A PERFORMANCE MODELLING OF WEB SERVICES SECURITY

Kezhe Tang, David Levy
School of Electrical and Information Engineering, University of Sydney, Australia

Shiping Chen, John Zic, Bo Yan
Networking Technologies Laboratory, CSIRO ICT Centre, Australia

Keywords: Web Service Security, WSS, Performance Modelling, SOAP Encryption, SOAP Signature.

Abstract: While Web Services Security (WSS) enhances the security of web services, it may also introduce additional performance overheads to standard web services due to additional CPU processing and larger message sizes. In this paper, we present a simple performance model for WSS. Based on the observations of WSS performance in our previous work, we extend a web service performance model by modelling WSS extra security operations and increased messages sizes into the existing model. As fitting the parameters on one testing environment, we validate our performance model on another different environment with different messages sizes and WSS security policies. Our testing results show that our performance model is valid and can be used to predicate the performance of web services with a variety of WSS configurations.

1 INTRODUCTION

Web services provide a loosely coupled architecture for building distributed systems with universal interoperability. It uses XML to pack data into XML messages defined by SOAP (Simple Object Access Protocol) and also uses XML to describe the data types and services in the SOAP message, called WSDL (Web Service Description Language). With web services, applications owned by different organizations can be easily integrated; even if they are developed in different programming languages and deployed on different platforms (Middleware/OS). As a result, web services have been widely adopted in the industry as a standard platform-independent middleware technology. (Tang, Chen, Levy, Zic and Yan, 2006)

Since SOAP itself does not provide secure transmission protocol for messages, it brings high risks to both sides of the message exchanger. Although traditional security technologies such as SSL and HTTPS can partially resolve this problem by encrypting messages transferred between two points (Booth, Haas, McCabe and etc, 2004), these point-to-point transport-layer security technologies cannot insure end-to-end security along the entire path from client to a web service in a complicated multi-tiers distributed system. Furthermore, these point-to-point security technologies are all based on a specific transport protocol/layer, such as TCP/IP for SSL and HTTP for HTTPS. Since SOAP is a transport-independent messaging protocol for web services, the capacity and application of web services would be limited if its security relies on these transport-dependent technologies. As a result, OASIS developed Web Services Security (WSS) specification (Web Services Security: SOAP Message Security 1.0, 2004) to provide message-level protection between two ends (clients and web services) through message integrity, message confidentiality and message authentications. WSS makes use of SOAP’s composable and extendable architecture by embedding security-related information (security token, signatures etc.) in the SOAP header without affecting the data stored in the SOAP’s body (but maybe encrypted/signed). This design allows WSS to integrate with SOAP as a plug-in and still retain SOAP’s compositability and extensibility for other purposes. Today more and more web services products are beginning to support the WSS standard (Web Services Security: SOAP...

While WSS enhances the security of web services, people may be concerned with its performance overheads. The overheads can come from: (a) extra CPU times to process WSS-related elements/operations at both client and services ends; (b) longer networking times to transport larger SOAP messages due to additional WSS contents. (Tang, Chen, Levy, Zic and Yan, 2006)

In our previous paper (Tang, Chen, Levy, Zic and Yan, 2006), we evaluated the performance of WSS by benchmarking a web service with and without applying the WSS basic security policies, i.e. encryption, signature, and authentication, and their combinations. We observed that both encryption and signature added significant performance overheads to web services, as there are little performance differences between using user names and X509 certificates. These observations motivate and guide us to develop a simple performance model for WSS.

In this paper, we present the development and validation of the simple WSS performance model. Based on the observations in our previous paper (Tang, Chen, Levy, Zic and Yan, 2006), we extend the existing web services performance model (Chen, Yan, Zic, Liu and Ng, 2006) by adding the extra overhead for each basic WSS security operations into the performance model. As fitting the parameters on one testing environment, we validate our performance model on another different environment with different messages sizes and WSS security policies. Our testing results show that our performance model is valid and can be used to predicate the performance of web services with a variety of WSS configurations.

The rest of this paper is organized as follows: Section 2 gives an overview of WSS and introduction to the web services performance modelling in (Chen, Yan, Zic, Liu and Ng, 2006). Section 3 discusses how to extend the existing web services performance model for WSS. In Section 4, the benchmark and approaches used for fitting the parameters in our performance are described in Section 4. We also discuss some observations found during the tests in Section 4. We present the results of the validation in Section 5 and conclude in Section 6.

2 BACKGROUND

2.1 SOAP vs. WSS

SOAP is the core messaging protocol for web services. A SOAP message is constructed as an *envelope*, which consists of a *header* and a *body*. While the *body* is mandatory and usually is used to carry application-level data, the *header* provides a flexible mechanism as an option to compose any schemas for extensions. One of the OASIS standards for Web Service Security, WS-Security, leverages this flexibility to provide security mechanisms that enhance the message integrity and message confidentiality. For example, it enables security tokens, which carry security credentials for authentication, to be attached to the message and specify the manner of which the binary tokens are encoded. (Web Services Security: SOAP Message Security 1.0, 2004)

By implementing XML Encryption and XML Digital Signature in association with security tokens, WSS keeps the sensitive portions of message confidential from intermediaries and guarantees the message integrity while the message is on wire (XML Encryption Syntax and Processing) (XML Signature Syntax and Processing). Figure 1 (a) lists a plain SOAP message from a ‘CustomerService’ web service, while the SOAP message in Figure 1 (b) is captured from the same web service but deployed with WSS Encryption policy. It can be seen that the <wsse: Security> element and its descendants in the encrypted message make the SOAP message much larger in size than the original message.

2.2 Performance Modelling of Web Services

The work done by Dr. Chen and etc (Chen, Yan, Zic, Liu and Ng, 2006) is a study on web services performance by evaluating the current implementations of web services and comparing them with a number of alternative technologies. A performance model of Web Services is also introduced to estimate the web services latencies (Chen, Yan, Zic, Liu and Ng, 2006).

According to the Modelling analysis in (Chen, Yan, Zic, Liu and Ng, 2006), the performance of web service is modelled as follows:
Latency = T_{msgProc} + T_{msgTrans} + T_{Synch} + T_{app} \quad (1)

Where:
- $T_{msgProc}$ represents the total cost of processing the messages, including coding/encoding, security checking, data type marshalling;
- $T_{msgTrans}$ represents the total cost to transfer a specific amount of messages over network;
- $T_{Synch}$ represents the overhead of the extra synchronization required by protocols;
- $T_{app}$ represents the time spent in business logic at application level.

Following assumptions are made for simplicity:
- The transmission speeds of data on wire are even on the whole path and approximated as the light speed in glass.
- All network devices (routers/switches) involved into the transmission have the comparable capacity and thus no network overflow/retransmission occurs at any point.
- The message complexity is proportional to message size, and thus overhead of processing a message can be modelled by message sizes.

Based on the above assumptions, the three terms in (1) are modelled in (Chen, Yan, Zic, Liu and Ng, 2006) as follows:

$$T_{msgProc} = \sum_{i=1}^{w} \sum_{j=1}^{n} (\alpha_j + \beta_j M_i) \frac{refP_j}{P_j} \quad (2)$$

Where:
- $w$ is the number of transits between client and server, e.g. $w=1$ for one way sending and $w=2$ for an normal request/response call;
- $refP_j$ specifies the CPU capacity of the reference platform;
- $P_j$ represents the CPU capacity of the machine where client/server is deployed;
- $\alpha_j$ represents an identical inherent overhead of processing/parsing a message for client/server built on a specific middleware running on the reference platform;
- $\beta_j$ represents the overhead of processing/parsing an unit amount of messages (say 1KB) for the same middleware $j$ also running on the reference platform;
- $n$ the total number of network devices involved.

$$T_{msgTrans} = \sum_{i=1}^{w} \sum_{j=1}^{n} (\tau_j + \frac{M_i W_j}{N_j}) \frac{D_i}{L} + W \quad (3)$$
The actual message size transferred on wire;
• \( N \) the bandwidth of the network devices;
• \( \tau \) message routing/switching delay at each network device;
• \( D \) the distance between the client and server;
• \( L \) the speed of light in glass, i.e. \( L = 200,000 \) km/s;
• \( W \) the delay on the core WAN;
\[
T_{\text{sync}} = \sum_{j=1}^{s} \left( \sum_{i=1}^{n} (\tau_j + \frac{m_i}{N_j}) + \frac{D}{L} + W \right) \quad (4)
\]
\( s \) is the number of synchronizations occurred during messaging
• \( m \) the message size for each synchronization
• \( W_s \) the TPC window size ranging from 16K to 64K.

3 WSS PERFORMANCE MODELLING

WSS is an additional security deployment that is added on a web service, by which the SOAP messages are encrypted/signed, transmitted to the recipient and decrypted/verified. Likewise, the performance of WSS can be regarded as the performance of a web service plus additional time cost on SOAP message transmission and additional time cost on processing the security content of the SOAP message. Thus, the Performance Model of Web Services from (Chen, Yan, Zic, Liu and Ng, 2006) is a good model to be based on for modelling performance of WSS.

We analysed the performance of WSS and developed a Performance Model for WSS by extending the Performance Model of Web Services from (Chen, Yan, Zic, Liu and Ng, 2006) for estimating the latency of a web service with a specific WSS setting in a certain hardware and software environment.

Figure 2 illustrates a web service call secured by WSS. There are eight major processes taken places on client and web service machine. They are:

• \( Pr1 \): The computational procedures to encode the data object to generate a plain SOAP request.
• \( Pr2 \): The computational procedures to encrypt and/or sign a plain SOAP request.
• \( Pr3 \): The computational procedures to decrypt and/or verify the signature of an encrypted and/or signed SOAP request.
• \( Pr4 \): The computational procedures to decode a plain SOAP request.
• \( Pr5 \): The computational procedures to encode the data object to generate a plain SOAP response.
• \( Pr6 \): The computational procedures to encrypt and/or sign a plain SOAP response.
• \( Pr7 \): The computational procedures to decrypt and/or verify the signature of an encrypted and/or signed SOAP response.
• \( Pr8 \): The computational procedures to decode a plain SOAP response.

Pr1, Pr4, Pr5 and Pr8 are the processes of a web service call without WSS, while Pr2, Pr3, Pr6 and Pr7 are the additional security related (encryption, decryption, signing or verification) processes that are required by WSS deployments. Thus, we can model the \( T_{\text{msgProc}} \) of the WSS Performance by

![Figure 2: A WSS secured web service call.](image-url)
adding additional time cost on the security related processes to the term (2).

The performance modelling of WSS Encryption and WSS Signature combination (WSS Encryption+Signature) is inspired by our previous work in 0. Figure 3 shows the LIP (Latency Increment Percentage) of WSS Encryption, Signature and Encryption+Signature from 0. The LIP is defined as a metric to evaluate the performance overhead for a specific WSS deployment:

\[
LIP = \frac{L_{\text{WSSDeployment}} - L_{\text{NonWSS}}}{L_{\text{NonWSS}}} \times 100\%
\]

Where:
- \(L_{\text{WSSDeployment}}\) is the latency of the web service with a specific type of WSS deployment, e.g. \(L_{\text{WSSEncryption}}\) for encryption.
- \(L_{\text{NonWSS}}\) is the latency of the web service without any WSS deployment.

![Figure 3: LIP of WSS Encryption, Signature and Encryption+Signature.](image)

By comparing the LIP of three WSS deployments, we observed that the sum of Encryption LIP and Signature LIP is roughly equals to the Encryption+Signature LIP. Thus, we model the additional time cost on encryption+signature/decryption+verification combination as the sum of the additional time cost on encryption/decryption and the additional time cost on signature/verification.

In order to generalize the model, we introduce \(\alpha_{\sec}\) and \(\beta_{\sec}\) to represent the total processing time on WSS as following:

\[
T_{\text{msgProc}} = \sum_{j=1}^{N} \sum_{i=1}^{M} (\alpha_j + \alpha_{\sec} + (\beta_j + \beta_{\sec}) \frac{P_i}{P_j}) \tag{5}
\]

Where:
- \(\alpha_{\sec} = \alpha_{\text{enc}} + \alpha_{\text{dec}} + \alpha_{\text{sig}} + \alpha_{\text{veri}}\)
- \(\beta_{\sec} = \beta_{\text{enc}} + \beta_{\text{dec}} + \beta_{\text{sig}} + \beta_{\text{veri}}\)
- \(\alpha_{\text{enc}}, \alpha_{\text{dec}}, \alpha_{\text{sig}}\) and \(\alpha_{\text{veri}}\) represent the additional identical inherent overhead of encrypting, decrypting, signing and verifying a SOAP message for client/server running on the reference platform respectively;
- \(\beta_{\text{enc}}, \beta_{\text{dec}}, \beta_{\text{sig}}\) and \(\beta_{\text{veri}}\) represent the additional identical inherent overhead of encrypting, decrypting, signing and verifying an unit amount of a SOAP message for client/server running on the reference platform respectively;

4 MODELLING TESTS

This Section introduces the benchmark for the modelling tests and describes how the parameters are fit into the performance model. Some observations on the performance of WSS are also discovered during the parameter fitting.

4.1 Benchmark

We reuse the benchmark in (Tang, Chen, Levy, Zic and Yan, 2006) for the modelling tests to fit the parameters. In addition to the benchmark in (Tang, Chen, Levy, Zic and Yan, 2006), we also add a test driver for testing CPU load to both client and server side.

As shown in Figure 4, the client application sends a SOAP request for an array of customer records from the web service on the server machine. The web service receives the request and generates an array of random objects containing customer records. The array is encapsulated in the SOAP response and the SOAP response is processed (encrypted / signed) according to the WSS policy that is deployed on the web service. The test drivers measure the average value of the latency of the round-trip web service calls on the client machine and measure the average value of CPU load during the calls on both machines.
4.2 Fitting \( \alpha_{sec} \) and \( \beta_{sec} \)

\( \alpha_{sec} \) and \( \beta_{sec} \) are the two parameters related to time spent on processing security contents of the SOAP message. They include different elements in different security deployment. For example, in the case of WSS encryption, \( \alpha_{sec} \) and \( \beta_{sec} \) is one pair of \( \alpha_{enc} \) and \( \beta_{enc} \) on the machine where encryption of the message happens while they are a pair of \( \alpha_{dec} \) and \( \beta_{dec} \) on the machine where decryption of the message happens. However, in the case of WSS Encryption+Signature combination, \( \alpha_{sec} \) is \( \alpha_{enc} + \alpha_{sig} \) and \( \beta_{sec} \) is \( \beta_{enc} + \beta_{sig} \) on the machine where encryption and signature of the message happens while \( \alpha_{sec} \) is \( \alpha_{enc} + \alpha_{veri} \) and \( \beta_{sec} \) is \( \beta_{enc} + \beta_{veri} \) on the machine where decryption and signature verification of the message happens. For the sake of simplicity and convenience in description, we can call \( \alpha_{sec}, \alpha_{enc}, \alpha_{sig} \) and \( \alpha_{veri} \) as \( \alpha^* \) and \( \beta_{sec}, \beta_{enc}, \beta_{sig} \) and \( \beta_{veri} \) as \( \beta^* \) in the rest of the paper.

In order to fit every \( \alpha^* \) and \( \beta^* \) for each WSS deployment, we need to isolate them from other objects in term (5):

- \( M_i, refP_j \) and \( P_j \) are the constants that we are able to obtain from each web service call.
- \( T_{msgProc} \) for each test can be calculated as following by applying Utility Law:

\[
T_{msgProc} = \frac{\lambda}{\text{Throughput}}
\]

\[
\text{Throughput} = \frac{1}{\text{Latency}}
\]

\( \lambda \) is the CPU load of the machine running the web service or the client application.

\( \alpha_1 \) and \( \beta_1 \), or we can call them \( \alpha_{soap} \) and \( \beta_{soap} \), are fitted in the same way described in (Tang, Chen, Levy, Zic and Yan, 2006) by running the modelling tests on a web service without any WSS deployment.

Therefore, a pair of \( \lambda \) and \( \text{Latency} \) for each WSS deployment needs to be tested to obtain \( T_{msgProc} \) for fitting \( \alpha^* \) and \( \beta^* \).

We run the modelling tests to fit \( \alpha^* \) and \( \beta^* \) on the benchmark described in Section 4.1. The results of the tests are listed in Table 1.

<table>
<thead>
<tr>
<th>Number of Customers</th>
<th>T10</th>
<th>T100</th>
<th>T500</th>
<th>T1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-WSS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mi (byte)</td>
<td>3.34</td>
<td>26.04</td>
<td>65.04</td>
<td>130.26</td>
</tr>
<tr>
<td>Latency (sec)</td>
<td>7.43</td>
<td>20.15</td>
<td>35.92</td>
<td>116.38</td>
</tr>
<tr>
<td>%</td>
<td>22.9</td>
<td>26.2</td>
<td>27.4</td>
<td>27.9</td>
</tr>
<tr>
<td>Sampler</td>
<td>30.6</td>
<td>30.4</td>
<td>31.3</td>
<td>31.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entrie Username</th>
<th>T10</th>
<th>T100</th>
<th>T500</th>
<th>T1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mi (byte)</td>
<td>3.34</td>
<td>26.20</td>
<td>65.04</td>
<td>130.26</td>
</tr>
<tr>
<td>Latency (sec)</td>
<td>32.5</td>
<td>197.3</td>
<td>259.0</td>
<td>466.5</td>
</tr>
<tr>
<td>%</td>
<td>25.9</td>
<td>7.7</td>
<td>6.0</td>
<td>7.9</td>
</tr>
<tr>
<td>Sampler</td>
<td>27.9</td>
<td>9.6</td>
<td>7.1</td>
<td>8.7</td>
</tr>
</tbody>
</table>

With the tests results, we can calculate the \( T_{msgProc} \) for each test case to work out \( \alpha^* \) and \( \beta^* \). The results of fitting \( \alpha^* \) and \( \beta^* \) are listed in Table 2.

<table>
<thead>
<tr>
<th>Non-WSS</th>
<th>( \alpha )</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_{soap} )</td>
<td>1.0032</td>
<td>0.0003</td>
</tr>
<tr>
<td>( \beta_{soap} )</td>
<td>0.0003</td>
<td>1.0006</td>
</tr>
<tr>
<td>( \alpha_{WSS+Sign} )</td>
<td>0.7850</td>
<td>0.0002</td>
</tr>
<tr>
<td>( \beta_{WSS+Sign} )</td>
<td>0.3689</td>
<td>0.0020</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSS + Sign Username</th>
<th>WSS + Sign Username</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_{WSS+Sign} )</td>
<td>1.1165</td>
</tr>
<tr>
<td>( \alpha_{WSS+Sign} )</td>
<td>1.4759</td>
</tr>
</tbody>
</table>
4.3 Encryption vs. Decryption and Signature vs. Verification

As we have tested \( \lambda \) and \( \text{Latency} \) for each WSS deployment in the parameter fitting tests of \( \alpha^* \) and \( \beta^* \), the \( T_{msgProc} \) of each test allow us to observe the differences in performance between encryption and decryption of a SOAP message, and also between signature and verification.

As illustrated in Error! Reference source not found., based on encryption algorithm RSA1.5 and signature algorithm RSA-SHA1 used in our tests, we can make the following observations,

- Encryption takes more time than decryption for WSS with Username token.
- In the cases using X509 certificate token, when the data size of the message is less than the turning point (83071 bytes), encryption is faster than decryption; while when data size of the message is larger than the turning point, encryption is slower than decryption.
- Signature generation is faster than verification of the signature in both of the cases of Username and X509 token.

4.4 Username vs. X509

According to the results of fitting \( \alpha^* \) and \( \beta^* \) in Table 2, the following observations can be made,

- The \( \alpha^* \) in the performance model of WSS with Username token is always smaller than corresponding \( \alpha^* \) of WSS with X509.
- The \( \beta^* \) in the performance model of WSS with Username token is always the same as corresponding \( \beta^* \) of WSS with X509.

Thus, we can make the following conclusions,

- The difference in \( T_{msgProc} \) of WSS for a certain message size between using Username and X509 does not vary.
- The performance gap of WSS between using Username and X509 might be the additional time required for certificate-related operations, such as, time spent on retrieving an X509 certificate from the system certificate store.

5 MODEL VALIDATION

We run a few latency tests in a different hardware and network environment to validate the WSS Performance Model, which are listed as follows:

- Client
  - CPU: 1.7 GHz Intel Celeron
  - Memory: 256MB
- Web Service
  - CPU: 3.00GHz Intel Pentium
  - Memory: 1.00GB
- LAN: 10Mbps switched

We run tests with three different message sizes:

- Small: 10 customer records in the SOAP message
- Medium: 50 customer records in the SOAP message

![Figure 5: TmsgProc of encryption/signature and decryption/verification.](image-url)
Large: 100 customer records in the SOAP message.

Predicated results of Latency are calculated from the extended model of web service and compared with actual testing results after the validation tests. Both of the results are shown in figure 6.

As shown in Figure 6, our model maintains valid on WSS Encryption with small, medium and large sized messages. The validation results are also positive on WSS Signature and WSS Encryption+Signature combination with small and medium sized messages. However, there is still space for improvements on the accuracy of the model with large sized message.

6 SUMMARY

In this paper, we developed a simple performance model for web services security. Based on the observations made in our previous paper, we extended our existing web services performance model by modelling the basic WSS security operations into the mode. We instanced our model by fitting the performance parameters on a testing environment and validated the model by using these parameters on another different testing environment. The testing results show that our model is able to provide approximate performance estimation for a web service with a variety of WSS configurations and message sizes. This WSS performance model can be used by web services architects and/or developers to evaluate the performance cost of applying WSS.