ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.9, September 2009, pp.2558−2573
doi: 10.3724/SP.J.1001.2009.00575

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种抵御拒绝服务攻击的自适应客户端难题*

陈瑞川[1,2], 郭文嘉[1,2], 唐礼勇[1,2+], 陈 钟[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

Adaptive Client Puzzle Scheme Against Denial-of-Service Attacks

CHEN Rui-Chuan[1,2], GUO Wen-Jia[1,2], TANG Li-Yong[1,2+], CHEN Zhong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: tly@infosec.pku.edu.cn, http://infosec.pku.edu.cn/

Chen RC, Guo WJ, Tang LY, Chen Z. Adaptive client puzzle scheme against denial-of-service attacks. *Journal of Software*, 2009,20(9):2558−2573. http://www.jos.org.cn/1000-9825/575.htm

Abstract: This paper studies the traditional client puzzle scheme and proposes an adaptive scheme which performs a lightweight client-server interaction to flexibly adjust the puzzle difficulty according to the real-time statuses of both client and server. To evaluate the applicability, the authors combine the two schemes and develop an adaptive DoS-resistant security framework for Peer-to-Peer networks. The theoretical analyses and experimental results show that the adaptive client puzzle scheme can effectively defend against various DoS attacks without significantly influencing legitimate clients' experiences even in a highly malicious environment.

Key words: denial-of-service attack; client puzzle; peer-to-peer network

摘 要: 研究传统的客户端难题方案,之后提出一种自适应客户端难题方案.该方案采用一种轻量级的协议交互方式来获取客户端和服务器双方的实时状态信息,并据此自适应地调整客户端难题的难度.为了评估该方案的适用性,结合传统和自适应两种客户端难题方案,在对等(P2P)网络中提出了一种抵御 DoS 攻击的自适应安全框架.理论分析和实验结果表明,甚至在高度恶意的网络环境中,自适应客户端难题方案都可以在不明显影响合法客户端性能的前提下有效地抵御各种 DoS 攻击.

关键词: 拒绝服务攻击;客户端难题;对等网络

中图法分类号: TP393 **文献标识码**: A

1 Introduction

Denial-of-Service (DoS) attacks aiming at exhausting a target host's resources have become a major security threat to the Internet. As a large number of incidents have illustrated, even a powerful host may be vulnerable to

* Supported by the National Natural Science Foundation of China under Grant No.60773163 (国家自然科学基金)
Received 2008-02-20; Accepted 2008-12-10

DoS attacks.

In general, on the Internet, a DoS attack is an attack in which one or more hosts attempt to thwart genuine hosts from having access to legitimate services[1]. For instance, when a DoS attacker overloads a host through initiating a large number of requests, the victim will consume its resources (e.g., memory space, computational resources and network bandwidth) and deny its services to genuine hosts. In most cases, sufficient scale can be achieved by compromising enough hosts and using those compromised hosts to perpetrate the attack. Such an attack is known as a Distributed Denial-of-Service (DDoS) attack.

Currently, several countermeasures have been proposed. Among them, the client puzzle scheme is an intriguing solution, and it is particularly well-suited for defending against DoS attacks[2]. Generally, in the scheme, a client has to compute a moderately expensive but not intractable puzzle in order to gain access to the resources allocated by a server. This prevents a DoS attacker from consuming a large portion of the target server's resources without investing considerable resources. Specifically, the computation cost for the client should be much higher than the generation and verification costs for the server; moreover, the client's computation cost, i.e., the puzzle difficulty, should be effortlessly adjusted by the server.

In this paper, we first study the traditional client puzzle scheme[3], and then we propose a novel adaptive client puzzle scheme which performs a lightweight client-server interaction to adjust the puzzle difficulty. Specifically, our scheme takes not only server status but also client status and time factor into consideration. That is, our scheme has the capacity of adaptively adjusting the puzzle difficulty based on the real-time statuses of both client and server. To evaluate the applicability, we further combine the adaptive client puzzle scheme with the traditional scheme, and develop an integrated DoS-resistant security framework for Peer-to-Peer (P2P) networks to defeat various DoS attacks including IP spoofing attack, reflection attack, replay attack and pre-computation attack. Both theoretical analyses and experimental results demonstrate that the adaptive client puzzle scheme can effectively handle various DoS attacks without significantly influencing legitimate clients' experiences even in a highly malicious environment.

The rest of this paper is organized as follows. Section 2 gives an overview of related work. We describe the environmental assumptions in Section 3. The traditional and the adaptive client puzzle schemes are elaborated in Section 4. We then specify the details of the adaptive DoS-resistant security framework for P2P networks in Section 5. The simulation methodology and performance evaluation are presented in Section 6. Finally, we conclude and describe the future work in Section 7.

2 Related Work

To the best of our knowledge, Merkle was the first to come up with the idea of client puzzles. However, he only applied puzzles for key agreement rather than access control[4]. Then, Dwork and Naor presented client puzzles as a general solution to regulate junk mails[5]. Afterwards, Juels and Brainard utilized client puzzles to handle TCP SYN flooding[6], and Aura, *et al*. applied client puzzles to authentication protocols in general[3]. However, most of the above schemes merely consider the server status, so they cannot reflect the network environment completely. Recently, a novel client puzzle scheme, Portcullis[2], was proposed. In Portcullis, since a server gives priority to requests containing puzzles with higher difficulty levels to gain access to the requested resources, each client—no matter legitimate or malicious—has to compete with each other and solve difficult puzzles under attacks. This may influence legitimate clients' experiences significantly. Compared with the existing puzzle schemes, our adaptive client puzzle scheme treats each client distinctively by performing a lightweight interaction to flexibly adjust the puzzle difficulty according to the real-time statuses of network environment. In our scheme, each legitimate client

only needs to compute an easy puzzle, while a malicious client has to compute a very hard puzzle. This guarantees that our adaptive client puzzle scheme does not influence legitimate clients' experiences significantly, and it also prevents a malicious client from attacking the server without investing considerable resources.

Besides the basic CPU-bound client puzzle, some other forms of client puzzles are actively studied, such as memory-bound puzzle[7], time-lock puzzle[8], threshold puzzle[9] and puzzle auction[10].

3 Environmental Assumptions

For convenience, if host *i* requests the resources of host *j* in a transaction, we define host *i* and host *j* as *client* and *server* respectively in this transaction. Then, we can make several environmental assumptions. Specifically, the first two assumptions describe the properties of server and client respectively, and the other two specify the capabilities of attackers.

- Each server has ample network bandwidth, and no attacker can saturate a server by simply issuing a large number of messages. Whatever happens, the server is at least capable of rejecting the incoming messages and sending reply;
- The legitimate clients seeking access to a heavily loaded server would like to perform a moderately expensive computation;
- Attackers cannot modify, delay or drop the messages transmitted between any legitimate client and the associated server. For instance, if attackers have the capacity of tampering with messages, they can launch DoS attacks by simply corrupting these messages;
- Attackers can perform IP spoofing, moreover, they are able to eavesdrop or replay any message sent between client and server.

In particular, the first environmental assumption seemingly limits the application field since many DoS attacks are characterized by flooding a server with huge amounts of requests; however, we can defeat such DoS attacks by coordinating multiple routers to check the attack flows before they converge to the server. A thorough investigation of the router coordination is important, but it is beyond the scope of this paper.

4 Client Puzzle Schemes

In this section, we first describe a set of desirable properties that should be possessed by a good client puzzle. Then, we study the traditional client puzzle scheme (abbr., TCPS), and propose a novel adaptive client puzzle scheme (abbr., ACPS). Finally, we analytically compare the two schemes under various typical DoS attacks.

4.1 Puzzle poperties

As described in previous studies[3,6,11], regardless of the specific implementation, a good client puzzle should have the following fundamental properties:

- Generating a puzzle and verifying the solution are inexpensive for the server, i.e., these operations must avoid compromising the server's resources;
- The puzzle difficulty can be effortlessly adjusted from zero to impossible;
- Having solved many other puzzles does not help the client solve new given puzzles;
- The puzzle must be time-dependent so that the client has only a limited time to solve the puzzle;
- The puzzle must be able to be verified in a stateless way, i.e., while the client is solving a puzzle, the server does not need to store any session-specific data.

4.2 Traditional client puzzle scheme (TCPS)

4.2.1 Design rationale

To generate a traditional puzzle, the server periodically creates a short-term nonce N_S to limit the time that clients have for computing puzzle solutions. Specifically, the nonce should not be a predictable value and should have at least 64 bits of entropy. This entropy guarantees that an attacker usually does not have enough resources to create a database by pre-computing all the ⟨nonce, result⟩ pairs; furthermore, it ensures that the occasional matches caused by birthday attacks are not very harmful. Besides, the server needs to determine the puzzle difficulty k based on its current status (described in Section 4.2.2). Consequently, the puzzle broadcasted to clients is the $\langle N_S, k\rangle$ pair.

To solve the puzzle, the client first generates a nonce N_C, and then performs a brute-force search to find the 32-bit solution X of the following equation:

$$h(IP_C \mid N_S \mid N_C \mid X) = Y^{(k)} \tag{1}$$

where

- h: A cryptographic hash function, e.g., MD5 or SHA1;
- IP_C: The client's IP address;
- N_S and N_C: Two nonces generated by the server and the client, respectively;
- X and k: The solution and the difficulty of the puzzle;
- $Y^{(k)}$: A hash value with the first k bits being equal to 0.

Somewhat interestingly, the client can reuse the latest puzzle by generating a new N_C. This is a favorable characteristic to enhance the system performance.

To gain access to the server's resources, the client should subsequently submit a message, including the client nonce N_C and the puzzle solution X, to the server. Then, the server needs to check the received message. As long as the server accepts the solution, it must keep book of the correctly solved puzzle instance in the form of $\langle IP_C, N_S, N_C\rangle$, where the client's IP address IP_C can be extracted from the IP header of the received message. This booking operation ensures that the solution cannot be replayed.

4.2.2 Adjusting the puzzle difficulty

When a server comes under attacks, it imposes the computational loads on the clients by distributing puzzles to these clients. On one hand, puzzles cannot be so hard that legitimate clients will experience a harmful degradation of services; on the other hand, puzzles cannot be too simple since attackers can rapidly solve the puzzles and compromise the server's resources. Generally, TCPS parameterizes the puzzle difficulty according to the server status, i.e., the ratio of consumed resources to total resources possessed by the server. The more resources the server consumes, the harder puzzles it will issue in the future. The actual generation of puzzle difficulty k is described as follows:

$$k = \min\left\{\alpha \times \underbrace{\left(\frac{R_{consumed}}{R_{total}}\right)^{\beta}}_{Server\ Status}, k_{\max}\right\} \tag{2}$$

where

- k: The puzzle difficulty;
- α and β: Two positive parameters to tune the puzzle difficulty k into an appropriate interval;
- $R_{consumed}$ and R_{total}: The consumed resources and the total resources possessed by the server, respectively;
- $k_{\max}$: The difficulty of a puzzle which is impossible to be solved in a limited period of time.

4.2.3 Scheme analysis

TCPS satisfies all the fundamental properties of a good puzzle[3]. However, this scheme can only periodically adjust the unified puzzle difficulty based on the server status, and it cannot distribute puzzles with different difficulties to different clients according to the real-time statuses of both client and server.

In real-world networks, each host has its unique property, e.g., the properties of legitimate and malicious hosts are completely different from each other. Therefore, we should independently tune each puzzle's difficulty appropriately to defeat the attacks and minimize the legitimate clients' costs. In the next subsection, we propose an adaptive client puzzle scheme that can flexibly adjust the puzzle difficulty.

4.3 Adaptive client puzzle scheme (ACPS)

4.3.1 Design rationale

Since we are interested in how messages are processed as well as what messages are sent, for the sake of clarity and simplicity, we utilize the annotated Alice-and-Bob specification[12] to describe the ACPS. As shown in Fig.1, the scheme performs a lightweight client-server interaction as follows:

Stage 1: If a client C wants to gain access to the server S' resources, the client first generates a 64-bit nonce N_C as its session identifier SI_C. Then, the client stores the session identifier locally, and sends it to the server.

Stage 2: On receiving the message consisting of SI_C sent by the client C, the server S determines the puzzle difficulty k based on the real-time statuses of both client and server (described in section 4.3.2). Subsequently, the server generates its 64-bit session identifier SI_S according to the client's IP address IP_C (extracted from the IP header of the received message), the client's session identifier SI_C and the puzzle difficulty k. The detailed generation process is specified as follows:

$$SI_S = HMAC_{secret}(IP_C \mid SI_C \mid k) \tag{3}$$

where

- SI_S: The server's session identifier;
- *HMAC*: A keyed hash function for message authentication[13];
- *secret*: A 32-bit key which is periodically changed and only known to the server itself;
- IP_C and SI_C: The IP address and the session identifier of the client;
- k: The puzzle difficulty.

Note that, since a single IP address may represent a number of actual clients (e.g., clients behind a NAT), we should additionally take port number into consideration; however, for the sake of clarity, we omit the port number related information in the description of ACPS.

Stage 1 (***C*→*S***): generate SI_C, store SI_C ||SI_C|| (*null*)
Stage 2 (***S*→*C***): generate k, generate SI_S ||SI_C,SI_S,k|| check SI_C, store SI_S
Stage 3 (***C*→*S***): retrieve ⟨SI_C,SI_S⟩, solve *puzzle* ||SI_C,SI_S,k,*solution*,*request*|| check ⟨SI_C,SI_S⟩, check SI_S, check *solution*, store ⟨SI_C,SI_S⟩, process *request*

where

- $\boldsymbol{SI_C}$: The client's session identifier;
- $\boldsymbol{SI_S}$: The server's session identifier;
- $\boldsymbol{k}$: The puzzle difficulty.

Fig.1 Adaptive client puzzle scheme

After the above generation process, the server replies to the client at IP_C with the client's session identifier SI_C, the server's session identifier SI_S and the puzzle difficulty k. Once the client has received this reply message, it first checks whether the received SI_C is really generated by itself. If the received SI_C is bogus, the client simply drops the message; otherwise, the client stores the server's session identifier SI_S immediately. Such replies and checking

operations could enhance the robustness of ACPS under IP spoofing attacks and reflection attacks.

Stage 3: The client C retrieves the $\langle SI_C,SI_S\rangle$ pair as the global session identifier, and then it tries to solve the puzzle according to the following equation by brute force:

$$h(SI_C \mid SI_S \mid X) = Y^{(k)} \tag{4}$$

where

- h: A cryptographic hash function, e.g., MD5 or SHA1;
- SI_C and SI_S: Session identifiers of the client and the server, respectively;
- X and k: The *solution* and the difficulty of the puzzle;
- $Y^{(k)}$: A hash value with the first k bits being equal to 0.

After the brute-force computation, the client sends the server a message including the global session identifier (i.e., the $\langle SI_C,SI_S\rangle$ pair), the puzzle difficulty, the puzzle solution and the actual *request*. Once the server has received this message, it should perform the following operations in turn:

- Check whether the global session identifier $\langle SI_C,SI_S\rangle$ is really fresh based on the database of the past global session identifiers;
- Check whether the server's session identifier SI_S can be correctly generated according to Eq.(3). Specifically, this operation can additionally check whether the difficulty k reported by the client is the original k determined by the server;
- Check whether the puzzle solution is correct according to Eq.(4);
- Store the global session identifier $\langle SI_C,SI_S\rangle$, and process the request submitted by the client.

Note that, in the sequence of operations, if one operation succeeds, the server continues to perform the next; otherwise, the server cancels all the following operations, and the entire transaction ends.

4.3.2 Adjusting the puzzle difficulty

The characteristic part in ACPS is the method adaptively adjusting the puzzle difficulty based on the real-time statuses of both client and server. Besides the *server status* which has already been involved in TCPS, we consider two more factors: *client status* and *time*.

Client status: Firstly, most of the legitimate clients should merely perform a simple computation to access the requested resources. Secondly, according to the second environmental assumption, some legitimate clients which frequently gain access to the server's resources (i.e., these legitimate clients have solved a relatively large number of puzzles) should perform a moderately expensive computation. Thirdly, according to the first environmental assumption, each server has sufficient network bandwidth, and the attackers cannot overwhelm a server simply by flooding the server with a sheer volume of messages. That is, in order to launch a DoS attack effectively, the attackers have no alternative but to solve a large number of puzzles and send these puzzle solutions to force the server to perform lots of expensive operations. Therefore, the more puzzles a client has solved in the latest period of time, the higher the probability that this activity is malicious and DoS-like, thus the harder puzzles the server issues to the client in the future; whereas the pattern is reversed. Note that, since an attacker could simply spoof its IP address, in order to effectively utilize the client status, our client puzzle scheme should have the capability of defending against IP spoofing attacks, which we have described in Section 4.3.1, and will further analyze in Section 4.4.

Time: As described in the previous subsection, the server periodically changes its secret. We define the latest secret change-point as T_{start}, and the forthcoming secret change-point as T_{end}. In real-world networks, attackers can acquire many puzzles from a server even a little after T_{start}, and then they manage to solve these puzzles as many as possible, finally at some time before T_{end}, the attackers submit all the puzzle solutions to overload the server. This is

a typical kind of pre-computation attacks. To counter such attacks, we distribute harder puzzles if the current time point is closer to T_{start}; otherwise, if the current time is closer to T_{end}, the pattern is reversed.

Combining the three factors, i.e., server status, client status and time, the difficulty of a puzzle which is distributed to a specific client at the current time can be calculated from the following equation:

$$k_i^{now} = \min\left\{\alpha \times \underbrace{\left(\frac{R_{consumed}}{R_{total}}\right)^{\beta}}_{Server\ Status} \times \underbrace{\left(\frac{N_i+1}{N_{total}+1}\right)^{\gamma}}_{Client\ Status} \times \underbrace{\left(\frac{T_{end}-T_{now}}{T_{end}-T_{start}}\right)^{\delta}}_{Time}, k_{\max}\right\} \quad (5)$$

where

- T_{now}: The current time point;
- k_i^{now}: The difficulty of the puzzle which is distributed to client i at T_{now};
- α, β, γ and δ: Four positive parameters to weigh various factors, and to tune the puzzle difficulty into an appropriate interval;
- $R_{consumed}$ and R_{total}: The consumed resources and the total resources possessed by the server;
- T_{start} and T_{end}: The latest and the forthcoming secret change-points;
- N_i: The count of puzzles which have been solved by client i during the latest time period $[T_{now}-T_{period}, T_{now}]$, where $T_{period}=T_{end}-T_{start}$;
- N_{total}: The total count of puzzles which have been solved by all the clients during the latest time period $[T_{now}-T_{period},T_{now}]$;
- $k_{\max}$: The difficulty of a puzzle which is impossible to be solved in a limited period of time.

In real-world networks, clients' computation capabilities vary a lot, e.g., the time to solve a puzzle will be much different between a client with multiple fast CPUs and a client with just one slow CPU. To decrease the computational disparity, some other kinds of puzzles, such as memory-bound puzzle[7] and time-lock puzzle[8], could be complementary to our scheme.

4.3.3 Scheme analysis

In ACPS, the server only needs to perform one arithmetic operation and one HMAC operation to generate a new puzzle; furthermore, it can merely execute a one-way hash function to verify the puzzle solution in a constant time. Therefore, ACPS satisfies the first fundamental property of a good puzzle.

To solve a puzzle, the client has no way to figure out the solution other than brute-force searching the solution space until a solution is found. The cost of solving the puzzle depends exponentially on the puzzle difficulty k, which can be effortlessly adjusted by the server. If k=0, no operation is required; if $k=k_{\max}$, the client needs to perform a brute-force search for $k_{\max}$ bits of the inverse of the hash function, which is computationally impossible in a limited period of time. Moreover, due to the essential feature of hash function, the client cannot solve the puzzle more efficiently with many other solved puzzles. Hence, the second and the third fundamental properties of a good puzzle are also satisfied.

Since the server secret is periodically changed and the time factor plays an important role in determining the puzzle difficulty, ACPS is time-dependent. Besides, in ACPS, the server stores the session-specific data and processes the actual request only after it has verified the client's solution to the puzzle. That is, the server does not commit its resources until the client has demonstrated the sincerity. Consequently, ACPS satisfies the last two fundamental properties.

To sum up, ACPS has satisfied all the fundamental properties of a good puzzle. Compared with TCPS, the ACPS can distribute different puzzles to different clients by adaptively adjusting each puzzle's difficulty based on

the real-time statuses of both client and server. This enhances the effectiveness of ACPS in the malicious environment. Besides, ACPS needs to perform a lightweight client-server interaction to adjust the puzzle difficulty. This interaction slightly reduces the system efficiency; however, it is worthwhile for the security enhancement it brings.

4.4 Analytical comparison under various DoS attacks

Currently, various DoS attacks have been found existing in real-world networks, such as IP spoofing attack, reflection attack, replay attack and pre-computation attack.

Clearly, since TCPS does not verify whether the client's IP address and the client nonce are genuine, it may seriously suffer from IP spoofing attacks and reflection attacks. On the contrary, as shown in Fig.1, ACPS checks the validity of the client's session identifier in the second stage. This checking operation can effectively defend against IP spoofing attacks and reflection attacks.

To counter replay attacks, ACPS checks the freshness of the global session identifier $\langle SI_C, SI_S \rangle$ in the third stage; while TCPS keeps book of the correctly solved puzzle instances to defeat such attacks. That is, both the two schemes are able to defend against replay attacks. Furthermore, the two schemes also use those periodically changed data, such as the server nonce and server secret, to mitigate pre-computation attacks. Specifically, ACPS additionally makes use of client status and time to adaptively adjust the puzzle difficulty to cope with pre-computation attacks without significantly influencing legitimate clients' experiences, but TCPS can only broadcast a puzzle with the unified difficulty to all clients. Therefore, ACPS can handle pre-computation attacks more effectively.

In summary, ACPS can perform more effectively than TCPS under various DoS attacks.

5 Case Study

P2P computing has emerged as a popular model aiming at further utilizing Internet resources, and goes beyond services offered by the traditional client-server model. Nowadays, the use of P2P applications is growing dramatically, and meanwhile, the P2P traffic has become the major traffic over the Internet[14]. However, due to the decentralized and unauthenticated nature, each participating peer has to manage the risks involved in the transactions without adequate experience and knowledge about other peers. This enables malicious peers to misuse resources and mount DoS attacks against arbitrary peers.

In terms of both the number of participating peers and the traffic volume, KaZaA[15] is one of the most important P2P networks on the Internet today. To evaluate the applicability of ACPS representatively, we utilize ACPS complementary with TCPS to develop an adaptive DoS-resistant security framework for KaZaA overlay network (abbr., ADSF).

5.1 KaZaA overlay network

In KaZaA, as shown in Fig.2, the overlay network consists of *ordinary peers* and *super peers*. Each ordinary peer provides several shared files, and transmits the meta-data of these files to a super peer. Some powerful ordinary peers with high bandwidth, enough disk space, strong processing power and sufficient uptime can be selected as super peers to facilitate searching by caching the meta-data. Each super peer covers a set of ordinary peers, and acts as a Napster-like[16] proxy to the KaZaA overlay network. A super peer and all the ordinary peers constitute a *cluster*. For searching a file, an ordinary peer issues a query to the associated super peer, and then the super peer performs a Gnutella-like[17] broadcasting in a highly pruned overlay network of super peers to lookup the requested file.

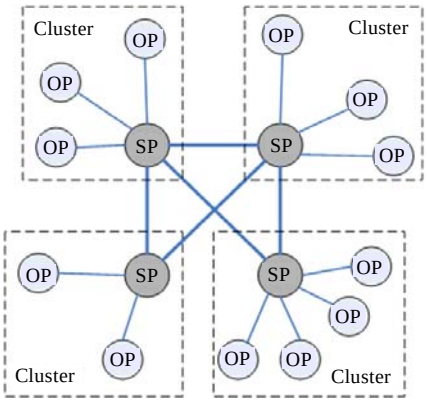


Fig.2 KaZaA overlay network, where SP and OP denote super peer and ordinary peer respectively

5.2 Adaptive DoS-resistant security framework for KaZaA overlay network (ADSF)

When there is no evidence of attack, a super peer accepts queries normally, that is, indiscriminately; otherwise, when a super peer comes under attacks, it should accept these queries selectively. As indicated in Fig.3, ADSF consists of two modules, i.e., *detection module* and *protection module*. Both modules are deployed at super peers for security enhancement.

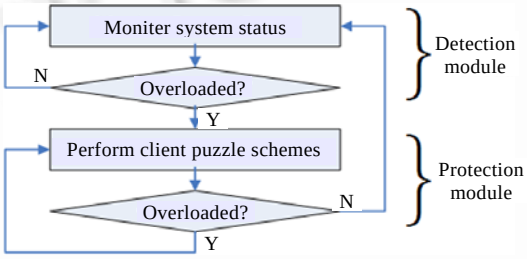


Fig.3 Modules of the adaptive DoS-resistant security framework for KaZaA overlay network

Detection module: This module is the front end of ADSF. It monitors the super peer's status, and adopts suitable response policy based on the system load of the super peer. Here, the *system load* is defined as the ratio of consumed resources to total resources possessed by the super peer, i.e.,

$$L_i = \frac{R^i_{consumed}}{R^i_{total}} \tag{6}$$

where

- L_i: The system load of super peer i;
- $R^i_{consumed}$ and R^i_{total} : The consumed resources and the total resources possessed by super peer i respectively.

If the system load of super peer i exceeds a predefined threshold Θ_{det}, the detection module deployed at super peer i considers that the super peer comes under DoS attacks, and it immediately triggers the protection module of the super peer.

Protection module: This module integrates both ACPS and TCPS to defend against various DoS attacks. Specifically, it adopts ACPS to handle the *intra-cluster queries*, and utilizes TCPS to cope with the *inter-cluster queries*.

- *Intra-Cluster queries*: These queries are generally transmitted between ordinary peers and super peers. If an ordinary peer wants to issue a query to the heavily loaded super peer, it has to initiate a transaction using ACPS. Since ACPS has the capacity of effectively defending against all kinds of typical DoS attacks, it is well-suited for the complicated intra-cluster environment;
- *Inter-Cluster queries*: This kind of queries must be efficiently performed between super peers to guarantee the system performance. To defeat inter-cluster DoS attacks, the heavily loaded super peer periodically distributes a traditional puzzle to all its neighboring super peers. Once a neighboring super peer wants to issue a query to the heavily loaded super peer, it has to solve the specific puzzle and submit the correct puzzle solution in a limited time interval. Sometimes, if a non-neighboring super peer expects to issue a query to the heavily loaded super peer, it should first initiate a request acquiring the specific puzzle; then, the non-neighboring super peer solves the puzzle and submits the solution as usual. Since the puzzle is reusable in a period of time, TCPS can efficiently handle a considerable number of queries; however, it cannot effectively defend against IP spoofing attacks, reflection attacks and pre-computation attacks launched by malicious super peers. To overcome this shortcoming, we should elect a super peer considering not only bandwidth, disk space, processing power and uptime, but also some other information reflecting peers' previous behaviors. In particular, we take the count of puzzles solved by a peer during a recent period of time into consideration.

During the working time of protection module, the super peer frequently checks its own status. If the system load falls below another predefined threshold Θ_{pro}, the protection module is terminated, and the detection module is triggered simultaneously. Commonly, $\Theta_{pro}<\Theta_{det}$, this prevents the super peer from triggering and terminating the protection module too often during continuous DoS attacks.

6 Experimental Evaluation

In this section, we first present the performance metrics, and then we describe the simulation setup of our following experiments, finally we evaluate the performance of ADSF and compare it with the performance of some other frameworks in suppressing various DoS attacks.

6.1 Performance metrics

A well-designed DoS-resistant security framework should seek to optimize its effectiveness under various DoS attacks. In our experiments, we characterize the system effectiveness by calculating the following two performance metrics:

Fraction of legitimate performed queries is defined as the ratio of legitimate queries, which are successfully performed by super peers, to all the performed queries. This metric reflects whether a framework can effectively filter out DoS attacks.

Fraction of failed legitimate queries is simply the failure rate of legitimate queries. It actually indicates whether a framework influences the legitimate peers' experiences.

6.2 Simulation setup

To evaluate the performance, we need to generate a KaZaA overlay network with various parameters, all of which should follow certain distributions. Specifically, all the following experiments are simulated on an OpenPower720 with four-way dual-core POWER5 CPUs and 16GB RAM running SLES 9.1.

Cluster size: In the experiments, we simulate 10 000 peers existing in the system. Previous measurement study implies that about 1% of all the participating peers are suitable to be super peers[15]. In our experiments, each

ordinary peer is attached uniformly at random to one of those super peers. That is, every cluster maintained by a super peer consists of about 100 ordinary peers.

Peer constitution: Since we have already revised the super peer selection algorithm by considering some information reflecting peers' previous behaviors as described in Section 5.2, it is relatively difficult for an attacker to be selected as a super peer. In our experiments, we assume that attackers take up 5% of all the super peers. However, the constitution of ordinary peers is more complicated, so we evaluate the system performance by changing the fraction of malicious ordinary peers.

Peer degree: P2P overlay topologies have the power law property[18]. In our experiments, peer degree, i.e., the count of a participating peer's inter-peer connections, obeys power law, and it ranges from 5 to 7.

Replication ratio of a file: The replication ratio of a file is proportional to the file's popularity, and it follows Zipf distribution[19]. In the following experiments, the replication ratio of a file follows Zipf distribution over [1‰, 5%] with Zipf parameter being equal to 0.8[20].

Count of files stored at each peer: A peer may share several files, and it follows the distribution as shown in Table 1[21].

Table 1 Count of files stored at each peer

Count of files	Percentage of peers (%)
0	25
[1,10]	20
[10,100]	30
[100,1000]	18
[1000,10000)	7

Query file frequency: Queries with a certain TTL (=4) for different files are initiated at random peers on the network topology, and the query frequency of a certain file is proportional to the count of the file's replicas[19].

Query frequency: Individual peers issue queries using a Poisson process with an average rate of 0.3 queries per minute, i.e., 12 805 unique IP addresses have issued 1 146 782 queries during 5 hours[22].

Resources of a super peer: Each super peer possesses 50 resources to perform queries; for the sake of simplicity, each performed query needs to occupy a resource for 20 seconds. Specifically, if a super peer's resources are exhausted, the super peer will discard all the newly incoming queries.

Other parameters: The change cycle of server nonce and server secret is 5 minutes; the two thresholds, Θ_{det} and Θ_{pro}, are set to 0.8 and 0.6, respectively.

6.3 Experiments

6.3.1 Computation cost

In this experiment, we simulate an attacker to solve the adaptive puzzle. Each simulation is run 1 000 times and the results of all runs are averaged. Note that, we use SHA1 as the cryptographic hash function h.

To solve an adaptive puzzle with difficulty k, a peer needs to perform a brute-force search to find the solution. Analytically, the cost of solving the puzzle depends exponentially on the puzzle difficulty k. The experimental results shown in Table 2 validate this analysis. Furthermore, these results also indicate that if the puzzle difficulty is lower than 18, the puzzle can be quickly solved; whereas, if the puzzle difficulty exceeds 26, a peer has to spend more than 24.728 seconds (71 810 216 SHA1 computations) in solving the specific puzzle. Such a long time prevents any attacker from launching a successful DoS attack. Thus, we set $k_{\max}$ used in Eq.(2) and Eq.(5) to 27, and tune the corresponding weight parameters to ensure that the puzzle difficulty varies between 18 and 27. Specifically, in the following experiments, α and β used in Eq.(2) are set to 27 and 1; α, β, γ and δ in Eq.(5) are set to 2 700, 1, 1 and 1, respectively.

Table 2 Computation cost

k	Minimum		Maximum		Average	
	Cycles	Time (s)	Cycles	Time (s)	Cycles	Time (s)
18	9	0.000	2 484 101	0.852	261 521	0.090
19	54	0.000	4 707 211	1.614	517 495	0.177
20	55	0.000	13 656 169	4.683	1 028 252	0.353
21	298	0.000	26 198 737	8.989	2 127 627	0.730
22	321	0.000	44 798 185	15.365	4 205 440	1.442
23	391	0.000	84 362 683	28.937	8 495 798	2.915
24	463	0.000	153 380 067	52.959	17 303 478	5.935
25	424	0.000	284 934 572	98.254	34 881 510	12.023
26	638	0.000	565 717 647	194.823	71 810 216	24.728

6.3.2 Performance evaluation and comparison

In the following experiments, we evaluate the performance of ADSF under various DoS attacks, and compare it with the performance of some other frameworks.

1) IP spoofing attack and reflection attack: Since the reflection attack is usually associated with IP spoofing attack, we combine the two attacks as a kind of mixed attack. Concretely, with 50% of all attacks being IP spoofing attacks and the other 50% being reflection attacks, we evaluate the system performance and compare it with the performance of TCPS, secure overlay services[23] (SOS) and non DoS-resistant framework (NONE). Especially, TCPS framework utilizes the traditional client puzzle scheme to cope with both intra-cluster and inter-cluster queries; the SOS framework allows all super peers to be its secure overlay access points (SOAPs), and all those SOAPs compose a Chord[24] ring to route the queries.

As shown in Fig.4, while the attack frequency is set to 5Hz, we simulate these four frameworks with the fraction of malicious ordinary peers changing from 0% to 30% in steps of 5% for each run of the experiment. The results show that TCPS suffers from a severe degradation of services, SOS can relatively effectively defeat IP spoofing attacks and reflection attacks, and the system without any DoS-resistant framework cannot survive under these two kinds of DoS attacks. Obviously, ADSF can defend against almost all these attacks no matter how many malicious ordinary peers exist in the network. The results also indicate that the performance of SOS is robust to the scale of malicious ordinary peers; however, it is drastically affected by the malicious super peers because even a minority of malicious super peers (=5% in our experiment) existing in the system can induce a serious performance degradation.

With the same parameters, we evaluate whether these frameworks influence legitimate peers' experiences by calculating the fraction of failed legitimate queries. Figure 5 indicates that ADSF can successfully perform almost all the legitimate queries (the small quantity of those failed legitimate queries are mostly due to the fact that some peers cannot solve the specific puzzles before the forthcoming change-point); TCPS has the capacity of performing more than 80% of all the legitimate queries; NONE denies almost all the legitimate queries, and it cannot work normally at all; SOS can perform most of legitimate queries when the fraction of malicious ordinary peers is lower than 20%, whereas its performance decreases rapidly when the fraction exceeds 20%.

Next, we fix the fraction of malicious ordinary peers to 10%, and calculate the fraction of legitimate performed queries with different attack frequencies. Figure 6 indicates that both ADSF and TCPS are robust to the change of attack frequency, and ADSF significantly outperforms TCPS; NONE cannot work normally at all. Somewhat interestingly, the experimental results additionally show that SOS can work well when the attack frequency is relatively low, but its performance decreases quickly with the growth of attack frequency. This phenomenon is due to the fact that SOS verifies the queries by utilizing IPsec or TLS which are considerably expensive and create new opportunities for attackers to launch DoS attacks.

Finally, we evaluate the fraction of failed legitimate queries with the same simulation parameters as described

in the previous experiment. The results shown in Fig.7 demonstrate that ADSF performs almost all the legitimate queries, and it is superior to TCPS; the performance of SOS decreases gradually with the increase of attack frequency; NONE works poorly under IP spoofing attacks and reflection attacks.

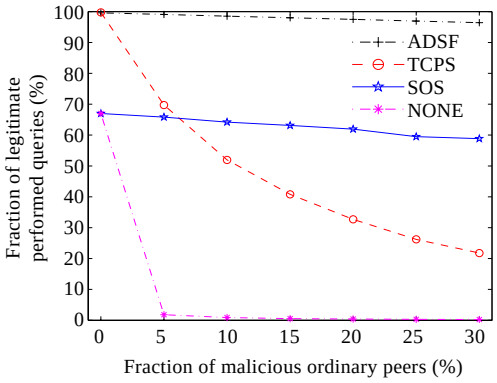


Fig.4 Fraction of legitimate performed queries vs. fraction of malicious ordinary peers, under IP spoofing attacks and reflection attacks (attack frequency=5Hz)

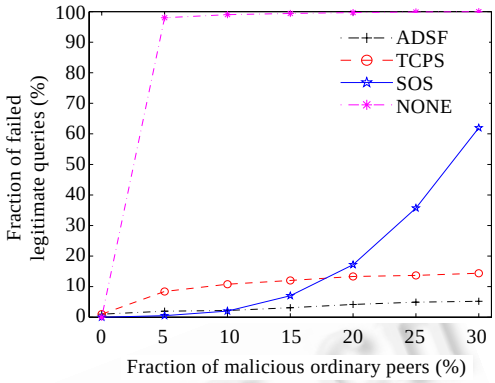


Fig.5 Fraction of failed legitimate queries vs. fraction of malicious ordinary peers, under IP spoofing attacks and reflection attacks (attack frequency=5Hz)

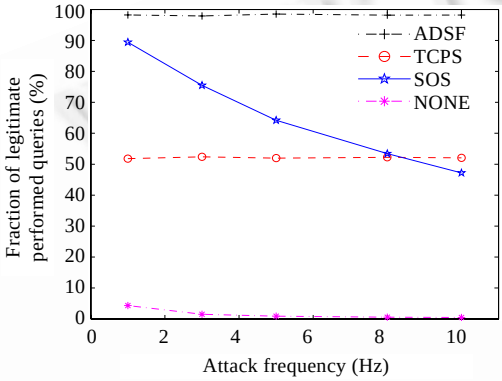


Fig.6 Fraction of legitimate performed queries vs. attack frequency, under IP spoofing attacks and reflection attacks (fraction of malicious ordinary peers=10%)

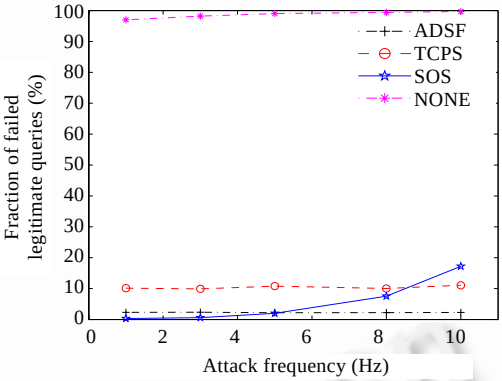


Fig.7 Fraction of failed legitimate queries vs. attack frequency, under IP spoofing attacks and reflection attacks (fraction of malicious ordinary peers=10%)

2) Replay attack and pre-computation attack: These two kinds of DoS attacks have the capacity of subverting client puzzle based DoS-resistant frameworks. In the following experiments, we further evaluate the effectiveness of ADSF under replay attacks and pre-computation attacks, and directly compare it with that of TCPS.

First, we simulate both frameworks under replay attacks with different fractions of malicious ordinary peers and different attack frequencies. The experimental results definitely prove that ADSF and TCPS can effectively defend against all the replay attacks without discarding the legitimate queries except for some overtime legitimate queries which are not submitted before the forthcoming change-point of server nonce or server secret. This validates our analysis described in Section 4.4.

Subsequently, we tune the attack frequency to 5Hz, and evaluate the fraction of legitimate performed queries

under pre-computation attacks. Fig.8 demonstrates that those pre-computation attacks influence the performance of both ADSF and TCPS; however, ADSF greatly outperforms TCPS, and it can still work well in the highly malicious environment because its fraction of legitimate performed queries exceeds 75% even with 30% of all the ordinary peers being attackers. This improvement is achieved because the ACPS applied by ADSF additionally utilizes client status and time to adjust the puzzle difficulty to mitigate pre-computation attacks.

Then, we fix the fraction of malicious ordinary peers to 10%, and evaluate whether ADSF and TCPS can effectively defend against pre-computation attacks with different attack frequencies. As shown in Fig.9, both the two frameworks are robust to the change of the frequency of pre-computation attacks, and ADSF can defend against pre-computation attacks more effectively.

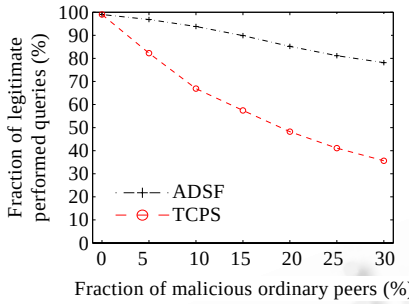


Fig.8 Fraction of legitimate performed queries vs. fraction of malicious ordinary peers, under pre-computation attacks (attack frequency=5Hz)

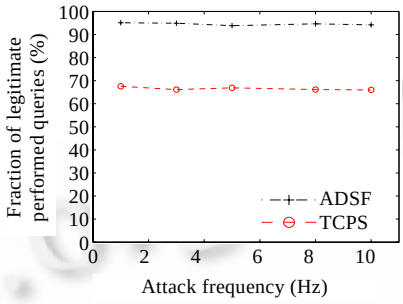


Fig.9 Fraction of legitimate performed queries vs. attack frequency, under pre-computation attacks (fraction of malicious ordinary peers=10%)

Finally, we evaluate the fraction of failed legitimate queries under pre-computation attacks. As shown in Fig.10 and Fig.11, the experimental results are similar to the results under IP spoofing attacks and reflection attacks, i.e., ADSF is superior to TCPS, and it always successfully performs more than 94% of all the legitimate queries. That is, ADSF does not significantly influence legitimate peers' experiences under pre-computation attacks.

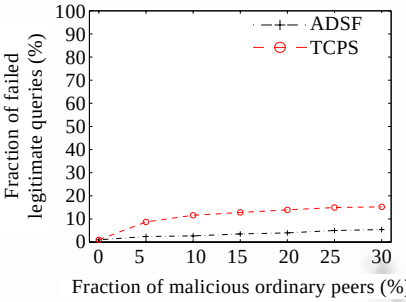


Fig.10 Fraction of failed legitimate queries vs. fraction of malicious ordinary peers, under pre-computation attacks (attack frequency=5Hz)

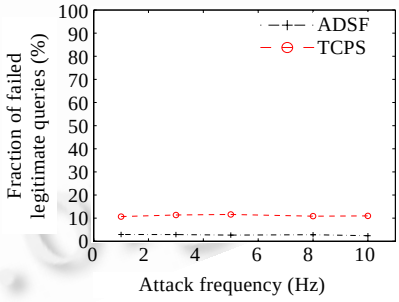


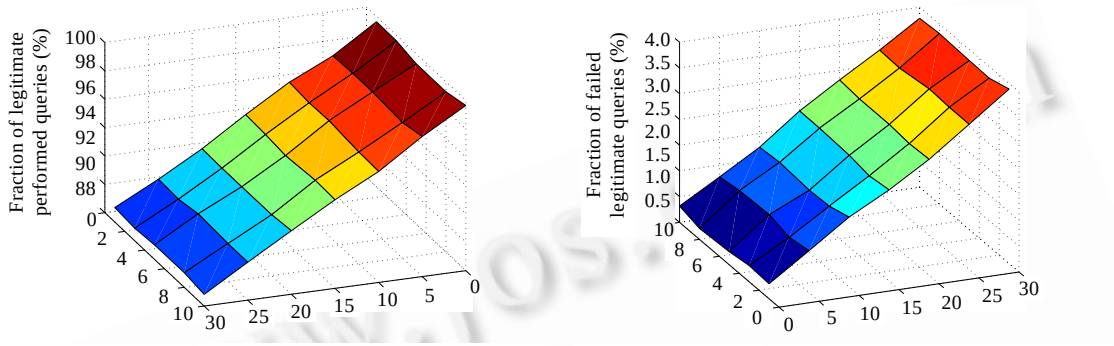
Fig.11 Fraction of failed legitimate queries vs. attack frequency, under pre-computation attacks (fraction of malicious ordinary peers=10%)

3) Combined attack: All of the above experiments independently evaluate the performance of ADSF from the viewpoints of different attack categories. However, in real-world networks, these four kinds of DoS attacks may mix with each other, and act as a combined attack to destroy the system. Therefore, we need to evaluate the effectiveness of ADSF under combined attacks to reflect the system performance more realistically.

In this experiment, each of these four DoS attacks takes up one quarter of the combined attacks. To evaluate the performance, we first calculate the fraction of legitimate performed queries with the fraction of malicious

ordinary peers changing from 0% to 30%, and the attack frequency varying from 1Hz to 10Hz simultaneously. The result surface plotted in Fig.12(a) demonstrates that the performance of ADSF decreases slightly with the growth of the fraction of malicious ordinary peers; moreover, it is robust to the change of attack frequency. Furthermore, ADSF can work well under intensive combined attacks since the fraction of legitimate performed queries exceeds 88% even in the highly malicious environment.

Since ADSF always has the capacity of successfully performing almost all the legitimate queries under various individual attacks, we expect that ADSF will hold the same property under combined attacks. The experimental results shown in Fig.12(b) prove our expectation, i.e., ADSF always works well and performs more than 96% of legitimate queries even under highly intensive combined attacks. That is, legitimate peers will experience only a small degradation of services.



(a) (b)

Fig.12 ADSF under combined attacks, with fraction of malicious ordinary peers changing from 0% to 30% and attack frequency varying from 1Hz to 10Hz

7 Conclusion and Future Work

In this paper, we study the traditional client puzzle scheme, and propose a novel adaptive client puzzle scheme which can perform a lightweight client-server interaction to flexibly adjust the puzzle difficulty based on the real-time statuses of both client and server. To evaluate the applicability, we further develop an adaptive DoS-resistant security framework for P2P networks. The theoretical analyses and experimental results indicate that the adaptive client puzzle scheme can effectively defend against various DoS attacks including IP spoofing attack, reflection attack, replay attack and pre-computation attack without significantly influencing legitimate peers' experiences even in a highly malicious environment.

For future work, we plan to integrate our adaptive client puzzle scheme with other advanced DoS defense mechanisms to further improve its performance, and extend its application field to some other kinds of networks, e.g., sensor networks and ad-hoc networks.

References:

[1] Handley M, Rescorla E. Internet denial-of-service considerations (RFC4732). 2006. http://www.ietf.org/rfc/rfc4732.txt

[2] Parno B, Wendlandt D, Shi E, Perrig A, Maggs B, Hu YC. Portcullis: Protecting connection setup from denial-of-capability attacks. In: Proc. of the ACM SIGCOMM 2007. 2007. 289−300.

[3] Aura T, Nikander P, Leiwo J. DOS-Resistant authentication with client puzzles. In: Proc. of the 8th Int'l Workshop on Security Protocols. 2000. 170−177.

[4] Merkle R. Secure communications over insecure channels. Communications of the ACM, 1978,21(4):294−299.

[5] Dwork C, Naor M. Pricing via processing or combatting junk mail. In: Proc. of the CRYPTO'92. 1992. 139−147.

[6] Juels A, Brainard J. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: Proc. of the 1999 Network and Distributed System Security Symp. (NDSS). 1999. 151−165.

[7] Abadi M, Burrows M, Manasse M, Wobber T. Moderately hard, memory-bound functions. ACM Trans. on Internet Technology (TOIT), 2005,5(2):299−327.

[8] Rivest R, Shamir A, Wagner D. Time-Lock puzzles and timed-release crypto. Technical Report, MIT-LCS-TR-684, MIT, 1996.

[9] Bocan V. Threshold puzzles: The evolution of DOS-resistant authentication. Trans. on Automatic Control and Computer Science, 2004,49(63).

[10] Wang X, Reiter M. Defending against denial-of-service attacks with puzzle auctions. In: Proc. of the 2003 IEEE Symp. on Security and Privacy. 2003. 78−92.

[11] Laurens V, Saddik A, Nayak A. Requirements for client puzzles to defeat the denial of service and the distributed denial of service attacks. Int'l Arab Journal of Information Technology, 2006,3(4):326−333.

[12] Meadows C. A formal framework and evaluation method for network denial of service. In: Proc. of the 12th IEEE Computer Security Foundations Workshop. 1999. 4−13.

[13] Krawczyk H, Bellare M, Canetti R. HMAC: Keyed-Hashing for message authentication (RFC2104). 1997. http://www.ietf.org/rfc/rfc2104.txt

[14] Cho K, Fukuda K, Esaki H, Kato A. The impact and implications of the growth in residential user-to-user traffic. In: Proc. of the ACM SIGCOMM 2006. 2006. 207−218.

[15] Liang J, Kumar R, Ross K. The KaZaA overlay: A measurement study. Computer Networks Journal (Special Issue on Overlay Distribution Structures and their Applications), 2005.

[16] Napster. http://www.napster.com/

[17] Gnutella. http://www.gnutella.com/

[18] Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Shenker S. Making Gnutella-like P2P systems scalable. In: Proc. of the ACM SIGCOMM 2003. 2003. 407−418.

[19] Merugu S, Srinivasan S, Zegura E. Adding structure to unstructured peer-to-peer networks: the role of overlay topology. In: Proc. of the Networked Group Communication (NGC). 2003.

[20] Liang J, Kumar R, Xi Y, Ross K. Pollution in P2P file sharing systems. In: Proc. of the IEEE INFOCOM 2005. 2005. 1174−1185.

[21] Saroiu S, Gummadi P, Gribble S. A measurement study of peer-to-peer file sharing systems. In: Proc. of the Multimedia Computing and Networking. 2002.

[22] Sripanidkulchai K. The popularity of Gnutella queries and its implications on scalability. In: Proc. of the O'Reilly Peer-to-Peer and Web Services Conf. 2001.

[23] Keromytis A, Misra V, Rubenstein D. SOS: An architecture for mitigating DDoS attacks. IEEE Journal on Selected Areas in Communications, 2004,22(1):176−188.

[24] Stoica I, Morris R, Karger D, Kaashoek M, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. of the ACM SIGCOMM 2001. 2001. 149−160.

CHEN Rui-Chuan was born in 1982. He is a Ph.D. candidate at Peking University. His current research areas are distributed computing and network security.

TANG Li-Yong was born in 1972. He is an associate professor at Peking University. His current research area is network security.

GUO Wen-Jia was born in 1983. He is a master student at Peking University. His current research area is distributed computing.

CHEN Zhong was born in 1963. He is a professor at Peking University and a CCF senior member. His current research areas are network security and software engineering.