PAPER *Special Section on Internet Technology VI*

# FCAN: Flash Crowds Alleviation Network Using Adaptive P2P Overlay of Cache Proxies

**Chenyu PAN**[†a)], **Merdan ATAJANOV**[†], **Mohammad BELAYET HOSSAIN**[††], **Toshihiko SHIMOKAWA**[††], ***Nonmembers, and*** **Norihiko YOSHIDA**[†], ***Member***

**SUMMARY** With the rapid spread of information and ubiquitous access of browsers, flash crowds, a sudden, unanticipated surge in the volume of request rates, have become the bane of many Internet websites. This paper models and presents FCAN, an adaptive network that dynamically optimizes the system structure between peer-to-peer (P2P) and client-server (C/S) configurations to alleviate flash crowds effect. FCAN constructs P2P overlay on cache proxy server layer to distribute the flash traffic from origin server. It uses policy-configured DNS redirection to route the client requests in balance, and adopts strategy load detection to monitor and react the load changes. Our preliminary simulation results showed that the system is overall well behaved, which validates the correctness of our design.

***key words:*** *Internet load distribution, flash crowds, content delivery networks, peer-to-peer systems*

## 1. Introduction

With the rapid spread of information and ubiquitous access of browsers, flash crowds, a sudden, unanticipated surge in the volume of request rates, have become the bane of many Internet websites. Flash crowds exert the heavy load to tens or hundreds times more than normal on target web server, causing the server temporarily unreachable. As a result, most or all users perceive intolerable poor performance.

Due to its overall unpredictability and relatively short duration, the traditional server-side over-provisions lead to the severe under-utilization of resources most of the time. An alternative is to let the clients share the popular objects among themselves, forming client-side P2P overlay. However, it loses the client transparency and some P2P systems remain low efficiency when the demand of flash objects decrease.

This paper focuses on modeling and simulating a self-tuning adaptive network, FCAN: Flash Crowds Alleviation Network, which dynamically transits system structure between peer-to-peer (P2P) and client-server (C/S) configurations to reach the optimum efficiency in protecting website from unexpected load surge, such as flash crowds. FCAN employs an Internet-wide infrastructure of cache proxy servers to perform P2P functions in dealing with the flash crowds effects and gets it out of the way when normal C/S architecture works well. Through this way the unjustified over-provision is avoided as well as the overheads caused by P2P functions can be minimized, while client transparency remains.

FCAN has been previously introduced elsewhere [1], [2]. As an updated and improved version, this paper makes the following contributions. First, we study the flash crowds and summary the main characteristics. Second, we present the design idea of FCAN system along with the mechanisms for object dissemination, DNS redirection, load balancing, load detection, and dynamic transition. Third, we model a flash traffic generator and use a home-grown simulator to do the preliminary simulation in verifying the correctness of the design and the results confirmed our idea.

The rest of the paper proceeds as: Sect. 2 takes a look at the nature of flash crowds. Section 3 studies related works. We present the overall design in Sect. 4 and overview the implementation issues in Sect. 5. Preliminary simulation results and discussions are shown in Sect. 6, and the last is the conclusion and future works.

## 2. Characteristics of Flash Crowds

First, let's study closer to flash crowds. Several real traces collected during flash crowd events have been presented and analyzed in previous research. These events include the FIFA 1998 World Cup [3], September 11th terrorist attack [4], play-along show TV site and Chilean presidential election [5], and new Linux Redhat image distribution [6]. Through these traces, some significant flash characteristics can be observed as follows.

1. Sudden events of great interests, whether planned or unplanned, such as links from popular web sites (i.e. Slashdot effect [7]) or breaking news stories (ex. Sep. 11th terrorist attack), trigger the flash crowds.
2. The volume of request for the popular objects increases dramatically to tens or hundreds times more than normal, which is far beyond the capacity of normal web servers.
3. The increase of the request rate is dramatic but relatively in short duration. Thus, traditional over-provision to handle the peak load may result servers stay practically idle most of the time.
4. The volume of request increases, while rapidly, is far from instantaneous. In play-along [5] cases, the rate

Manuscript received July 1, 2005.

[†]The authors are with the Department of Information and Computer Sciences, Saitama University, Saitama-shi, 338-8570 Japan.

[††]The authors are with the Department of Social Information Systems, Kyushu Sangyo University, Fukuoka-shi, 813-8503 Japan.

a) E-mail: pan@ss.ics.saitama-u.ac.jp

DOI: 10.1093/ietcom/e89–b.4.1119

increase occurs gradually over the 15-minute interval. This gives the time for a system to react before a flash crowd reaches its peak.
5. Network bandwidth is the primary constraints bottleneck. The closer the links to the server, the worse affected. So we must alleviate the bandwidth bottleneck at the server.
6. The distribution of requested objects is Zipf-like. The number of clients is commensurate with the request rate. These are big differences to rule out the DDoS attack from flash crowds [5].
7. A small number of objects, less than 10%, is responsible for a large percentage of requests, more than 90%. It indicates that caching of these documents might be a possible solution.
8. Over 60% of objects is accessed only during flash crowd. It implies normal web caches may not have these objects at the beginning of the flash crowd.

These flash characteristics give us the fundamental basis when designing the FCAN system.

## 3. Related Works

Generally speaking, the taxonomy of solutions for flash crowds includes three categories: server-layer, client-layer and intermediate-layer solutions, according to a typical architecture of network.

Server-layer solutions are straight-forward but costly approaches. They extend object's availability by providing excessive capacity of the servers based on peak demand or by CDN technology [8], [9] to increase server locations in advance. However, the average low traffic of the surrogate servers that experiences a flash crowd once in a lifetime but stay idle most of the time makes these solutions unsuitable.

The idea of flash crowd alleviation via building client-side P2P overlay was previously described in [10]–[12]. This kind of solutions enables the hot objects propagate quickly among the overlay and distributes the load burden from centralized server. However, these solutions mainly rely on the client-side cooperation. They have to be deployed on user's desktop which cannot be accepted in some occasions. In addition, some remain overheads, such as flooding problem, which cannot be neglected while facing the leaving of flash crowds.

There have been similar works of cache proxy layer solutions against flash crowds. BackSlash [7] is a web mirroring system built on a distributed hash table overlay, where hot objects are cached as FCAN does. However, the content on mirror servers have to be pre-placed and well-organized in advance which incurs the operation complexity and low extensibility of the system. The solution using multi-level caching [13] argues that with proper replacement algorithms a caching infrastructure designed to handle normal web loads can be enough to handle flash crowds. The system needs a dedicated deployment of hierarchical caching placement and currently lacks adaptive mechanism to handle flash crowd flexibly.

Other works related to adaptive network include: Adaptive CDN [5], focusing on the CDN network, NEWS [14], imposing congestion control on application level which scarifies some user requests to achieve a high network performance, and Dynamic Resource Allocation [4], allocating the server resources by a front-end load balancer. We benefit from these researches extensively and propose our own improved design.

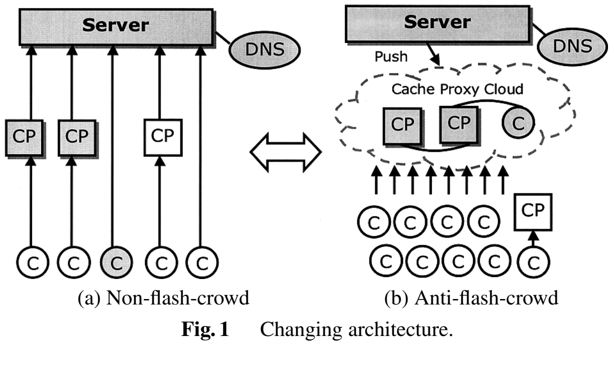
## 4. Design Description

Different from the server-side solutions and client-side solutions, our design is an intermediate-layer approach. We focus on a P2P-based cache proxy server layer where hot objects requested during a flash crowd are cached in, and delivered to end users after conducting P2P searches. To deal with the short-term of flash crowds, we tune the network to be adaptive by invoking P2P mode only if the C/S mode fails. To some extent, FCAN organizes a group of forward cache proxies into a temporary and wide-area-based layer of reverse cache proxy. In addition, we also welcome the clients who voluntarily share their resource as a peer member on the purpose of assisting the system providing better service.

### 4.1 Basic Operation

Figure 1 illustrates how FCAN alleviates the symptoms associated with flash crowds. The server who wants to be alleviated from flash crowds is called Member Server. Member Peers consist of Member Cache Proxies (Member CPs) and volunteer clients. All Member Peers, Member Server and special DNS are shown in deep color to distinguish from other clients who connect to the web server directly or through common cache proxies.

In peaceful time, the typical C/S architecture satisfies most of the client requests. Member Server and Member Peers do little more than what normal ones do. Once a flash crowd comes, Member Server detects the increase in traffic load. It triggers all Member Peers to form P2P overlay, the cloud shown in Fig. 1(b), through which all requests are conducted instead of bugging the origin server. Moreover, all subsequent client requests are redirected to this cloud by



(a) Non-flash-crowd (b) Anti-flash-crowd

**Fig. 1** Changing architecture.

DNS redirection automatically.

In the following subsections, we present the design detail of the P2P-based cache proxy cloud, DNS redirection and dynamic transition.

## 4.2 Cache Proxy Cloud

### 4.2.1 Implicit Overlay

Cache proxy cloud is a quick-formed P2P overlay conducted by all Member Peers during flash traffic. The cloud is responsible for absorbing the huge amount requests so that the flashed server can be free from overwhelming.

A quick-formed P2P overlay has the features of being simple and lightweight. It can be quickly started and stopped with the change of the network traffic. We introduce the conception of implicit P2P overlay. Each Member Peer maintains a part of network relationship which contributes to combine a whole network overlay. This overlay is unknown to the outside but only invoked to be explicit during anti-flash-crowds status so that the P2P search can be conducted with a minimized warm-up time. Through normal communication process, Member Peers and Member Server discover their existence by adding special head fielder in request/response http packet. Member Peers get the initial neighbor list from Member Server and update it after each transition to keep on the implicit overlay. The detail protocol design can be found in [2].

In FCAN, each Member CP is primarily a regular cache proxy during its normal mode of operation. It serves a fixed set of users in usual time but serves the requests arriving at the cloud from any user during the anti-flash-crowd status. In reality, each cache proxy serves for several content servers, and there is a case that any server suffers from flash crowds while the others do not. Therefore, each Member CP has the functionality of mixed-mode operations for the normal C/S mode and the anti-flash-crowds P2P mode. The modes are switched according to requested contents.

### 4.2.2 Object Dissemination

FCAN adopts a TTL-scoped searching schema and "push" service to disseminate objects among the cache proxy cloud.

P2P systems have evolved from the first generation system, using simple flooded searching algorithm in locating objects, to the second generation system, in contrast, using a variety of well-organized distributed hash tables. Considering the caching nature that the contents on proxy servers are determined randomly, not well organized, by client requests, first generation system is a simple and lightweight schema to disseminate hot objects among the cache proxy cloud. Mathematical and simulation analysis in [15] shows that the searches of the first generation P2P systems can be designed to have low expected inner traffic when searching for objects that are the interest of a flash crowd.

Through normal communication processes, Member Peers notify their existence to Member Server and Member

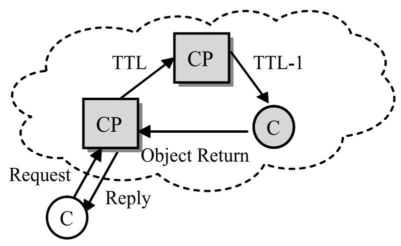


**Fig. 2** Scoped search.

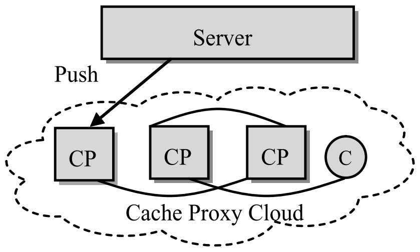


**Fig. 3** Push service.

Server is able to send/restore the alarm of a coming flash crowd to the corresponding Member Peers. Once a Member Peer receives the alarm, it creates or updates a neighbor list according to the information received from Member server to active the P2P overlay explicit. Then Member CP conducts the requests either from client users or neighbor peers, while volunteer client only conducts the requests from neighbor peers or itself. If the requested hot object is found, Member Peers caches a copy and returns the object to the requester, otherwise delivers the scoped search query to its neighbors with a decreased TTL time (See Fig. 2).

We also notice that over 60% of objects are new to a flash crowd, they may not been cached at the beginning stage. A "push" service is employed as a supplementary to improve the P2P search efficiency (See Fig. 3). Member Server has its file access references and trace histories of Member Peers. When there is a flash crowd for specified objects which have not been accessed by Member Peers, it pushes these objects to the cloud by connecting to just one or two Member Peers. The delivered objects will soon be propagated in the cloud because of the P2P functions and the high demands.

## 4.3 DNS Redirection

To protect server and network from overload, flooded requests must be redirected. We use a DNS-based request redirection. DNS redirection gives out the address of the cache proxy cloud instead of the origin server address when a client tries to resolve the server name through its local DNS server. The address of the cloud can be any address of Member Cache Proxies, and appropriate addresses are picked up to the client by adopting certain selecting policies, such as load-weighted algorithm, proximity-based algorithm or as simple as round robin fashion. The redirected

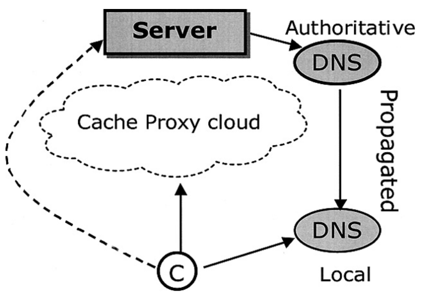


**Fig. 4** DNS redirection.

requests for flashed objects are conducted by P2P search among the cloud, while the requests for common objects are processed as normal client request that the cache proxy fetches the objects directly from server.

As Fig. 4 shows, Member Server determines when to modify the entries address and when to restore it back. Once being modified, the new addresses are propagated through the Internet to the client side. One problem is that the DNS cache may delay that propagation and leave the load unbalanced among the cloud. We use a specialized DNS server, TENBIN [16], on server site which allows DNS lookup entries to be modified dynamically. TENBIN is our research product, and already used in practice, for example, “Ring Servers” and “LIVE! ECLIPSE” projects [17]. We have obtained several experiences on DNS propagation from experiments and practical use of TENBIN. It spends only 1 or 2 minutes to complete the propagation, which can be neglected compared with a typical flash crowd which occupies several hours or several days.

### 4.4 Dynamic Transition

To react the network condition, FCAN does overload/underload detection periodically and performs dynamic transition accordingly. The higher the overload detection frequency, the more responsive our system is to load changing. However, it is a trade-off that achieving fast response takes the expense of possible false alarm at overload detection, while incurs rapid fluctuations at underload detection.

Currently FCAN uses a threshold-based scheme to trigger dynamic transition. The goal of this scheme is to maintain the average load on the Member Servers.

When detecting the coming of a flash crowd, it observes the volume of server load periodically and computes the slope of the tangent line for that load curve. Once the server load increase exceeds to the predefined threshold ($T_{high}$) with a considerable growth slope in the past ($\delta 1$) history seconds, Member Server treats it as a coming flash crowd. It sends the alarm to all Member Peers to transit into anti-flash-crowd status with necessary information, such as flash object lists, neighbor lists and etc., then pushes uncached hot objects to the cloud to ensure the object existence. After that, Member Server modifies the DNS lookup entries of the web site address to those of the Member CPs’ and waits for new addresses being propagated through the Internet. As a result, DNS gradually redirects subsequent requests from the server to the cache proxy cloud and makes load distribution inside the cloud.

When detecting the leaving of a flash crowd, Member Server collects the load information from each peer to make a global decision. However, it slows the interval monitor time, enlarges the history seconds, and lowers the threshold mark. The purpose of this strategy detection is to 1) decrease the inner-traffic caused by continuous monitoring, 2) ensure the flash crowd does leave away, 3) and consequently avoid fluctuation of system transition. Once the global load among cloud decreases under a pre-defined threshold ($T_{low}$) in the past ($\delta 2$) seconds, FCAN treats it as the leaving of the flash crowd. Member Server restores the DNS lookup entries and notifies all Member Peers to stop P2P search. Then, everything is back to normal.

## 5. Implementation Overview

We put special wrappers for intercepting the requests on normal web servers and cache proxies, and employ special DNS server to distribute the requests.

The wrappers on Member Server and Member CP are designed as a module for the popular Apache web server and Squid cache proxy. Both Apache and Squid are built from a modular design, which is easy to add new features. Thus converting normal elements with FCAN member features should require no more than compiling a new module into Apache or Squid and editing a configuration file. The wrappers on volunteer client does not require client to install any other cache proxy software, besides the common member functions such as P2P search, load monitor, it acts as a simple cache proxy as well.

To implement P2P overlay construction, the PROOFS system presented in [10] can be a possible schema. As we mentioned before, P2P overlay construction should be fast started and stopped with the change of the network conditions. PROOFS is a simple, lightweight and naturally robust approach. It shuffles peer neighbors to achieve the randomness of P2P overlay and uses scoped search to deliver objects atop that overlay. However, this simple searching algorithm incurs flooding problem when searching for unpopular objects. We offset this weakness by invoking PROOFS only for the duration when the searching object is heavily demanded, i.e. flash crowds period. Besides, PROOFS assumes a large number of participated nodes. In practice, the number of contributed cache proxy for FCAN may not reach that amount. Therefore, our P2P overlay can be a simplified PROOFS system with a server-made, instead of shuffling-made, randomness. Moreover, all inner-communications among Member Server and Member Peers, such as search query, load report, are conducted through IP address instead of http domain name. That is to avoid the infinite loop once DNS server entries have been modified.

We adopt TENBIN system to implement the special DNS server. With TENBIN, we can limit the DNS cache problem, dynamically modify the lookup entries and configure special policy to achieve load balance among P2P cloud.

## 6. Preliminary Simulation

In this section, we present the preliminary simulation results to verify the design of FCAN and convince its correctness. The simulator is a home-grown, thread-based, java program and mainly composed of five parts:

1. A package of core classes including the main roles of system such as Member Server.
2. A flash crowds traffic generator which models a flash crowd with load growth rate, growth level and three duration phases.
3. Underlying network architecture which supports high-level application with the TCP and UDP communication and creates unique network address for each node.
4. Network monitors which collects the load information from active roles periodically and writes the load data to the log files for analysis.
5. Console User Interface which enables parameters configuration and runs test case for simulator user.

### 6.1 Flash Crowds Generator

Flash crowds can differ from each other in several respects, for examples, some flash crowds exhibit dramatic growth rates, while some last comparatively long-duration, and some have very high magnitude peaks. We generate flash traffic based on the model illustrated in Fig. 5.

Previous works have been done in modeling flash crowds. Our model simplifies the three-phases model of [13] and extends it by adding a growth rate parameter, enlightened by [4]. Three major phases can be distinguished in a flash crowd. They are 1) a ramp-up phase, where flash crowd starts at time t0, more and more people are interested in event and the traffic level rises from its normal level to a sustained maximum level at time t1, 2) a sustained traffic phase, where people come in and leave out the web site at a roughly same rate until time t2 3) and a ramp-down phase, where people leave out gradually and the traffic level decreases to its normal level at time t3.

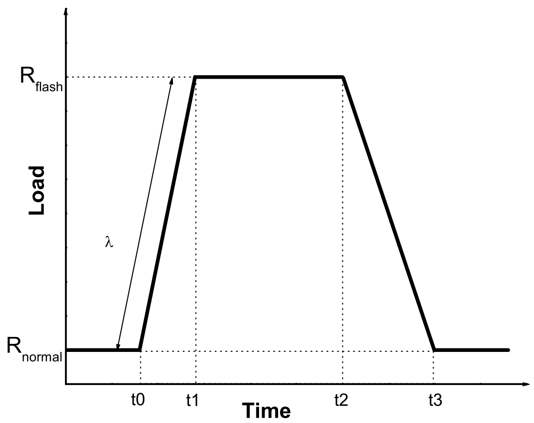


**Fig. 5** Flash crowd model.

We calculate the ramp-up time l1 = t1–t0 by: l1 = (Rflash-Rnormal)/ ($\lambda$), where ($\lambda$) is the rate at which the load increases before the flash crowd stabilizes. This parameter relates to the speed at which FCAN needs to detect and react. The larger the ($\lambda$) value is, the short time it takes the load to reach the peak. Rnormal and Rflash stand for the average load in peaceful period and the peak load in flash period respectively. A growth level parameter is defined as shock-level = Rflash/Rnormal that shows the magnitude increase in the average request rate seen by a web site. This parameter relates to the maximum resource capacity required to successfully handle the flash crowd.

The sustained time l2 = t2–t1 and ramp-down time l3 = t3–t2 are less sensitive, and currently they are given simply by n*l1, where n is a constant.

### 6.2 Simulation Methodology

Quantitative and rigorous evaluations are not included in this preliminary simulation. We take it as one of our future works, and did simulation-based experiments to validate the correctness of our design at current stage.

The simulator defines several system roles such as Member Server, Member CP, Client and etc., each runs in a separated thread. The network is built on an application level with Member Server and Member Peers running continuously, and client thread is created to send a request and destroyed after getting the result back from server or from cloud. Either server or peer rejects the incoming requests if the load exceeds its capacity. There is no hot object copy on any Member CP at the beginning stage. Since TENBIN needs short time for DNS propagation. We neglected DNS cache problem in the simulation.

Table 1 shows the detail parameters configuration. The most important flash characteristic for studying the performance of FCAN is the load growth rate ($\lambda$). We set $\lambda$ values

**Table 1** Simulation parameters.

| Description | Value |
|---|---|
| Load growth rate $\lambda$ | 3 |
| Load growth level | 10 |
| Normal traffic rate (req/sec) | 10 |
| Threshold for alarm Thigh (req/sec) | 20 |
| Threshold for restore Tlow (req/sec) | 10 |
| Threshold for push (req/sec) | 4 |
| History $\delta$1 size (sec) | 0 |
| History $\delta$2 size (sec) | 4 |
| Starting time of flash crowd (sec) | 10 |
| Server capacity | 30 |
| Peer capacity | 15 |
| Number of peers | 10 |
| Copies of pushing object | 1 |
| Number of flash object | 2 |
| Number of common object | 8 |
| P2P search ttl | 2 |
| P2P search fanout | 1 |

of 3 that increase the load 30% per second. The flash growth level is set to 10, thus according to a normal traffic rate of 10, the peak load is expected to reach around 100 req/sec. We start the flash crowd at the 10th second and the ramp-up duration l1, sustained load duration l2, ramp-down duration l3 are given by (Rflash-Rnormal)/($\lambda$), 1 * l1 and 2 * l1. To better investigate the behavior of FCAN, we study only a single flash crowd toward certain number of flash objects, value of 2, on one Member Server, but the same scenario could be recreated for multiple flash crowds toward to other objects on same Member Server or different objects on multiple Member Servers.

According to a server load capacity of 30 req/sec, the threshold for alarm is set to 20 req/sec. However, the server may still suffer from increased high request rate for several seconds since the transition takes some time to complete. The restore C/S threshold is set to the value of 10, the same as normal traffic rate, combined with a referenced history size of 4, so as to ensure the flash traffic does leave.

## 6.3 Simulation Results

Figure 6 illustrates the comparison of two servers which suffered from modeled flash crowd traffic, where the total traffic duration was 150 s. Flash crowd started at the 10th second and ended at the 140 second. The ramp-up phase lasted for 30 s, starting from the load of 10 req/sec to 100 req/sec, then the load traffic sustained around 100 req/sec load for 30 s and the ramp-down phase lasted for 60 s until the load back to the normal value. Compare with the server without FCAN, the server with FCAN controls the request rate all under its capacity 30 req/sec and the server load degraded during the flash crowd period because the P2P cloud cached objects by the clients gradually. From this figure, we can see the adaptive transition does alleviate the flash traffic from server side.

The two transition points are around the time of 15 s and the time of 133 s, as Fig. 7 shows. Table 2 lists the detail request load information of each individual at those two transition points. At the time of 13 s the number of requests on Member Server exceeded the threshold 20 and reached 21. It took 2 seconds to complete the transition, and at the time of 15 s P2P cloud began to receive the redirected client requests. The time for restoring the system from P2P to C/S configuration is relatively short. At the time of 130 s, the total load on P2P cloud summed up to 10, and in the later 3 seconds it remained no more than 10 req/sec which triggered system to restore back. One second passed, Member Server restored the system and no subsequent request was redirected to cloud.

Next let's study the behaviors of peers during flash crowds. Table 3 reflects the load distribution among FCAN. It gives the detail handled request information of Member Peers as well as Member Server. The FC column is the modeled flash crowd traffic collected on client sides. From these data, we can find that server load was distributed among P2P

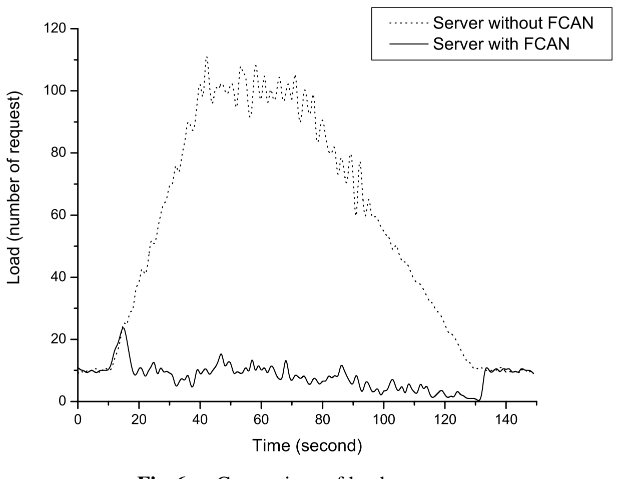


**Fig. 6** Comparison of load on servers.

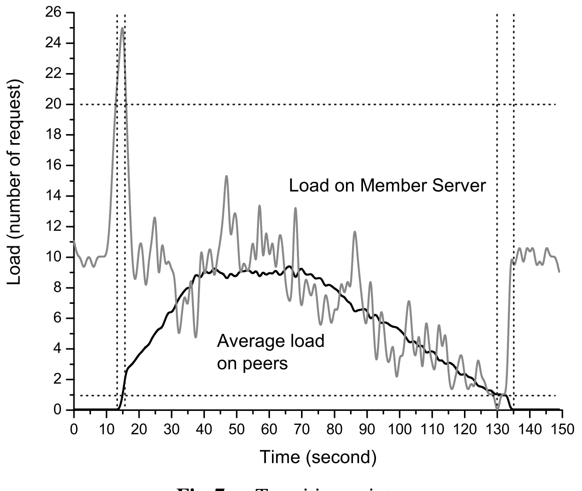


**Fig. 7** Transition points.

**Table 2** Detail data at transition points.

| Time(s) | MS | CP1 | CP2 | CP3 | CP4 | CP5 | CP6 | CP7 | CP8 | CP9 | CP0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 25 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 130 | 0 | 1 | 0 | 4 | 1 | 0 | 0 | 3 | 1 | 0 | 0 |
| 131 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 132 | 1 | 0 | 2 | 1 | 1 | 3 | 0 | 0 | 1 | 0 | 1 |
| 133 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 134 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 3** Comparison of handled requests by each individual.

| | FC | MS | CP1 | CP2 | CP3 | CP4 | CP5 | CP6 | CP7 | CP8 | CP9 | CP0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total requests | 8361 | 1206 | 726 | 769 | 705 | 710 | 706 | 695 | 735 | 707 | 662 | 740 |
| Avg request rate | 55.74 | 8.04 | 4.84 | 5.13 | 4.70 | 4.73 | 4.71 | 4.63 | 4.90 | 4.71 | 4.41 | 4.93 |
| Peak request rate | 116 | 25 | 15 | 17 | 14 | 14 | 14 | 14 | 16 | 15 | 14 | 14 |

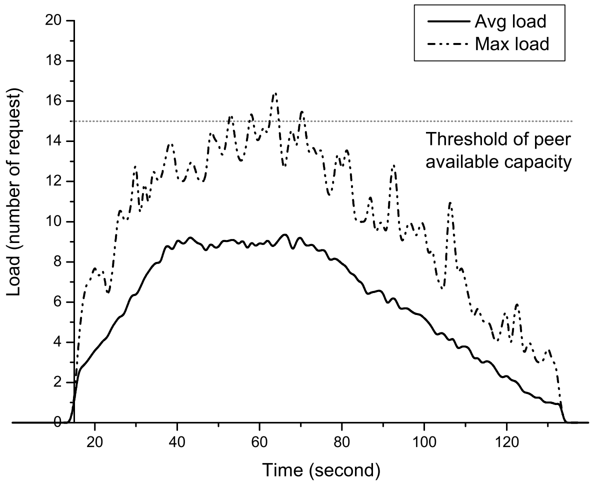


**Fig. 8** Load on peers.

**Table 4** Percentage of fulfilled requests.

| | Total | Success | Percentage |
|---|---|---|---|
| Without FCAN | 8361 | 3830 | 45.81% |
| With FCAN | 8361 | 8354 | 99.92% |

cloud generally balanced by each peer. Figure 8 shows the average load curve on peers, which is under the threshold of peer max capacity, however, the max load curve which picks up the maximum load of peers at each second exceeds the threshold at the peak time. These exceeded requests were rejected. We statistic the percentage of successful requests against total requests on client sides both using FCAN and without FCAN. The result is shown in Table 4 that FCAN helps the success rate up from 46% to nearly 100%.

## 6.4 Implications

The above presented simulation results indicate that:

1. FCAN is well behaved to the coming and leaving of flash crowd. Both the warm-up time and cool-down time are short to take the transition into effect.
2. Server protected by FCAN is free from the sustained high flash traffic, and its load degraded because of the cache in cloud.
3. The load distributed among P2P cloud is overall balanced by each peer, with an acceptable average load.
4. The number of participated Member Peers determines the magnitude of flash crowd that the cloud can absorb.

The last one is an important observation. Once each Member Peer has successfully searched and cached a copy of flashed object, the remaining problem is how to service the subsequent huge requests. Member Peer spares a part of its CPU cycles and disk space to deal with the redirected requests. We have to limit the load on any participating peer so as not to overload it as a secondary victim of a flash crowd. If a flash crowd is large in magnitude or duration, FCAN may have low performance when there are few participated peers. We leave this problem as one of our future works.

## 7. Conclusions

This paper presented FCAN, a prototype system designed to alleviate flash crowds effect by adaptive changing system structure between C/S to P2P configuration. FCAN constructs P2P overlay on cache proxy server layer to distribute the flash traffic from origin server. It uses policy-configured DNS redirection to route the client requests and adopts strategy load detection to monitor the changing load traffic. Our preliminary simulation results showed that the system is overall well behaved, which validates the correctness of our design and points out the way for future research.

FCAN is designed for flash crowds protecting but not limited to this. It adjusts server load under a predefined threshold facing against any unexpected traffic surges, thus we assume some kinds of DDoS attacks can also be handled.

The next steps in this research involve the studies of online capacity planning, multiple work load situation, possible solution to non-cacheable objects, and a real network based quantitative evaluations.

## References

[1] C. Pan, M. Atajanov, T. Shimokawa, and N. Yoshida, "Design of adaptive network against flash crowds," Proc. Information Technology Letters, vol.3, pp.323–326, Sept. 2004.

[2] C. Pan, M. Atajanov, T. Shimokawa, and N. Yoshida, "Flash crowds alleviation via dynamic adaptive network," Internet Conference 2004, pp.21–28, Tsukuba, Japan, Oct. 2004.

[3] M. Arlitt and T. Jin, "A workload characterization of the 1998 world cup web site," IEEE Netw., vol.14, no.3, pp.30–37, May 2000.

[4] A. Chandra and P. Shenoy, "Effectiveness of dynamic resource allocation for handling Internet flash crowds," Technical Report TR03-37, Department of Computer Science, University of Massachusetts Amherst, Nov. 2003.

[5] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites," Proc. 11th International World Wide Web Conf., pp.252–262, May 2002.

[6] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalis," Proc. ACM SIGCOMM Internet Measurement Workshop, San Francisco, CA, Nov. 2001.

[7] T. Stading, P. Maniatis, and M. Baker, "Peer-to-peer caching schemes to address flash crowds," Proc. 1st International Workshop on Peer-to-Peer Systems, pp.203–213, Cambridge,MA, USA, March 2002.

[8] Akamai, http://www.akamai.com
[9] Digital Island, http://www.digitalisland.com
[10] A. Stavrou, D. Rubenstein, and S. Sahu, "A lightweight, robust P2P system to handle flash crowds," IEEE J. Sel. Areas Commun., vol.22, no.1, pp.6–17, Jan. 2004.
[11] V.N. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," Proc. 1st International Workshop on Peer-to-Peer Systems, pp.178–190, Cambridge, MA, USA, March 2002.
[12] S. Lyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," Proc. 21th ACM Symposium on Principles of Distributed Computing, 2002.
[13] I. Ari, B. Hong, E.L. Miller, S.A. Brandt, and D.E. Long, "Managing flash crowds on the Internet," Proc. 11th IEEE/ACM International Symposium on MASCOTS, Orlando, FL, Oct. 2003.
[14] X. Chen and J. Heidemann, "Flash crowd mitigation via an adaptive admission control based on application-level measurement," Technical Report ISI-TR-557, USC/ISI, May 2002.
[15] D. Rubenstein and S. Sahu, "An analysis of a simple P2P protocol for flash crowd document retrieval," Technical Report, EE011109-1, Columbia University, Nov. 2001.
[16] T. Shimokawa, N. Yoshida, and K. Ushijima, "Flexible server selection using DNS," Proc. International Workshop on Internet 2000 in IEEE-CS 20th International Conference on Distributed Computing Systems, pp.A76-A81, Taipei, April 2000.
[17] T. Shimokawa, Y. Koba, I. Nakagawa, B. Yamamoto, and N. Yoshida, "Server selection mechanism using DNS and routing information in widely distributed environment," IEICE Trans. Commun. (Japanese Edition), vol.J86-B, no.8, pp.1454–1462, Aug. 2003.
[18] B.Y. Zhao, J. Kubatowicz, and A.D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Technical Report, UCB/CSD-01-1141,U.C. Berkeley, April 2001.

**Chenyu Pan** received her B.Sc. and M.Eng. degree in computer science from Donghua University, Shanghai China, in 2000 and 2003 respectively. From 2003 October she has been working toward the Ph.D. degree of Information and Mathematical Sciences at Saitama University. Her research interests are distributed computer networking, especially Content Distribution Network and P2P overlay network.

**Merdan Atajanov** graduated from Middle East Technical University in Turkey/Ankara in 2001. He received his M.Eng. degree in information and computer science from Saitama University in 2005. He is now enrolled to doctoral course of Department of Information and Mathematical Sciences in Saitama University. His research interests include computer networking, especially Content Distribution Networks and Load Balancing.

**Mohammad Belayet Hossain** received the B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka in 2002. Now he is a M.Eng. student in Graduate School of Information Science in Kyushu Sangyo University. His research interests include network programming, P2P networking and load distribution. He is a student member of IEEE.

**Toshihiko Shimokawa** received the M.Eng. degree in computer science and communication engineering and the Ph.D. degree in computer science from Kyushu University in 1992 and 2001 respectively. He is Associate Professor of information science at the Department of Social Information Systems in Kyushu Sangyo University since 2002. His research interests include parallel and distributed computation, and wide area networking. He is a member of Information Processing Society of Japan.

**Norihiko Yoshida** received the M.Eng. degree in mathematical engineering and information physics in 1981 from the University of Tokyo, and the Dr. Eng. degree in computer science and communication engineering in 1990 from Kyushu University. He is Professor of computer systems at the Department of Information and Computer Sciences in Saitama University since 2002. His research interests include system design methodologies, and parallel and distributed computation. He has been served on program committees and organizing committees of various international conferences and symposia. He is a member of IPSJ, IEICE, JSSST, ACM and IEEE.