisons against algebraic evaluation on Maple have shown this to cause no anomalous results.

4.5. Precomputation of alpha, beta, etc.

Due to the heavy use of α_i, β_i, θ_i, and ϕ_i, it is obviously worthwhile to precompute these and put them away in a table. These tables tend to store values up to $i = 20$ since the truncation error in this case, namely:

$$\sum_{i=0}^{\infty} \alpha_i x^i - \sum_{j=0}^{20} \alpha_i x^i = \sum_{i=21}^{\infty} \alpha_i x^i$$

is bounded above by $\varepsilon_{\mathrm{mach}}$ for all $|x| \le 1$.

4.6. Use of Recursion Down to Base Case of Power Series

Many of the series have terms of the form: $k_n \dfrac{(-x)^n}{n!}$. For large x and small n, these terms can get very large. Later terms of the series get smaller, but the nature of floating point arithmetic will cause substantial loss of precision when trying to add these terms. In addition, these terms alternate, introducing "catastrophic cancellation".

Fortunately, the recursive forms of the various formulae cause a reduction in the size of x. To be precise, each recursive call results in x being divided by 2. Thus, enough recursive calls will result in an x that is of manageable size. In particular, $|x| \le 1$ is quite comfortable.

So, computation of badly-behaved power series was performed by recursive calls to bring down the magnitude of x, and then using power series evaluation for the "base case".

4.7. Minor Tweaks

The basic rule of thumb for accumulating sums on a machine is, "add the smallest terms first". In a few cases, it was found that rearranging the way terms of certain formulae were summed did improve accuracy. Indeed, in an attempt to extract the most precision possible, quicksorting terms was tried. This turned out to be too time-consuming and did not yield significant gains in precision. In most cases, sufficient accuracy was preserved by the machine to obviate the need for such tweaking.

4.8. Development

Maple was an invaluable tool for prototyping. Although too slow for the purposes of final computation, it allowed the verification of the various required derivations. Its interpreted nature allowed us to make quick changes as well as allowing us to ignore the problems of machine precision and compilation. In addition, Maple's ability to evaluate to arbitrary precision allowed the incremental verification of the precision of the C code.

4.9. Evaluation of other possibilities

We were rather fortunate that utilizing the recursion to bring terms down to reasonable size managed to be quite numerically stable. However, the benefits of Maple being seen early on did make me investigate options that would deliver the arbitrary precision of Maple while being substantially faster.

With this in mind, I posted to sci.math.symbolic and was referred to the following:

- bc

  It never hurts to look at what's available free on the system. It supports fixed-point precision to 100 decimal places. Alas, it does not include matrix support.

- Form

  This package is heavily oriented towards mathematics through "symbol shuffling". Its programming language, somewhat reminiscent of assembly language crossed with Prolog (if you can believe that), allows the user to define operations on formulae. Indeed, differentiation is provided in this way. The feature in which I was interested is its arbitrary precision floating point capability.

The syntax was rather cryptic, and we never did discover how to create a matrix (though we're sure it's possible). This tool is probably quite powerful in the hands of an expert, but the only experts that seemed to be available were all from the university at which this package originates (Nikhef?) in the Netherlands.

For problems involving **serious** computation, this package may be worth learning. It attempts to make the best use of the available computing resources.

It is available via anonymous-ftp from nikhefh.nikhef.nl in the "form" directory. Since the developers intend to develop it for commercial use, it is only available in binary form.

○ Jacal

This package is Scheme-based and thus, is slow in that it runs on top of an general-purpose language interpreter. It also lacked proper support for rational numbers. Numerators and denominators were represented by long ints, and thus were not capable of representing quantities to the desired precision.

○ Pari

This package would have probably been my eventual choice, had the system not been sufficiently stable for the floating point arithmetic supported by C.

It originally began life as a math library for C that allowed manipulation of abstract entities, like formulae and matrices, in addition to providing arbitrary-precision fixed-point arithmetic. More modern releases of Pari include a front-end to allow one to use it as a calculator. Some simple scripts would probably have sufficed to generate all the necessary quantities.

Given that it is a general-purpose tool, Pari was still orders of magnitude slower than the C implementation. However, it is not as general a tool as Maple, and thus is probably two to 5 times faster for simple numeric tasks.

This package is available in source code format from math.ucla.edu.

○ APML (Arbitrary Precision Math Library)

This C library, and others like it, were considered, but there never came a time where any code was implemented that utilized this. Given the current availability of matrix and floating-point libraries for C++, and the ability of C++ to "transparently" "overload" operators, I would think that C++ would be a better choice than C for any numeric applications that may require the underlying number representation to suddenly change.

This library is available from any comp.sources.misc archive, but the GNU Multi-Precision (GMP) library is newer and looks quite a bit better. GMP should be available on prep.ai.mit.edu.

5. Numerical Results and Discussion

Figure 4 shows the mean and standard deviation of the packet delay in the STA and MTA under gated access. Our calculated data is also shown in Table 3.

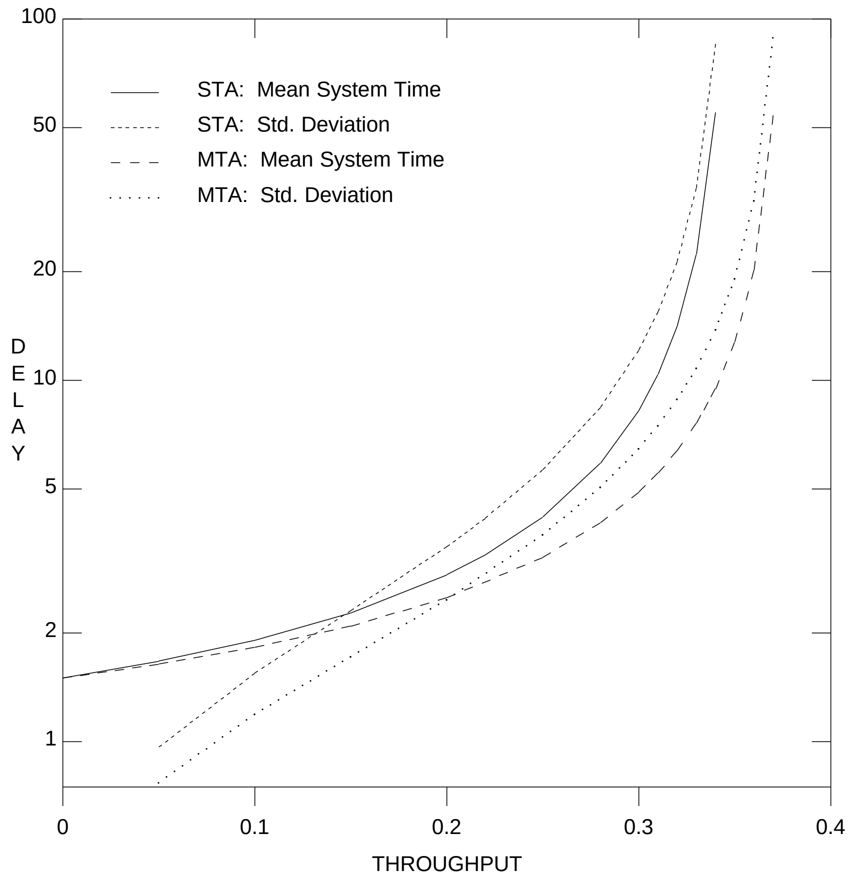

Figure 4: Packet Delay in the STA and MTA with Gated Access

λ	Standard Tree Algorithm		Modified Tree Algorithm	
	$\overline{T}$	σ_T	$\overline{T}$	σ_T
0.0	1.5	0.08333	1.5	0.08333
0.05	1.671	0.9659	1.640	0.7682
0.10	1.909	1.549	1.825	1.190
0.15	2.273	2.306	2.089	1.716
0.20	2.896	3.467	2.502	2.470
0.22	3.288	4.153	2.740	2.886
0.25	4.184	5.659	3.238	3.727
0.28	5.892	8.419	4.038	5.042
0.30	8.246	12.14	4.912	6.454
0.31	10.38	15.48	5.543	7.466
0.32	14.11	21.29	6.392	8.827
0.33	22.28	34.04	7.603	10.76
0.34	55.11	85.44	9.471	13.77
0.35	–	–	12.76	19.07
0.36	–	–	20.17	31.24
0.37	–	–	54.13	88.46

Table 3: Calculated Delay Statistics for Tree Algorithms with Gated Access

In Figure 5 we compare our results for the mean packet delay in the STA with Massey's bounds, after correcting a minor error in his analysis (which is described in the Appendix). It is interesting to note that, although Massey thought the upper bound was much tighter than the lower bound, the mean packet delay is actually closer to his lower bound. Massey also gave an approximate lower bound, in which he replaced a (loose) lower bound on t_0 by a (tight) upper bound in his lower bounding argument. As we can see, the resulting expression is only a lower bound when $\lambda \leq 0.22$. Nevertheless, it is a remarkably good approximation to $\overline{T}$ for all λ.

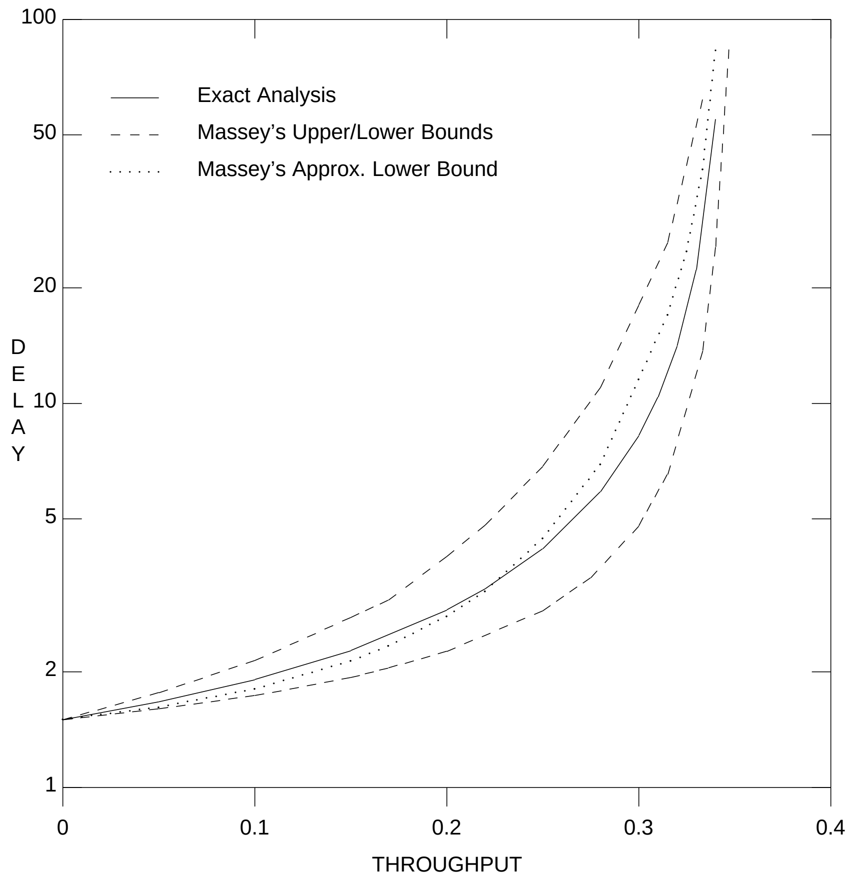

Figure 5: Comparison of STA Results to Massey's Bounds

And finally, in Figures 6–7, we compare our results for the STA using gated access with the corresponding results for window access and constant-sized window access, obtained by Polyzos [1]. In Figure 6, we compare the mean system times, where we see that gated access is comparable to window access when $\lambda<0.2$, and much worse than even constant-sized window access when $\lambda>0.3$. Figure 7 shows the corresponding results for the standard deviation. As expected, gated access looks much worse under this metric: it is comparable to window access only when $\lambda<0.1$, and becomes much worse than even constant-sized window access when $\lambda>0.2$.

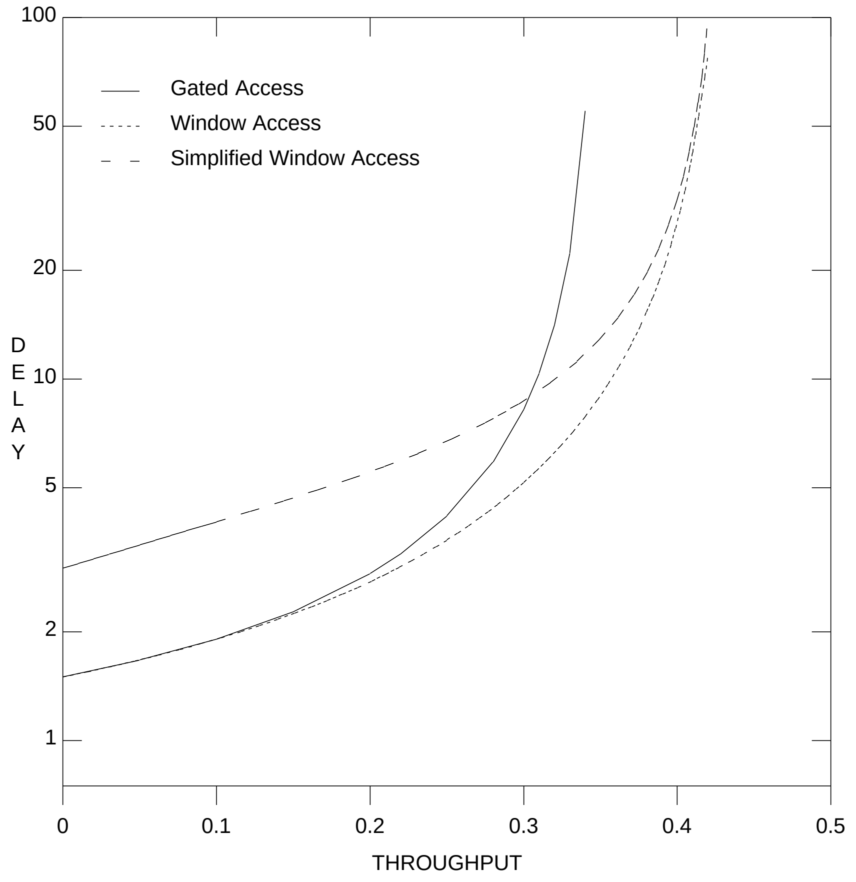

Figure 6: Comparison of Mean Delay in STA with Gated and Window Access

6. Appendix: Correction to Massey's Bounds on $\overline{T}$ for the STA

In this section, we correct a minor error in Massey's derivation of upper and lower bounds on the mean delay in the STA, as shown in [2]. Massey's approach begins by recognizing that the delay for a randomly chosen "tagged" packet depends on both the length, Y_a, of the ongoing CRI when the "tagged" packet arrives, and the length, Y_d, of the next CRI during which the "tagged" packet departs. In particular, its system time is the sum of the residual life of first and the age of the second, which we denote by t_0 and t_2, respectively. The difference in Massey's approach is that, instead of solving a Markov chain to find the distribution of Y_a and Y_d, he merely finds upper and lower bounds on $E[Y_a]$, and on $E[Y_d \mid Y_a]$, and uses these to estimate $E[t_0]$, and $E[t_2]$.

The error in Massey's analysis crept in at Eq. (4.29), where he used the Poisson arrival property to deduce that, X_d, the mean number of packets served in the CRI where the "tagged" packet departs, is simply $\lambda \cdot Y_a$. However, since we are conditioning on the fact that the "tagged" packet happened to arrive in *this* CRI, *the "tagged packet" is not accounted for in the Poisson term.* Thus, we have:

$$E[X_d \mid Y_a = L, \text{tagged departure}] = \lambda \cdot E[Y_a] + 1\,, \tag{4.29}$$

and

$$E[X_d \mid \text{tagged departure}] = \lambda \cdot E[Y_a] + 1\,. \tag{4.30}$$

In other words, the probability that $X_d = N$ is given by the Poisson density evaluated at $N-1$. Carrying this change through Massey's derivation yields:

$$E[Y_d \mid \text{tagged departure}] \le 2.8867 \cdot E[X_d \mid \text{tagged departure}] - 1.8867 \cdot P[X_d = 1 \mid \text{tagged departure}] \tag{4.32}$$

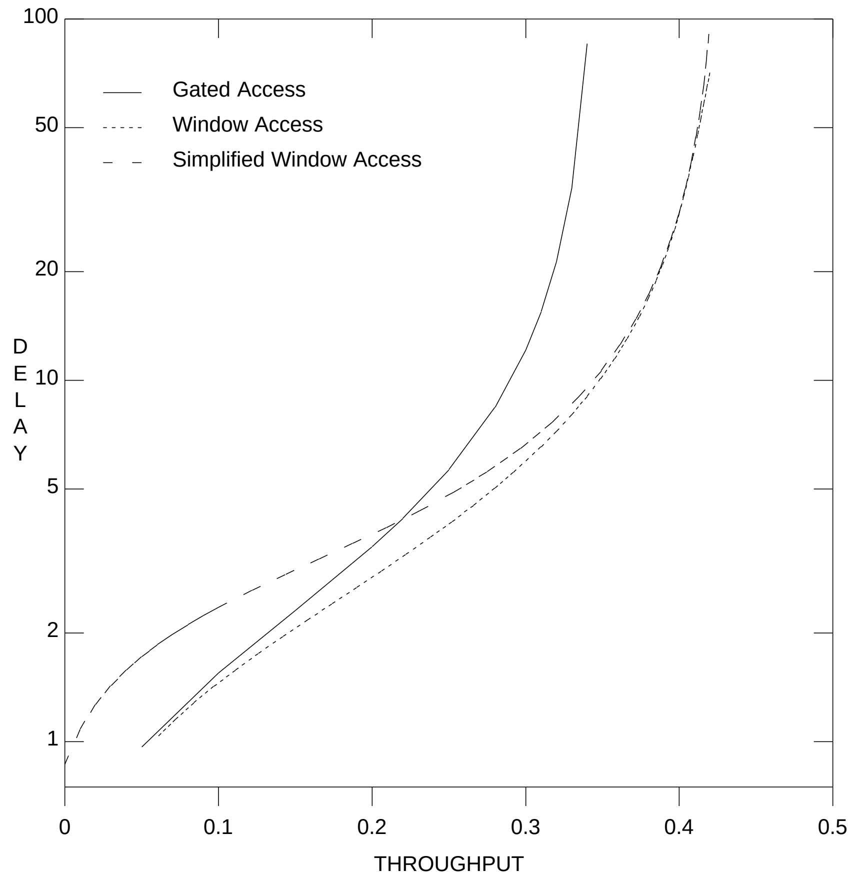

Figure 7: Standard Deviations of Delay in STA with Gated and Window Access

Since Jensen’s inequality can be used to show that

$$P[X_d = 1 \mid \text{tagged departure}] \geq e^{-\lambda \cdot E[Y_a]},$$

we can substitute Eq. (4.30) into Eq. (4.32) to obtain the following upper bound:

$$E[Y_d \mid \text{tagged departure}] \leq 2.8867 \cdot (1 + \lambda \cdot E[Y_a \mid \text{tagged arrival}]) - 1.8867 \cdot e^{-\lambda \cdot E[Y_a]}. \qquad (4.36)$$

The lower bounds begins from Eq. (4.37), which reduces to:

$$E[Y_d \mid \text{tagged departure}] \geq 2.8810 \cdot E[X_d \mid \text{tagged departure}] - 1.0 - 0.8810 \cdot P[X_d = 1 \mid \text{tagged departure}],$$

since we know that $X_d > 0$ because of the "tagged" arrival. But since

$$P[X_d = 1 \mid Y_a = L, \text{ tagged departure}] = e^{-\lambda \cdot L} \leq e^{-\lambda},$$

we arrive at the following lower bound:

$$E[Y_d \mid \text{tagged departure}] \geq 2.8810 \cdot (1 + \lambda \cdot E[Y_a \mid \text{tagged arrival}]) - 1.0 - 0.8810 \cdot e^{-\lambda}. \qquad (4.40)$$

These bounds on $E[Y_a]$ and $E[Y_d]$ are translated into upper and lower bounds on the packet waiting time by Eq. (4.41), which is equivalent saying that

$$E[t_0] = E[Y_a \mid \text{tagged arrival}]]/2,$$

and that

$$\frac{1}{2} E[Y_d - 1 \mid \text{tagged departure}] \leq E[t_2] - 1 \leq E[Y_d - 1 \mid \text{tagged departure}]$$

Below, in Table 3, we have recalculated Massey's bounds for the mean time in system for a packet, $\bar{T}$. These data differ from Massey's results (Table 4.1 in [2]) because we add one more slot to account for the transmission time for each packet (where Massey showed only the waiting time), and because we have used the corrected versions of Eqs. (4.36) and (4.40). (We have also used Eq. (4.81), instead of $\lambda(1-\lambda)$, to represent π_1 in Eq. (4.60) in the range $0.22 < \lambda < 0.3$, to avoid an anomaly in the bound in this range.) Table 3 also includes Massey's strict lower bound on the waiting time, which was not shown in [2]. This is because our results show his approximate lower bound is actually an upper bound when $\lambda > 0.22$.

λ	$\bar{T}$ upper bound	$\bar{T}$ approx. lower bound	$\bar{T}$ lower bound
0.0	1.5	1.5	1.5
.05	1.764976	1.6158	1.602
.1	2.13967	1.801	1.733
.15	2.77037	2.1297	1.932
.1696	3.0768	2.331	2.047
.2	3.9955	2.786	2.26
.22	4.82	3.239	
.25	6.864	4.4405	2.885
.28	11.008	6.93	3.694
.3	18.065	11.517	4.798
.315	26.306	16.975	6.525
.324		24.863	
.33333333	61.84	40.645	13.73
.34	125.09	82.773	25.554
.347	565.3	374.5	83.578

Table 3: Massey's Bounds on the Mean System Time (corrected)

7. References

[1] George C. Polyzos, *A Queueing Theoretic Approach to the Delay Analysis for a Class of Conflict Resolution Algorithms*, Technical Report CSRI-224, Computer Systems Research Institute, University of Toronto, Toronto (January 1989).

[2] James L. Massey, "Collision-Resolution Algorithms and Random-Access Communications", in *Multi-User Communications*, G. Longo (ed)., Springer-Verlag, New York (1981).

[3] George E. Forsythe and Cleve B. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, Englewood Cliffs (1967).