An Adaptive Approach to Optimizing Tradeoff between Service Performance and Security in Service-based Systems

Stephen S. Yau, Yin Yin and Ho An
Information Assurance Center, and
School of Computing, Informatics, and Decision Systems Engineering

Arizona State University, Tempe, Arizona, USA
{yau, yin.yin, ho.an}@asu.edu

ABSTRACT:

The message-based communication among services in Service-based Systems (SBS) is vulnerable to various security attacks, and has to be well protected by security mechanisms, which may affect the service performance due to available system resources. In this paper, an adaptive approach is presented to optimizing the tradeoff between service performance and security according to the SBS users' requirements and preferences on performance and security. This adaptive approach is based on a tradeoff algorithm with quantitative performance and security metrics and the tradeoff objective function. An SBS example with a security service and a traffic service is used to illustrate the approach.

KEY WORDS:
Tradeoff approach, service-based systems, service performance, security, and quantitative metrics

1. Introduction

Service-oriented architecture (SOA) facilitates dynamic organization of needed services to compose a system for performing application functions, where each service communicates with other services through messages. Distributed systems based on SOA are called *service-base systems (SBS)*. Although message-based communication makes SBS easier to use services, the protection of the communication messages is a serious security concern of SBS.

Current research on SOA security mainly focuses on the protection of services from malicious consumers through authentication and authorization (Godik & Moses, 2003) (Hallam-Baker & Mysore, 2005) (Bajaj, et al., 2004) (Pashalidis & Mitchell, 2003), and the protection of messages through XML encryption and signature (Mactaggart, 2001). However, the impact of these security mechanisms on the systems' performance has not been well addressed. Security requirements are often in contrast to other performance requirements, like timeliness and throughput, which are usually determined by the availability of system resources. For example, when a VoIP service needs to provide secure voice communication, it sends the voice data to an encryption service, which will encrypt the voice data and then forward the encrypted voice data to the users of VoIP service. The encryption service increases the delay of VoIP service in two ways. First, the VoIP users need to wait for the encryption service to encrypt voice data, which increases the delay. Second, if the VoIP service and the encryption service are hosted on the same server, the encryption service may compete with the VoIP service for system resources, such as CPU and memory, and causes longer delay. Because VoIP service has strict timeliness requirements (Cheng & Li, 2000), without careful control on the encryption service's negative impact on the VoIP service's delay, the VoIP service may either do not use the encryption service at all or became useless due to unaffordable delay caused by the encryption service.

In this paper, we will present an adaptive approach to optimizing tradeoff between service performance and security in order to satisfy service performance and security requirements in service-based systems simultaneously (Yau, Ye, Sarjoughian, & Huang, 2008). Typically, the tradeoff between performance and security is implemented through controlling the number of system resources allocated for performance and security. This approach needs to model all system resources and control resource allocation strategy, which is usually controlled by operating systems. Our approach to the tradeoff is to adjust the security parameters, such as key length and encrypting percentage, which are much easier to control. The major distinction between our approach and existing approaches (Lie & Satyanarayanan, 2007) (Yurcik, Woolam, Hellings, Khan, & Thuraisingham, 2007) (Lu, Lu, Abdelzaher, Stankovic, & Son, 2006) (Kang & Son, 2006) (Son, Zimmerman, & Hansson, 2000) (Spyropoulou, Levin, & Irvine, 2000) (Yau, Yan, & Huang, 2007) is that our approach can achieve the best tradeoff by minimizing a tradeoff objective function developed from service performance and security metrics, instead of intuitively trying all possible combinations of security parameters and monitoring the resulting performance and security until the desirable tradeoff is reached. Hence, our approach can achieve the best tradeoff fast and does not need to change security parameters frequently. Our approach is based on a model with a set of metrics to quantitatively measure performance and security and a tradeoff objective function, which enable us to consider performance and security simultaneously. Our approach will be illustrated using a SBS composed of a security service that provides AES encryption for confidentiality and a traffic service that simulates communication in applications.

This paper is organized as follows. The background and related work will be summarized in Section 2. The overall adaptive approach will be presented in Section 3, which is elaborated in Section 4 presenting the quantitative metrics for service performance and security, and Section 5 presenting tradeoff model between service performance and security. The estimation of parameters with experiment results will be presented in Section 5. Conclusions and future work will be discussed in Section 6.

2. Related Work

With the increasingly diverse applications of service-based systems, it is important to investigate resource management for improving performance. Application level differentiated services (Eggert & Heidemann, 1999) (Rao & Ramamurthy, 2001) allocate fewer system resources to normal users, and reserve more resources to premium users to guarantee good performance, when the system resources are fewer. Furthermore, when resources are seriously limited and even premium consumers' required performance couldn't be fully satisfied. Feedback controlled web services (Lu, Lu, Abdelzaher, Stankovic, & Son, 2006) (Abdelzaher, Stankovic, Lu, Zhang, & Lu, 2003) have been developed to adjust resource allocation to meet the most important performance requirement, such as the delay in real-time systems. These systems mainly focus on the tradeoff among various users' performance requirements or users' requirements on various performance aspects, but do not consider users' security requirements that also consume resources.

To control the tradeoff between performance and security, we first need to develop security metrics to compare the security strengths of different security mechanisms, especially the security mechanisms for the same security functionality. Existing qualitative security metrics (Son, Zimmerman, & Hansson, 2000) (Spyropoulou, Levin, & Irvine, 2000) classify security mechanisms to several discrete levels, such as low, medium, and high. Security mechanisms in the higher level can provide better protection than those in the lower level, but security mechanisms within the same security level are treated with them same security strength and

cannot be compared**.** Furthermore, qualitative security metrics are too coarse for fine control of the tradeoff between performance and security.

A quantitative security metric generates a security strength value for each security mechanism from its parameter configurations, and hence is more accurate and can compare the security strength of any two security mechanisms. However, the difference between the values generated by existing quantitative security metrics (Lie & Satyanarayanan, 2007) (Kang & Son, 2006) cannot correctly reflect the actual difference between the security strengths of corresponding security mechanisms because they only consider the key length and ignore other important factors, such as security algorithms, attacking approaches, and attackers' computing power. For example, symmetric encryption algorithms and asymmetric encryption algorithms have different requirements on the key length. To achieve the same security strength of the symmetric encryption algorithm *AES* with a *256*-bit key, the asymmetric encryption algorithm *RSA* needs a *15,360*-bit key, and the elliptic encryption needs a *571*-bit key (Lenstra & Verheul, 2001). Furthermore, the security strength of security mechanisms is not absolute, but relative to the capability of attackers, which is not considered in existing quantitative metrics. To achieve the same security strength, systems with more powerful attackers have to use longer keys than systems with weak attackers. Hence, the National Security Agency suggests that *128*-bit *AES* encryption is sufficient for SECRECT level data, but TOP SECRECT level data requires at least *192*-bit *AES* encryption (Hathaway, 2003).

Because it is difficult to improve encryption algorithms' efficiency, some special encryption algorithms named as selective encryption algorithms were developed to reduce encryption algorithms' negative impact on performance (Cheng & Li, 2000) (Droogenbroeck & Benedett, 2002) (Shi, Wang, & Bhargava, 1999) (Zeng & Lei, 2003). These selective encryption algorithms are based on the fact that encrypting partial information may be already sufficient to prevent attackers from knowing the entire information, especially for multimedia information. Hence, encrypting information as little as possible is used in order to significantly reduce the encryption's negative impact on performance. However, it is difficult to decide which partial information should be encrypted to provide sufficient security protection, and hence selective encryption algorithms so far developed are not for general applications, but for specific applications, such as image selective encryptions (Cheng & Li, 2000) (Droogenbroeck & Benedett, 2002) and video selective encryptions (Shi, Wang, & Bhargava, 1999) (Zeng & Lei, 2003).

3. Our Overall Approach

For an SBS with security mechanisms providing security protection, our approach to the adaptive tradeoff between service performance and security helps SBS developers to control these security mechanisms' negative impacts on the performance, and provide best user experience for the SBS users according to their requirements and preferences on performance and security. The SBS developers use our approach to can be summarized as follows:

Step 1. Specify security mechanisms' parameters as *security configuration vectors* discussed in Section 4.1.

Step 2. Generate security and performance metrics as discussed in Sections 4.2 and 4.3.

Step 3. Estimate the parameters for the generated security and performance metrics as presented in Sections 4.2 and 4.3.

Step 4. Develop the tradeoff objective function to optimize tradeoff between performance and security with the tradeoff strategies according to users' preferences on performance and security to be presented in Section 5.2.

Step 5. Use the tradeoff algorithm to be presented in Section 5 as follows: For each incoming service request with the performance and security requirements from users, use the tradeoff algorithm to check whether the user's minimum performance and security requirements can be satisfied with current system resources as discussed in Section 5.1. If both minimum performance and security requirements can be satisfied, the tradeoff algorithm determines the best parameters for security mechanisms' security configuration vectors. If both minimum performance and security requirements cannot be satisfied, the SBS rejects the user's service request

4. Security and Performance Metrics

Tradeoff between security and performance is to degrade security for better performance or vice verse. Hence, our tradeoff model should be able to measure and adjust the security and performance. For performance, there usually are common metrics, like measuring delay by observing when the packet in and when the packet out, and measuring traffic by counting the number of bits sent and received. Similar to performance, security can also be quantitatively measured by observing or counting certain system data. For example, the security can be measured through the number of vulnerabilities found by vulnerability scanners, the number of viruses detected by anti-virus software, or the number of intrusions detected by intrusion detection systems. However, these measurements only reflect the system's current status, but not the actual strength to resist attackers. A system with no intrusion detected only means that there is no attacker right now, but does not guarantee that the system will be secure under attacks.

Because security metric needs to consider both security mechanisms applied by the system and attackers' capability, it is challenging to generate quantitative metrics for security. In this section, we will first use the security configuration vectors (*SCV*) to specify the parameters of security mechanisms, and then estimate possible attackers' computing power and their attacking technologies. The quantitative security metric developed in this section measures security as the probability of protecting a packet from attackers' one attack attempt. Hence, a larger security value means that attackers need more effort to crack the protected packet, and hence more secure.

With the security metric based on *SCV*, our tradeoff model can adjust a security mechanism's strength and the consumed resources by adjusting the parameters in the corresponding *SCV*. For performance, the straightforward way to adjust the performance is to adjust resource allocation. However, it is difficult to control how much CPU, memory, bandwidth, or other system resources can be used by a specific service, which are usually controlled by the underlying operating systems. Hence, in this section, we will develop the quantitative performance metric also based on *SCV*. Then, by adjusting the parameters of *SCV*, our tradeoff model can adjust both performance and security simultaneously, and then make tradeoff between them.

4.1 Security Configuration Vector

The security configuration vector SCV is a set of security configuration parameters that determine security strength. In this paper, we define SCV as $\{F, A, l, p\}$ where F is the security functionality, A is the security algorithm, l is the key length, and p is the protection percentage.

F specifies the type of protection the security enforcing mechanism provides, like confidentiality, integrity, and non-repudiation. *A* is the algorithm name specifies which algorithms used by the security mechanism. For confidentiality, the security mechanism can use DES or AES, and a security mechanism can provide the integrity of the protected information with SHA-1 Hash or Digital Signature. *l* is another important factor that influences the security strength. A longer key provides higher security, but consumes more system resources. Hence, specifying key length in *SCV* gives the security mechanism more flexibility in the tradeoff between performance and security. *p* defines how much packets will be protected by *A*.

While cryptography has normally been applied to protect every bit of information, sometimes such extensive protection is unnecessary and unaffordable. Encrypting a part of packets, instead of encrypting all packets, will dramatically reduce the overhead caused by security mechanisms and still prevent attackers from learning the information. Some selective encryption algorithms have been proposed for specific applications, like image selective encryptions (Cheng & Li, 2000) (Droogenbroeck & Benedett, 2002) and video selective encryptions (Shi, Wang, & Bhargava, 1999) (Zeng & Lei, 2003). Because we are developing a tradeoff approach for general applications that have no information about the traffic content, and hence cannot figure out which part of the traffic is the most important and need protection, we implement the selective encryption as encrypting packets randomly with probability *p*.

In each SCV, one packet will be protected with probability *p*. That is, for a packet

- With probability *1-p*, the packet will not be protected and hence can be cracked with probability *1*.
- With probability *p*, the packet will be protected but can still be cracked with certain probability because there is no perfect protection.

For a security mechanism with a *SCV* = {*F, A, l, p*}, we define the *vulnerability function v*(*A, l*) of the *SCV* to measure the probability of cracking a packet protected by this security mechanism with one attack attempt, which varies with the design and implementation of the security mechanism and the evolution of attacking techniques. For *AES* encryption algorithm and brutal force attack, $v(AES, 128) = 2^{-127}$, $v(AES, 192) = 2^{-191}$, and $v(AES, 256) = 2^{-255}$. For *RSA* encryption algorithm and the general number field sieve attack (GNFS) (Pomerance, 1996), which is the known most efficient integer factoring algorithm for integers with more than *100* bits, the vulnerability functions are $v(RSA, 768) = 0.93 \times 10^{-23}$, $v(RSA, 1024) = 0.77 \times 10^{-26}$, and $v(RSA, 2048) = 0.65 \times 10^{-33}$.

The other consideration in the security metric is the attacker's computing power, which specifies how many times the attacker can attack the packet within one second (i.e., the attacking speed) using a given attacking approach. The attacker's computing power can be estimated based on the sensitivity of the protected information and the expected protection period. First, the computing power of potential attackers for military systems and commercial systems are different due to different sensitivity of the information. Second, if the security mechanism is handling information that should be protected for a very long period, the security mechanism needs to estimate the capability of attackers in the expected period.

4.2 Security Metric

Given the vulnerability function *v*(*A, l*) of a *SCV* and the attacker's computing power *c*, the probability of the attacker to successfully crack a protected packet in one second is *cv*(*A, l*). The

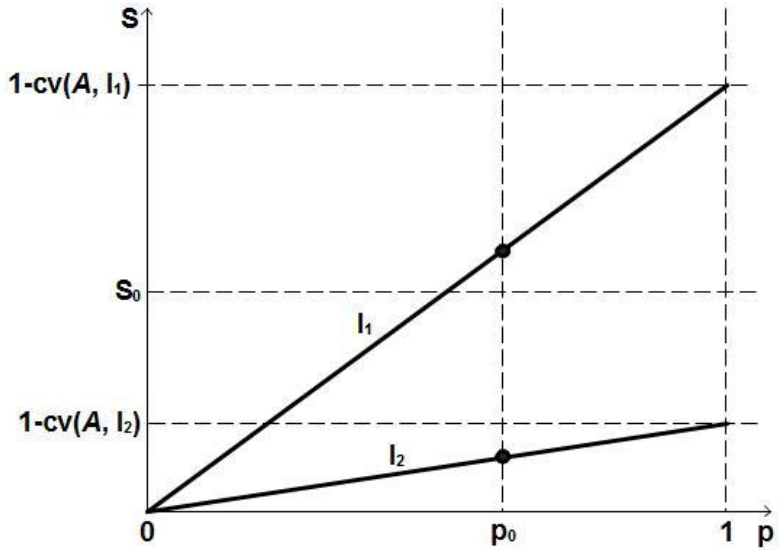

Figure 1. The relations between security *S* and protection percentage *p* for the same algorithm *A* with two different key lengths l_1 and l_2, where $l_1 > l_2$. S_0 is the minimum security requirement given by the user.

overall security *S*(*SCV*,*c*), which is defined as the probability that a packet protected by a security mechanism with *SCV* can resist an attacker with computing power *c* for one second, is given as follows:

$$S(SCV, c) = 1 - ((1-p) \times 1 + p \times cv(A, l)) = p(1 - cv(A, l)) \quad (1)$$

For the same security algorithm with two different key lengths $l_1 > l_2$, Figure 1 shows two relations between the security metric (1) and the protection percentage *p*. As shown in Figure 1, the same algorithm with longer l_1 can reach the same security level with a smaller protection percentage.

4.3 Performance Metric

Through observing or accounting system data, it is easy to measure performance, but difficult to adjust it because the resource allocation is usually controlled by the underlying operating systems. Now, we would like to develop a quantitative performance metric also from *SCV* like the security metric, which enable our approach to adjust both security and performance simultaneously through *SCV*.

Generally, adjusting the parameters in *SCV* to achieve better security will lead to allocating more resources for security enforcing mechanism and sacrifice the performance when resources are limited. However, the exact relation between performance and *SCV* depends on the specific security mechanisms and applications. For example, for an authentication mechanism that only authenticates users when they are login the system, its *SCV* has less impact on performance than that of an encryption mechanism that protects all traffics. Furthermore, for the same security mechanism, its impact on different performance aspects may also be different. For example, the above encryption mechanism's *SCV* has much impact on CPU usage and delay, but little impact on bandwidth when the encrypted packets have the same size of the original packets.

Without loss of generality, we use the encryption mechanism *AES* to show how to develop the metric to measure the *average delay D* with the traffic *t* and *SCV*={*F, A, l, p*}. We define *D* as the average time required by *AES* for accepting a packet, protecting the packet with the parameters specified in *SCV*, and returning the packet back. We define the *traffic t* as the number of packets handled by *AES* per second. There may be various metrics to define *AES*'s performance metric using *D*, *t* and *SCV*, but all these metrics should satisfy the following conditions:

- The delay *D* consists of the delay of receiving and sending packets, and the delay caused by protecting packets. While the first part of the delay involves all incoming packets, the second part of the delay only involves packets that are protected. Let the current traffic be *t* and the percentage of protected packets specified in *SCV* be *p*. The number of protected packet is *tp*.
- When *t=0*, *D=0* for any *SCV*. Hence, we have *D*(*SCV, 0*) = *0*.
- When *t* is small, the security mechanism can complete the processing of one packet before the next packet coming in. That is, the security mechanism has sufficient time to handle every packet. In this case, *D* is mainly determined by the *SCV*. Let T_1 be the traffic threshold under which the security enforcing mechanism can complete the processing of one packet before the next packet coming in. Then, when *t* is smaller than T_1, i.e., $0 < t < T_1$, we have *D*(*SCV, t*) = D_1(*SCV*), where D_1(*SCV*) is a constant related to *SCV*.
- When *t* exceeds T_1, the security mechanism does not have sufficient system resources to protect a packet on time and send the encrypted packet out before the next packet coming in. In this case, the security mechanism can either create multiple threads to process packets concurrently, or put packets in a waiting queue. Hence, when *t* is increasing, more threads will be created to compete for system resources, or more packets will be put in the queue, both of which will lead to longer waiting time. That is, for the traffic t_1 and t_2 of two threads with $T_1 < t_1 < t_2$, we have *D*(*SCV*, t_1) < *D*(*SCV*, t_2).
- The larger the *t*, the faster *D* increases as *t* increases. Hence, if $T_1 < t_1 < t_2$, $\partial D(SCV, t_1)/\partial t_1 > \partial D(SCV, t_2)/\partial t_2 > 0$.
- When *t* keeps increasing and approaches the maximum bandwidth capability, the system will start to drop packets. Hence, from the security mechanism's point of view, there is an upper limit for *t*, which then leads to an upper limit for the delay *D*. For some applications, such as VoIP that have time-out mechanisms, *t* may reach these two upper limits before the system resources are exhausted. The increasing speed of *D* will slow down when *t* approaches its upper limit.
- When *t* is fixed, more efficient algorithm *A*, shorter key length *l*, or smaller percentage *p* for protecting the packets will lead to smaller *D*. However, the effect of the combination of different *A*, *l*, and *p* needs to be investigated through experiments.

Based on the above observation, we obtain the following performance metric *D*(*SCV, t*) for a given aspect (in the above discussion, we use delay *D* as the performance aspect) as follows:

$$D(SCV,t) = \begin{cases} 0, & t = 0 \\ D_1(SCV), & 0 < t \le T_1 \\ D_1(SCV) + a_1(1 - e^{-a_2(t-T_1)^{a_3}}), & t > T_1 \end{cases} \quad (2)$$

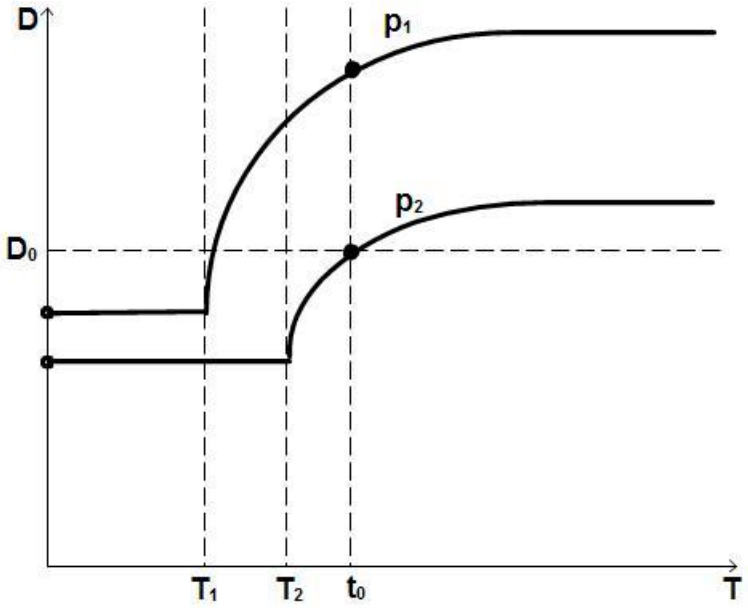

Figure 2. The relations between delay *D* and traffic *T* for two SCVs with the same algorithm and key length, but different protection percentages p_1 and p_2, where $p_1 > p_2$. D_0 is the minimum delay requirement given by the user.

where p is the protection percentage, a_1, a_2, a_3, T_1 and $D_1(SCV)$ are five parameters related to SCV, but independent of t. These five parameters can be found using regression on the SBS's historical performance data on D.

The above sigmoid function is usually used to describe a progression starting from a small value and then accelerating and approaching an upper limit. For the metric (2), when t is less than T_1, D is a constant $D_1(SCV)$. When the traffic exceeds T_1, D starts to accelerate and approaches the upper limit $D_1(SCV)+a_1$.

Figure 2 shows the delay D for two security configuration vectors SCV_1 and SCV_2, which have the same encryption algorithm and key length, but different protection percentage p_1 and p_2. Because $p_1 > p_2$, the security mechanism using SCV_1 will encrypt more packets with the same traffic than SCV_2, and hence D with SCV_1 starts to increase earlier and faster than D with SCV_2.

5. Adaptable Tradeoff Algorithm between Performance and Security

In this section, we will present three different tradeoff strategies to maximize performance, maximize security, and balance performance and security according to users' preferences, respectively. To check whether the tradeoff between performance and security is possible, we first need to make sure that the users' minimum performance and security requirements can be satisfied.

5.1 Minimum Requirement Validation

Chen *et al.* (Chen, Farley, & Ye, 2004) specified and analyzed the performance requirements of various types of network applications with consideration of both human factors and technology

factors. For example, a voice communication application has strict timeliness requirements measured by delay and jitter, but can tolerate certain data loss and errors. Furthermore, the performance of *SBS* is usually measured from various aspects, such as timeliness, throughput, used bandwidth, allocated memory, and used percentage of CPU time. A performance model was presented (Yau, Ye, Sarjoughian, & Huang, 2008) to analyze the interrelations among these different performance requirements and different performance aspects. This model has been demonstrated using non-binary classification and regression trees on experimental data to predicate the value of a performance aspect from other performance aspects.

We assume that the *SBS* requires that the delay D is smaller than D_0, the traffic t of the system will be up to T_0, and the security protection against an attacker with capability c should be at least S_0. Due to the minimum security requirement, we have

$$S(l,p,c) \geq S_0 \Rightarrow p \geq \frac{S_0}{1-cv(A,l)} \tag{3}$$

To satisfy the minimum performance requirement, from the metric of delay (2), we have

$$\begin{array}{l} D(SCV,t) \leq D_0 \\ \Rightarrow \quad \begin{cases} D_1(SCV) \leq D_0, & if\ T_0 \leq T_1 \\ p \leq (\sqrt[a_3]{(-ln(1-\frac{D_0-D_1(SCV)}{a_1})/a_2} + T_1)/T_0, & if\ T_0 > T_1 \end{cases} \end{array} \tag{4}$$

Note that (3) gives a lower bound for p, and (4) gives an upper bound for p. Hence, to check whether the minimum performance and security requirements can be satisfied, we only need to check whether the p between (3) and (4) exits, i.e., whether there exists an algorithm A and key length l satisfying

$$\frac{S_0}{1-cv(A,l)} \leq \frac{\sqrt[a_3]{-ln(1-\frac{D_0-D_1(SCV)}{a_1})/a_2} + T_1}{T_0}, \quad if\ T_0 > T_1 \tag{5}$$

All parameters, including a_1, a_2, a_3, c, $v(A, l)$, and T_1, are determined by the security algorithm A and key length l. Because an *SBS* only supports a limited number of security algorithms and key lengths, we can check whether both the minimum performance and security requirements can be satisfied by enumerating all supported security algorithms and key lengths to see if (5) can be satisfied. For example, Figure 1 shows that the *SCV* with the key length l_2 cannot satisfy the minimum security requirement S_0. Figure 2 shows that the *SCV* with the protection percentage p_1 cannot satisfy the minimum performance requirement D_0 when the traffic is t_0.

5.2 Tradeoff Objective Function

When both the minimum performance and security requirements are satisfied, the *SBS* can use the available resources for better performance or security. To improve security, as shown in the security metric (1), the *SBS* can use either a stronger algorithm with a longer key, or a larger protection percentage, or a better security algorithm with smaller vulnerability. On the other hand, the performance metric (2) shows that a stronger algorithm with a longer key will generate larger a_1, a_2, a_3 and smaller T_1, and a larger protection percentage p will make the delay to increase faster with the traffic t. Both of these effects will downgrade the performance. Hence, to achieve good tradeoff between performance and security, we have to incorporate the performance metric

(2) and security metric (1) in a tradeoff objective function between (1) and (2). Such a tradeoff objection function $G(SCV, t)$ can be defined as follows:

$$G(SCV, t) = w_1 D(SCV, t) + w_2 S(SCV, c) \quad (6)$$

where *SCV* = {*F, A, l, p*}, w_1 and w_2 are two weighting factors representing the user's preferences on performance and security, respectively.

To normalize *G*, we assume that $w_1+w_2=1$ and both w_1 and w_2 are within the range [*0, 1*]. Given D_0 and S_0, the optimized tradeoff between *D* and *S* is defined as minimizing *G* with the constraints $D(SCV, T_0) \leq D_0$ and $S(SCV, c) \geq S_0$. With the objective function (6), the system can compute the best *SCV* with algorithm *A*, key length *l*, and protection percentage *p* according to users' requirements and current traffic *t* to achieve the best tradeoff between performance and security.

5.2.1 Performance Biased Objective Function

The performance biased objective function is a tradeoff objective function to maximize the SBS performance without violating the minimum security requirement. For *G*(*SCV, t*) given in (6), the performance biased objective function sets $w_2=0$. In this case, to minimize *G*(*SCV, t*) is equivalent to minimizing *D*.

When the encryption algorithm and the key length are fixed, the performance biased tradeoff should use the minimum protection percentage $S_0/(1 - cv(A, l))$ computed in (3).

5.2.2 Security Biased Tradeoff Function

The security biased tradeoff function is a tradeoff objective function to maximize the security without violating the minimum performance requirements. For *G*(*SCV, t*) given in (6), the security biased tradeoff function sets $w_1=0$. In this case, to minimize *G*(*SCV, t*) is equivalent to minimizing *S*. When the encryption algorithm and the key length are fixed, we can compute the upper bound for the protection percentage from the minimum performance requirements according to (4).

5.2.3 Dynamic tradeoff objective function

If the *SBS* user's preferences on performance and security are stable and do not change with the real-time performance and security conditions, we call such a tradeoff objective function (6) as a *static tradeoff objective function*, like the performance biased tradeoff function and the security biased tradeoff function. On the contrary, if users' preferences on performance and security are related to the current performance and security, we call such a tradeoff objective function a *dynamic tradeoff objective function*. There may be various ways to define a dynamic tradeoff model, but all definitions should have the following properties:

- The weighting factor of performance (security) increases when the performance (security) approaches the minimum performance (security) requirement.

- The minimum performance or security requirements are critical for the SBS. Hence, when the minimum performance (security) requirements are not satisfied, the weighting factor of performance (security) should increase to infinity to show the unsatisfactory status

Thus, a possible dynamic tradeoff objective function for the performance metric (1) and the security metric (2) can be defined as follows:

$$G'(SCV,t) \quad = \quad e^{a'/(D_0-D(SCV,t))}D(SCV,t) + e^{b'/(S_0-S(SCV,t))}S(l,p,c)$$

where a' and b' are two constants. The weighting factors of performance and security will increase faster with larger performance and security, and will become infinite when the minimum delay and security requirements are not satisfied.

5.3 The Tradeoff Algorithm

The tradeoff algorithm is to find the best *SCV* parameters for the security mechanism, which minimize the tradeoff objective function and hence achieve the optimized tradeoff between performance and security.

For each request, the user specifies the minimum requirements on both performance and security. Considering these minimum requirements as inputs, the tradeoff algorithm first analyzes the current traffic *t* whether both minimum requirements can be satisfied by evaluating the metric (5) under current traffic. If both minimum requirements can be satisfied, the algorithm calculates and returns the optimized parameters for *SCV*; otherwise, the request cannot be satisfied and will be rejected. The details of the tradeoff algorithm are given in Algorithm 1.

Algorithm 1: Tradeoff Algorithm.

Input: Minimum performance requirement D_{min}, minimum security requirement S_{min}

Output: The optimized $SCV_o = \{F, A, k_o, p_o\}$

```
1  Initialize G_o ← −1.0, G ← 0.0;
2  Monitoring current traffic t;
3  foreach key length k' supported by A do
4      Mimimum requirement validation;
5      if (Do not pass the validation) then
6          Reject this session;
7      else
8          foreach Percentage p' supported by A do
9              SCV' ← {F, A, k', p'};
10             Evalaute Performance Metric D ← D(SCV', t);
11             Esitmate Attachers' Computing Power c;
12             Evaluate Security Metric S ← S(SCV', c);
13             Calcualte Goal Fucntion G ← w1 D + w2 S;
14             if (G < G_o or G_o < 0) then
15                 G_o ← G;
16                 SCV_o ← {F, A, k', p'};
17 Return SCV_o;
```

6. An Experimental Example

We have implemented an *SBS* with two services using C# in .NET environment and Microsoft Windows Communication Foundation (WCF). The first service is a security service supporting AES encryption algorithm for confidentiality. Hence, its security configuration vector SCV_{enc} is defined as {*Confidentiality, AES, k, p*}, where *k* is the key length selected from {*128*-bits, *192*-bits, *256*-bits}, and *p* is the protection percentage selected from {*0%, 25%, 50%, 75%, 100%*}. That is, there are total *15* possible parameter configurations for SCV_{enc}. The second service in *SBS* is a traffic service, which generates traffic to simulate the communication in applications. The traffic service generates packets with *1000*-bits size and sends them out with a rate from *10* to *5,000* packets per second. To protect these packets' confidentiality, the traffic service invokes the security service to encrypt packets. In this section, we use this *SBS* to show how to estimate the parameters for security metric (1) and delay metric (2), and then use our adaptive tradeoff approach to achieve an optimal tradeoff between confidentiality and delay.

For the security metric (1), *p* is determined by the *SCV*, where the parameters *c* and $v(AES, l)$ are not assigned by the system but estimated from users' expectation on the attackers and the security mechanisms used in the SBS. The most efficient AES implementation (Bernstein & Schwabe, 2008) requires about 10 CPU cycles to encrypt one byte. In our experimental example, each packet has 1000 bits, and then a 3GHz CPU needs about 0.02 microseconds to encrypt one packet. That is, an attacker with a 3GHz CPU can search about 50,000 keys every second. Hence, in our experiment, we assume that $c = 50{,}000$.

The vulnerability of AES is measured through the size of key space needed to search. For a brute-force attack, we have $v(AES, l) = 2^{-l}$. The most recent related-key analysis for AES (Biryukov & Khovratovich, Related-key cryptanalysis of the full AES-192 and AES-256, 2009) shows that the key space needed to search can be reduced to 2^{176} for 192-bits key length, and 2^{119} for 256-bits key length. Note that the complexity of AES with 256-bits key length is even smaller than AES with 192-bits key length because of the bad key schedule design for 256-bits key length. The AES with 128-bits key length is not affected by the related-key analysis, and all existing attacks that are better than the brute force attack are designed for reduced transformation rounds. For example, the complexity of the most efficient attack for AES with 128-bits key length is 2^{22} with 7 rounds, and 2^{44} with 8 rounds (Biryukov & Khovratovich, A New Security Analysis of AES-128, 2009), while the standard implementation for AES with 128-bits key length requires 10 rounds. Hence, in our experiment, we assume that

$$v(AES, l) = \begin{cases} 2^{-128}, & l = 128 \\ 2^{-176}, & l = 192 \\ 2^{-119}, & l = 256 \end{cases}$$

For the performance metric (2), we need to predict the delay of a packet given security service's SCV_{enc} and current traffic *t*. Generally, a longer key length requires more operations in encryptions and a higher percentage requires the security service to encrypt more packets. Hence, the security service with a longer key length or a larger protection percentage in SCV_{enc} is expected to cause a longer delay on packets. To study the relation among SCV_{enc}, *t* and the delay *D*, we run the experiment by gradually increasing traffic *t* and collecting the average packet delay *D* under different traffic.

For each SCV_{enc}, we run the experiment for 10 seconds to find the average delay of packets as training data, which includes twelve curves for the relations between delay and traffic as shown in

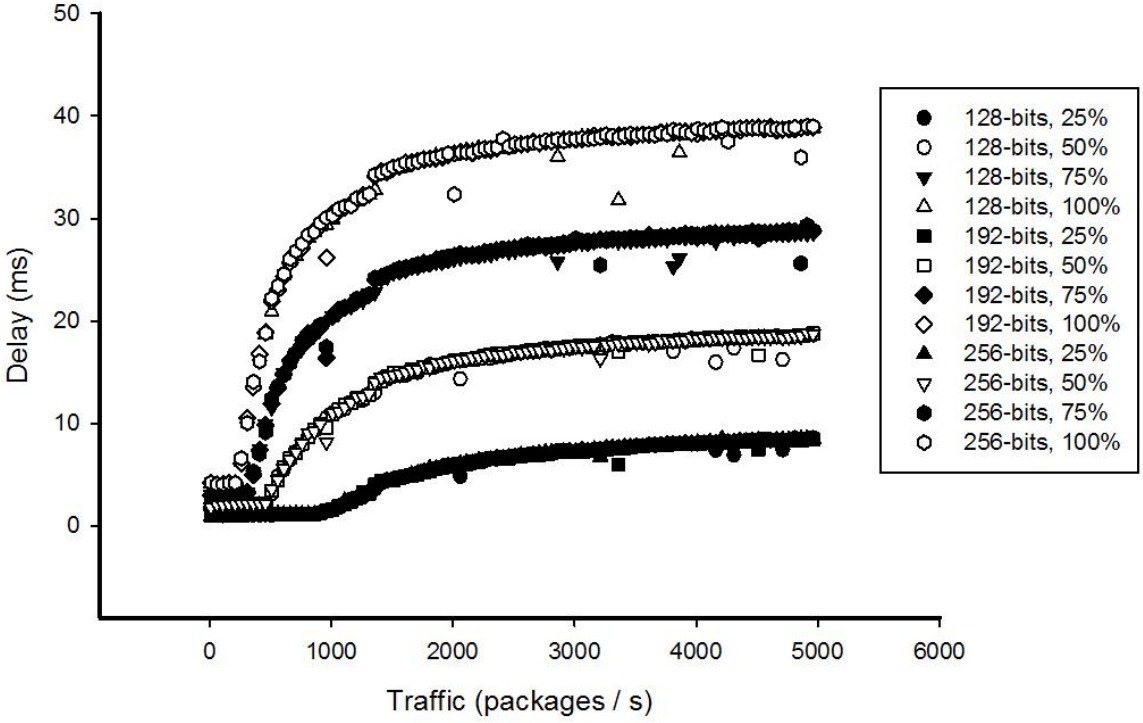

Figure 3. The relations between traffic and delay with different key lengths. Note that curves with the same protection percentage but different key lengths almost coincide together because the protection percentage dominates the security strength.

Figure 3. Figure 4 shows the delay metric *D* for SCV_{enc} = {*Confidence, AES, 128, 100*}. To make *D* clearer for small *t*, Figure 5 shows the same data of Figure 4, but with log x-axes. From Figures 4 and 5, it is clear that *D* increases with *t* as a sigmoid function like (2), which starts from a small value and then accelerates and approaches to an upper limit.

The parameter regression results for all SCV_{enc} are shown in Table 1, in which the last two columns are the coefficient of determination and the adjusted coefficient of determination (Steel & Torrie, 1960), which are very close to 1. It indicates that our regression results match the training data very well. Specifically, the *D* for SCV_{enc} = {*Confidence, AES, 128, 100*} shown in (2) becomes

$$D(SCV,t) = \begin{cases} 0, & t = 0 \\ 4.5187, & 0 < t \leq 283.6797 \\ 4.5187 + 34.4231 \times (1 - e^{-0.019504((t-283.6797))^{0.6521}}), & t > 283.6797 \end{cases} \quad (7)$$

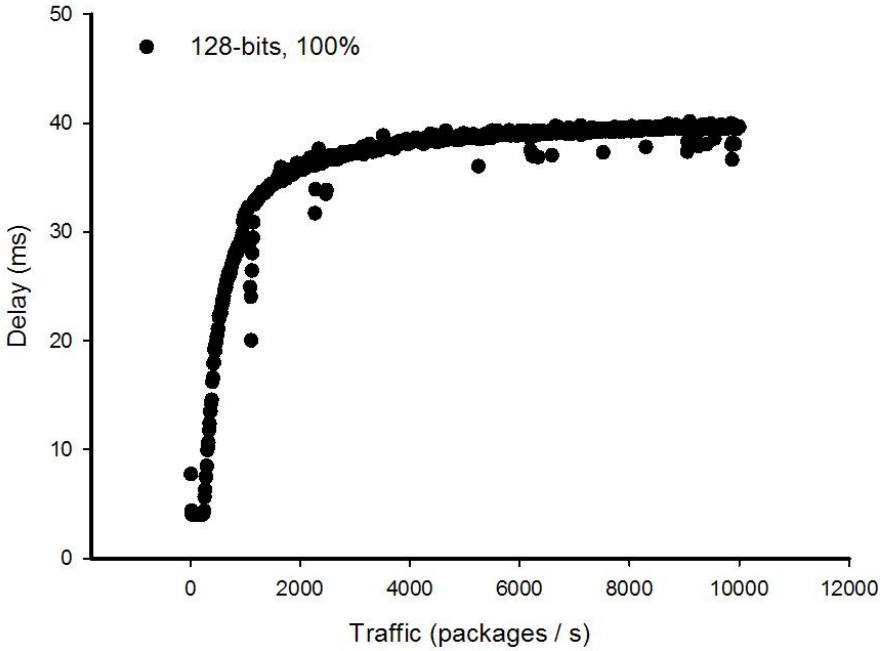

Figure 4. The delay metric for SCV_{enc} = {*Confidentiality, AES, 128, 100*} with linear X-axes

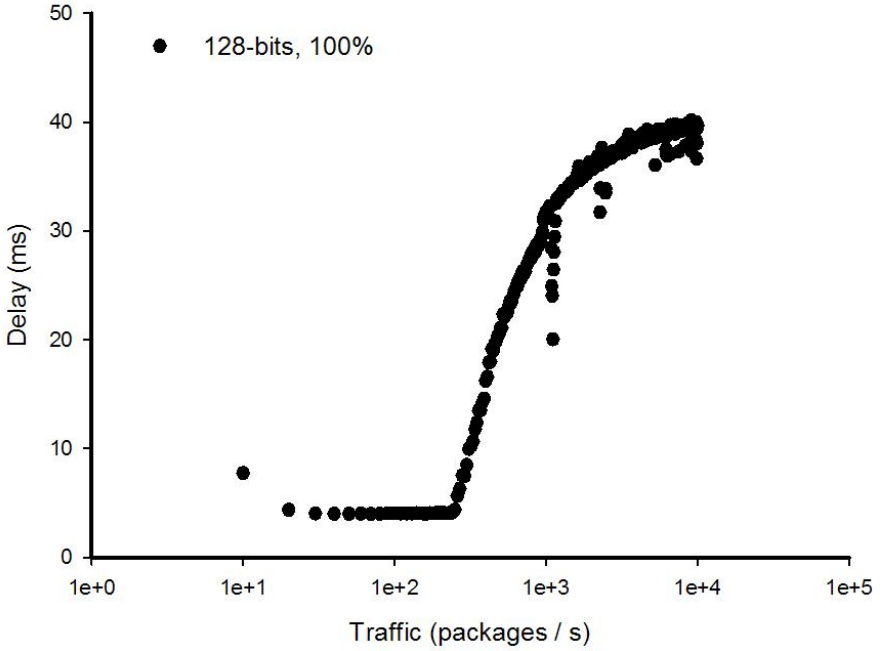

Figure 5. The delay metric for SCV_{enc} = {*Confidentiality, AES, 128, 100*} with Log X-axes

Table 1. Parameter Estimation for all SCV_{enc}

kl	P	a_1	a_2	a_3	T_1	D_1(SCV)	Rsqr	Adj Rsqr
25%	128	7.4962	0.003022483	0.8364	995.5745578	1.1166	0.9929	0.9926
	192	7.8844	0.006025605	0.7293	1031.053106	1.1237	0.9936	0.9933
	256	7.8718	0.003969303	0.7872	992.1825803	1.0953	0.9976	0.9975
50%	128	16.309	0.004038537	0.8381	479.0610806	2.0487	0.994	0.9938
	192	16.365	0.003569859	0.8565	473.8439465	2.0651	0.9974	0.9973
	256	16.4489	0.003141091	0.8723	468.426792	2.0512	0.9962	0.996
75%	128	25.4971	0.009690801	0.7377	345.3320472	3.0559	0.9962	0.9961
	192	25.7254	0.00932398	0.7394	344.0336194	3.0773	0.9964	0.9963
	256	25.5483	0.008280744	0.76	341.559387	3.0899	0.9945	0.9943
100%	128	34.5291	0.01273608	0.7141	252.706246	4.0311	0.9933	0.9931
	192	34.8059	0.013063287	0.7072	253.4914386	4.0651	0.9971	0.997
	256	34.4231	0.019503999	0.6521	283.6797251	4.5187	0.9952	0.995

Table 2. Selected experimental data for SCV_{enc} = {*Confidence, AES, 128, 100*}

Traffic	# Packets	Observed Avg. Delay D (ms)	Predicated Avg. Delay D' (ms)	Deviation Error (%) |D-D'|/D
10	100	7.739	4.0311	47.91 %
30	300	4.0237	4.0311	0.18 %
50	500	3.995	4.0311	0.90 %
80	800	3.9963	4.0311	0.87 %
100	1000	4.0363	4.0311	0.13 %
300	3000	8.4901	10.2899	21.20 %
500	5000	21.072	20.5645	2.41 %
800	8000	27.5223	27.6184	0.35 %
1000	10000	31.6228	30.3422	4.05 %
3000	30000	37.1973	37.6511	1.22 %
5000	50000	39.0438	38.4003	1.65 %
8000	80000	39.5463	38.5434	2.54 %
10000	100000	39.6344	38.5559	2.72 %

These experimental results show the following properties:

- The key length *l* has no significant effect on the relation between the delay and traffic. That is, the parameters a_1, a_2, a_3, T_1 and $D_1(SCV)$ of the delay metric (2) are not significantly affected by *l*. In Figure 3, the twelve curves are grouped into four groups according to *p*. For each group, there are three data sets representing different *l* with same *p*, which coincide with each other. In Table 1, when *p* is the same, all parameters' values are very close for various key lengths.

- Because *p* determines the actual number of packets encrypted, with the same *l*, *p* dominates the relation between *D* and *t*. A larger *p* generates a longer *D* than a smaller *p* and makes the *D* to increase faster than smaller *p*. In Figure 3, the curve group with larger p is above that with smaller p, and is stepper. In Table 1, when *p* increases, the parameters a_1 and a_2 also increases, which speed up the increasing of the *D*.

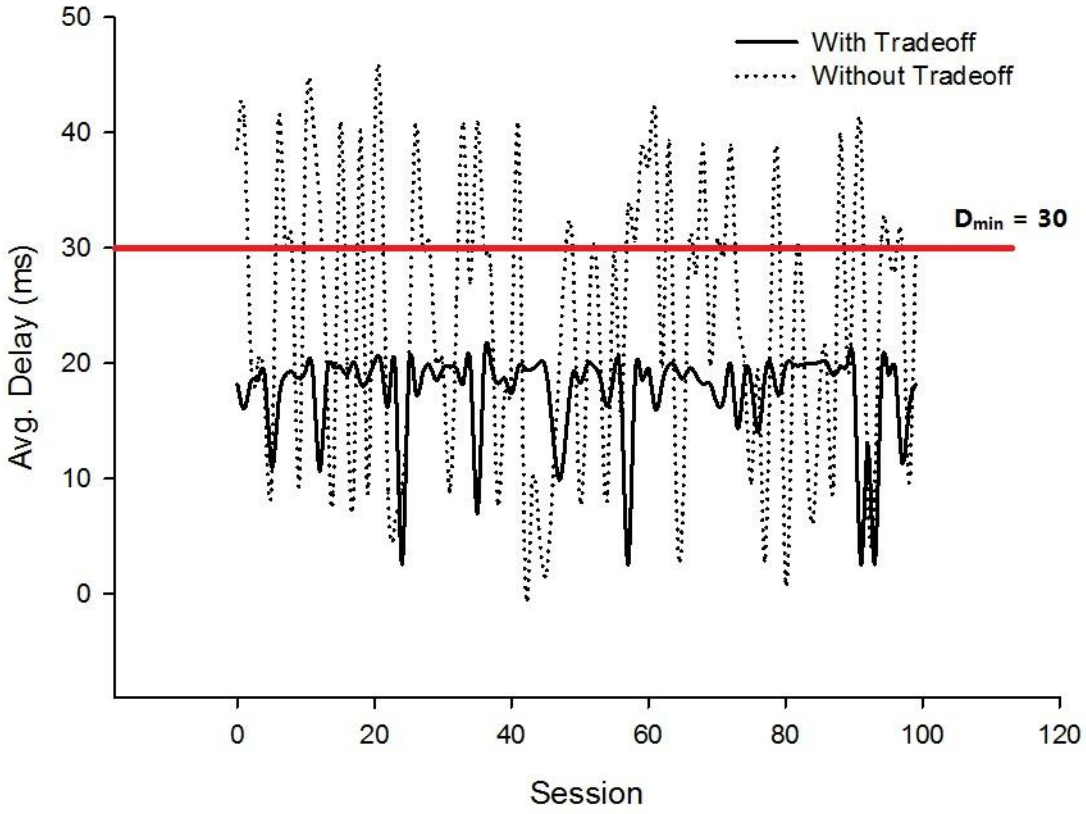

Figure 6. The evaluation of the tradeoff algorithm with D_{min} = 30 and S_{min} = 75.

- Because the time that the delay starting to increase exponentially is determined by the incoming traffic of the security service, but not the overall traffic of the whole system, a larger *p* generates a larger incoming traffic for the security service, when the traffic of the system is the same. In Figure 3, the curve group with a larger p starts to increase earlier than the curve group with a smaller p. In Table 1, the parameter T_1 for larger *p* is smaller.

- Even when *p* is not 100%, all packets including the unprotected packets still consume some resources. Because for a smaller p, the traffic service sends fewer packets to the encryption service, in Figure 3, when the traffic is small and the *D* has not started to increase exponentially, the curve group with larger p is also above the curve group with smaller p. In Table 1, the parameter $D_1(SCV)$ for larger *p* is larger.

To verify the regressed metric, we ran the experiments again and collected the data for SCV_{enc} = {*Confidence, AES, 128, 100*} with *t* from 10 to 10,000 as testing data, and compare the observed average delay with the predicted average delay computed from (7). Some sampling data is listed in Table 2, where the experiment was run 10 seconds for each traffic level. The deviation error in Table 2 shows that the above regressed delay metric matches the real delay data very well. Because the security service needs to create the encryption key at the beginning of encryption, when we started the experiment from t = 10, the observed average delay includes both the delay caused by key generation and encryption which leads to a large deviation error. The large deviation error for t=300 is due to the noises from system's unstable performance.

To evaluate the performance of our tradeoff algorithm *Alg*, we set the minimum security requirement as that the attacker cannot crack a packet with p > 25%, and the minimum security requirement as that the average delay of packets D ≤ 30 million seconds. That is, in the tradeoff algorithm *Alg*, $S_{min}=75$ and $D_{min}=30$. We sample 100 communication sessions with random traffic and SCV parameters. If the session satisfies the minimum security requirement and performance requirement simultaneously, the session successes; otherwise, the session fails. We ran this evaluation twice with and without *Alg*, and compare their successful ratios. The evaluation results are shown in Figure 6, which shows that all sessions' delays are controlled to be less than D_{min} and the success ratio is 100% with *Alg*. Without *Alg*, some sessions' delays exceed the D_{min} and the success ratio is only 37%. Note that because our SCV parameters are discrete and all sessions' traffics were randomly sampled, we could not control the sessions' delays precisely and make them very close to the D_{min}. Hence, there may be a gap between D_{min} and the sessions' delays with *Alg* as shown in Figure 6.

7. Conclusions and Future Research

In this paper, we have presented an adaptive tradeoff approach to improving service performance and security with limited resources in SBS. We have developed quantitative performance and security metrics, and combined them in a tradeoff objective function with two weighting factors representing the preferences on performance and security. We have shown how to achieve the optimized maximum performance, the maximum security, and the optimized balance between performance and security by adjusting the weighting factors. We also have illustrated our approach with a SBS with minimum performance and security requirements how to estimate the parameters for our approach.

Future research along this line includes extending our approach for optimizing tradeoff between security and a set of performance aspects, such as the usage of CPU times, memory, and bandwidth.

Acknowledgement

This work is supported by National Science Foundation under grant number CCF-0725340.

REFERENCES

Abdelzaher, T. F., Stankovic, J. A., Lu, C., Zhang, R., & Lu, Y. (2003). Feedback Performance Control in Software Services. *IEEE Control Systems Magazine , 23* (3), 74-90.

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., et al. (2004). *Web Services Policy Framework (WS-Policy).* BEA Systems Inc., Int'l Business Machines Corporation, Microsoft Corporation, Inc., SAP AG, Sonic Software, and VeriSign Inc.

Bernstein, D. J., & Schwabe, P. (2008). New AES Software Speed Records. *9th International Conference on Cryptology in India* (pp. 322-336). Springer.

Biryukov, A., & Khovratovich, D. (2009, Aug). *A New Security Analysis of AES-128*. Retrieved from http://rump2009.cr.yp.to/b6f3cb038135799a7ea398f99faf4a55.pdf

Biryukov, A., & Khovratovich, D. (2009). *Related-key cryptanalysis of the full AES-192 and AES-256.* Retrieved from https://cryptolux.org/mediawiki/uploads/1/1a/Aes-192-256.pdf

Chen, Y., Farley, T., & Ye, N. (2004). QoS Requirements of Network Applications on the Internet. *Info., Knowledge, Systems Management , 4* (1), 57-76.

Cheng, H., & Li, X. (2000). Partial Encryption of Compressed Images and Video. *IEEE Trans. on Signal Processing , 48* (8), 2439-2451.

Droogenbroeck, M. V., & Benedett, R. (2002). Techniques for a Selective Encryption of Uncompressed and Compressed Images. *Proc. of Adv. Concepts for Intelligent Vision Systems (ACIVS)*, (pp. 9-11).

Eggert, L., & Heidemann, J. (1999). Application-Level Differentiated Services for Web Services. *J. World-Wide Web , 2* (3), 133-142.

Godik, S., & Moses, T. (2003, February). eXtensible Access Control Markup Language (XACML) Version 1.0. *OASIS Standard* .

Hallam-Baker, P., & Mysore, S. (2005). XML Key Management Specification (XKMS 2.0). *W3C Recommendation* .

Hathaway, L. (2003). *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information.* National Security Agency.

Kang, K., & Son, S. (2006). Systematic Security and Timeless Tradeoffs in Real-Time Embedded Systems. *Proc. 12th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Application* , 183-189.

Lenstra, A., & Verheul, E. (2001). Selecting Cryptographic Key Sizes. *J. of Cryptology , 14*, 255-293.

Lie, D., & Satyanarayanan, M. (2007). Quantifying the Strength of Security Systems. *Proc. of 2nd USENIX Workshop on Hot Topics in Security*, (pp. 1-6).

Lu, C., Lu, Y., Abdelzaher, T. F., Stankovic, J. A., & Son, S. H. (2006). Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE Trans. on Parallel and Distributed Systems , 17* (9), 1014-1027.

Mactaggart, M. (2001). Enabling XML Security: An Introduction to XML Encryption and XML Signature. *IBM Developer Works , 9*.

Pashalidis, A., & Mitchell, C. J. (2003). A Taxonomy of Single Sign-On Systems. *Proc. of Info. Security and Privacy*, (pp. 249-264).

Pomerance, C. (1996). A Tale of Two Sieves. *Notices of the AMS , 43* (12), 1473-1485.

Rao, G., & Ramamurthy, B. (2001). DiffServer: Appliction Level Differentiated Services for Webservers. *Proc. of IEEE Int'l Conf. on Communication*, *5*, pp. 1633-1637.

Shi, C., Wang, S. Y., & Bhargava, B. (1999). MPEG Video Encryption in Real-Time Using Secret key Cryptography. *Proc. of Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA).*

Son, S. H., Zimmerman, R., & Hansson, J. (2000). An Adaptable Security Manager for Real-Time Transactions. *Proc. 12th Euromicro Conf. on Real-Time Systems*, (pp. 63-70).

Spyropoulou, E., Levin, T., & Irvine, C. (2000). Calculating Costs for Quality of Security Service. *Proc. 16th Annual Conf. Computer Security Applications*, (pp. 334-343).

Steel, R., & Torrie, J. (1960). *Principles and Procedures of Statistics.* New York: McGraw-Hill.

Yau, S. S., Yan, M., & Huang, D. (2007). Design of Service-based Systems with Adaptive Tradeoff between Security and Service Delay. *Proc. 4th Int'l Conf. on Autonomic and Trusted Computing*, (pp. 394-401).

Yau, S. S., Ye, N., Sarjoughian, H., & Huang, D. (2008). Developing Service-based Software Systems with QoS Monitoring and Adaptation. *Proc. of 12th Int'l Workshop on Future Trends of Distributed (FTDCS)*, (pp. 74-80).

Yurcik, W., Woolam, C., Hellings, G., Khan, L., & Thuraisingham, B. (2007). SCRUB-tcpdump: A Multi-level Packet Anonymizer Demonstrating Privacy/Analysis Tradeoffs. *Proc. 3rd Security and Privacy in Commu. Networks and the Workshops (SecureComm)*, (pp. 49-56).

Zeng, W., & Lei, S. (2003). Efficient Frequency Domain Selective Scrambling of Digital Video. *IEEE Trans. on Multimedia , 5* (1), 118-129.

ABOUT THE AUTHOR (Important)

Stephen S. Yau is currently the director of Information Assurance Center and Professor of Computer Science and Engineering at Arizona State University, Tempe, Arizona, USA. He served as the chair of the Department of Computer Science and Engineering from 1994 to 2001. He was previously with the University of Florida, Gainesville and Northwestern University, Evanston, Illinois. He served as the president of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE) and the editor-in-chief of IEEE Computer magazine. His current research is in distributed and service-oriented computing systems, software engineering and trustworthy computing, and data privacy. He received the Ph.D. degree in electrical engineering from the University of Illinois, Urbana. He is a life fellow of the IEEE and a fellow of American Association for the Advancement of Science. Contact him at yau@asu.edu.

Yin Yin is a Ph.D. student in the School of Computing, Informatics and Decision System Engineering at Arizona State University, Tempe, Arizona, USA. His research interests include privacy protection, trustworthy computing, and cryptography. He received the B.S. degree in mathematics from Wuhan University in China, and the M.S. degree in computer science from Chinese Academy of Science. Contact him at yin.yin@asu.edu.

Ho An is a Ph.D. student in the School of Computing, Informatics and Decision System Engineering at Arizona State University, Tempe, Arizona, USA. His research interests include service and cloud computing, security and privacy. He received the M.S. degree in computer science from Arizona State University. Contact him at ho.an@asu.edu.