基于参数优化批处理的 TLS 协议*

齐 芳[1,2]，贾维嘉[1,3+]，鲍 丰[2]，吴永东[2]，王国军[1]

[1](中南大学 信息科学与工程学院,湖南 长沙 410083)

[2](Agency for Science, Technology and Research, Institute for Infocomm Research, Singapore 119613, Singapore)

[3](Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong)

Parameter Optimization-Based Batching TLS Protocol

QI Fang[1,2], JIA Wei-Jia[1,3+], BAO Feng[2], WU Yong-Dong[2], WANG Guo-Jun[1]

[1](School of Information Science and Engineering, Central South University, Changsha 410083, China)

[2](Agency for Science, Technology and Research, Institute for Infocomm Research, Singapore 119613, Singapore)

[3](Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong)

+ Corresponding author: Phn: +86-731-8836152, Fax: +86-731-8877711, E-mail: itjia@cityu.edu.hk

Qi F, Jia WJ, Bao F, Wu YD, Wang GJ. Parameter optimization-based batching TLS protocol. *Journal of Software*, 2007,18(6):1522−1530. http://www.jos.org.cn/1000-9825/18/1522.htm

Abstract: The primary goal of the Transport Layer Security (TLS) protocol is to provide confidentiality and data integrity between two communicating entities. Since the most computationally expensive step in the TLS handshake protocol is the server's RSA decryption, it is introduced that optimal batch RSA can be used to speedup TLS session initialization. This paper first indicates that the previous batch method is impractical since it requires a multiple of certificates, then it proposes the unique certificate scheme to overcome the problem. It is also introduced that the batching parameter is optimized when integrating users' requirements for Internet Quality of Service (QoS). To select the optimal batching parameters, not only the server's performance but also the client's tolerable waiting time is considered. Based on the analysis of the mean queue time, batching service time and the stability of the system, a novel batch optimal scheduling algorithm which is deployed in a batching Web server is proposed. Finally, the proposed algorithm is evaluated to be practical and efficient through both analysis and simulation studies.

Key words: transport layer security; batch RSA; TLS handshake; mean response time; tolerable waiting time

摘 要: TLS(transport layer security)协议的基本设计目标是为两个通信实体之间提供数据的保密性和完整性.由于在传输层安全握手协议中最耗费计算资源的步骤是服务器 RSA 解密运算,优化的批处理的 RSA 方法提出可以用于加速 TLS 会话的初始化.首先指出了以前的批处理方法由于要求多证书实现而实用性不强.然后提出了单一证书策略的方法,从而克服了这一问题.还提出结合用户对于因特网服务质量的要求优化了批处理参数.为了选择优化的

* Supported by the National Natural Science Foundation of China under Grant No.60503007 (国家自然科学基金); the CityU Research Project (APR) 9610027; the National Basic Research Program of China under Grant No.2003CB317003 (国家重点基础研究发展计划(973)); the Program for New Century Excellent Talents in University of the Chinese Ministry of Education under Grant No.NCET-06- 0686 (2006 年度教育部"新世纪优秀人才支持计划")

Received 2005-11-01; Accepted 2007-01-26

批处理的参数,不仅考虑了服务器的性能,而且还考虑了客户可容忍的等待时间.通过分析并在阐述平均排队时间、批处理服务时间和系统稳定性的基础上提出了一种新颖的优化批处理调度算法,已部署在服务器上.最后通过分析和模拟两种方法验证了所提出方案的实用性和有效性.

关键词: 传输层安全协议;批处理 RSA;TLS 握手协议;平均响应时间;可容忍的等待时间
中图法分类号: TP309 文献标识码: A

1 Introduction

Secure communications such as Internet banking and e-commerce are an intrinsic demand of today's world of online transactions. TLS protects communications by encrypting messages with a secret key negotiated in the TLS handshake protocol. Such a protocol allows the server and the client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before transmitting and receiving the first byte of data[1]. However, such a protocol needs intensive computational resource due to the cost of public-key operations.

The proposed scheme in this paper focuses on the batching technology which is a software-only approach for speeding up TLS's performance on a web server. Starting-point of the proposed scheme is a technique due to Fiat[2] for batch RSA decryption. Shacham and Boneh proved that it is impossible to use a single certificate in the present TLS system[3]. Whereas, this paper adapts the certificate mechanism so as to provide TLS setup with unique certificate issued by Certificate Authority (CA). Batch size is equal to four in the scheme of Shacham and Boneh[3]. However, it ignores the satisfaction of the users' requirements for Quality of Service such as tolerable waiting time. Tolerable waiting time is defined as the delay time a client can tolerate between a request for a secure web page and receiving the page. In addition, the proposed scheme in this paper models the client request as an M/D/1 queue[4] and uses approximate analytical solution of mean response time to optimize the batch size of the server. Consequently, the proposed optimal batch scheduling algorithm is employed in a batching web server.

The rest of the paper is organized as follows. Section 2 describes the existing batch method with Batch RSA and describes its problems. The proposed unique certificate scheme in batching TLS is presented in Section 3. The analytical model of parameter optimization in batching TLS handshake is presented in Section 4. Section 5 validates the solutions through both analysis and simulation studies.

2 Preliminaries

Definition 1. Given b distinct and pairwise relatively prime public keys $e_1,...,e_b$ all sharing a common modulus $N=pq$, relatively prime to $\phi(N)=(p-1)(q-1)$. n is the bit length of the public modulus N and k the bit length of the bigger of e_i. Given plaintext messages $m_1,...,m_b$. Furthermore, we have b encrypted messages $v_1,...,v_b$ (i.e. $v_i = m_i^{e_i} \bmod N$) one encrypted with each key. The multiplication computation phase is to generate the product $v = \prod_{i=1}^{b} v_i^{e/e_i} \bmod N$, where $e = \prod_{i=1}^{b} e_i$. The exponentiation phase yields $v^{1/e} \bmod N = \prod_{i=1}^{b} v_i^{1/e_i} \bmod N$, which is stored as m. The division computation phase is to break up the product m to obtain the plaintexts $m_i = v_i^{1/e_i}$ which we wish to decrypt simultaneously. This procedure is called Batch RSA[2].

Example. Fiat observed that when using small public exponent e_1 and e_2, it is possible to decrypt two cipher texts for approximately the price of one[3]. Suppose v_1 and v_2 is a cipher text obtained by encryption using the public key (N,3) and (N,5). Then the product $m = (v_1^5 \cdot v_2^3)^{1/15} \bmod N$ can be computed according to Definition 1. To decrypt m_1 and m_2, we must compute $v_1^{1/3}$ and $v_2^{1/5}$ mod N as follows:

$$v_1^{1/3} \bmod N = \frac{m^{10}}{v_1^3 \cdot v_2^2},\ v_2^{1/5} \bmod N = \frac{m^6}{v_1^2 \cdot v_2} \tag{1}$$

Shacham and Boneh showed that it is not possible to batch when the same public key is used for more than one message[3]. They provide a multiple certificate to implement the batch method by assigning the different public key in TLS handshake. The method has following disadvantages: Requirement for many different RSA certificates; additional payment for certificates; extra maintenance works of multiple certifications. The proposed unique certificate scheme solves the impractical of multiple certificate method in next section.

3 Unique Certificate Scheme in Batching TLS Handshake

In the standard TLS protocol, each client encrypts a 48-byte pre-master secret using e_i as the encryption exponents, and the server decrypts the cipher text independently so as to get the *Pre-master secret*. But batch RSA obtains the *Pre-master secrets* from multiple clients and hence improves the performance significantly.

Our unique certificate method is to reuse the message *ServerHello.random* in the protocol (see Fig.1). For simplicity, we only show the related processes and the modified information in the standard TLS handshake protocol. The following procedure is the unique certificate scheme for SSL handshake protocol: (1) Clients send "*Client hello*" massage that includes the cipher suits to the server and create random nonce r_c respectively. (2) Server responds with a "server hello" massage that includes server's public-key certificate and a random nonce r_s. In this improvement, e_i is actually a part of *ServerHello.random*. Server only needs to send unique certificate to all the clients. (3) Clients choose a secret random 48-byte *Pre-master secret* m_1 and m_2 by inputting values m_1, m_2, r_c, r_s into hash function $f()$. It then encrypts m_i with e_i which is different from server's public-key and attaches the ciphertext to a "*Client key exchange*" message that is sent to server. (4) Server decrypts the *Pre-master secret* m_1 and m_2 simultaneously using batch RSA, and uses it to compute the *Shared master secret* s_1 and s_2 respectively.

The client will verify the certificate as usual, but encrypt the *pre-master secret* with received e_i instead of the public exponent in the certificate. Therefore, no extra charge is required, and it is easy to manage the certificate. On the other hand, since the certificate is used to prove the owner who knows the factors of the RSA moduli N only, this adaptation does not undermine the security strength of TLS protocol.

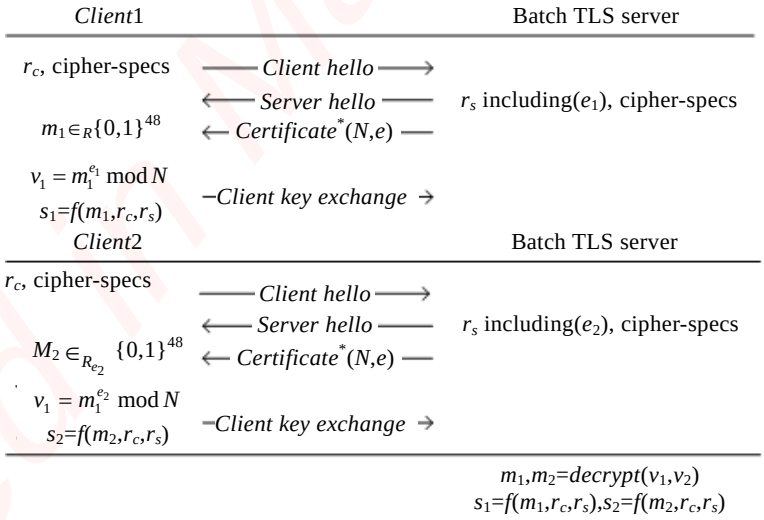


Fig.1 Unique certificate scheme

4 Analytical Model of Parameter Optimization in Batching TLS Handshake

In this section, analytical model of parameter optimization is constructed. The proposed scheme models the client request as an M/D/1 queue and uses approximate solution of mean response time to optimize the batch size of

the server. Consequently, based on the analysis of the stability of the system, the proposed optimal batch scheduling algorithm is employed in a batching Web server.

4.1 Analytical mean queue time in batching queue model M/D/1

Suppose the client arrival process is Poisson distributed with an arrival rate, and the server response is regarded as an M/D/1 queue. Let the batching service time be τ which is determined by the batching size b.

The M/D/1 model can be approximated with a semi-Markov process (see Fig.2). The state of semi-Markov process is described by (i) where i indicates the number of clients waiting in the queue with i=idle and idle indicates the server is idle. Let the mean residual service time be T_r=0.5τ[4].

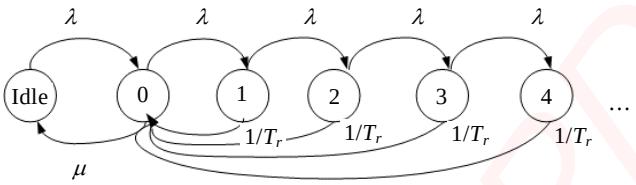


Fig.2 Semi-Markov process model of batching server

Let A_0 be the mean holding time in state idle, A_1 be the mean holding time in state (0), and A_2 be the mean holding time in state (i), for i>0. Thus

$$A_0=\int_0^\infty t\lambda e^{-\lambda t}\mathrm{d}t=1/\lambda,\ A_1=\int_0^\tau t\lambda e^{-\lambda t}\mathrm{d}t+\int_\tau^\infty \lambda \mathrm{e}^{-\lambda t}\mathrm{d}t=\eta_1/\lambda\quad A_2=\int_0^{T_r} t\lambda e^{-\lambda t}\mathrm{d}t+\int_{T_r}^\infty \lambda e^{-\lambda t}\mathrm{d}t=\eta_2/\lambda \tag{2}$$

where, η_1=1–e$^{-\lambda\tau}$ denotes the Markov transition Probability from idle to state (0), and η_2=1–e$^{-0.5\lambda\tau}$ denotes the Markov transition state (i) to state (i+1), for i>0.

Let π_0^* corresponds to the state (0). Then, the steady distribution π_0^* for semi-Markov can be solved as π_0^*=(1–η_1)(1–η_2)/(1+$\eta_1\eta_2$–η_2) due to Ref.[5] (see Fig.2). *Prob*(*idle*) is approximated by π_0^*, which is obtained by solving the semi-Markov processed. Then the mean queue time with batching T_q is solved using the residual service time T_r and the steady state distribution π_0^* for semi-Markov when the server is idle.

$$T_q=(1-Prob(idle))T_r=(1-\pi^*)T_r=(1-(1-\eta_1)(1-\eta_2)/(1+\eta_1\eta_2-\eta_2))T_r \tag{3}$$

4.2 Analytical batching service time

Let the batching RSA decryption time in TLS handshake time be T_b. Let n be the bit length of the public modulus N and k be the bit length of the bigger of exponent. Because the public exponent e_i is chosen as small as possible to make auxiliary exponentiations cheap, the low-cost operations in the computation cost estimation can be ignored.

Asymptotic behavior of analysis of batch RSA can be estimated as $3n^3+(42b+k(3b^3+3b)-1)n^2+o(n^2)$[6].

The classic RSA's decryption mainly includes exponentiation computation which costs $3n^3+n^2+o(n^2)$.

It is assumed that the classic RSA decryption time is computed in advance and denoted as T_{rsa} with 1024 bits public modulus N and the exponent e 65537. The batching RSA decryption time in TLS handshake T_b can be estimated as the following.

As a result, the batching RSA decryption time is

$$T_b=\left(\frac{3n^3+n^2(42b+k(3b^3+3b)-1)}{b(3n^3+n^2)}\right)bT_{rsa}=\left(\frac{3n+42b+k(3b^3+3b)-1}{b(3n+1)}\right)bT_{rsa} \tag{4}$$

Since T_b is the majority of service time, the batching service time of the server τ is T_b roughly. As a TLS server waits for more than one RSA decryption request and performs one big computation for all decryptions, it can save a lot of running time capacity, being able to perform more TLS handshakes.

Theorem 1. To satisfy the client's requirement for the stability of the system, the batching service time is less than the batch size multiplying mean Poisson distributed arrival time interval when the time in the Batch Queue Model M/D/1, thus

$$\tau \approx T_b < b/\lambda \tag{5}$$

Proof: Let X_i(i=1,2,...) be the arrival time interval of two consecutive requests, and Y be the time interval of b consecutive requests. If the system achieves the stability when the time $t \to \infty$ for M/D/1 queue model, $T_b < E(Y)$, where $E(Y)$ is the expected value of Y. Because the X_i is a random variable with independent identical distribution, the average arrival time interval of b consecutive requests is

$$E(Y) = E\left(\sum_{i=1}^{b} X_i\right) = bE(X_i) = b/\lambda \tag{6}$$

Then Theorem 1 is proved.

Lemma 1. In the Batch Queue Model M/D/1, to satisfy the client's requirement for the stability of the system, thus

$$T_q < b/2\lambda \tag{7}$$

Proof: In the Batch Queue Model M/D/1, the value of T_q is derived following Eq.(3)

$$T_q = \left(1 - \frac{(1-\eta_1)(1-\eta_2)}{(1+\eta_1\eta_2-\eta_2)}\right)T_r = \left(\frac{1-e^{-\lambda\tau}}{1-e^{-\lambda\tau}+e^{-1.5\lambda\tau}}\right)T_r = \left(\frac{e^{-\lambda\tau}-1}{e^{-\lambda\tau}-1+e^{-0.5\lambda\tau}}\right)T_r = \left(\frac{1}{1+\frac{1}{(e^{-\lambda\tau}-1)e^{0.5\lambda\tau}}}\right)T_r \tag{8}$$

where, T_r=0.5τ. Due to Theorem 1, it can be easily described as

$$T_q = \left(\frac{0.5\tau}{1+\frac{1}{(e^{-\lambda\tau}-1)e^{0.5\lambda\tau}}}\right) < \left(\frac{1}{1+\frac{1}{(e^{b}-1)e^{0.5b}}}\right)\left(\frac{b}{2\lambda}\right) \tag{9}$$

It can be easily described as when $b \geq 2$, $0.944 \approx 1+\frac{1}{(e^2-1)e^{0.5\times 2}} \leq 1+\frac{1}{(e^b-1)e^{0.5b}} < 1$. Then the value bound of the upper limit of T_q is estimated as [0.944b/2λ, b/2λ]. Then Lemma 1 is proved.

4.3 Optimal batch scheduling algorithm

If the constants and practical value of the client's tolerable waiting time T_t are known, the upper limit of batch size b can be estimated according to different arrival rates of the client. It is usually expected that the ideal response time of web sites is 1 or 2 seconds[7] and 50% people apparently will wait only 8 seconds for a website to download before they get fed up and move on[7]. It is assumed that the value T_t is equal to 1 second, 2 seconds and 8 seconds as examples both in the analytical model and simulation. The total customer response time T is denoted as the sum of T_q, T_c and T_b, where T_c is the time for waiting other client in the same batching. It is easily derived that the max value of T_c is (b–1)λ. Otherwise, the upper limit of T_b has been estimated as b/λ in Theorem 1. We can also estimate the upper limit of T_q as b/2λ derived from Lemma 1. It is assumed that T will not exceed T_t. The upper limit of b is derived as the following

$$T = T_q + T_c + T_b = \frac{b}{2\lambda} + \frac{(b-1)}{\lambda} + \frac{b}{\lambda} < T_t \Rightarrow b < 0.4(\lambda T_t + 1) \tag{10}$$

The pseudo code (see Fig.3.) shows the optimal batch scheduling algorithm which is employed in a batching web server. Step one sorts b to satisfy max solution of $T_b < b/\lambda$ in ascending order from two to the upper limit using Eq.(10). The computation of T_b is performed using Eq.(5). In step two, the batch server performs a set of related tasks to optimal schedule. The server firstly constructs b queues ($Q(e_1)$,$Q(e_2)$,….,$Q(e_b)$) for every exponent e_i. A

heavily loaded web server using a round strategy when dispatching the exponent to clients would incur minimal latency before receiving b TLS handshake requests. When new requests arrive, the server adds the client to the corresponding queue and initializes a timer for the client. If every queue has client, it means that the condition of optimal batch is satisfied. Then, the server excuses *batch_decryption*(). The meaning of function *batch_decyption* () is to do batch decryption with the client in the head of every not empty queue in the ($Q(e_1)$,$Q(e_2)$,…,$Q(e_b)$). If the condition of optimal batch can not be satisfied, the server waits for a period of time that is equal to T_t surplus the max value of timer of the client in the head of every queue. Then the server does the *batch_decryption*(). The batch server has the ability to fall back on *conventional_RSA_decryption*() when only one client is in queues. If all queues do not have clients, the algorithm terminates.

Step1: Find out the solution of b
Input: T_t, λ;
Output: T_b, optimal batch size.
Begin
1. **Compute** the max batch size. max*batchsize*= *int*(0.4(λT_t+1)) (refer to Eq.(10)).
2. **If** (max*batchsize*<=1) **then** do
3. **conventional_RSA_decryption**(); return;
4. **For** (*b*=2;*b*<=max*batchsize*;*b*++)
5. {Success=false;
6. **Compute** T_b (refer to Eq.(4));
7. **If** (T_b<b/λ) **then** {*Optimalbatchsize*=b; Success=true;}}
8. **If** (!success) **then** return;
9. **If** (success) **then** goto step2();

End

Step2: Optimal batch scheduling
Input: Optimal batch size, T_t;
Output: optimal batch scheduling.
Begin:
1. **Construct** b queues ($Q(e_1)$,$Q(e_2)$,...,$Q(e_b)$) for every exponent e_i. max*timer*=0;
2. **Assign** the exponents {e_1,e_2,…,e_b} to different clients using round robin strategy in *serverhello* message (refer to Fig.1.)
3. **while** ($Q(e_1)$!=Null **or** (e_2)!=Null ... **or** (e_b)!=Null){
4. **If** Client arrived **then** {**match** client. *exponent*=e_i, **enqueue** ($Q(e_i)$,client); **initialize** Client.timer}
5. **If** ($Q(e_1)$!=Null **and** $Q(e_2)$!=Null ... **and** $Q(e_b)$!=Null)
6. **then** {Do **batch_decryption**(); **reset** server_waiting_time; **update** queues ($Q(e_1)$,$Q(e_2)$,...,$Q(e_b)$;)
7. **Else** { **for** (j=1; j<=b; j++)
8. {**If** (($Q(e_j)$!=Null) **and** ($Q(e_j)$.*head*.*timer*>=max*timer*))
9. {max*timer*=($Q(e_j)$.*head*.*timer*); } }
10. **If** (server waiting time>=T_t–max*timer*) **then**
11. {do **batch_decryption**(); **reset** server_waiting_time; **update** queues ($Q(e_1)$,$Q(e_2)$,...,$Q(e_b)$;)
12. **Else** continuing waiting for request of client;}

End

Fig.3 Optimal batch scheduling algorithm

5 Validation of Analytical Models and Performance Evaluation Study

5.1 Experiment configuration

The analytical models are executed on a machine with a Dell Intel Pentium IV processor clocked at 3.20GHz and 1GMB RAM. Specifically, this paper performs the simulation of batching RSA with very small public exponents, namely e=3,5,7,11,13,17 etc. The simulation result of the conventional RSA decryption time T_{rsa} with larger public exponent, namely e=65537 is about 32 ms which is tested using reiterative results. It is assumed public modulus N is 1024 bits length.

5.2 Validation of analytical models

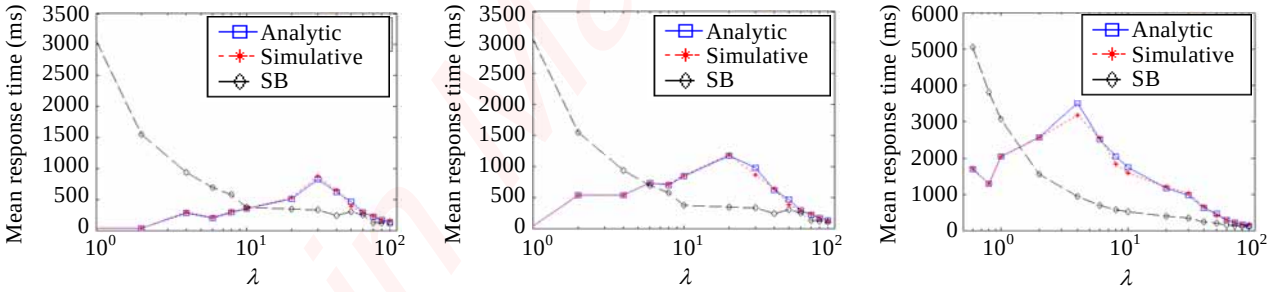
Table 1 validates optimal batch size described by the analytical model. As small arrival rates, b is almost uniform calculated by our analytical model (Table 1). Since the arrival rates are small (i.e., T_t=8, λ<0.6), there is very little opportunity to batch, and therefore, the solution of b is relative small (Table 1). Even at higher arrival rate, the analytical result and simulation result are very close. The solution of the optimal batch size is increased with λ both in analytic and simulation when λ<30 (i.e., T_t=1) approximately. Otherwise, T in this case is not increased obviously. The solution of b is decreased with the λ when λ>30 approximately.

Table 1 Optimal batch size results validation

λ	Optimal batch size					
	Analytical model			Simulation results		
	T_t=1	T_t=2	T_t=8	T_t=1	T_t=2	T_t=8
0.6	–	–	2	–	–	2
1	–	–	3	–	–	3
2	–	2	6	–	2	6
3	–	2	10	–	2	10
4	2	3	13	2	3	12
5	2	4	13	2	4	12
10	4	8	13	4	8	13
20	8	13	13	8	12	13
30	12	12	12	12	12	12
40	11	11	11	10	11	11
50	10	10	10	10	9	10
60	8	8	8	8	8	8
80	6	7	6	6	6	6
90	5	5	5	6	5	6
100	5	5	5	6	4	6

5.3 Performance evaluation

Fig.4(a)~Fig.4(c) illustrates the analytical mean response time T, our simulation results and that of SB (Shacham and Boneh) scheme. When λ is equal to 30 approximately, T reaches Maximum whereas the value is less than 1 second and decreased with the λ when using optimal batch size b (see Fig.4(a)). In the SB scheme, T of a batching system behaves poorly especially when λ does not exceed 10 requests/sec (see Fig.4(a)). When λ is larger than 10 and less than 100 approximately, the performance of the novel scheme and SB scheme are all satisfied with the clients' requirement of tolerable waiting time. However, it is shown that the solutions of b (Table 1) are larger than four. That means the optimal scheme can submit more decryption requests once to decryption device than SB scheme. The other two cases with T_t=2 (see Fig.4(b)) and T_t=8 (see Fig.4(c)) can be analyzed similarly as that with T_t=1 (see Fig.4(a)). The analytical and simulation results of T shows the novel batch scheme behaves nicely.



(a) T_t=1 (b) T_t=2 (c) T_t=8

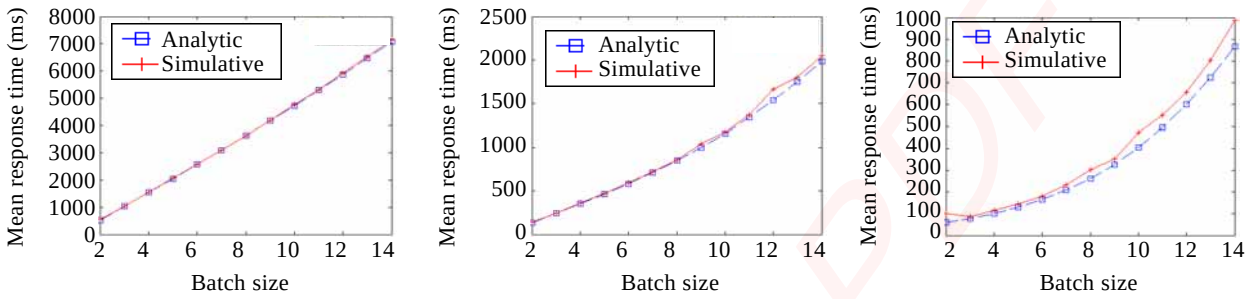
Fig.4 Mean response time validation over client's tolerable waiting time

It is assumed that T_t is equal to 8 seconds in Fig.5(a)~Fig.5(c). These figures show that T is almost linear when λ is relatively small. This is due to the fact that $T=T_q+T_c+T_b$. When λ is relatively small, the main contribution to T is made by T_c. It is evident that the time T_c is increased linearly with b. T_b is also increased with b. Therefore T is also increased with b when λ is relatively large (i. e., λ=80).

A non-batching system becomes unstable when $\lambda>1/T_{rsa}=1/0.032=31.25$ due to the fact that a non-batching system becomes unstable when $\lambda>\tau$. But with batching, a batch system behaves nicely even at high loads. When the non-batching system is stable, the mean response time T' can be estimated as Eq.(11) (see Fig.6).The mean service time τ' is deterministic in the non-batching in M/D/1 queue model. Since T_{rsa} is the majority of service time, the mean service time τ' of the server is roughly T_{rsa}[5].

$$T' = \tau' + \tau'\left(\frac{\tau'\lambda}{2(1-\tau'\lambda)}\right) \approx T_{rsa} + T_{rsa}\left(\frac{T_{rsa}\lambda}{2(1-T_{rsa}\lambda)}\right) \tag{11}$$

Figure 7 (i.e. T_t=8) illustrates the comparison of the mean response time of the batching schemes with the non-batching scheme. The vertical axis in each graph is the mean response time over batch size divided by the mean response time with non-batching scheme. Refering to Table.1, it is shown that the optimal batch size is equal to 12 when λ=30. The speedup of mean response time is an optimal one which equals to 5.23 approximately. It is clear that with the optimal batch size the batching system has considerable advantages whereas costs little.



(a) λ=2 (b) λ=10 (c) λ=80

Fig.5 Mean response time validation over batch size

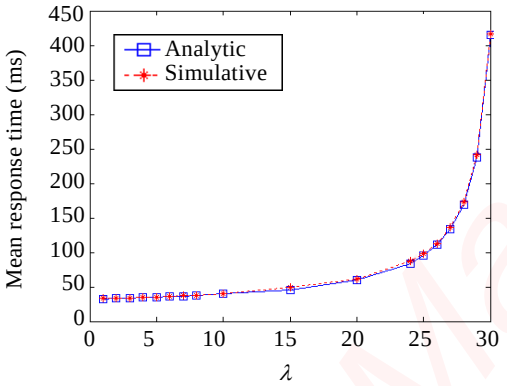


Fig.6 Mean response time for non-batching scheme

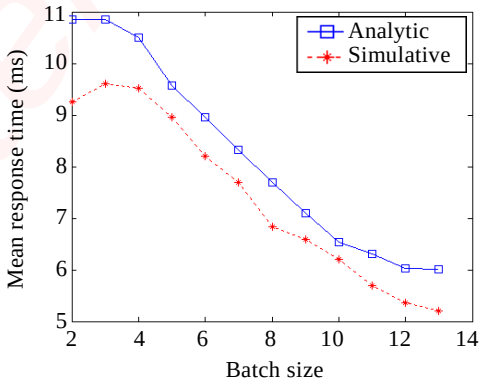


Fig.7 Mean response time speedup against non-batching

6 Conclusions

In conclusion, this paper proposes a novel method of assigning the set of public exponent e_i only using unique certificate in batching TLS handshake protocol. This paper also optimizes the batch size by developing an analytical model which satisfies the stability conditions of the system for the batching TLS handshake. The novel optimal batch scheduling algorithm is employed in a batching web server which satisfies the clients' requirement of stability of system and tolerable waiting time. The parameter optimization-based batching TLS handshake is a viable option for secure communications.

References:

[1] Sun LH, Ye DF, Lü SW, Feng DG. Security analysis and improvement of TLS. Journal of Software, 2003,14(3):518−523 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/518.htm

[2] Fiat A. Batch RSA. Journal of Cryptology, 1997,10(2):75−88.

[3] Shacham H, Boneh D. Improving SSL handshake performance via batching. In: Proc. of the RSA 2001. LNCS 2020, San Francisco: Spring-Verlag, 2001. 28–43.

[4] Lin C. Performance Evaluation of Computer Network and Computer System. Beijing: Tsinghua University Press, 2001. 26–65 (in Chinese).

[5] Cheng WC, Chou CF, Golubchik L. Performance of batch-based digital signatures. In: Proc. of the 10th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Fort Worth: IEEE Computer Society, 2002. 291–299.

[6] Vuillaume C. Side channel attacks on elliptic curve cryptosystems [MS Thesis]. Berlin: Technological University of Berlin, 2004.

[7] Shan ZG, Lin C, Xiao RY, Yang Y. Web quality of service: Survey. Chinese Journal of Computers, 2004,27(2):145–156 (in Chinese with English abstract).

附中文参考文献:

[1] 孙林红,叶顶锋,吕述望,冯登国.传输层安全协议的安全性分析及改进.软件学报,2003,14(3):518–523. http://www.jos.org.cn/1000-9825/14/518.htm

[4] 林闯.计算机网络和计算机系统的性能评价.北京:清华大学出版社,2001.26–65.

[7] 单志光,林闯,肖人毅,杨扬.Web QoS 控制研究综述.计算机学报,2004,27(2):145–156.

QI Fang was born in 1978. She is a Ph.D. candidate at Central South University. Her current research areas are network security, mobile computing and wireless communications.

JIA Wei-Jia was born in 1957. He is a professor and doctoral supervisor at Central South University and a CCF Senior member. He is also associate professor at City University of Hong Kong. His research areas are designing routing protocols, wireless communications, mobile computing, parallel and distributed computing.

BAO Feng was born in 1962. He is a principal scientist in Singapore. He is also the manager of security department of the Institute for Infocomm Research. His current research areas are cryptography and information security.

WU Yong-Dong was born in 1970. He is a research scientist and the head of information security laboratory of the Institute for Infocomm Research. His current research areas are multimedia security, network security and digital right management.

WANG Guo-Jun was born in 1970. He is a professor and doctoral supervisor at Central South University and a CCF senior member. His research areas are computer networks, fault tolerant and dependable computing and software engineering.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究
65. 具有 MVB 接口的输入输出设备的分析
66. 基于 STM32 的 GSM 模块综合应用
67. 基于 ARM7 的 MVB CAN 网关设计
68. 机车车辆的 MVB CAN 总线网关设计
69. 智能变电站冗余网络中 IEEE1588 协议的应用
70. CAN 总线的浅析 CANopen 协议
71. 基于 CANopen 协议实现多电机系统实时控制
72. 以太网时钟同步协议的研究
73. 基于 CANopen 的列车通信网络实现研究
74. 基于 SJA1000 的 CAN 总线智能控制系统设计
75. 基于 CANopen 的运动控制单元的设计
76. 基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现

77. 分布式控制系统精确时钟同步技术
78. 基于 IEEE 1588 的时钟同步技术在分布式系统中应用
79. 基于 SJA1000 的 CAN 总线通讯模块的实现
80. 嵌入式设备的精确时钟同步技术的研究与实现
81. 基于 SJA1000 的 CAN 网桥设计
82. 基于 CAN 总线分布式温室监控系统的设计与实现
83. 基于 DSP 的 CANopen 通讯协议的实现
84. 基于 PCI9656 控制芯片的高速网卡 DMA 设计
85. 基于以太网及串口的数据采集模块设计
86. MVB1 类设备控制器的 FPGA 设计
87. MVB 接口彩色液晶显示诊断单元的显示应用软件设计
88. IPv6 新型套接字的网络编程剖析
89. 基于规则的 IPv4 源程序到 IPv6 源程序的移植方法
90. MVB 网络接口单元的 SOC 解决方案
91. 基于 IPSec 协议的 IPv6 安全研究
92. 具有 VME 总线的车载安全计算机 MVB 通信板卡
93. SD 卡的传输协议和读写程序
94. 基于 SCTP 的 TLS 应用

VxWorks:

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动

18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现
43. 实时操作系统 VxWorks 在微机保护中的应用
44. 基于 VxWorks 的多任务程序设计及通信管理
45. 基于 Tilcon 的 VxWorks 图形界面开发技术
46. 嵌入式图形系统 Tilcon 及应用研究
47. 基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现
48. 基于嵌入式的 Tilcon 用户图形界面设计与开发
49. 基于 Tilcon 的交互式多页面的设计
50. 基于 Tilcon 的嵌入式系统人机界面开发技术
51. 基于 Tilcon 的指控系统多任务人机交互软件设计
52. 基于 Tilcon 航海标绘台界面设计
53. 基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发
54. VxWorks 环境下内存文件系统的应用
55. VxWorks 下的多重定时器设计
56. Freescale 的 MPC8641D 的 VxWorks BSP
57. VxWorks 实验五[时间片轮转调度]
58. 解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题
59. 基于 VxWorks 系统的 MiniGUI 图形界面开发

60. VxWorks BSP 开发中的 PCI 配置方法
61. VxWorks 在 S3C2410 上的 BSP 设计
62. VxWorks 操作系统中 PCI 总线驱动程序的设计与实现
63. VxWorks 概述
64. 基于 AT91RM9200 的 VxWorks END 网络驱动开发
65. 基于 EBD9200 的 VxWorks BSP 设计和实现
66. 基于 VxWorks 的 BSP 技术分析
67. ARM LPC2210 的 VxWorks BSP 源码
68. 基于 LPC2210 的 VxWorks BSP 移植
69. 基于 VxWorks 平台的 SCTP 协议软件设计实现
70. VxWorks 快速启动的实现方法[上电到应用程序 1 秒]

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究

26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略
52. Linux 基于 W83697 和 W83977 的 UART 串口驱动开发文档
53. 基于 AT91RM9200 的嵌入式 Linux 系统的移植与实现
54. 路由信息协议在 Linux 平台上的实现
55. Linux 下 IPv6 高级路由器的实现

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码

7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现
28. 基于 XPE 的数字残币兑换工具开发
29. Windows CENET 下 ADC 驱动开发设计
30. Windows CE 下 USB 设备流驱动开发与设计
31. Windows 驱动程序设计
32. 基于 Windows CE 的 GPS 应用
33. 基于 Windows CE 下大像素图像分块显示算法的研究
34. 基于 Windows CE 的数控软件开发与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置

8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现
27. 基于 PowerPC 和 DSP 的电能质量在线监测装置的研制
28. 基于 PowerPC 架构多核处理器嵌入式系统硬件设计
29. 基于 PowerPC 的多屏系统设计
30. 基于 PowerPC 的嵌入式 SMP 系统设计
31. 基于 MPC850 的多功能通信管理器
32. 基于 MPC8640D 处理系统的技术研究
33. 基于双核 MPC8641D 处理器的计算机模块设计
34. 基于 MPC8641D 处理器的对称多处理技术研究

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码

9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现
36. UBoot 在 AT91RM9200 上的移植简析
37. 基于工控级 AT91RM9200 开发板的 UBoot 移植分析

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备

8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法
34. 基于 GPU 的瑕疵检测
35. 基于 GPU 通用计算的分析与研究
36. 面向 OpenCL 架构的 GPGPU 量化性能模型
37. 基于 OpenCL 的图像积分图算法优化研究
38. 基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究
39. 基于 OpenCL 的异构系统并行编程
40. 嵌入式系统中热备份双机切换技术研究
41. EFI-Tiano 环境下的 AES 算法应用模型
42. EFI 及其安全性研究
43. 基于 UEFI Shell 的 PreOS Application 的开发与研究

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案
10. 基于小波变换与偏微分方程的图像分解及边缘检测
11. 基于图像能量的布匹瑕疵检测方法
12. 基于 OpenCL 的拉普拉斯图像增强算法优化研究
13. 异构平台上基于 OpenCL 的 FFT 实现与优化
14. 数据结构考题 - 第 4 章 串
15. 数据结构考题 - 第 4 章 串答案
16. 用 IPv6 编程接口实现有连接通信的方法

FPGA / CPLD:

1. 一种基于并行处理器的快速车道线检测系统及 FPGA 实现
2. 基于 FPGA 和 DSP 的 DBF 实现
3. 高速浮点运算单元的 FPGA 实现
4. DLMS 算法的脉动阵结构设计及 FPGA 实现
5. 一种基于 FPGA 的 3DES 加密算法实现
6. 可编程 FIR 滤波器的 FPGA 实现
7. 基于 FPGA 的 AES 加密算法的高速实现
8. 基于 FPGA 的精确时钟同步方法
9. 应用分布式算法在 FPGA 平台实现 FIR 低通滤波器
10. 流水线技术在用 FPGA 实现高速 DSP 运算中的应用
11. 基于 FPGA 的 CAN 总线通信节点设计
12. 基于 FPGA 的高速时钟数据恢复电路的实现
13. 基于 FPGA 的高阶高速 FIR 滤波器设计与实现
14. 基于 FPGA 高效实现 FIR 滤波器的研究
15. FPGA 的 VHDL 设计策略
16. 用 FPGA 实现串口通信的设计
17. GPIB 接口的 FPGA 实现