Performance of Batch-based Digital Signatures*

[Appeared in Proceedings of IEEE MASCOTS 2002]

William C. Cheng† bill.cheng@acm.org

Cheng-Fu Chou‡ ccf@csie.ntu.edu.tw

Leana Golubchik§ leana@cs.usc.edu

Abstract

A Digital Signature is an important type of authentication in a public-key *(or asymmetric) cryptographic system, and it is in wide use. The performance of an Internet server computing digital signatures online is limited by the high cost of modular arithmetic. One simple way to improve the performance of the server is to reduce the number of computed digital signatures by combining a set of documents into a batch in a smart way and signing each batch only once. This reduces the demand on the CPU but requires extra information to be sent to clients.*

In this paper, we investigate performance characteristics of online digital signature batching schemes. We give a semi-Markov model of a gated batch-based digital signature server and its approximate solution. We validate the solutions of the analytical model through both emulation and simulation. Our study shows that significant computational benefits can be obtained from batching without significant increases in the amount of additional information that needs to be sent to the clients.

1 Introduction

A Digital Signature is an important type of authentication in a *public-key* (or asymmetric) cryptographic system, and it is in wide use [13, 16]. If Alice would like to send an authenticated (but not encrypted) message M to Bob, Alice can compute a digital signature from M, denoted by DS[M], concatenate M with DS[M], denoted by M+DS[M], and send M+DS[M] to Bob. Bob, with possession of Alice's public key, can verify the following important security properties of the message.

- *Integrity* – that not a single bit in the message has been altered.
- *Authentication* – that the message was truly sent by Alice.
- *Nonrepudiation* – that Alice cannot deny that she has sent the message.

Another important property of a digital signature is that it is *not* vulnerable to the so-called *man-in-the-middle* attack, i.e., no one (other than Alice) can change a single bit of either M or DS[M] without Bob noticing that the message, M+DS[M], has been altered.

In a client/server-based application, a server, which offers a set of services, can play the role of Alice and a client can play the role of Bob. Often, a client would like to obtain a receipt from the server, describing the service rendered, and signifying the completion of the prescribed transaction. A digital signature can act as such a receipt due to the nice security properties listed above. There are many client/server-based applications where a digital signature is desirable. For example, in a lottery ticket selling service (or a concert tickets purchasing priority numbers issuing service), a server can timestamp and digitally sign each ticket it issues; in a pay-per-view stock tip service, a server can generate the latest report on a stock symbol from its database and digitally sign the report; in an income tax form collection service, a proposal collection service, a conference paper collection service, or a bid collection service for contract bidding, a server can generate a timestamp and send the digitally signed timestamp as proof that a client's submission has been received and that the client has made a deadline [7]. Note that for most of these online applications (stock tip type service being the exception), the document being signed is a timestamp or a small collection of numbers, i.e., the size of the document is fairly small.

A typical application is illustrated in Figure 1. In Figure 1(a), a server is shown to provide documents to a large number of clients spread across the network, such as the Internet. (This can be generalized to multiple services and multiple/mirrored servers). Each client j sends a request for a document, I_j, to the server. The server digitally signs document I_j, to produce DS[I_j] and sends the document together with the digital signature to client j (refer to Figures 1(b) and (c)).

*This work is supported in part by the NSF Digital Government Grant EIA-0091474.

†TeleGIF, Marina del Rey, California. This work was partly done while the author was with the Department of Computer Science and UMIACS at the University of Maryland.

‡Department of Computer Science and Information Engineering, National Taiwan University. This work was partly done while the author was with the Department of Computer Science and UMIACS at the University of Maryland.

§Computer Science Department, IMSC and ISI, University of Southern California. This work was partly done while the author was with the Department of Computer Science and UMIACS at the University of Maryland.

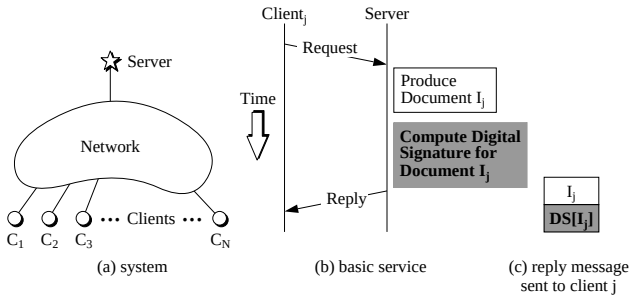

Figure 1. Digital Signatures Problem.

The main problem here is that the digital signature process is very computationally expensive. Therefore, if the particular service is very popular, under high loads, the response time for a client can be very large. Each of the applications mentioned above corresponds to a *real-life event* which may experience high demands at certain times. For example, demands for popular concert tickets can be high when the tickets first go on sale, while submission of income tax forms are often done close to the deadline. It is the main focus of this paper to model and analyze performance of approaches used to reduce a client's response time when the load on the server is high.

Let us examine the digital signature process more carefully to see where is the performance bottleneck. Figure 2 depicts the process of digitally signing and verifying a document M [16]. We focus on the left half (signing) in Figure 2. (We will denote the verification procedure, the right half of Figure 2 by VERIFY[M,DS[M]].) Let us denote the public and private keys used for a service $\mathcal{S}$ provided by the server as $K^{\mathcal{S}}_{pub}$ and $K^{\mathcal{S}}_{priv}$. (Please note that we will drop the $\mathcal{S}$ superscript for clarity.) The server applies a one-way hash (uninvertible) function H to document M to produce H(M). The size of H(M) is fixed (on the order of 20 bytes, depending on which digital signature system the server is required to use). The server then uses K_{priv} to digitally sign H(M), the output of which is referred to as the *digital signature* of document M, denoted by DS[M]. Although there are several existing different digital signature algorithms (e.g., RSA, DSA, etc. [13]), the basic signing process is the same. The differences among these algorithms are abstracted in the block labeled DS in Figure 2.

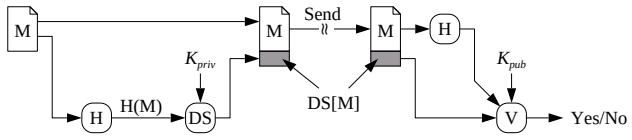

Figure 2. Digitally Signing and Verifying a Document M.

Since the creation of a digital signature mainly consists of the H and the DS blocks, let us examine the relative time spent in each block. The time to compute a one-way hash function is linearly proportional to the size of M. The time spent in the DS block is also linearly proportional to the size of its input. But since input to the DS block is H(M), whose size is fixed, the time spent in the DS block is constant. If the size of M is small, computing DS[M] is typically several orders of magnitude more expensive than computing H(M). For example, on a 800 MHz Pentium-III PC running Linux, digitally signing 20 bytes of data (with a 1024-byte private key) takes about 4 milliseconds while computing a hash of 20 bytes of data takes about 1 microsecond with OpenSSL [17].

Hence, under high workloads, there is a need for reducing a server's computational requirements which are due to digital signatures. To this end, we explore performance improvements due to the use of batching schemes (described next). We present an analytical model of a gated batch-based digital signature server and its approximate solution. We validate the solution of the analytical model through both emulation and simulation. Our study shows that, using standard cryptographic techniques, the server's performance under high loads can be significantly improved. That is, even under high load (near 100% utilization), the server can keep up with the demands (without sacrificing security) while keeping computational and networking overhead at a minimal.

2 Background on Batching Schemes

To reduce the computational requirements of digital signatures for large numbers of clients we explore the use of a previously proposed approach [11, 12], in a systems setting.

First, we describe a non-batched system, as in the scheme used in [2], as a baseline for comparison. This system is depicted in Figure 3. (This scheme is commonly used in Internet servers.) In order to guard against *replay*

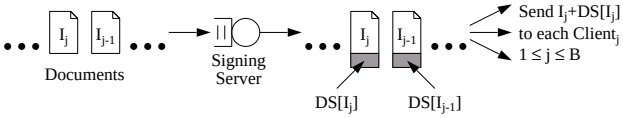

Figure 3. No batching.

attacks [13], we require that each request j is accompanied by a nonce, r_j, whose length is comparable to the length of a hash value, and we also require that, in the response, the corresponding document I_j includes r_j in its header, so that the client can verify that I_j was indeed intended for it. (The header in I_j may include additional information such as a server timestamp.) For each document I_j, the signing server produces DS[I_j] and sends I_j + DS[I_j] to the client which requested I_j in a first-come-first-served fashion. The digital signature of a document is computed independently of other documents and signatures. The digital signature scheme used here is depicted, in general, as shown in the left half of Figure 2. Clearly, at high loads, a large queue

can build up at the signing server since computing digital signatures is CPU intensive. Note that in this scheme, there is no *wasted* network bandwidth, in the sense that a client C_j only receives I_j + DS[I_j], i.e., it does not receive information that does not belong to it. (The verification procedure is performed by the client and is not described here; please refer to [16].)

2.1 Simple Batching

A very simple way to batch requests is to use a *gated* server, as depicted in Figure 4. We use the term *customer* and *client request* interchangeably. When the server becomes free, it closes a gate behind the last customer in the queue and serves all the customers inside the gate in a batch. All newly arrived customers queue up behind the gate. When the server finishes serving the batch, all customers inside the gate depart from the server. The process then repeats. If the server is free when a customer arrives, it closes the gate behind this customer and serves this customer only.

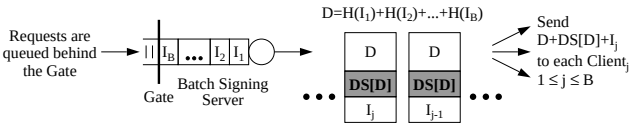

Figure 4. Simple Batching Scheme.

Let B be the number of client requests (or customers) waiting when the server completes the previous computation of a digital signature (B is also referred to as the *batch size*). Each client j requires document I_j and a digital signature to verify the security properties related to I_j. One approach to reducing the computational cost of signing each document independently is to first concatenate the B documents corresponding to the B clients waiting in the queue and then sign them all together, i.e., produce D=I_1+I_2+...+I_B, and DS[D] and send D+DS[D] to each client. However, client j will be able to see I_k where $k \neq j$. To get around this problem, instead of concatenating the documents, we can concatenate the hashes of the documents, i.e., produce D=H(I_1)+H(I_2)+...+H(I_B), and DS[D] and send D+DS[D]+I_j to client j, for $1 \leq j \leq B$. (Note that this D can be considerably smaller than one obtained by concatenating the actual documents.) Then, our client and server algorithms are:

Server algorithm (refer to Figure 4):
1) let D = H(I_1) + H(I_2) + ... + H(I_B);
2) compute DS[D];
3) construct message D+DS[D]+I_j for each client $1 \leq j \leq B$.

Client j algorithm (upon receiving DS[D]+D+I_j):
1) VERIFY[D,DS[D]];
2) verify that the nonce, r_j, is in the header of I_j;
3) compute H(I_j);
4) verify H(I_j) can be found in D.

This is referred to as the Simple Batching scheme. It can give a significant CPU speed improvement since it only computes 1 digital signature for each batch as oppose to doing B digital signature computations in the non-batched scheme. However, this requires larger message sizes. The total overhead on the system, as far as message sizes are concerned, is B times the size of D (since sizeof(DS[D]) equals sizeof(DS[I_j])). The size of D is just B times the size of a hash. Therefore, the message size overhead is $B^2 \cdot sizeof(hash)$.

2.2 Tree-based Batching

Motivated by the potential need to save network bandwidth (not just CPU speed) and hence reduce message sizes, we employ a tree-based batching scheme [11] which builds a tree of hashes, where leaves of the tree are hashes of documents. Only the root of the tree, which is denoted by R, needs to be digitally signed. The value of an internal node of the tree is computed by concatenating the values of its children and then applying the hash function. The security properties of the tree-based batching scheme are the same as those of the non-batched scheme [12]. For this scheme, we also use a *gated* server which operates similarly to the Simple Batching scheme depicted in Figure 4. The main difference is in the content of the out-going messages. Let $P_M = [x_1, x_2, \ldots, x_h]$ denote a list of nodes along the path from a leaf node that corresponds to document M to the root node, excluding the root node, and let h denote the depth of the leaf node (given that the root is at depth 0). Let $S_M = [y_1, y_2, \ldots, y_h]$ denote a list of nodes, where y_i is the sibling of the corresponding node x_i in P_M. (S_M is called the *authentication path* in [11].) The client and server algorithms are modified as follows.

Server algorithm:
1) build a complete binary tree with leaf nodes H(I_1), H(I_2), ..., H(I_B); the value of an internal node X with children Y and Z is simply X = H(Y+Z);
2) compute DS[R] (where R is the root of the tree);
3) construct message R+DS[R]+S_{I_j}+I_j for each client $1 \leq j \leq B$.

Client j algorithm (upon receiving R+DS[R]+S_{I_j}+I_j):
1) VERIFY[R, DS[R]];
2) verify that the nonce, r_j, is in the header of I_j;
3) let $S_{I_j} = [y_1, y_2, \ldots, y_h]$; compute H($I_j$) and run the following simple algorithm:
   $r \leftarrow$ H(I_j);
   for (i=1 to h) do $r \leftarrow$ H($r + y_i$);
   verify that $r = R$.

Compared with the Simple Batching scheme, the Tree-based Batching scheme computes only one digital signature, but it computes twice as many hashes. Compared with

the non-batched scheme, the total overhead on the network for the Tree-based Batching scheme, as far as message sizes are concerned, is B times the size of a hash times the height of the complete binary tree. Therefore, the message size overhead is $B \times \log_2(B) \times sizeof(hash)$.

In general, an m-ary tree can be used instead of a binary tree to reduce the number of additional hashed needed. However, the message size overhead is then $B \times (m-1) \times \log_m(B) \times sizeof(hash)$. It can easily be shown that the above expression is monotonically increasing for $m > 1$ (by taking the derivative of the expression with respect to m), and therefore, is minimized when $m = 2$.

3 Performance Evaluation

Above batch-based digital signing schemes are motivated by the need to improve Internet servers' performance, especially under high workloads. Hence, there is a need for evaluation of the resulting performance characteristics. Moreover, there are tradeoffs to be considered in the design of batch-based schemes, such as the tradeoff between computational improvements and increases in messages sizes (and hence network bandwidth needs). These tradeoffs also motivate the need for performance evaluation studies. Thus, in what follows, we propose the use of analytical models for performance evaluation of batch-based digital signing schemes. We construct such models for both batching schemes and validate them against emulation and simulation. In this validation, we consider documents of various sizes. Such models are a useful tool in capacity planning and system sizing as well as in studying design tradeoffs.

3.1 Analysis

We begin with the analysis of the digital signature *gated* server with batching schemes described in Sections 2.1 and 2.2.

The following analysis is carried out under the assumption that the digital signature is computed using a private key type operation (such as in RSA and DSA), and that the one way function is computed through a typical hash function such as SHA1 or MD5. Hence, the digital signature computation is *significantly* more costly than the hash function computation, given the same size document. For instance, the OpenSSL benchmark [17] results in more than 3 orders of magnitude difference between digital signature computation and hash function computation for a 20 byte message.

3.1.1 System Model

We first consider the case where the size of the document is small. By small, we mean that the time it takes to hash a document is less than 10% of the time it takes to digitally sign a 20-byte string (or a 16-byte string if MD5 is used). A different (but simpler) model can be used for larger documents, which is described later.

Given the above assumption, we model the service time for computation of a digital signature of a batch of B customers as being deterministic and independent of the batch size (i.e., most of the time is spent in computing the digital signature so we ignore the fact that the hash function computation time is a function of the batch size). That is, we can think of the server as an M/D/1 queue with batching, i.e., where the arrival process is Poisson with rate λ and the service time is deterministic and equal to $1/\mu$.

We *approximate* this model with the semi-Markov process, $\mathcal{SM}$, depicted in Figure 5. The state of $\mathcal{SM}$ is described by (i, j), where i indicates the number of customers waiting in the queue and j indicates whether the server is busy or idle, with $j = 0$ indicating that it is idle and $j = s$ indicating that it is busy. Lastly T is the *mean residual* service time, and it is equal to $1/(2\mu)$ (refer to [8]). (That is, if we let r.v. X represent the digital signature plus hash function computation time, and we let r.v. Y represent the corresponding residual computation time, then $E[Y] = E[X^2]/2E[X]$.)

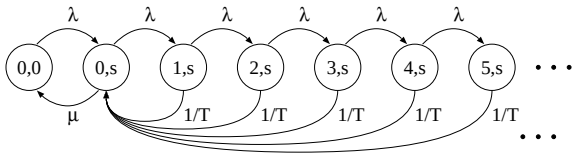

Figure 5. Semi-Markov Process Model of Server.

Let τ_0 be the mean holding time in state $(0, 0)$, τ_1 be the mean holding time in state $(0, s)$, and τ_2 be the mean holding time in state (i, s), for $i > 0$. Then,

$$\tau_0 = \int_0^\infty t\lambda e^{-\lambda t} dt = \frac{1}{\lambda}$$

$$\tau_1 = \int_0^{\frac{1}{\mu}} t\lambda e^{-\lambda t} dt + \frac{1}{\mu} \int_{\frac{1}{\mu}}^\infty \lambda e^{-\lambda t} dt = \frac{1}{\lambda}(1 - p_1)$$

$$\tau_2 = \int_0^T t\lambda e^{-\lambda t} dt + T \int_T^\infty \lambda e^{-\lambda t} dt = \frac{1}{\lambda}(1 - p_2)$$

where $p_1 = e^{-(\lambda/\mu)}$ and $p_2 = e^{-(\lambda/2\mu)} = e^{-\lambda T}$. Then, we can solve this model by constructing the corresponding Discrete-Time Markov Chain (or DTMC), $\mathcal{M}$, as illustrated in Figure 6, as follows. Let $\boldsymbol{\pi} = \{\pi_0, \pi_1, \pi_2, \ldots\}$ be the steady state distribution for $\mathcal{M}$, where π_0 corresponds to state $(0, 0)$, π_1 corresponds to state $(0, s)$, and π_i corresponds to state $(i - 1, s)$ for $i \geq 2$. Then we have the following set of equations:

$$\begin{aligned} \textstyle\sum_{i=0}^\infty \pi_i &= 1 \\ \pi_0 &= \pi_1 p_1 \\ \pi_1 &= \pi_0 + \textstyle\sum_{i=2}^\infty \pi_i p_2 \end{aligned}$$

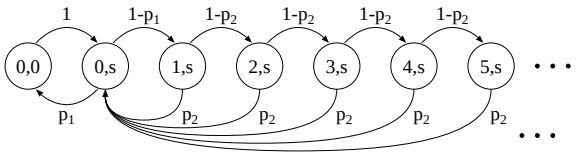

Figure 6. DTMC corresponding to $\mathcal{SM}$.

$$\pi_i = \pi_1(1-p_1)(1-p_2)^{i-2} \quad \forall i \geq 2$$

which gives us

$$\begin{aligned}
\pi_0 &= \frac{p_2 p_1}{p_1 p_2 + p_2 + 1 - p_1} \\
\pi_1 &= \frac{p_2}{p_1 p_2 + p_2 + 1 - p_1} \\
\pi_i &= \frac{p_2(1-p_1)(1-p_2)^{i-2}}{p_1 p_2 + p_2 + 1 - p_1} \quad \forall i \geq 2
\end{aligned}$$

Let $\boldsymbol{\pi}^* = \{\pi_0^*, \pi_1^*, \pi_2^*, \ldots\}$ be the steady state distribution for $\mathcal{SM}$. Then,

$$\begin{aligned}
\pi_0^* &= \frac{\pi_0 \tau_0}{\pi_0 \tau_0 + \pi_1 \tau_1 + \sum_2^\infty \pi_i \tau_2} \\
\pi_1^* &= \frac{\pi_1 \tau_1}{\pi_0 \tau_0 + \pi_1 \tau_1 + \sum_2^\infty \pi_i \tau_2} \\
\pi_i^* &= \frac{\pi_i \tau_2}{\pi_0 \tau_0 + \pi_1 \tau_1 + \sum_2^\infty \pi_i \tau_2} \quad \forall i \geq 2
\end{aligned}$$

which gives

$$\pi_0^* = \frac{p_1 p_2}{1 - p_1 + p_1 p_2}.$$

Finally, we compute the mean waiting time of the M/D/1 queue with batching as:

$$\overline{W} = (1 - \text{Prob[system is empty]}) \cdot T$$

where Prob[system is empty] is the probability of an arrival finding the M/D/1 batching system empty and $T = 1/(2\mu)$ is the mean residual service time of the M/D/1 batching system. We approximate Prob[system is empty] by π_0^*, which is obtained by solving the semi-Markov processed given in Figure 5. Then, the corresponding mean response time is approximated as:

$$\begin{aligned}
\overline{T} &= \frac{1}{\mu} + \overline{W} = \frac{1}{\mu} + \frac{1}{2\mu}(1 - \pi_0^*) \\
&= \frac{1}{\mu} + \frac{1}{2\mu}\left(\frac{1 - e^{-\frac{\lambda}{\mu}}}{1 - e^{-\frac{\lambda}{\mu}} + e^{-\frac{3\lambda}{2\mu}}}\right)
\end{aligned}$$

3.1.2 Model Parameters

What remains is to derive the model parameters as a function of the digital signature and hash function operations. Specifically, we need to derive the service time for a batch of size B. Let the digital signature computation for a fixed size (around 20 bytes) string take t_{ds} units of time. Let the computation of a hash function for a similar size document (i.e., similar to the size of the output of a hash function) take t_h^s time units. And, let the computation of a hash function for an "average" size document sent in the reply message in our system take t_h units of time, i.e., we simplify the following derivation by assuming that the documents being sent by our system are of reasonably comparable size and thus the hash function computation of an average size document is representative of the time it takes to compute a hash function on some document I_j. We further simplify the parameter derivation by assuming an average size batch and approximating it as λ/μ (i.e., $B \approx \lambda/\mu$). Given that the digital signature computation is significantly more costly, the above simplifications are reasonable. Referring back to Figure 2, t_{ds} is the time spent in the DS block, t_h is the time spent in the H block for an average-sized message M, and t_h^s is the time spent in the H block if M has the size of a hash (i.e., around 20 bytes).

We further approximate the time it takes to compute a hash function of a document of size $k \times L$ by k times the time it takes to compute a hash function of a document of size L. (This is a reasonable assumption given currently used hash functions and the results from the OpenSSL benchmark [17].)

Then, in the case of the Simple Batching scheme, the time required to compute D is $(\lambda/\mu)t_h$. To digitally sign D, the time spent in the H block of Figure 2 is approximately $(\lambda/\mu)t_h^s$ (by the above assumption) and the time spent in the DS block of Figure 2 is t_{ds}. Therefore, the time to compute D+DS[D] is:

$$\frac{1}{\mu} = t_{ds} + \frac{\lambda}{\mu}t_h + \frac{\lambda}{\mu}t_h^s = \frac{t_{ds}}{1 - \lambda(t_h + t_h^s)} \qquad (1)$$

Similarly, in the case of the binary Tree-based Batching scheme, we have:

$$\frac{1}{\mu} = t_{ds} + \frac{\lambda}{\mu}t_h + 2(\frac{\lambda}{\mu} - 1)t_h^s = \frac{t_{ds} - 2t_h^s}{1 - \lambda(t_h + 2t_h^s)} \qquad (2)$$

It can easily be shown that the stability regions for the schemes described above are:

$$\lambda < \begin{cases} 1 \,/\, (t_{ds} + t_h) & \text{for non-batched} \\ 1 \,/\, (t_h + t_h^s) & \text{for Simple Batching} \\ 1 \,/\, (t_h + 2t_h^s) & \text{for Tree-based Batching} \end{cases} \qquad (3)$$

We note that, in the case of the m-ary Tree-based Batching scheme, we have:

$$\begin{aligned}
\frac{1}{\mu} &= t_{ds} + \frac{\lambda}{\mu}t_h + \frac{\lambda}{\mu}m t_h^s \left(\sum_{i=1}^{log_m(\lambda/\mu)} \frac{1}{m^i}\right) \\
&= t_{ds} + \frac{\lambda}{\mu}t_h + (\frac{\lambda}{\mu} - 1)\frac{m}{m-1}t_h^s \\
&= \frac{t_{ds} - (\frac{m}{m-1})t_h^s}{1 - \lambda[t_h + (\frac{m}{m-1})t_h^s]}
\end{aligned}$$

In the above equation, although a higher value of m will reduce $1/\mu$, it only reduces $1/\mu$ by a very small amount because t_{ds} is several orders of magnitude larger than t_h^s. Based on the discussion presented at the end of Section 2.2, the message size overhead increases as m increases. Therefore, it is not worthwhile to use a value of $m > 2$.

3.2 Validation of Analytical Models

In this section we validate our analysis by comparing the analytical results obtained above with those obtained through emulation as well as simulation.

Specifically, we perform the emulation by executing the digital signature schemes described in Section 2 using the OpenSSL [17] implementation on an 800 MHz Pentium-III PC running Linux. This is an emulation since we still use a Poisson arrival stream with rate λ as our customer requests workload. The simulation of these schemes is performed under the same conditions as emulation, using CSIM [15], except that the service times used in the simulation are computed using the OpenSSL benchmark. We note that we validate through simulation, in a subset of cases, in addition to emulation, since in the emulation environment we cannot guarantee that no other workload would be running on the emulation machine at the time of emulation experiments (more details are given below). All emulation and simulation results are reported with $95\% \pm 5\%$ confidence intervals.

We compute the parameters of the analytical models using equations given in Section 3.1.2, where t_{ds} and t_h are set to the values produced by the benchmark provided with OpenSSL [17] and executed on the same machine as the emulation.

The comparison of analytical and emulation results are given in Figure 7, for mean response time, and Figure 8, for mean batch size. Figures 7(a)-(d) depict the results corresponding to the Simple Batching scheme, Figures 7(e)-(h) depict the results corresponding to the (binary) Tree-based Batching scheme, Figures 8(a)-(d) validate our approximation of an average batch size as λ/μ for the Simple Batching scheme, and Figures 8(e)-(h) validate our approximation of an average batch size as λ/μ for the (binary) Tree-based Batching scheme (refer to Section 3.1.2).

For relatively small document sizes (20 bytes, 1KB, and 10K), mean response times predicted by our analytical model are fairly close to that of emulation for both batching schemes, especially for small arrival rates. Although at higher arrival rates the errors can be larger, they are no more than 10% for both batching schemes. For document size of 100KB, the assumption that hashing time is not a function of batch size (or rather just using the approximate mean batch size to approximate the hashing time) is no longer a good approximation, and the model performs worse, as shown in Figures 7(d) and 7(h). (For even larger document sizes, please see discussion below.) We also note a problem with emulation experiments. The emulation is running on a machine where the background load fluctuates. As emulation times get longer (i.e., for larger document sizes), additional background load interference with the emulation experiments is more significant. Therefore, in Figures 7(d) and 7(h), we also added simulation results obtained using the same parameters used for the emulations. We show that the error is reduced. However, the assumption that hashing time is not a function of batch size still accounts for the larger errors.

Validation results for mean batch size, given in Figure 8, show that the error of our approximation of mean batch size by λ/μ is fairly small for both batching schemes. The error characteristics for larger document sizes are similar to the validation results for mean response time.

For even larger document sizes (e.g., 1MB), the time it takes to perform hash calculations begins to dominate the time it takes to perform digital signatures. The comparison of analytical and emulation results are given in Figure 9 for mean batch size and mean response time. Clearly, the approximation that the batch size is λ/μ is a poor one, as demonstrated in Figures 9(a)-(b). Nevertheless, the mean response time prediction is shown to be less sensitive to the batch size estimation, as shown in Figures 9(d)-(e). We also note that since the mean batch size for large document sizes (such as 1MB) is fairly close to 1 even under relatively high arrival rates (as is evident from Figures 9(a)-(b)). Hence, a simple M/D/1 model without batching can be used in these cases (its mean response time is given in Eq.(4) below). This is validated in Figures 9(d)-(e) where the M/D/1 model results in very small errors as compared to simulation.

Since we also would like to make comparisons with the original non-batched digital signature scheme, i.e., one which signs each requested document independently (refer to Section 2), we also model it as an M/D/1 queue, but without batching, whose mean response time is given by (refer to [8]):

$$\overline{T} = \frac{1}{\mu} + \frac{1}{\mu}\left(\frac{\frac{\lambda}{\mu}}{2(1-\frac{\lambda}{\mu})}\right) \tag{4}$$

The parameters of this model can be computed as in Section 3.1.2, i.e., $1/\mu = t_{ds} + t_h$. And, the validation of this model through our emulation is illustrated in Figure 10 as well as Figure 9(c) for the 1MB document size.

In addition to using a Poisson arrival processes, we have performed emulation and simulation studies where request inter-arrival times are distributed according to uniform or normal distributions. These emulations and simulations show that the results are fairly insensitive to the arrival process distribution (at least for the distributions we tried). That is, the difference in mean response time and mean batch size between the results obtained using a Poisson process and other processes is comparable to the difference between the analytical model results and the emulation/simulation results using a Poisson process. For instance, for smaller document sizes the difference tends to be around 5%, with differences getting somewhat larger for larger document sizes. However, we note that in all such experiments, the largest observed difference is under 15%.

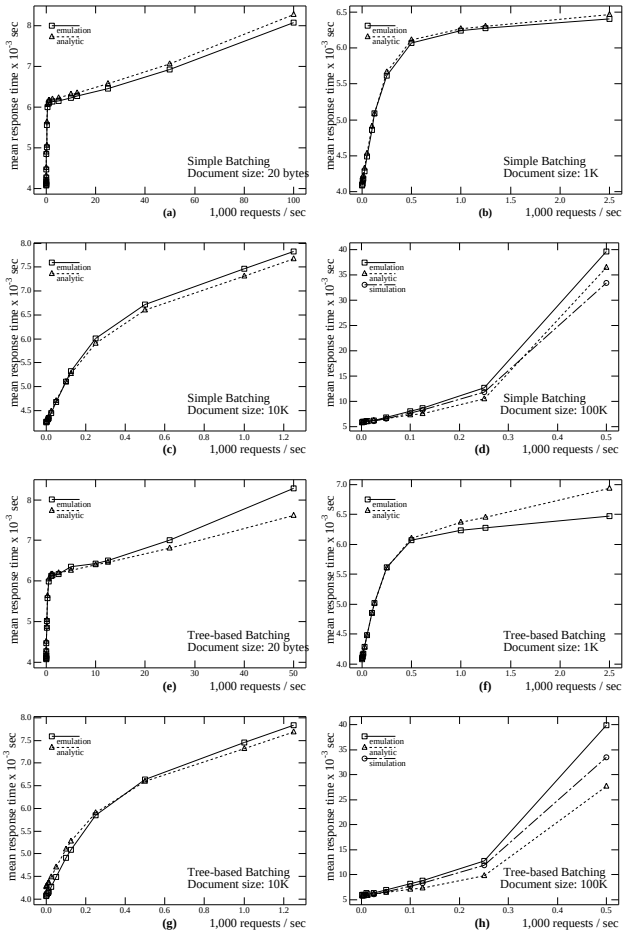

Figure 7. Mean Response Time Validation of the Analytic Model against Emulation for Simple and Tree-based Batching Schemes.

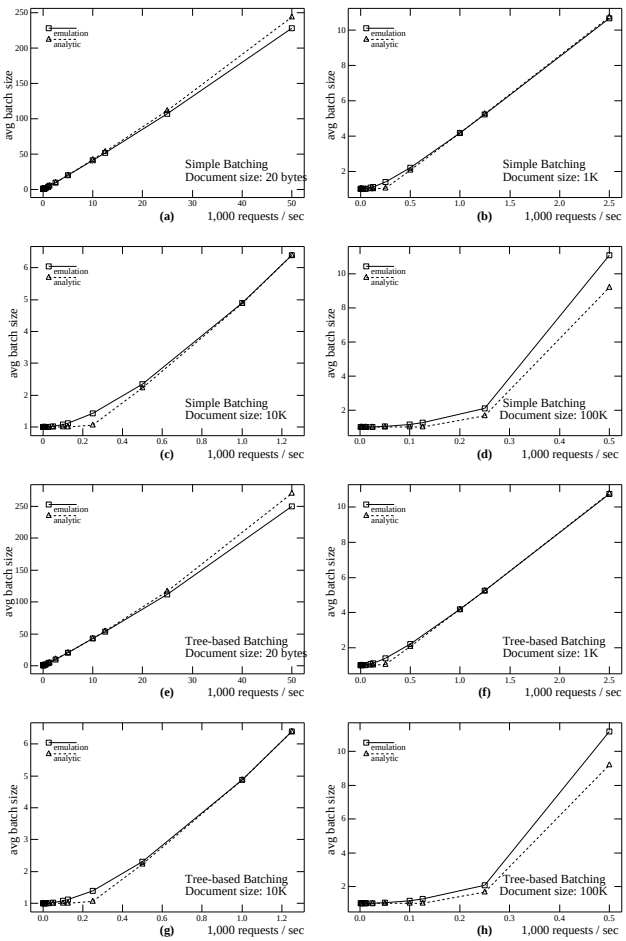

Figure 8. Mean Batch Size Validation of the Analytic Model against Emulation for Simple and Tree-based Batching Schemes.

In summary, as can be seen in all these figures, analytical results match emulation results closely for document sizes $\leq 10K$. These sizes suffice for applications, such as the ones mentioned in [2], which require secure timestamps and which motivated our work. Given the goodness of our analytical results, below we use analytical models to evaluate performance of the batching digital signature schemes.

3.3 Performance Evaluation Study

In all analytical results presented here we used $t_{ds} = 0.0041$ seconds, as measured by the OpenSSL benchmark. Based on Eq. (3), a non-batched system becomes unstable when the arrival rate exceeds $1/0.0041 = 244$ requests/sec for small documents ($\leq$ 10KB). Figures 10(a)-(d) and 9(c) show that the response time curve has a convex shape for such a system. Comparing that with Figures 7(a)-(c) and 7(e)-(g), the response time of a batched system behaves nicely even at very high loads for small documents. Note that the arrival rates in Figure 10 are given in units of 1/sec while the arrival rates in Figure 7 are given in units of 1000/sec. Also note that the stability conditions, for the batched and the non-batched schemes, given in Eq. (3), also illustrate a similar comparison as Figures 7 and 10. That is, the stability of the non-batched system is a function of t_{ds} while the stability of the batch-based systems is not.

For medium size documents (i.e., 100KB), as shown in Figures 7(d) and 7(h), the advantages of batching start to diminish. For large documents (i.e., 1MB), as shown in Figures 9(c)-(e), batching does not improve performance (perhaps only slightly in some cases). However, it does not reduce performance either.

Let us now look at batch sizes. For large documents, all systems start to perform poorly at small arrival rates. Since arrival rates are small, there is very little opportunity to batch, and therefore, average batch sizes are small, as shown in Figures 9(a)-(b). For small documents, Figures 8(a)-(c) and 8(e)-(g) show that the average batch size

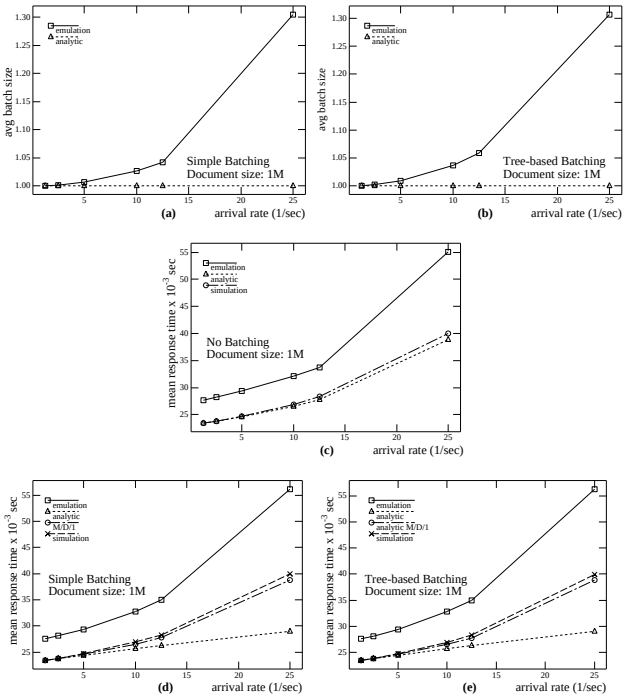

Figure 9. Validation of the Analytic Model against Emulation for Large Documents.

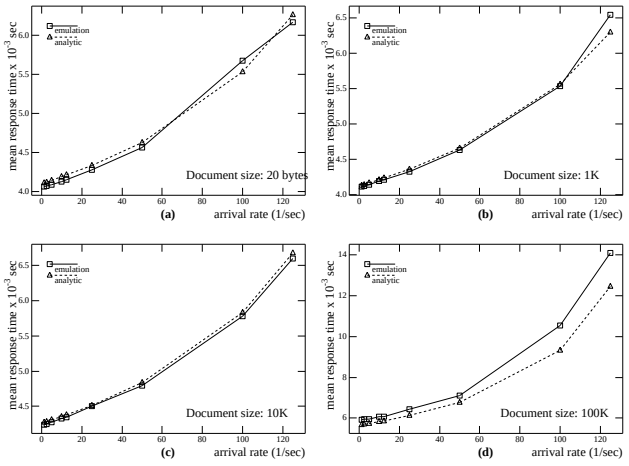

Figure 10. Mean Response Time Validation of a Simple Analytic Model against Emulation for the No Batching Case.

increases almost linearly with the arrival rate. This is due to the fact that the service time is almost constant in these systems. This supports our earlier observations about the performance of the various schemes.

Lastly, Figure 11 compares the network overhead of the batching schemes with the non-batched scheme. The vertical axis in each graph is the total number of bytes sent to the clients normalized by the total number of bytes sent to the clients if no batching is used. It is clear from these graphs that the Tree-based Batching scheme results in considerably less network overhead as compared with the Simple Batching scheme. For a given batch size B, the difference in computation time between the Tree-based Batching scheme and the Simple Batching scheme is simply $(B-1) \times t_h^s$, (refer to Eqs. (1) and (2)), where t_h^s is typically on the order of a few microseconds. It should be clear that Tree-based Batching has considerable advantages but costs very little.

4 Related Work

We are mainly concerned with *systems* and *performance* issues in producing digital signatures under high loads in the context of trusted servers. By that we mean that we explore performance improvements through the use of previously proposed batching schemes for a gated server under *standard* cryptographic techniques, i.e., those in [13, 16] using readily available software (e.g., OpenSSL [17]). There has been some work in the cryptography literature on batch decryption and batch verification schemes, in the context of public-key cryptographic systems, e.g., [1, 5, 14]. Most of these proposals require modifications to cryptographic algorithms. Regarding batch signing, Merkle first introduced the idea of *authentication trees* in [11] (although it was developed for an alternate, non-public-key-based, cryptographic system). This is what we used in Section 2.2. In [12], Pavlovski and Boyd showed that the tree-based batching scheme introduced in [11] has security characteristics identical to a scheme without batching, and that a binary tree structure has minimum residue.

In [6], Gennaro et al. studied the case where a single receiver is to receive a stream of blocks online and each block needs to be authenticated. They mentioned that hashes of all blocks can be put in a table and the table can be digitally signed and sent to the receiver. The Simple Batching scheme presented in Section 2.1 is similar, but in our case, each block is sent to a different receiver.

Since the slowness of digital signatures mainly stems from the high cost of modular arithmetic, an alternative approach is the so-called *one-time signatures* used in *secret-key* (or symmetric) cryptographic systems, e.g., [3, 9, 10]. Although one-time signatures are very fast to compute, this approach requires a large number of keys to be generated, managed, and distributed (since a signature can only be used once). Therefore, they are not widely used in practice. Another approach of mixing private-key digital signatures and one-time signatures also exists [4]. It has some of the same drawbacks as one-time signatures.

In all the work mentioned above, some have given complexity analysis of batching schemes, but, to the best of our knowledge, none have considered performance modeling and analysis of an online system using batch-based digital signatures. Given the applications in Section 1, there is a need for significant performance improvement of digital

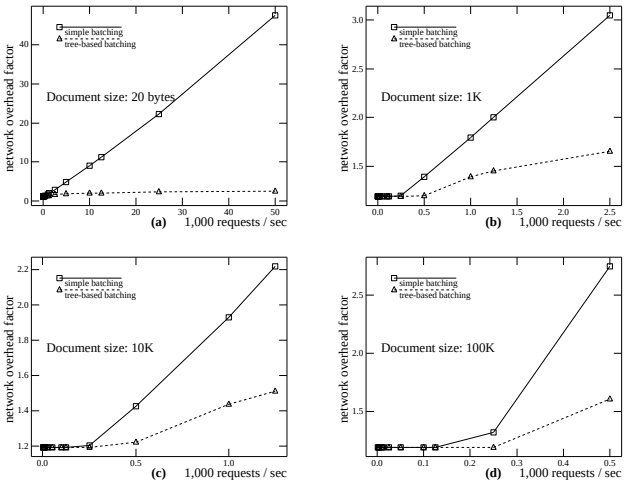

Figure 11. Normalized Network Overhead of Batching Schemes.

signatures under high loads in a system setting. This leads to a need for analytical models and performance evaluation techniques which can facilitate capacity planning and system sizing of such systems. Hence, performance modeling and evaluation of online systems using batch-based digital signatures is the focus and the contribution of this paper.

5 Conclusions

We developed an analytical model for online batch-based digital signature schemes for Internet servers, validated this model against emulations and simulations, and used it to compare performance of the batching schemes against a non-batched system. Using this model, we have shown that significant computational benefits can be obtained from batching schemes without significant increases in the amount of additional information that needs to be sent to the clients.

We have also established stability conditions for the batching schemes. From the stability conditions, it is fairly easy to see that as the document sizes grow, the performance of the gated server will be limited by hash functions calculations, and the benefit of batching will diminish. However, for applications such as the ones mentioned in [2] which require secure timestamps, batch signing with gated service can relieve the CPU bottleneck at the server.

References

[1] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. *Advances in Cryptology – Eurocrypt 98 Proceedings, Volume 1403 of Lecture Notes in Computer Science, Springer Verlag*, 1998.

[2] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A secure and scalable wide-area upload service. In *Proceedings of the 2nd International Conference on Internet Computing, Volume 2*, pages 733–739, June 2001.

[3] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[4] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In G. Brassard, editor, *Proceedings of CRYPTO'89*, pages 263–277. Springer-Verlag, 1990.

[5] A. Fiat. Batch RSA. pages 175–185, August, 1989.

[6] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Proceedings of CRYPTO'97*, pages 180–197, Santa Barbara, CA, August 1997.

[7] L. Golubchik. Scalable data collection for Internet-based Digital Government applications. *To appear in Advances in Digital Government: Systems, Human Factors, and Policy*.

[8] L. Kleinrock. *Queueing Systems, Volume I*. Wiley-Interscience, 1975.

[9] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[10] R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Proceedings of CRYPTO'87*, pages 369–378. Springer-Verlag, 1988.

[11] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proceedings of CRYPTO'89*, pages 218–238. Springer-Verlag, 1989.

[12] C. Pavlovski and C. Boyd. Efficient batch signature generation using tree structures. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99)*, pages 70–77, City University of Hong Kong Press, 1999.

[13] B. Schneier. *Applied Cryptography, Second Edition*. Wiley, 1996.

[14] H. Shacham and D. Boneh. Improving SSL handshake performance via batching. *RSA 2001, Volume 2020 of Lecture Notes in Computer Science, Springer Verlag*, pages 28–43, 2001.

[15] Mesquite Software. *CSIM18*. www.mesquite.com.

[16] W. Stallings. *Cryptography and Network Security: Principles and Practice, 2nd Edition*. Prentice Hall, 1999.

[17] E. A. Young. *OpenSSL: The Open Source Toolkit for SSL/TLS*. www.openssl.org.