# Turning Flash Crowds into Smart Mobs with Real-Time Stochastic Detection and Adaptive Cooperative Caching

Jay A. Patel, Charles M. Yang and Indranil Gupta
*Department of Computer Science*
*University of Illinois at Urbana-Champaign*
*Urbana, IL 61801*
jaypatel,cmyang2,indy@cs.uiuc.edu

## Abstract

**A flash crowd occurs when a stampede of clients attempt to access a resource, e.g., a Web page. It can cause drastic performance degradation at servers and clients, and even outage of the service. In this paper, we first use traces collected from six different flash crowds to characterize the properties of flash crowds, showing in the process that ideal caching schemes can in fact mask flash crowds. We then present a stochastic detector that can alert a server about a flash crowd while the event is happening, i.e., in real-time. This scheme is then combined with cooperative caching among the clients to design and implement a system that adaptively detects and masks the effect of flash crowds. This mechanism effectively turns the clients in a flash crowd event into a "smart mob". We make our study comprehensive by comparing the performance of the above design experimentally with representative systems from three classes of existing designs for content providing - single server, server cooperation, and server-client modification.**

## 1 Introduction

The initial half of this decade has seen a continued increase in the popularity and use of Web services and content delivery mechanisms. Unfortunately, this growth has simultaneously produced several sources of stress with the potential to cripple performance at both the client side and the server side [2, 3, 9].

Informally, a flash crowd is a sudden spike in traffic (i.e., number of clients) to a resource such as a Web page. For instance, flash crowds are common at news websites (due to planned or unplanned news events) and in the weblog world (where it is often called the "slashdot" effect).

This paper considers one such type of stress, called a *flash crowd*. We first study six traces to characterize flash crowd behavior, then we experimentally compare the performance of several existing schemes under flash crowd stress. Finally, we propose a new solution that uses stochastic real-time detection and adaptive server-client cooperation to turn a flash crowd into a "smart mob", effectively masking the effects at both clients and servers.

Figure 1 shows the bandwidth utilization due to a flash crowd on the *Bell* website (see section 2.1). The flash crowd starts with the most noticeable spike in bandwidth consumption; its short-term effects can range from unavailability of the content, to a server crash. The long-term effects can last up to several hours after the initial stampede. For instance, on the day of Saddam Hussein's capture, ever major news sites like MSNBC were forced to disable sections of their website.

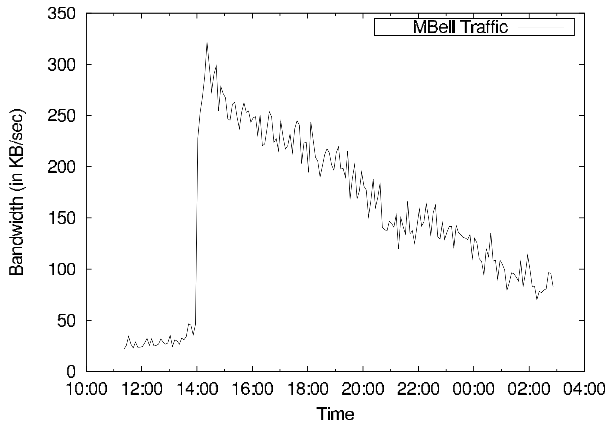


Figure 1: **Flash crowds are chiefly characterized by a sudden jump (of an order of magnitude or more) in resource utilization. The increased used of bandwidth consumption (over a 24-hour span) is visible from the flash crowd afflicted portion of the logs of *Bell* site (for details see Section 2.1). The flash crowd consumes resources at an extremely rapid rate, reaching peak utilization very rapidly. The longer-term effects of a flash crowd generally last up to 12-24 hours, with peak utilization lasting up to a few hours.**

Zooming in around the start of the flash crowd in Figure 1 shows that the spike occurs over the period of only a few minutes - see Figure 2. This gives clients and servers only *a few tens of seconds* to adapt to the incoming traffic! In addition, flash crowds are infrequent and unpredictable.

All of these factors require that any solution deployed to mitigate flash crowds consist of: (1) proactive and *real-time* mechanisms to detect the flash crowd *while (or even before) it happens*, and (2) an initiation of immediate action that can *mask the effects of the stress.*

A variety of system designs for content delivery have emerged over the past few years. Based on their underlying design methodologies, we classify these existing designs into *four* classes. We will later use this taxonomy to compare the effectiveness of the existing designs to mitigate flash crowds. The four classes are:

1. **Single Server solutions:** These solutions modify server code only (and not client code). SEDA [22] and Capriccio [21] fall under this category - although these systems

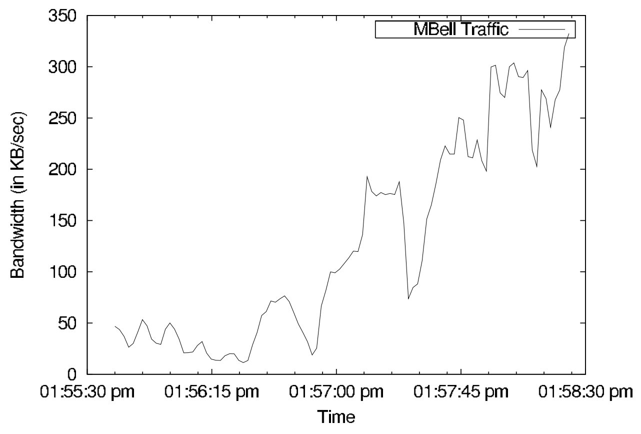


Figure 2: **Flash crowds induce a rapid jump in resource utilization. On a finer granularity time-scale, the beginning of the flash crowd shown in Figure 1 forces a drastic jump in the bandwidth utilization in approximately one minute. An ill-prepared web site, one without extra resource provision, can experience severe performance degradation or even come to a virtual standstill during the initial stampede. An effective solution must mitigate the effects within the first few seconds of a flash crowd.**

do not focus on flash crowds explicitly, they are aimed at improving server throughput.

2. **Server Cooperation solutions:** These schemes distribute content with the aid of cooperation and content sharing among the servers. This class includes systems such as [6, 11, 19, 24].
3. **Server-Client solutions:** These solutions require modifications on both the client and the server side. These include cooperative caching solutions requiring clients to run extra code, e.g., [16, 20]. This category also includes systems like the one designed later in this paper.
4. **Client Cooperative Caching solutions:** Systems in this class are those based on peer-to-peer routing substrates, e.g., [7, 12]. Although [23] showed that the marginal utility of cache hit rate diminishes with increasing population size, our study in this paper reveals cooperative caching may in fact work rather well under flash crowds.

Bringing together our discussion so far, we follow a four-step approach in our study of flash crowds in this paper.

- **Characteristics of flash crowds:** We study the properties of six real flash crowds based on traces collected from the respective websites. To the best of our knowledge, there is any existing trace-based study of this problem. One of the surprising results from this study is that *ideal caching can actually avoid flash crowds.*
- **Real-time Flash Crowd Detector:** We present the design and implementation of a real-time detector of flash crowds, running at the server. A detector must continuously monitor visitor traffic and resource utilization, and be able to immediately and accurately flag a flash crowd in real-time. Additionally, the detector must run efficiently, without consuming significant computational and memory resources, especially during normal operations. *We find that our design enables us to detect a flash crowd while the event is happening.*
- **Comparison of existing solutions:** We use the above flash crowd traces to compare the effectiveness of the three of the four methodologies discussed above. We measure both resource utilization that effect the server (and directly the content provider's costs) along with the quality of service to the clients (i.e., the latency to satisfy incoming requests).
- **Design of an Adaptive Cooperative Caching Solution:** We combine our real-time flash crowd detector at the server, with a cooperative caching scheme among clients, to design an adaptive cooperative caching solution for flash crowds. We compare the performance of this scheme with the existing methodologies. *Our results reveal that such an approach is extremely effective in turning a flash crowd into a smart mob.*

## 2 Characterization

Many previous studies [2, 9] have documented stress caused by flash crowds. These studies have shown that the load on content delivery infrastructure increases by an order of magnitude or more during the peak of the stress. The behavior of the clients has also been analyzed, with observations regarding traffic patterns (they are widely spread throughout the Internet topology) and popularity of requests (it still follows a Zipf distribution).

In this section, we briefly describe the nature of the web sites from which we collected our trace data. Furthermore, we make key observations regarding the dynamic generation of the content. We also show, somewhat surprisingly, that an ideal organizational caching model suffices to mitigate flash crowds.

### 2.1 Trace Data

To make broad observations relating to flash crowds, we acquired trace logs from a variety of web sites. A summary of our trace collection is shown in Table 1. All traces are from actively deployed web sites which experienced one or more flash crowds during the recording period.

In our efforts to collect traces, we found that many web site administrators are averse to sharing their traffic logs. However, we convinced a few administrators by providing a sanitizing script to anonymize potentially sensitive information such as client IP addresses. Additionally, a few administrators requested that we not reveal particulars of their site. In order to respect their privacy, we will only refer to the traces by pseudonyms.

The following is a brief description of the websites in our trace collection:

- *Building*: A specialized academic website created to facilitate the logistics of moving faculty, staff, and students

| Website | Traffic | Start | End | IP Anonymized? |
|---|---|---|---|---|
| *Building* | Extremely Low | Apr 13, 2004 | Dec 07, 2004 | Yes |
| *Webnote* | Low to Moderate | Aug 08, 2004 | Aug 21, 2004 | No |
| *Bell* | Moderate | Jan 30, 2005 | Feb 27, 2005 | Yes |
| *Community* | Moderate to High | Dec 31, 2004 | Jan 26, 2005 | No |
| *Kernel* | High | Jul 29. 2003 | Feb 09, 2005 | No |
| *WCup98* | Extremely High | Jul 01, 1998 | Jul 07, 1998 | Yes |

Table 1: **A brief summary of the collected traces. Most of the traces were collected from actively deployed websites that experienced one or more flash crowds within the past year.**

to a new building. The inauguration of the building and related press release triggered a flash crowd.

- *Webnote*: Includes a newly launched note-taking application. The launch created a flash crowd as the site was linked by multiple sources.
- *Bell*: Personal web log ("blog") of a popular individual, read by numerous regular users. Frequently linked by numerous other sources, especially when interesting articles are posted.
- *Community*: A consumer-based community site with focus on news and events relating to a particular company and its products.
- *Kernel*: A news site with a focus on technical events, articles, and discussions relating to a popular open source operating system kernel.
- *WCup98*: Traces from the FIFA Soccer World Cup 1998, the world's most popular sporting event. Because of the extremely high traffic experienced by this web site, we used this trace as a reference to verify our experiments validity on extremely high traffic web sites.

Most of the traces were recorded in the Common Log Format (CLF), which records the client IP address[1], the resource (URL), the time (with 1-second granularity), HTTP information like protocol version, request method and server response, along with the number of bytes transferred. A few of the traces also had extended information about the referring web site (i.e., hyperlink source) and the web browser utilized.

## 2.2 Dynamic Content

Modern websites have evolved from being comprised primarily of static documents, to a rich confluence of static and dynamically generated documents. Technologies such as CGI, PHP, ASP, JSP, JavaScript, etc., are utilized to build highly interactive and user-personalizable content. Additionally, many contemporary websites use a database back-end to store information.

Figure 3 shows that a significant portion of the bandwidth consumed by contemporary websites is to satisfy requests for dynamically generated documents. Dynamically generated content creates an interesting problem for the Web infrastructure because it must be regenerated for each request, defeating caching mechanisms. We discuss the effectiveness of an ideal caching solution next.

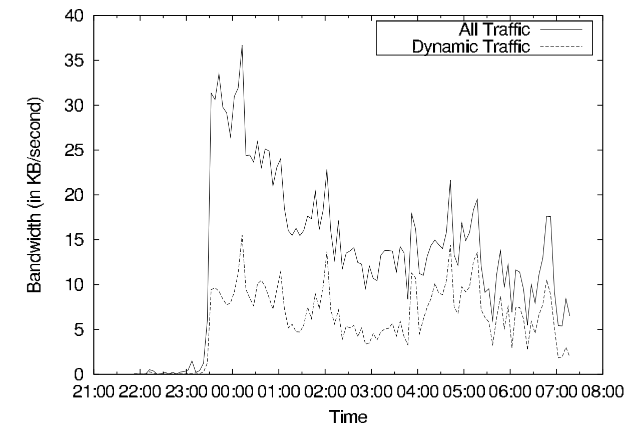


Figure 3: **The flash crowd afflicted traces of the *Webnote* site show that a significant portion of a modern web site is dynamically generated. Because of the high computational expense incurred in generating dynamic pages, the bottleneck may shift from the network to the CPU.**

## 2.3 Caching Effectiveness

If we assume that each corporation and ISP deploys an ideal caching infrastructure that spans the entire organization; Figure 4 shows that a flash crowd stress can be mitigated. Here, an organization refers to a single class A, B, or C network. We construct the bandwidth utilization of the Web server as if an ideal caching infrastructure is deployed at all organizations.

We map each client IP address (for non-anonymized traces) to a particular organization based on its network address. The network address is calculated from the first 8 bits of class A IP address, the first 16 bits of class B IP address, and the first 24 bits of a class C IP address. As many class A organizations are multinational corporations with offices all around the globe, an organization-wide cache may not be plausible. To circumvent this possible, problem, we also use a conservative ideal caching model, which uses the first 16 bit of a class A IP address to calculate its network address (i.e., relegating it to a class B type network). On the other hand many small organizations operate on multiple class C networks, however, in both our models, we equate an organization to only one class C network.

Even with the conservative ideal caching model and a complete cold-start of all proxy servers just moments before the flash crowd, the initial spike in bandwidth consumption drasti-

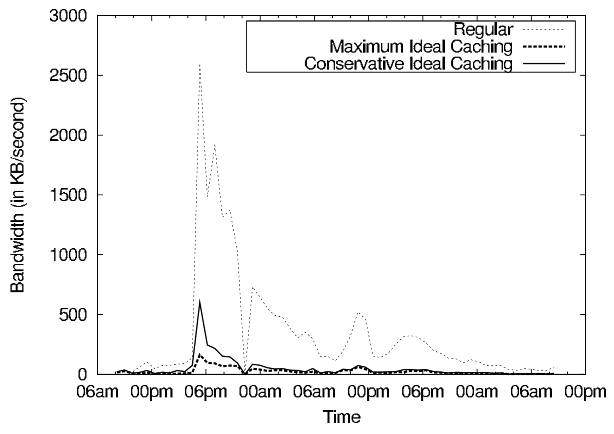


Figure 4: **The flash-crowd afflicted traces of the *Community* site demonstrate the effectiveness of caching in mitigating stress caused by flash crowds. The above shows the bandwidth utilization of the server if an ideal caching infrastructure was deployed at all pertinent organizations (individually identified by network address – see section 2.3).**

cally reduces (by a factor of 5), and the rest of the flash crowds presents minimal traffic.

## 3 Flash Crowd Autodetection

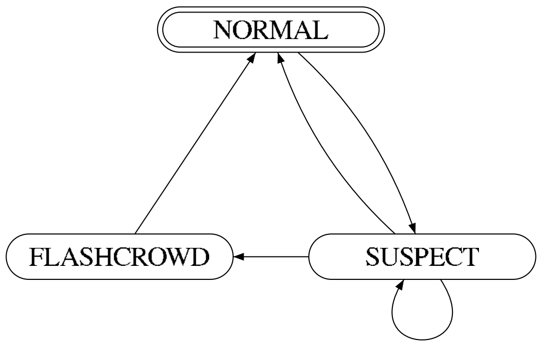
As a website's incoming traffic transitions from normal patterns to that of a flash crowd, substantial increases in resource utilization may serve as an early warning sign of an impending flash crowd. However, a simple hard-threshold based detection scheme may not suffice. Previous flash crowd detection schemes include detecting bandwidth aggregates [13], transmission latencies [5], or exponential moving averages to estimate load [5, 10].

Our goals of a flash crowd detection method are as follows: real-time detection, detection *prior* to the peak, ability to adapt to recent and long-term traffic patterns, low rates of false positives and no false negatives, ability to use different sets of applicable metrics, low resource utilization, applicability to both low and high traffic sites, and tolerance of jitter (rapid and repeated flagging on and off due to variance in load rates).

### 3.1 The Approach

Figure 1 illustrated a typical flash crowd. At the onset, there is a sudden spike in traffic, followed by a peak, then a gradual reduction of traffic back to steady-state levels. This archetypical flash crowd pattern motivates the design of our stochastic detector.

The canonical detector is only run on the server side, and runs at all times. It is represented as the state machine in Figure 5. There are three states: *NORMAL*, *SUSPECT*, and *FLASHCROWD*. Beginning in *NORMAL*, as load increases, the detector switches to *SUSPECT* where it may either continue in *SUSPECT*, return to *NORMAL*, or switch to *FLASHCROWD*. In *FLASHCROWD*, when the load reduces to steady-state levels, the detector finally returns to *NORMAL*.



$$\begin{tikzcd} & \text{NORMAL} \arrow[dr, bend left] & \\ \text{FLASHCROWD} \arrow[ur] & & \text{SUSPECT} \arrow[ul] \arrow[ll] \arrow[loop below] \end{tikzcd}$$

Figure 5: **The three state machine for the stochastic flash crowd detector.**

The stochastic detector we describe below works best if normal traffic actually follows a Poisson distribution. Our experiments show that the stochastic detector works well in practice too. Our approach uses the exponential moving average (EMA) and exponential moving standard deviation (EMSD) to capture the notion of a load and its history with respect to time.

**Exponential Moving Average (EMA)** - At time $t$ quanta, with observed sample $\lambda_t$, and weight $\omega$, the EMA is:

$$\bar{\lambda}_t = \omega\lambda_t + (1-\omega)\bar{\lambda}_{t-1}$$

**Exponential Moving Standard Deviation (EMSD)** - At time $t$ quanta (we expand on the quanta granularity later), with observed sample $\lambda_t$, and weight $\omega$, the EMSD is:

$$\sigma_t = \sqrt{\omega(\lambda_t - \bar{\lambda}_t)^2 + (1-\omega)\sigma_{t-1}^2}$$

#### 3.1.1 Variables and Parameters

The variables and parameters used by the stochastic detector are given as follows, along with an explanation of their intuition.

**Long-term and Short-term Load** - We maintain information about both long-term and short-term load. The long-term load represents the "typical" steady-state load patterns of a given site, i.e., when the site is not experiencing a flash crowd. The short-term load represents "current" load patterns. If the short-term load exceeds the bounds of the long-term load patterns, then there is reasonable suspicion that a flash crowd may be oncoming.

We represent the long-term load with 2 variables: an exponential moving average (EMA) and an exponential moving standard deviation (EMSD). The short-term load, in turn, is represented by one variable: an EMA. In summary, we maintain the following at all times:

1. $\bar{\lambda}_{lt}$ - Long-term EMA.
2. $\sigma_{lt}$ - Long-term EMSD.
3. $\bar{\lambda}_{st}$ - Short-term EMA.

Jitter is the high variance of the short-term load about a certain level. It can cause false positives with respect to detection (we elaborate on jitter later). In addition to having a notion of history, using an EMA and EMSD to represent load has the added benefit of minimizing it due to the weighting mechanism.

**Weights** - The ability of the EMA and EMSD to capture history is due to their weights. A relatively higher weight implies that more consideration is given to the present, while a lower weight implies more consideration to the past. The following weights are needed:

- $\omega_{lt}$ - Weight used to calculate $\bar{\lambda}_{lt}$ and $\sigma_{lt}$ while in state *NORMAL*.
- $\omega_{lt_{fc}}$ - Weights used to calculate $\bar{\lambda}_{lt}$ and $\sigma_{lt}$ while in state *SUSPECT* and *FLASHCROWD*. We set $\omega_{lt_{fc}} < \omega_{lt}$.
- $\omega_{st}$ - Weight used to calculate $\bar{\lambda}_{st}$ while in state *NORMAL*.
- $\omega_{st_{fc}}$ - Weight used to calculate $\bar{\lambda}_{st}$ while in state *SUSPECT* and *FLASHCROWD*. We set $\omega_{st_{fc}} < \omega_{st}$.

**Thresholds** - The detector transitions within *NORMAL*, *SUSPECT*, and *FLASHCROWD* whenever the short-term load matches a certain threshold.

- $\tau_{on}$ - The number of EMSD's that the short-term load's EMA must exceed the long-term load's EMA in order to transition from *NORMAL* to *SUSPECT*.
- $\Delta_{\tau_{on}}$ - The amount the short-term load's EMA must additionally increase to transition from *SUSPECT* to *FLASHCROWD*. In other words $\bar{\lambda}_{st}$ will need to eventually exceed $\tau_{on} + \Delta_{\tau_{on}}$. The motivation is to recognize when the load is experiencing the initial sudden spike characteristic of flash crowds.
- $\tau_{off}$ - The number of EMSD's the short-term load's EMA must fall below the long-term load's EMA in order to transition from *FLASHCROWD* to *NORMAL*. *We set $\tau_{off} < \tau_{on}$, thus giving the detector hysteretic behavior.*

The weights and thresholds are the tunable parameters of the detector. However we use constant values subject to the following discussion.

**Granularity** - Sampling granularity has direct consequences on our choices of values for the parameters. Finer granularity indicates more variance in any given sample. Hence, with finer granularity, we use a lower long-term weight.

As Figure 6 indicates, the weight effects the duration a metric contributes to the aggregate. For instance, at 1-second granularity, with an $\omega = 0.1$, a given metric will contribute over 1% of the EMA and EMSD for 21.85 seconds, and up to 0.1% until 43.70 seconds. The length of the "window" effectively smooths out jitter, and indicates the relevence of a given metric with respect to time.

For the short-term load, we wish to strike a balance between tolerance of jitter (especially at coarse granularity) by keeping the weight low, yet high enough to allow for sensitivity to actual changes in the load pattern. For the long-term load, a

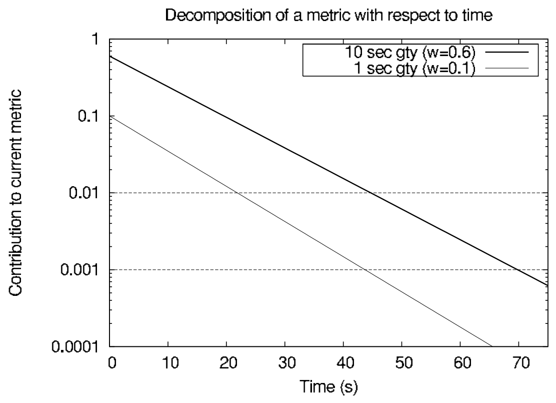


Figure 6: **Decomposition of a metric with respect to time for the short-term weights. For the 10-second granularity with $\omega = 0.6$, the contribution of a metric reaches 0.01 at 44.68 seconds, and 0.001 at 69.81 seconds. For the 1 second granularity with $\omega = 0.1$, 0.01 in 21.85 seconds, and 0.001 in 43.70 seconds.**

low weight is desirable so that "normal" traffic is expressed by a wide timespan of loads.

**Low-pass Filter** With finer granularity, the extreme variance of a low-traffic site may play havoc with accurate detection. We introduce a low-pass filter by adding a rule that if the long-term EMA and EMSD are extremely low (at 1 hit/sec each), then we do not trigger a flash crowd. The low-pass filter should only be necessary for detection on extremely low-traffic sites.

### 3.1.2 Measuring Load

We introduce a more concrete definition of *load*. While it is impossible to obtain an all-encompassing real-time picture of the infrastructure resources demanded during a flash crowd, a quick "snapshot" at various locations will suffice.

**Load** - Load may be determined by incoming client rate, number of unique clients, network bandwidth utilization, processor load, request service latency, failed connections. An observed load metric can thus be constructed from a normalized and weighted summation of the available metrics. Consider $n$ sampled metrics $x_1, x_2, \ldots, x_n$ with weights $\omega_1 + \omega_2 + \ldots + \omega_n = 1$, then let the observed load at a given time be:

$$\lambda = \sum_{i=1}^{n} \omega_i x_i$$

Metrics are sampled at a predetermined frequency as explained earlier. Hence, the load history of a server beginning at $t_0$ quanta would be $\lambda_{t_0}, \lambda_{t_1}, \lambda_{t_2}, \ldots, \lambda_{t_\infty}$.

The intuition behind the summation is to capture the effects of all the metrics being measured. However, a problem may arise when one metric is at high utilization while the others are not. For instance, in the case of a somewhat popular large file (such as a CD-ROM ISO), bandwidth utilization may be at peak capacity, while the actual number of clients stay relatively low.

One solution involves keeping track of $n$ separate short- and long-term loads, their respective thresholds, and to transition state if any on the short-term loads exceeds its threshold. Another solution would be to normalize each metric on an exponential scale — the intuition being that the aggregate load should reflect whether any one metric exceeds its threshold. This latter approach has the benefit of using only one comprehensive number to represent the load, thus obviating the need to keep parallel sets of variables. We urge the latter approach as it reduces resource overhead.

### 3.1.3 Variations of the Approach

We introduce and explore variants of the stochastic approach that were previously considered and tested.

**Jitter** - In terms of our stochastic detector, jitter is the rapid change of the short-term load above and below a threshold, thus yielding a large number of false positives. This behavior is due to the variance of a short-term load centered near the level of the threshold.

The detector deals with jitter by:

- inserting a *SUSPECT* state in-between *NORMAL* and *FLASHCROWD*
- using the short-term EMA $\bar{\lambda}_{st}$ with two different weights $\omega_{st_{fc}} < \omega_{st}$. This leads to a hysteresis-like behavior.

Two alternatives to dealing with jitter are:

- *sliding window* - With a *sliding window* of predetermined length $n$, the simplest manner to represent the the short-term load is as the average of the window. This reduces variance and thus smooths out the short-term load.
- *switch* - Instead of a *SUSPECT* state, a *switch* of predetermined length $n$ can be used to represent the number of consecutive rounds the short-term load $\lambda_{st}$ would need to exceed $\tau_{on}$ before transitioning from *NORMAL* to *FLASHCROWD* (note that this yields a two state machine). The intuition is similar to that of using a *SUSPECT* state. Likewise, a switch may also be used to transition the detector from *FLASHCROWD* to *NORMAL*.

Instead of a *sliding window*, an EMA was chosen for two reasons: the EMA supplies a more natural notion of history by giving more weight to current values, and the EMA does not require the overhead of maintaining and calculating a window of values.

$\Delta_{\tau_{on}}$ replaces the switch from *NORMAL* to *FLASHCROWD* due to its higher flexibility for flagging a flash crowd; the detector is capable of detecting a flash crowd within any number of rounds as opposed to the predetermined number a switch requires.

Likewise, lowering the short-term load weight to $\omega_{st_{fc}} < \omega_{st}$ while in the *FLASHCROWD* state obviates the need for a less flexible predetermined length switch.

**Detecting the spike** - The goal of flash crowd detection is to detect an oncoming flash crowd prior to the peak. $\Delta_{\tau_{on}}$ serves this purpose for the detector.

A prior version of the detector monitored the second derivative of the load, and flagged a flash crowd when the second derivative exceeded a certain threshold. The intuition is to detect when the load is increasing at an accelerated rate, thus marking the beginnings of a flash crowd.

While theoretically elegant and attractive, the practical results demonstrated a less desirable outcome. Prior implementations revealed that variance in actual consecutive load-rates hinders the ability of the detector to judge the short-term trends in real-time. Furthermore, low traffic sites tend to exhibit more traffic variance, and those are precisely the sites that need flash crowd detection.

Additionally, measuring the second derivative of the load rate has the added side-effect of making the threshold value dependent on the absolute magnitude of the site. To make the detector site-size agnostic, a subsequent version measured the second derivative of the short-term EMSD. However, the variance of the short-term loads, especially in low-traffic sites, again hampered detection.

We finally settled on $\Delta_{\tau_{on}}$, which is site-size agnostic, gives the notion of an increasing load, and yields good results.

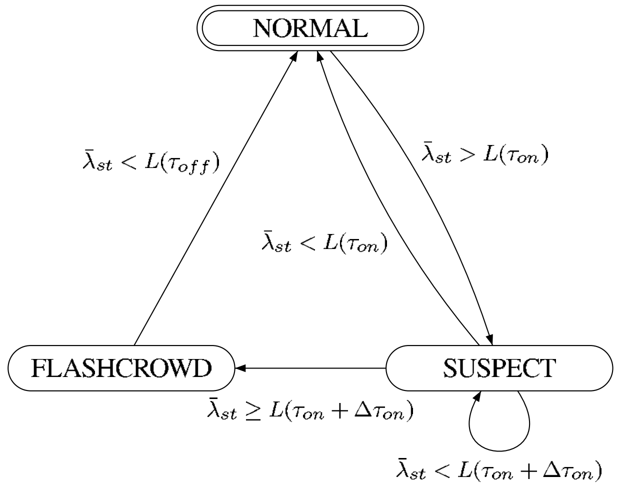

NORMAL
$\bar{\lambda}_{st} < L(\tau_{off})$
$\bar{\lambda}_{st} > L(\tau_{on})$
$\bar{\lambda}_{st} < L(\tau_{on})$
FLASHCROWD
SUSPECT
$\bar{\lambda}_{st} \geq L(\tau_{on} + \Delta\tau_{on})$
$\bar{\lambda}_{st} < L(\tau_{on} + \Delta\tau_{on})$

Figure 7: **State diagram for the stochastic flash crowd detector. Note that $L(\tau_{on}) = \bar{\lambda}_{lt} + \tau_{on}(\sigma_{lt})$. It represents the actual load that is calculated as $\tau_{on}$ EMSD's above the EMA. $L(\tau_{on} + \Delta\tau_{on})$ and $L(\tau_{off})$ are calculated similarly.**

## 3.2 Verification

We have tested and verified the stochastic detector against the site logs using only a small subset of the available resource utilization measures, namely hit-rate. While a comprehensive load can be calculated by normalizing and weighting metrics such as CPU utilization, bandwidth usages, we find that a lone hit-rate metric suffices to yield dependable results.

We present implementation details below, and compare and contrast the results with a strawman approach.

### 3.2.1 Implementation

Figure 7 updates the state machine with the transition cases. As normal load patterns continue, the mode remains *NORMAL*. As a high load occurs as defined by $\bar{\lambda}_{st} > \tau_{on}$, the detector enters into *SUSPECT* and $\omega_{lt}$ is reduced. At this point, if $\bar{\lambda}_{s.t}$ exceeds $\Delta\tau on$ then a flash crowd is flagged and the detector enters *FLASHCROWD*. If, on the other hand, $\bar{\lambda}_{s.t}$ drops below the $\tau_{on}$, the state returns to *NORMAL* and $\omega_{lt}$ is restored.

During *FLASHCROWD*, if the load drops to $\bar{\lambda}_{st} < \tau_{off}$, then the flash crowd is perceived to be over and the detector switches to *NORMAL* mode. If the load stays above the threshold $\tau_{off}$, then the detector remains in *FLASHCROWD*. However, if the load drops below $\tau_{off}$, then the detector returns to *NORMAL*.

**Hard Threshold Strawman** A naive approach would be to have two states, *NORMAL* and *FLASHCROWD*, with transitions from one to the other determined by two hard thresholds: $\tau_{hard_{on}}$ and $\tau_{hard_{off}}$. While in *NORMAL*, whenever the hit-rate $\lambda > \tau_{hard_{on}}$, the state transitions to *FLASHCROWD*. Likewise, when $\lambda < \tau_{hard_{off}}$, the state returns to *NORMAL*. Setting $\tau_{hard_{on}} > \tau_{hard_{off}}$, gives us hysteretic behavior.

We also introduce a modified strawman that is more tolerant of jitter by using a *switch* of length 5. In order to transition from *NORMAL* to *FLASHCROWD*, the hit-rate $\lambda$ would need to exceed $\tau_{hard_{on}}$ consecutively for 5 times. We also include a separate switch of the same length for the transition from *FLASHCROWD* to *NORMAL*.

| Gty | $\omega_{lt}$ | $\omega_{lt_{fc}}$ | $\omega_{st}$ | $\omega_{st_{fc}}$ | $\tau_{on}$ | $\Delta_{\tau_{on}}$ | $\tau_{off}$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.001 | 0 | 0.1 | 0.001 | 2.5 | 2 | 0.5 |
| 10 | 0.002 | 0 | 0.6 | 0.01 | 2.5 | 11 | 1 |

Table 2: **Parameters for the stochastic detector for 1 & 10 sec granularities. The detector is run with the same parameters for all sites. The finer granularity uses lower weights $\omega_{lt}$, $\omega_{lt_{fc}}$, $\omega_{st}$, and $\omega_{st_{fc}}$ in order to prolong the decomposition of the metric with respect to rounds. The thresholds $\Delta_{\tau_{on}}$ for both granularities are similar, with differing $\Delta_{\tau_{on}}$ — this is due to the high variance (and reduced possibility of consecutive increases of load) at finer granularities. The lower $\tau_{off}$ for the finer granularity is to prolong the rounds needed to flag off a flash crowd.**

### 3.2.2 Results

We ran the stochastic, strawman, and switched strawman detectors at 1 and 10-second granularity with the parameters in Tables 2 and 3.

The strawmen parameters were determined by an initial running of the stochastic detector on each site, and putting in detected values. For every given website, we set $\tau_{hard_{on}}$ to the stochastic detector's calculated threshold at the onset of the flash crowd. In other words, the strawmens' parameters are set to go off at precisely the thresholds that the stochastic model itself calculated and used. Likewise, we set $\tau_{hard_{off}}$ to the stochastic detector's calculated value.

| | 1s granularity | | 10s granularity | |
|---|---|---|---|---|
| Site | $\tau_{hard_{on}}$ | $\tau_{hard_{off}}$ | $\tau_{hard_{on}}$ | $\tau_{hard_{off}}$ |
| *Building* | 3.77 | 1.65 | 18.03 | 9.98 |
| *Webnote* | 4.11 | 1.83 | 25.53 | 11.74 |
| *Bell* | 5.57 | 3.55 | 52.42 | 35.66 |
| *Community* | 7.75 | 4.91 | 86.13 | 55.86 |
| *Kernel* | 6.86 | 4.76 | 72.99 | 52.11 |
| *WCup98* | 629.52 | 493.82 | 6143.65 | 4525.59 |

Table 3: **Parameters for the strawmen detectors. The values are calculated from an initial run of the stochastic detector on the respective sites. These values are more generous than those that were "eyeballed" by looking at a plot of the traffic of each respective website.**

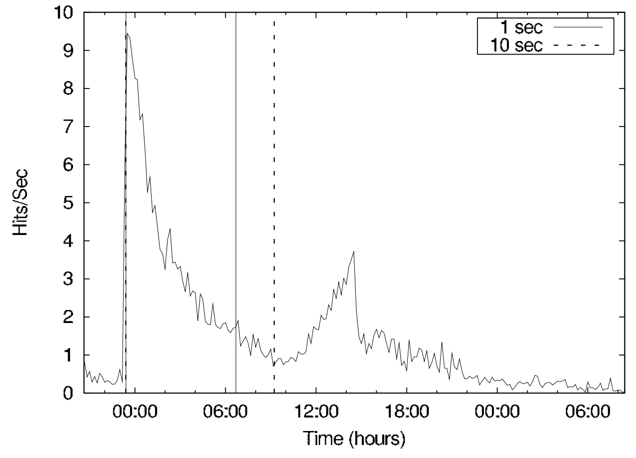
It is interesting to note that, after choosing these stochastic detector values, we proceeded to look at the graphs of each website, with the intention of selecting a separate set of hard thresholds to test. To our surprise, we discovered that the values calculated by the stochastic detector *were more generous to the strawman*, with respect to possible false positives, than our "eyeballed" values.

Figure 8 shows plots of the websites and their respective opening and closing boundaries of flash crowds as detected by the stochastic detector at 1 and 10-second granularity. The focus of the plots are the actual flash crowds, while the actual detectors were run on a much larger timespan of the logs. Tables 5 and 4 shows the results of running the detectors on the longer timespans with respect to false positives and false negatives, as well as timeliness.

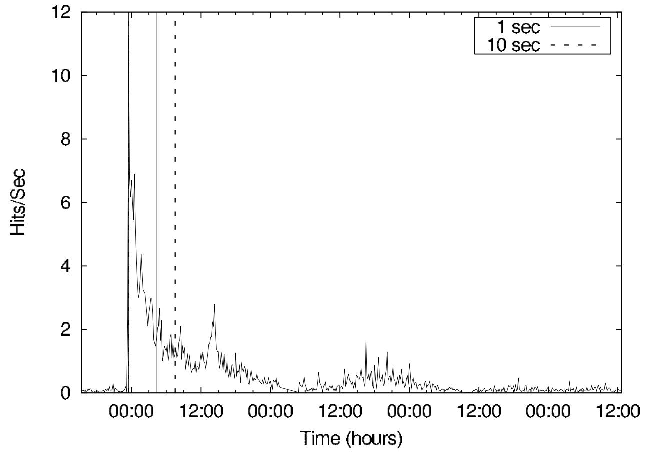
From the plots of Figure 8, we see that the stochastic detector is able to flag flash crowds during the initial spike, and *prior to the peak* with low rates of false positives. The same parameters were used for all the plots, with the exception of *WCup98*.

The failure to detect the *WCup98* flash crowd with the generic parameters may be a consequence of the size of *WCup98*. *The steady-state hit-rates are up to 2 orders of magnitude larger than any of the other websites.* At 1-second granularity, the stochastic detector flagged on hours prior to the actual onset of the flash crowd, and flagged off the flash crowd hours after the actual flash crowd had passed. In other words, the stochastic detector with a $\Delta_{\tau_{on}} = 2$ was too sensitive for *WCup98*. We re-ran the stochastic detector on *WCup98* with a $\Delta_{\tau_{on}} = 4$, and detected the flash crowd with no false positives. The failure to detect at 10-second granularity may be explained by the size of the site, with the sheer numbers buffering and restricting how fast the hit-rate can spike. Hence, $\Delta_{\tau_{on}} = 11$ is too high a setting. We re-ran the stochastic detector with $\Delta_{\tau_{on}} = 5$, and detected the flash crowd with no false positives.

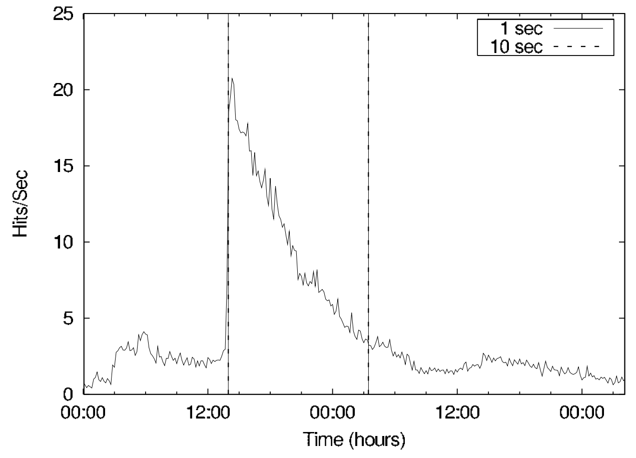
*Kernel* in Figure 8(e) contained two flash crowds. It also has a strong diurnal pattern as evidenced by the periodic mini-peaks in the plot, hence the tail-end of the second flash crowd blends in with the oncoming peakṘaising $\Delta_{\tau_{on}} > 2$ would solve the problem.



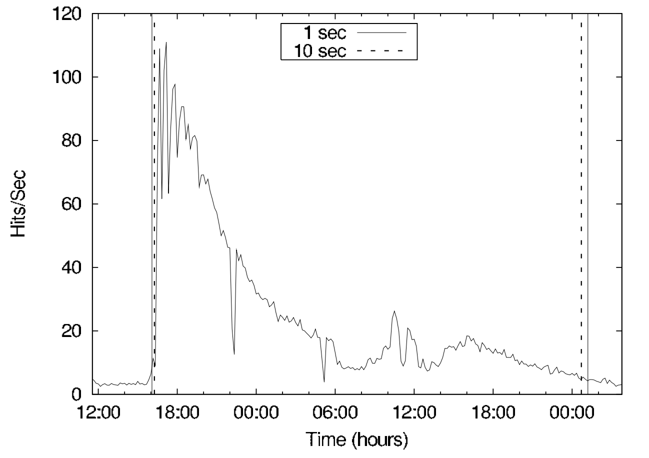
(a) *Building*: Note that the starting boundaries overlap for 1 and 10 second granularity.



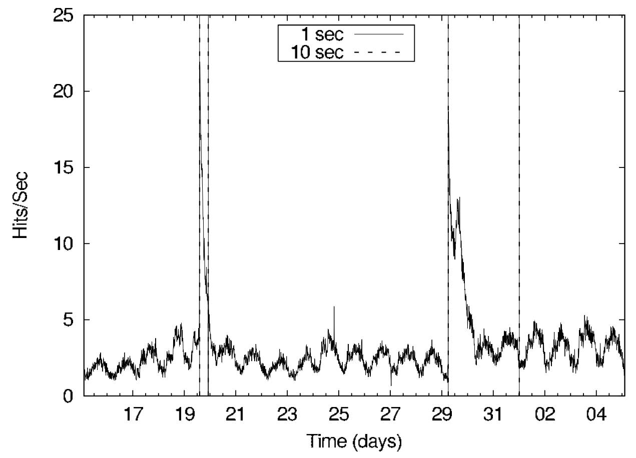
(b) *Webnote*: Note that the starting boundaries overlap for 1 and 10 second granularity.



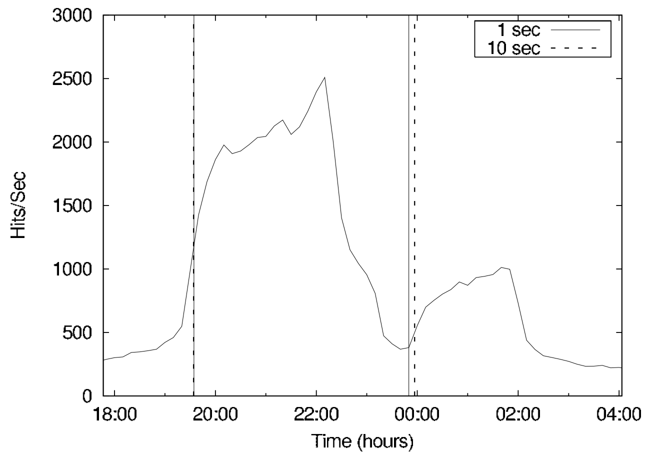
(c) *Bell*: Note that the starting and ending boundaries overlap for 1 and 10 second granularity.



(d) *Community*: Note that the starting boundaries overlap for 1 and 10 second granularity.



(e) *Kernel*: Two flash crowds are detected. The second flash crowd is flagged off later than expected due to the diurnal patterns of the website evident in the graph; the tail end of the flash crowd joins with the beginning of a diurnal peak.



(f) *WCup98*: Note that the starting boundaries overlap for 1 and 10 second granularity. Ran with $\Delta_{\tau_{on}} = 4$ at 1-second granularity, and $\Delta_{\tau_{on}} = 5$ at 10-second granularity.

Figure 8: **Flashcrowds and the detection boundaries of the stochastic detector at 1 and 10-second granularity. Note that for some plots, the 1 and 10-second granularity boundaries overlap.**

|  | Stochastic |  |  |  | Strawman |  |  |  | Switched Strawman |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 1s granularity |  | 10s granularity |  | 1s granularity |  | 10s granularity |  | 1s granularity |  | 10s granularity |  |
| site | 1st | peak | 1st | peak | 1st | peak | 1st | peak | 1st | peak | 1st | peak |
| *Building* | 155 | 1715 | 190 | 1750 | N/A | N/A | 230 | 1790 | 204 | 1764 | 190 | 1750 |
| *Webnote* | 142 | 142 | -20 | -20 | N/A | N/A | N/A | N/A | 156 | 156 | 230 | 230 |
| *Bell* | 80 | 1340 | 60 | 1320 | N/A | N/A | 100 | 1360 | 85 | 1345 | 60 | 1320 |
| *Community* | 679 | 2059 | 10 | 1390 | N/A | N/A | -2310 | -930 | -1034 | 346 | -1070 | 310 |
| *Kernel* #1 | 81 | 2241 | 10 | 2170 | N/A | N/A | 130 | 2290 | 122 | 2282 | 20 | 2180 |
| *Kernel* #2 | 49 | 469 | 60 | 480 | N/A | N/A | 80 | 500 | 58 | 478 | 40 | 460 |
| *WCup98* | 1531 | 9751 | 1560 | 9780 | 1903 | 10123 | 1900 | 10120 | 1887 | 10107 | 1860 | 10080 |

Table 4: **The timeliness of the stochastic and strawmen detectors. Each value is the number of seconds the flash crowd was detected** *prior* **to the 1st apex and maximum peak of the flash crowd, for 1 and 10 second granularities. Hence, the larger the value, the earlier the detection, and the better the performance.** ***Kernel*** **had two flash crowds as reflected in the table.**

**Comparison** - Table 5 shows the number of false positives and negatives for the stochastic and strawmen detectors. For both granularities, the stochastic detector performed with low rates of false positives (orders of magnitude lower than the strawmen detectors), and with the alternative $\Delta_{\tau_{on}}$ for *WCup98* as explained above, detected every flash crowd. The strawman fared poorly with huge rates of false positives. Detection at 1-second granularity was almost non-existent. The switched strawman fared better, detecting every flash crowd, albeit with rates of false positives orders of magnitude higher than that of the stochastic detector.

Table 4 indicates the timeliness of the flash crowd detection. Larger values indicate earlier detection, and hence better performance with respect to timeliness. The "1st" column indicates the number of seconds prior to the first apex of the flash crowd. The "highest" column indicates the timeliness of the detection with respect to the maximum peak rate that the flash crowd experiences.

For the majority of sites, at 1-second granularity, the stochastic detector detected the flash crowd in the range of 49–142 seconds prior to the first apex of the flash crowd. For *Community* and *WCup98*, the performance is markedly better with 679 and 1531 seconds prior to the first apex, respectively. From the peak, the performance is spread over a large range, and is fundamentally dependent upon the particular characteristics of each website (*Webnote*'s 1st apex is the same as it's maximum peak). With regards to *Webnote*, for 10-second granularity, the stochastic detector signaled a flash crowd 20 seconds *after* the initial apex. This is due to the relative short duration of *Webnote*'s initial spike relative to the 10-second granularity. The 10-second granularity is simply too wide to detect *Webnote*'s flash crowd in time; a finer granularity is needed. An alternative solution would involve setting a lower threshold value for *Webnote*.

The strawman detector fare poorly at 1-second granularity simply because it never detected the flash crowds. Actually, dozens, if not hundreds, of flash crowds were detected, each with durations on the order of single to tens of seconds. These detections are not useful in any practical sense, and so we discount them. At 10-second granularity, with the exception of *Community*, the strawman appears to perform well with the values supplied by the stochastic detector, though one must take into consideration the number of false positives also generated.

The switched strawman, with the thresholds determined by the stochastic detector, performs comparatively to the stochastic model with regards to timeliness. However, the considerable amount of false positives detracts from its overall performance and feasibility of practical use.

## 3.3 Summary

In this section, we introduced a stochastic approach of detecting flash crowds. We show that by keeping track of a few variables, real-time early detection of flash crowds, with a minimal number of false positives, is practical.

Future directions of work may include taking diurnal or weekly patterns into account, and a method of automatically tuning parameters. A more comprehensive detector would use historical reference to interpret the current load patterns and to tune the parameters with respect to site-size and developing load patterns.

The stochastic detector is simple and adaptable to many uses. We combine the stochastic detector with a flash crowd mitigation solution later on in Section 5.

## 4 Comparison of Content Providing Solutions

To the best of our knowledge, no previous research has studied the effectiveness of existing content providing solutions in fighting flash crowds. We perform a trace-based simulation to determine the effectiveness of different classes of solutions. We measure both resource utilization that effect the server (and directly the content provider's costs) along with the quality of service to the clients (i.e., the latency to satisfy incoming requests).

## 4.1 Taxonomy

Solutions designed to mitigate stress due to flash crowds can be categorized into four broad classes based on their under-

| | Stochastic | | | | Strawman | | | | Switched Strawman | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1s gty | | 10s gty | | 1s gty | | 10s gty | | 1s gty | | 10s gty | |
| Site | + | - | + | - | + | - | + | - | + | - | + | - |
| *Building* | 4 | 0 | 3 | 0 | 28229 | 1 | 1792 | 0 | 210 | 0 | 28 | 0 |
| *Webnote* | 1 | 0 | 2 | 0 | 9525 | 1 | 724 | 1 | 107 | 0 | 8 | 0 |
| *Bell* | 2 | 0 | 3 | 0 | 41074 | 1 | 879 | 0 | 476 | 0 | 7 | 0 |
| *Community* | 8 | 0 | 0 | 0 | 70978 | 1 | 2554 | 0 | 3152 | 0 | 31 | 0 |
| *Kernel* | 0 | 0 | 0 | 0 | 76083 | 1 | 4446 | 0 | 62 | 0 | 20 | 0 |
| *WCup98* | 6 | 0/1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *WCup98* * | 0 | 0 | 0 | 0 | | | | | | | | |

Table 5: **False positives (+) and false negatives (-) of the stochastic, strawman, and switched strawman detectors at 1 and 10-second granularity. *Note that for the stochastic detector, *WCup98* had false negatives (for 1-second granularity, the flash crowd was flagged on hours before the actual flash crowd, and flagged off hours after). We re-ran with a different parameter of $\Delta_{\tau_{on}} = 4$ and $\Delta_{\tau_{on}} = 5$ for 1 and 10-second granularity, respectively, which yielded much better results.**

lying design methodologies. We enumerate the taxonomy below. In doing so, we select one representative from each class.

### 4.1.1 Single Server Solutions

Solutions in this class modify server behavior only (leaving the client alone). These solutions can be deployed autonomously by the website administrator and are intended to work with the current infrastructure. Brecht et al [4] discuss multiple strategies to improve performance of standalone Web servers. Capriccio [21] is a user mode thread library that can be utilized to construct a high throughput server. The Haboob web server utilizes Staged Event-Driven Architecture (SEDA) [22] to improve performance.

The above solutions seek to achieve optimal performance out of the available resources. They provide better throughput and quality of service to visitors during peak traffic than popularly deployed servers like Apache [14]. For example, the Haboob server is able to handle thousands of concurrent connections (given the availability of required network resources). However, the effectiveness of such solutions degrades (or even vanishes) when constraints are applied on network resources.

On the other hand, admission control schemes have been proposed to mitigate flash crowds [10, 8], by controlling resource utilization. Admission control schemes improve the quality of service to a subset of clients, at the cost of providing sub par service to the remaining clients.

### 4.1.2 Server Cooperation

Pseudoserving [11], Coral [6], Backslash [19], and DotSlash [24] are schemes that distribute content with the aid of cooperative servers. These methods provide mechanisms to automatically replicate ("mirror") popular content.

Collectively, solutions in this category reduce the network bandwidth utilization by dividing up the work load transparently amongst a set of cooperating nodes (i.e., other servers). However, the sum of networking resources required to mask the effects of a flash crowd actually increases with this class of solutions due to the overhead of replicating content. Coral uses a set of volunteer nodes to circumvent the need for additional bandwidth (at least from the website administrator's point of view), while the other schemes require additional nodes controlled either directly by the website administrator, or indirectly through out-of-band agreements.

One problem with this class of solutions is that dynamic content is often still served from the primary web server. Platform and database dependencies make it simply inefficient to migrate content generation to another server. While DotSlash [24] does mitigate some of the issues relating to platform dependencies by using a resource discovery protocol, it fails to provide a consistent solution for database migration. In that respect, DotSlash just migrates the bottleneck from the web server to the database server, which may not be an acceptable solution for many Web sites.

However, if utilized effectively for static content, a flash crowd stress can be mitigated through these solutions. On the other hand, reliance on third parties (including volunteers) impairs self-reliability.

### 4.1.3 Cooperative Client Caching

One solution to mitigate dynamic stress is to exploit the synergy of clients to form a cooperative caching model. Squirrel [7] and Kelips-based cooperative caching [12] are constructed on top of peer-to-peer routing substrate and allow clients to share their local (i.e., web browser) caches amongst each other. Such systems can be used to provide hot content to cooperating clients. Traditional centralized web caching proxies also present an alternate form of cooperative caching (from the clients' perspective). The intermediate proxy server is as the aggregator of client caches, however, due to real-world limitations, the central proxy may not be as encompass the net total of all local caches.

As content popularity models a Zipf distribution, cooperative caching has been shown to have limited benefit [23] on a large scale. Hence, deployment of caching infrastructure does not cross organizational boundaries. However, even with those limitations, the ideal caching model shown in Section 2.3 is an effective solution to mitigate flash crowds.

However, many organizations feel that organization-wide central caching proxies are cost inhibitive. However, decentralized cooperative caching can reduce costs and improve the quality of service, if widely deployed.

We do not explicitly experiment with cooperative caching solutions; instead we note that the server-client solutions use cooperative caching within them. The success of server-client solutions could be used to argue for pure cooperative caching.

### 4.1.4 Server-Client Solutions

To improve performance of content delivery systems, DHTTP [17] was designed on enable caching along the Internet route. Others have proposed solutions that involve client cooperation with minimal aid from the Web server. For example, in CoopNet [15], the server maintains a list of volunteer clients willing to serve content to their peers. By constructing a churn-resistant overlay, PROOFS [20] allows for content distribution using a peer-to-peer protocol. Overhaul [16] breaks each requested document into "chunks" and distributes them sequentially amongst clients. The clients cooperate with each other by exchanging chunks to reconstruct a coherent document. Overhaul also supports one-hop search to retrieve additional documents from peer clients.

Such solutions reduce the network bandwidth utilization of a web site under flash crowd traffic. However, they present a challenge in widespread deployment due to the acceptance of established standards.

## 4.2 Simulation

To test the effectiveness of each class of solution in mitigating the effects of flash crowd, we chose a representative from each class to be modeled. However, to effectively test each solution, we simulated the models under real-world resource constraints.

### 4.2.1 Setup

For the first part of our setup, we determine the server resource provisions (i.e., network capacity) needed to provide good quality of service to the clients. Since the Web is primarily an interactive medium, users expect the documents to be fetched almost instantaneously. However, a small fraction of documents fetched are abnormally large. These documents tend to be large archive files, media files, or program utilities. This fact was verified by our traces. For example, in the *Bell* trace, the largest successful document transfer was approximately 52 MB, however, the average document request was only 48 KB the median document only 2 KB. The largest 1% of documents requested were all larger than 113 KB.

To provide good quality of service, most document requests should be completed within a few seconds. Obviously, it may not be possible to transfer a large document (like the 52 MB file) within that time span. We used this fact as the guideline to calculate the resource provisions for a website. We used the following maxim: 99% of incoming requests must be satisfied within 10 seconds (assuming the client is not the bottleneck), including concurrent connections from the same client. The support for providing good quality of service to concurrent connections is necessary because modern browsers establish multiple such connections to fetch embedded documents like images, style sheets, etc. From our traces, the network resources provisioned using the above maxim were consistently at least six to eight times more than average utilization. In other words, the provisions should be able to handle most of the traffic created by the flash crowd, except for a few hours around peak utilization.

Throughout our simulations, we assume that the client is not the bottleneck during the flash crowd event. This allows us to calculate the best (or ideal) response time for a solution. While the client is generally the bottleneck in a network, and can potentially degrade the quality of service for other clients, it is not the case during a flash crowd. Our traces show that when the server saturates its network capacity, it tends to become the bottleneck rather then the client.

**Regular** The behavior of the popular web server Apache is modeled by this category. Both Apache versions 1.x and version 2.x (with the pre-fork model) defaults to serving a maximum of 256 clients simultaneously. Apache is also configured to drop a connection (timeout) after 300 seconds, either due to client inactivity, or due to its inability to start serving the request (for example, under a flash crowd load).

While these defaults can be changed easily, we model the default Apache behavior.

**Admission Control** We model an admission control scheme where only a maximum of $x$ documents are served concurrently. Any extra concurrent hits are kept in the pending queue, only to be served when a slot from the serving queue is vacated. The next request from the pending queue is chosen using two different policies: the first-in first-out (FIFO) and the last-in first-out (LIFO).

We utilize $x = 5$ for our experiments, as it yields good results. The client timeout value of 300 seconds is also enforced. We perform admission control and other measurements at the HTTP level.

**Cooperative Server** The cooperative server scheme is fairly simple. We utilize $n$ servers, with the ability to transparently redirect requests amongst themselves. However, all dynamic content is served from the primary server.

We vary the number $n$ in our experiments. The client timeout value of 300 seconds is also enforced.

**Server-Client Collaboration** The client-server model is based on Overhaul [16]. A server (under flash crowd load) divides a document in to $c$ chunks and maintains a list of the last $m$ clients (per document). The clients are sent a single chunk (distributed sequentially) of the requested document along with an addendum header containing verification signatures for the $c$ chunks and the initial membership list.

The visitors form a p2p overlay on the fly, based on the initial membership list, where they exchange chunks and discover new peers (as other clients join in). No distributed hash table is used. Moreover, peers discover and fetch dependent documents autonomously, without server intervention. However, if a client is unable to locate additional chunks of a document (for example, for a dynamic document) within time $t$, it requests an additional chunk from the server. This process is repeated until the all requests are satisfied.

We vary the number of chunks, $c$ in our experiments. However, we use fixed values for other variables, $m = 50$ and $t = 5$ seconds. Additionally, to err on the conservative side, we assume that no client remains in the system for longer than 5 minutes after the last request. Previous studies [1, 18] on web sessions have shown that the most users remains on a web site for approximately 5 to 15 minutes.

### 4.2.2 Results

**Bandwidth Utilization** Figure 9 includes the graphical summary of resource utilization for simulation results from *Bell* trace data. Figure 9(a) plots the bandwidth utilization of the flash crowd as it happened, i.e., with sufficient over provision of resources to handle the flash crowd.

However, if the simulation is modeled with restriction on bandwidth resources as calculated from our maxim (see section 4.2.1), there is a severe degradation of service. From the pre-flash crowd history of the trace data, we calculated that 204 KB/second is ample bandwidth to sufficiently satisfy 99% of all incoming requests with 10 seconds.

With a strict limit on the bandwidth resources (see Figures 9(b)-9(f)), the network capacity is quickly saturated as the flash crowd arrives. The network remains saturated for nearly 6 hours. Admission control with FIFO policy fares about the same. The network remains saturated for a little less than 6 hours. However, admission control with LIFO policy fares a little better, when the network capacity is only permeated for about 5 hours. Cooperative server schemes fare much better, even with only 3 servers, as the network is never saturated. Finally, the client-server model utilizes the lowest amount of bandwidth when "chunking" document requests. Even with chunking disabled (to decrease client fetch latencies), the client-server model never reaches the network saturation point. However, as soon as document chunking is turned off, there is a sudden spike in bandwidth utilization. While the spike does saturate the network capacity, it is critical to have a detector that is able to turn off chunking only at the appropriate time.

**Client Latencies** The various schemes yield a varying range of service quality. The regular Web server model, with ideal amount of resources, is able to contain the flash crowd comfortably. It is able to serve 100% of the requests within 1 second. However, numerous client connections (approximately 10% each) were dropped for all three schemes that pierced the saturation point (see Figure 10). Both the regular and the FIFO-policy admission control servers had severely degraded

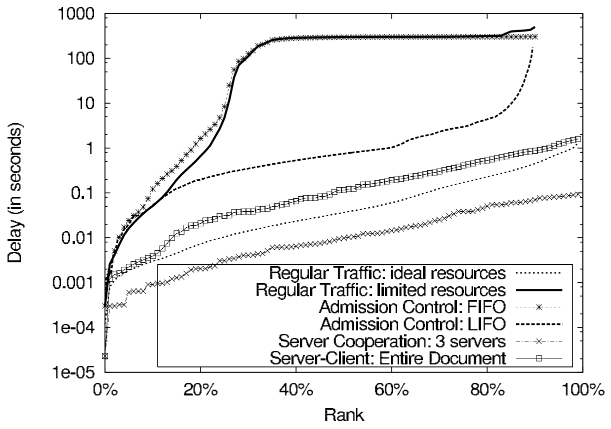


Figure 10: **A comparison of latencies (the minimum time needed to serve a request since its arrival) for various schemes under a flash crowd stress. The simulation results were conducted using the real traces of the *Bell* website.**

quality of service. Approximately 75% of client requests took over 10 seconds to be satisfied. On the other hand, the LIFO-policy admission control server fared significantly better: only about 20% of the client requests were not served within 10 seconds. The cooperative server scheme fared best overall (all requests satisfied within a second of arrival), as it was able to transfer a majority of its traffic to other servers. The server-client solution was also able to satisfy all of its request in time comparable to the cooperative server approach. However, this is due to the fact that most of the requests were satisfied by the p2p overlay.

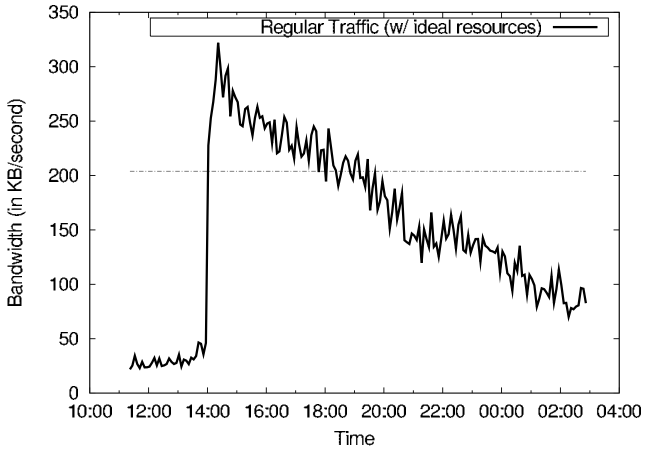
## 5 Adaptive Mitigation

We combine the stochastic flash crowd detector developed in Section 3, with a server-client solution to design an adaptive chunking solution. We test the performance of our adaptive solution via simulation.

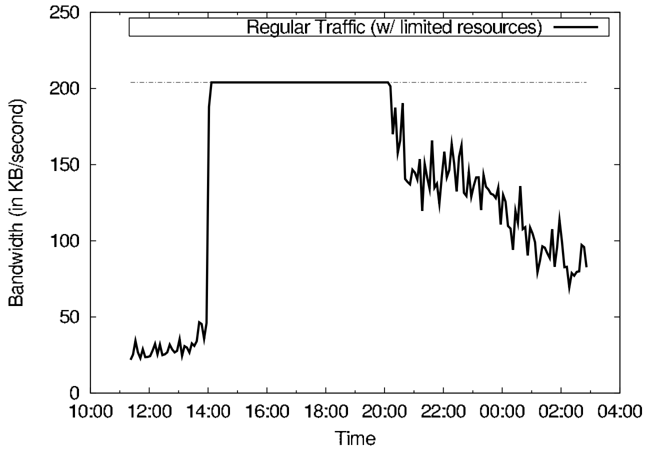
### 5.1 Adaptive Chunking

Presently, a server-client solution mitigates server load by breaking up requested web documents into a fixed number of chunks. As shown in section 4.2.2, the performance of such a scheme depends on the number of chunks, and the effectiveness of the flash crowd detector. We integrate the stochastic autodetector with Overhaul. In this hybrid solution, the stochastic detector enabled the client-server solution to adaptively determine the degree of chunking.

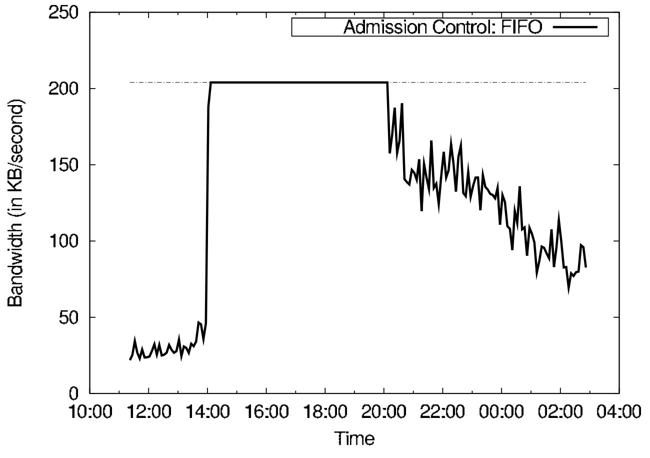
To review, when the short-term load $\bar{\lambda}_{st}$ exceeds the long term load $\bar{\lambda}_{lt}$ by $(\tau_{on} + \Delta_{\tau_{on}})$ long-term exponential standard deviations $(\sigma_{lt})$, a flash crowd is flagged. The adaptive mechanism works in conjunction with this: while in the *FLASHCROWD* state, let $\Delta_{st} = (\bar{\lambda}_{st} - \bar{\lambda}_{lt})/\sigma_{lt}$. In other words, $\Delta_{st}$ is the number of EMSD's the current load is above the long-term EMA. The detector calculates the current number of chunks as follows:



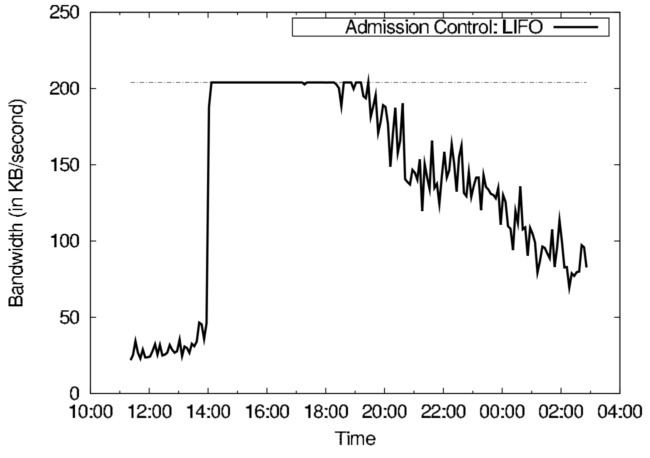
(a) Regular Web Server with an ideal provision of resources



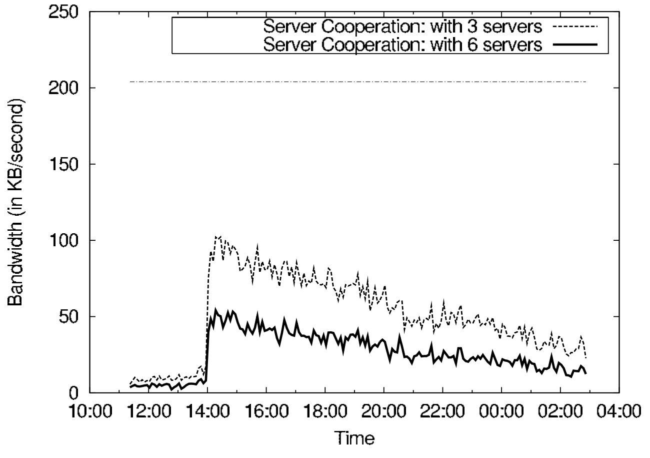
(b) Regular Web Server with limited provision of resources



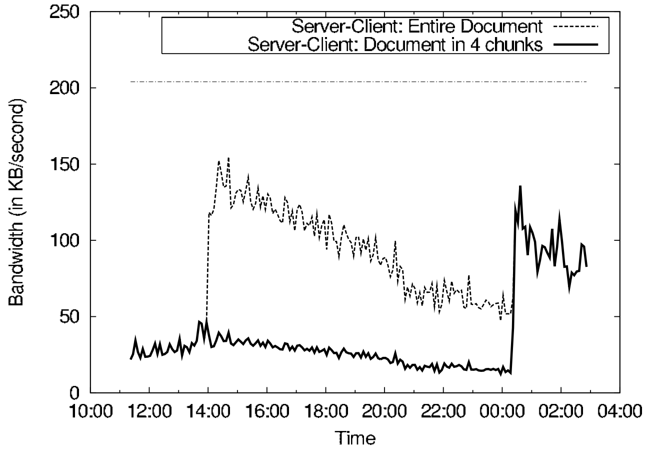
(c) Admission Control: FIFO



(d) Admission Control: LIFO



(e) Cooperative Server



(f) Client-Server

Figure 9: **The bandwidth utilization of various schemes under a flash crowd stress. The simulation results were conducted using the real traces of the *Bell* website.**

$$N_{chunks} = 2^{\lfloor \log_2 (\frac{\Delta_{st}}{\tau_{on}}) \rfloor + 1}$$

Intuitively, this calculates the number of times to double $\tau_{on}$ in order to reach $\Delta_{st}$. We keep the number of chunks as a power-of-2 so that clients possessing old chunks (smaller chunks), may still actively fetch chunks from newer clients, and vice versa.

## 5.2 Results

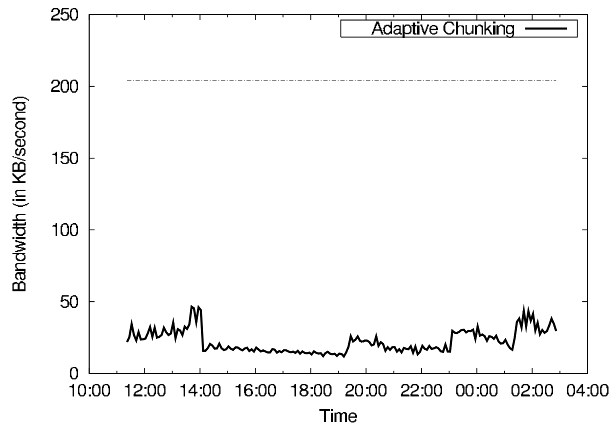


Figure 11: **The performance of our adaptive mitigation scheme under a flash crowd stress. The bandwidth utilization of the server remains in the normal range (even dipping below normal utilization levels) throughout the flash crowd. The simulation results were conducted using the real traces of the *Bell* website.**

We simulate our adaptive solution with the same parameters as the fixed-chunking server-client solution in Section 4.2.2. As shown in Figure 11, our solution performs exceedingly well under a flash crowd stress. Our adaptive method removes the sudden spike in bandwidth utilization near the end of the flash crowd, as the number of chunks degrades gradually. In fact, in this particular trace, the adaptive mitigation method utilizes less bandwidth then during normal operations. Moreover, the bandwidth use never approaches the saturation point.

## 6 Conclusion

We used traces collected from six different flash crowds to characterize the properties of flash crowds, showing that organizational deployment of ideal caching schemes can mask flash crowds. We developed a stochastic detector that works in real-time, i.e., 50-100 seconds before the first apex. We then combine the detector with cooperative caching to implement a system that adaptively masks the effect of flash crowds. This effectively turns the flash crowd in to a "smart mob". We also include a comprehensive study by comparing the performance of the above design with representative systems from three classes of existing designs for content providing - single server, server cooperation, and server-client modification. We show that, during a flash crowd, adaptive mitigation utilizes bandwidth equaling normal utilization levels.

## References

[1] M. F. Arlitt. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, Sept. 2000.

[2] M. F. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May 2000.

[3] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, Oct. 1997.

[4] T. Brecht, D. Pariag, and L. Gammo. accept()able strategies for improving web server performance. In *Proceedings of the USENIX 2004 Annual Technical Conference*, pages 227–240, June 2004.

[5] X. Chen and J. Heidemann. Experimental evaluation of an adaptive flash crowd protection system xuan chen and john heidemann.

[6] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI 2004)*, pages 239–252, Mar. 2004.

[7] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 213–222, July 2002.

[8] H. Jamjoom and K. G. Shin. Persistent dropping: An efficient control of traffic aggregates. In *Proceedings of SIGCOMM 2003*, pages 287–298, Aug. 2003.

[9] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of servie attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International World Wide Web Conference (WWW 2002)*, pages 293–304, May 2002.

[10] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI 2004)*, May 2005.

[11] K. Kong and D. Ghosal. Mitigating server-side congestion in the internet through pseudoserving. *IEEE/ACM Transactions on Networking*, 7(4):530–544, Aug. 1999.

[12] P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems*, pages 1–10, Oct. 2003.

[13] R. Mahajan, S. Bellovin, S. Floyd, J. Vern, and P. Scott. Controlling high bandwidth aggregates in the network, 2001.

[14] Netcraft. February 2005 web server survey. http://news.netcraft.com/archives/2005/02/ .

[15] V. N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, pages 178–190, Mar. 2002.

[16] J. A. Patel and I. Gupta. Overhaul: Extending HTTP to combat flash crowds. In *Proceedings of the Ninth International Workshop on Web Content Caching and Distribution (WCW 2004)*, pages 34–43, Oct. 2004.

[17] M. Rabinovich and H. Wang. DHTTP: An efficient and cache-friendly transfer protocol for web traffic. In *Proceedings of IEEE INFOCOM 2001*, pages 1597–1606, Apr. 2001.

[18] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa. A freamework for the evaluation of session reconstruction heurisitcs in web-usage analysis. *INFORMS Journal on Computing*, 15(2):171–190, 2003.

[19] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pages 203–213, Mar. 2002.

[20] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust, p2p system to handle flash crowds. In *Proceedings of the Tenth IEEE International Conference on Network Protocols*, pages 226–235, Nov. 2002.

[21] R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer. Capriccio: Scalable threads for internet services. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 268–281, Oct. 2003.

[22] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pages 268–281, Oct. 2001.

[23] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)*, pages 16–31, Dec. 1999.

[24] W. Zhao and H. Schulzrinne. DotSlash: A self-configuring and scalable rescue system for handling web hotspots effectively. In *Proceedings of the Ninth International Workshop on Web Content Caching and Distribution (WCW 2004)*, pages 1–18, Oct. 2004.