Effect of Malicious Synchronization

Mun Choon Chan, Ee-Chien Chang, Liming Lu, and Peng Song Ngiam

Department of Computer Science
National University of Singapore
{chanmc,changec,luliming,ngiampen}@comp.nus.edu.sg

Abstract. We study the impact of malicious synchronization on computer systems that serve customers periodically. Systems supporting automatic periodic updates are common in web servers providing regular news update, sports scores or stock quotes. Our study focuses on the possibility of launching an effective low rate attack on the server to degrade performance measured in terms of longer processing time and request drops due to timeouts. The attackers are assumed to behave like normal users and send one request per update cycle. The only parameter utilized in the attack is the timing of the requests sent. By exploiting the periodic nature of the updates, a small number of attackers can *herd* users' update requests to a cluster and arrive in a short period of time. Herding can be used to discourage new users from joining the system and to modify the user arrival distribution, so that the subsequent *burst attack* will be effective. While the herding based attacks can be launched with a small amount of resource, they can be easily prevented by adding a small random component to the length of the update interval.

Keywords. Network security, Distributed Denial of Service (DDoS) attacks, low rate DDoS attack, synchronization, periodicity, herding.

1 Introduction

There are many applications in the Internet that utilize periodic updates. Some common examples are stock quote update, news update and sport score update. Less common examples, but gaining popularity, are web cameras that provide images of highways, scenic views, or various sites under surveillance. Popular news web sites like CNN (www.cnn.com), Wall Street Journal (www.wsj.com) and The New York Times (www.nytimes.com) perform automatic refresh every 1800s, 900s and 900s respectively. For sport events, many sport related web sites provide periodic score updates, commonly in the intervals of 30s, 60s or 90s. In global events like the Olympics, tremendous amount of traffic reaches a relatively small number of servers for periodic updates. In this paper, we study the potential of malicious synchronization on such systems.

The main role of periodic updates is to "spread" the users over the update interval, so as to obtain a trade-off between the server's resources and timeliness of the service. An implicit assumption is that since users arrive randomly, it is likely the arrivals are also randomly distributed over the update interval.

In general, periodic updates can be performed in an *absolute* or *relative* manner. In absolute periodic update, once a user begins an update at T_0, subsequent updates are performed at time T_i^a, $i = 1, 2, 3...$, where $T_i^a = T_0 + i * P$ and P is the update period. On the other hand, in relative update, the next update is scheduled P seconds after the completion of the current update. Hence, $T_i^r = T_{i-1}^r + N_i + P$, where N_i is a positive component that reflects the network and processing delays. Relative update is easy to implement and is the common approach used. Many web-pages evoke automatic updates using the Refresh META tag. For instance, including the following line in a web-page automatically refreshes the page every 30 seconds in a relative manner.

```
<meta http-equiv="Refresh" content="30">
```

Relative update is not only easy to implement; in the absence of maliciously synchronized requests, it has an implicit adaptive behavior in response to various load levels and the distribution of initial user arrivals. Through the self-correction mechanism, it is more stable and outperforms the absolute update, as will be shown in section 4.

However, the adaptivity of relative update can be exploited by attackers. Through a process of *herding*, a small number of attackers can gather a significant portion of the normal users to arrive in a relatively small interval, causing temporary overload, even though the average load over the entire update period is low. As a result, some new users may receive highly degraded service and decide to leave the system. Furthermore, herding can condition the user arrival distribution so that subsequent Denial-of-Service attacks can be effective, even with a small number of attackers. The combined herd-burst attack can be executed even when each attacker adheres to the normal application semantics, without excessively consuming resources. Such behavior makes detection very difficult. The only "tool" needed by the attackers is the timing of the request arrivals. Experiments show that herding attacks can be successful even when the total users and attackers load is only a fraction of the server capacity.

The rest of the paper is organized as the following. In Section 2, the related work on DDoS and synchronization problems in the network is presented. In Section 3, the model of periodic update is presented. In Section 4, the behavior of relative update and its advantage over absolute update in high load condition is examined. The herding behavior and related attacks are discussed in Sections 5 and 6. Implementation results on test-bed are presented in Section 7. Finally, a prevention measure to stop such attacks is described in Section 8 and the conclusion is drawn in Section 9.

2 Related Work

One of the most prevalent forms of network attacks is Distributed Denial of Service (DDoS) attack. In DDoS, many compromised hosts send a large amount of network traffic to the victim network elements such that the resources of the elements are exhausted and the performance seen by legitimate users is severely

degraded. A comprehensive overview of common DDoS attacks can be found in [1].

Normal DDoS attacks are very different from the attacks described in this paper. Usually in a DDoS attack, the amount of attack traffic is extremely high which easily overwhelms the victims. Hence, DDoS defense mechanisms always assume that they are operating in an overloaded system. However, the attacks to be presented are effective even when the total traffic from attackers and users is a fraction of the system capacity. In this sense, they are similar to the low rate attacks described in [2] and [3]. In [2], a low rate attack designed to disrupt TCP connections is proposed. By sending a burst of well-timed packets, this attack is able to create packet loss and retransmission timeout for certain TCP flows (in particular, those with small RTTs). The signature for such an attack is the existence of a "square wave". Defense mechanisms like [4] have been proposed to detect such attacks. In [3], the attack proposed degrades the performance by disrupting the feedback mechanism of a control system, with a small amount of attack traffic. Interestingly, the authors also made the observation that the vulnerabilities resulting from adaptation of dynamics are potentially serious. The example used for illustration is a bottleneck queue with Active Queue Management(AQM) employing Random Early Detection (RED). The attack effectiveness is measured in terms of the reduction of quality (RoQ) compared to the original system. The impact of our attacks is similar to [2] and [3] in that using only low rate attack traffic, the victim's service is disrupted; and that the affected users experiencing prolonged delay are driven to leave the system.

The proposed attacks rely on the periodicity and synchronization of update requests. Risks caused by periodicity and synchronization have been explored in domains such as routing updates, NTP and Wireless Sensor Network (WSN) MAC protocols. [5] highlights the problem that periodic routing messages can be synchronized unintentionally, causing significant delay in the routing update. [6] discusses similar problems in ACK compression. It is reported in [7] and [8] that defective NTP configuration can direct a massive amount of synchronized requests at a particular NTP server. As a response, the server can explicitly send a "kiss-of-death" packet, requesting the clients to back off. In [9] and [10], several jamming attacks against representative WSN MAC protocols are presented. In these attacks, the sensor's sleep-listen schedule and the temporal pattern in packet inter-arrivals are exploited to create collisions energy efficiently. In [10], the proposed solutions include the use of link layer encryption to hide the schedule, spread spectrum hardware and TDMA.

3 Periodic Updates

Figure 1 illustrates a periodic update system. At any time, a new user may join the system; and users existing in the system may periodically send requests for updates, or leave the service. The delay of a request is measured as the duration between the initiation of the request and the reception of the reply by the user. In the case of *relative update*, after the request is served, the user waits for T

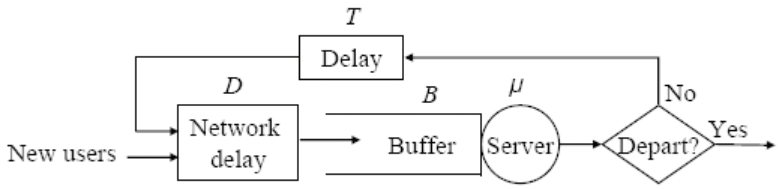

Fig. 1. Periodic updates

seconds before sending the update request. Due to network delay and the service time, the actual interval between two update requests is more than T. In the case of *absolute update*, the time between consecutive update request initiations is always T. An attacker behaves exactly like a user, except for the timing of the requests.

In this paper, two settings are considered:

1. The server's buffer is unlimited and the user requests do not timeout.
2. The server has a finite buffer of size B and a request is dropped when the buffer is full. Timeout occurs when a request is not served within the initial timeout period T_A. Timeout can occur either due to a dropped request or the request waiting for more than T_A seconds in the buffer. After a timeout occurs, the user retransmits his request. In addition, each subsequent timeout period is twice of the previous one. If the user's request is not served after N_T number of timeouts, the user will depart and is considered *lost*.

Below is a summary of the relevant parameters for the model:

- N_A: Number of attackers
- N_U: Number of users
- μ: Mean service time of a request
- T: Update period
- δ: One-way network delay
- B: Buffer size at the server
- N_T: Number of timeouts allowed
- T_A: Initial timeout period
- α: Probability of a user to depart

4 Absolute and Relative Update

Before presenting the low rate attack strategies, we first explore the advantage of relative update over absolute update.

Figure 2 shows the average processing delay vs. the normalized server load, where the normalized server load is computed as $\rho = \mu N_U/T$ and the mean service time $\mu = 50$ ms. For ρ below 0.90, the performances of the absolute and relative updates are very similar and have average processing delays below 0.3s. However, when ρ approaches 1, the average processing delay for absolute update

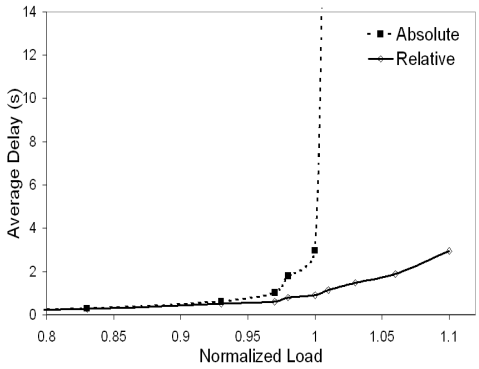

Fig. 2. Comparison of absolute and relative updates

increases rapidly. Yet for relative update, the average delay increases slightly to 0.90s when $\rho = 1$. Even at an extremely high load of $\rho = 1.1$, the average delay is only 2.95 seconds. Such robustness is due to the indirect "increase" of the update period by increasing waiting time in the queue.

While the self correcting behavior of relative update makes the approach more robust, this dynamic behavior, which is absent in the absolute update, can be exploited by attackers.

5 Herding

5.1 Concept of Herding

Herding allows attackers to influence the timing of the users such that the users become part of the attack. The herding action is executed by having a small amount of attackers performing *approximately synchronized* updates. Note that it is not necessary to have perfect synchronization. The idea of herding is illustrated in Figure 3. User arrivals are (initially) distributed over the update period. In each herding round, the attackers delay their update time by a duration of T_{off} in addition to the usual update period T.

To simplify the explanation, μ is assumed to be constant, $\delta = 0$, and the attackers are perfectly synchronized. Let N_A attackers commence herding at time T_h and send N_A simultaneous requests to the server. The buffer size B is assumed to be large enough that there is no request drop. Under such a deterministic scenario, all attacker requests arrive at the server at the same time and for a period of μN_A, only attackers are served. Users arriving during this period are queued behind the attackers. Their request completion times become more clustered. Since the updates are performed in a *relative* manner, the compact completion times entail compact update requests in subsequent rounds. In fact, a period of μN_A is removed from serving users in each herding round.

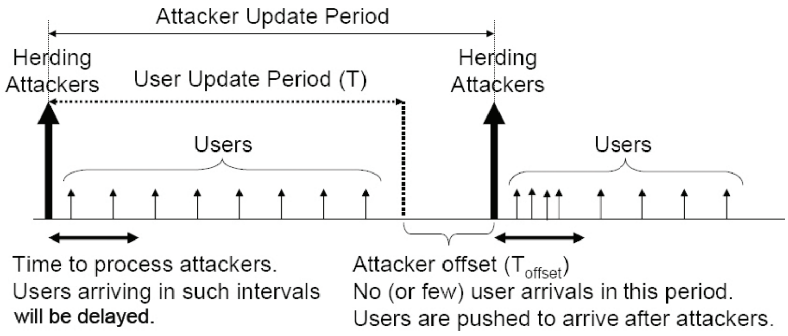

Fig. 3. The herding behavior

By setting $T_{\text{off}} \leq \mu N_A$, the attackers commence the next herding action at time $T_h + T + T_{\text{off}}$. With a constant μ and $\delta = 0$, there will be no user update requests between $T_h + T$ and $T_h + T + \mu N_A$. A herding offset of $T_{\text{off}} = \mu N_A$ per round for the herding scheme in this static scenario is the most efficient. A smaller offset reduces the speed of the herding process whereas a larger offset lets some users be served before the attackers and escape from the cluster of compact request arrivals.

Formally, we say that a user request q is *herded* if there is no delay between the completion time of the previously served request $\widetilde{q}$ and the time the server starts to serve q. Furthermore, the previously served request $\widetilde{q}$ is either (1) from an attacker, or (2) from another herded user.

In a probabilistic environment, the network delay and the service time are not deterministic. In addition, the number of users joining or departing the service varies. Hence, more analysis is required to determine the optimal offset.

5.2 Modelling of Herding Behavior

In this section, we present a model for the herding behavior. Such model is useful in estimating the optimal attack offset T_{off} and monitoring the effectiveness of herding. We first consider the effect of variable network and processing delay, and next handle the case with new and departed users. We make the simplification that the attackers are synchronized. This is a reasonable approximation as the attackers can estimate the network delay they experience. In the simulation, we evaluate the impact of synchronization error on the performance.

The two main components in the model are the *escape probability*, the probability that herded users become no longer herded, and the average number of freshly herded users in a period.

Variable Network and Processing Delay For each herded user, we want to estimate its escape probability which depends on the duration between the

arrival of the request and that of the attacker requests. Suppose the attackers arrive at T_0, they will return at $T_1 = T_0 + T + T_{\text{off}} + \delta$.

Consider the requests in the current period. Let the i-th herded user served after the attackers be u_i and its service time after T_0 be t_i. Hence, for u_i, it will have its service completed at $T_0 + t_i$ and will return at $T_0 + t_i + T + 2\delta$. The user u_i will not be herded in the next update period if it arrives (early) before the attackers, therefore if $T_0 + t_i + T + 2\delta < T_1$. The converse may not be true but the chances that u_i escapes by arriving late is low. Therefore, we can approximate,

$$Pr(u_i \text{ escapes}) = Pr(t_i + \delta < T_{\text{off}}). \tag{1}$$

Let us assume that the processing time is exponentially distributed with mean μ. The distribution of t_i is the Gamma distribution where

$$f_n(t) = \lambda e^{-\lambda t} \frac{(\lambda t)^{(n-1)}}{(n-1)!} \quad \text{with } n = N_A + i, \lambda = \frac{1}{\mu}. \tag{2}$$

Suppose there are k_1 herded users in the current update period, for a specific N_A, T_{off} and a constant δ, using equation 1, the expected number of users escaping in the next period can be written as

$$esp(k_1) = \sum_{j=1}^{k_1} Pr(u_j \text{ escapes}). \tag{3}$$

Note that the escape probability is heavily dependent on T_{off}. It increases with increasing T_{off} since users are more likely to arrive before the attackers but decreases with increasing δ since the reverse is true.

New and Departed Users After his request is served, a user may depart after any request with probability $\alpha > 0$. For simplicity, we consider a model where the number of users in an update period is kept constant. In other words, for every user departure during the current period, a new user will join in the next period, and its arrival time is uniformly distributed over $[T_0 + T, T_0 + 2T]$.

Let $r = (T_{\text{off}}/T)$. Suppose the number of herded users in the current update period is k_1, the expected number of users captured in the next period can be approximated by

$$cap(k_1) = r\alpha k_1 + r(N_U - k_1), \tag{4}$$

where the first term gives the average number of new users who join and are captured immediately; the second term gives the average number of unherded users that are captured in each round. Assuming a small α, the first term increases with i but the second term decreases with i. Overall, as i increases, $cap(k_1)$ decreases.

H_i, the expected number of herded users in the i-th update period can be computed using equations 3 and 4. We calculate H_i iteratively as

$$H_i = H_{i-1} - esp(H_{i-1}) + cap(H_{i-1}). \tag{5}$$

5.3 Simulation Results on Herding

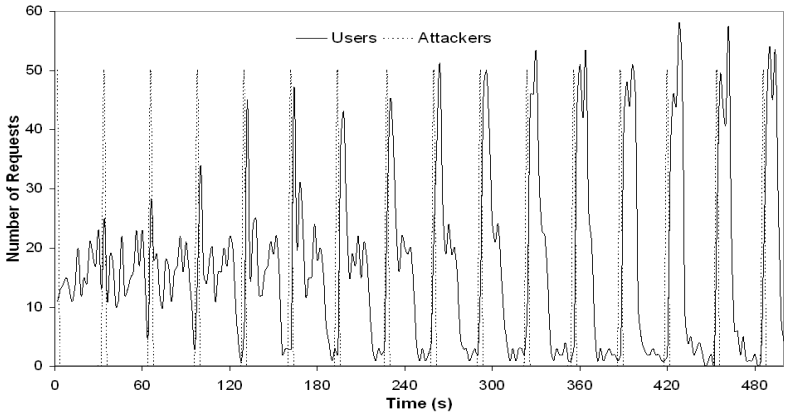

Fig. 4. Request arrivals with herding

In this section, we use simulations to demonstrate the effect of herding. For the experiments reported here and in Section 7, the parameters are set as (unless otherwise specified): the update period T=30s, the number of attackers N_A=50 and the number of users N_U=250. The service time of a request is exponentially distributed with mean μ=50ms. So the normalized server load is $\rho = (250 + 50)0.050/30 \approx 50\%$. The herding offset is chosen as 2.25s, which is slightly less than $\mu N_A = 50 \times 50$ms. In addition, in every update period, 5% of new users join the system and 5% of existing users leave.

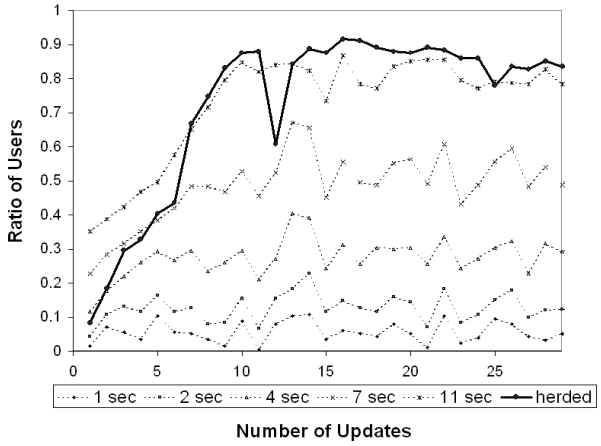

Fig. 5. The effect of herding as ratio of users herded

Figure 4 illustrates the user behavior during herding. As herding progresses, user requests become increasingly clustered after the attacker requests.

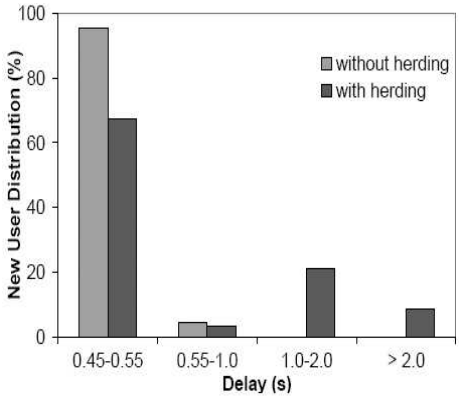

Fig. 6. The effect of herding on new user delay

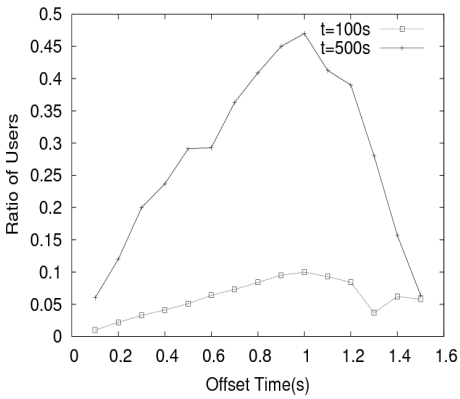

Fig. 7. Impact of T_{off}

Figure 5 presents the progress of herding in each period as the ratio of users clustered. The graph shows 6 lines, each line corresponds to the percentage of users herded or arriving within 1, 2, 4, 7 and 11 seconds after the attackers. Initially, user arrivals are uniformly distributed. The ratio of herded users gradually increases to 90% after 10 rounds and stabilizes. The partial escape of users from herding is caused by variation in the network delay and the server's processing time. Figure 6 compares the delay experienced by new users when herding is present and absent. For the case with herding, the delay experienced by new users is measured after 10 rounds of herding, when the herding effect has stabilized. When herding is absent, the new user delays average at 0.50s. With herding, close to 40% of the new users experience prolonged delay. Among which, 22.1% has delay greater than 1.5s and 8.8% has delay greater than 2s.

The choice of herding offset T_{off} greatly impacts the effectiveness of herding, because a small T_{off} takes much longer to herd the users; while a large T_{off} allows many users to escape herding. Figure 7 shows how the ratio of herded users varies with different T_{off} after 100 and 500 seconds of herding. In this simulation, the number of attackers $N_A = 100$, and the expected service time $\mu = 10$ms ($\mu N_A = 1$s). The result shows that when $T_{\text{off}} = 0.1$s, herding is too slow while the effect of herding decreases dramatically for T_{off} larger than 1.2s. Herding is most effective when T_{off} approximates μN_A, which is between 0.9 to 1.1s in this scenario.

6 Effect of Herding and Attacks

Herding achieved two results. Firstly, by herding most of the users into a much smaller time interval, the average delay of many normal users increases substan-

tially. In particular, the delay experienced by new users who arrive during the herding interval will be excessive. In the context of web server, the new users requesting for web pages are more sensitive to delays. A noticeable delay is sufficient to discourage a new user from browsing further. Recent studies show that a user usually decides whether he satisfies with the web page quality within 50ms [11]. Hence, herding alone is sufficient to turn away a significant number of new users.

Secondly, by making many users arrive in a small time interval, the impact of a burst attack can be magnified. In other words, herding can be used as a means to "condition" the user distribution so that subsequent attacks can be effectively carried out.

In the previous discussion, we assume the buffer size is unlimited. As mentioned in Section 3, limited buffer can lead to request drop, entailing timeouts and retransmissions, which in turn leads to user lost. In the next few subsections, we will consider limited buffer when comparing three attacks: *flood*, *burst* and *herd-burst* attacks.

6.1 Attacks without Herding

DDoS attacks are typified by *flood* attacks. In such attacks, a large amount of attack traffic is generated to overwhelm the server. Success of such attacks is achieved when the combined load from the users and attackers exceeds the server capacity.

In a *burst* attack, attackers are synchronized and the attack packets are sent at the same time to the server. Such attack achieves short term congestion, yet it still requires a large amount of attack traffic for an ongoing congestion.

6.2 Combining Herding and Burst Attack

Intuitively, burst attacks are effective when the aggregated user and attacker request rate is close to or exceeds the system capacity. On the other hand, with herding, short term congestion can be created. This motivates the following herd-burst attack.

The strategy is to alternate herding and burst attack. When herding creates sufficient short term congestion, burst attack can then be used for maximum impact. The attack strategy is present below:

- Perform herding using N_H ($< N_A$) attackers for R_1 rounds
- Repeat
  - Perform burst attack using N_A attackers
  - Perform herding for R_2 rounds

At the start of the attack, R_1 rounds of herding are performed to increase user density over a short period of time which will help in the later attack stages.

The above attack can be stealth, since during herding, *there is no request drop or excessive processing delay*. Hence, unless details on arrival time are captured and analyzed, it is difficult for the system administrator to notice that

the herding process is going on. When the initial "preparation" herding is done, burst attack commences.

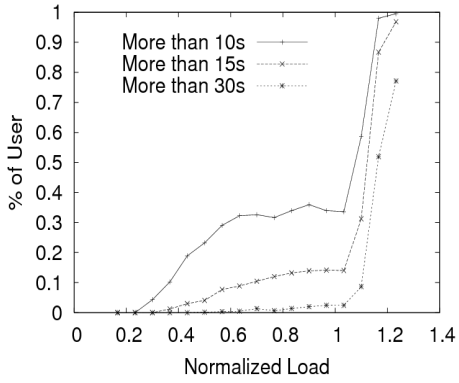

Fig. 8. Delay of user access

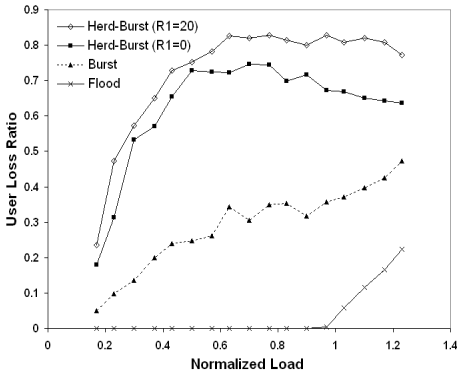

Fig. 9. Comparing user request lost

Figure 8 shows the result of a herd-burst attack with $R_1 = 20$ and $R_2 = 2$. The simulated duration is 1500 seconds, and the parameters are $N_H = 100$, $N_A = 300$, $B = 100$ and $T = 30$s. The μ and δ are exponentially distributed with mean 10ms and 50ms respectively. The initial timeout T_A is uniformly distributed between 1s and 2s. Each subsequent timeout value is twice of its previous one. The user always retransmits in case of timeout. Three lines are shown in Figure 8, indicating the percentage of users experiencing processing delay of more than 10, 15 and 30 seconds at least once. For example, at a normalized load of 60%, 40% of the users experience a processing delay of more than 10s, 8% experience delay of more than 15s and 0.5% experience delay of more than 30s. Depending on the application level timeout specified or user impatience, the number of users who feel unsatisfied with the service and leave the system can range from 0.5% to 40% if the application level timeout is between 10s to 30s.

6.3 Comparison of Herd-Burst, Flood & Burst Attacks

In this section, we compare the performance of the proposed attack to the flooding and the burst attacks. For the herd-burst attack, there are two experiments. The first experiment sets R_1 to 0, meaning there is no pre-herding and the alternating herd-attack starts immediately. In the second experiment, herding is first performed for 20 rounds. The number of users, N_U, is varied from 200 to 3400. Therefore, the normalized load, $\rho = (N_A + N_U)\mu/T$, varies from 0.17 to 1.23. The rest of the parameter values follow those from the previous subsection for all attacks, except $N_T = 1$, that is, one retransmission is allowed, a user will be lost if the retransmission also timeouts. This definition is used in the rest of the simulation in this section.

Figure 9 illustrates that the proposed herd-burst strategy is much more effective. With $R_1 = 0$, $R_2 = 2$, the user lost rate is 72.2% at 60% load. When 20 rounds of pre-herding are performed to cluster users ($R_1 = 20$), the attack efficiency is improved to 82.7% user loss at 60% load.

6.4 Effect of Network Delay and Attacker Synchronization Error

In this section, we study the effect of network delay and synchronization error on the effectiveness of herding.

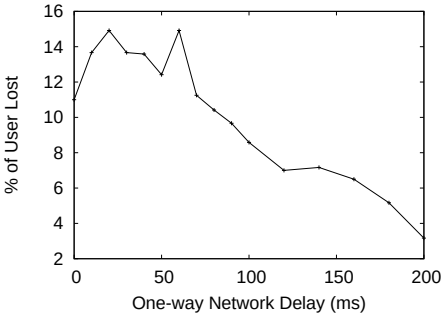

Fig. 10. Impact of network delay

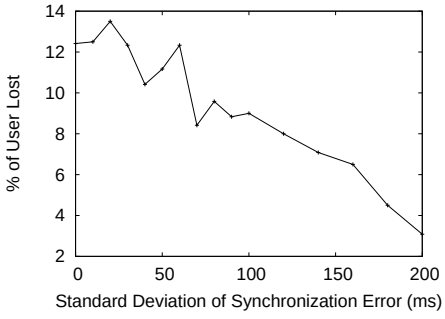

Fig. 11. Impact of attacker synchronization error

Figure 10 shows the impact of increasing the network delay variation. The network delay is exponentially distributed with mean varying from 0ms to 200ms. As expected, as network delay variation increases, the attack efficiency decreases.

In Figure 11, the attacker synchronization error is assumed to have a normal distribution with 0 mean. The standard deviation of the distribution is varied from 0ms to 200ms. The result is similar to the variation in the network delay. The attack efficiency remains high for standard deviation less than 50ms.

6.5 Effect of Buffer Size

In previous experiments, we assume that the buffer size is known and herding can be performed by sending the exact number of attacker requests. However, such values may not be available and needs to be estimated by probing. Figure 12 shows the impact of estimation errors in the server buffer size, with the attacker assuming that $B = 100$. For herding to work correctly, it is important for the attackers to not over-estimate the buffer size as it will result in missing too many users in the herding process. The figure shows that if the buffer is less than 60, the attackers are progressing too quickly and too many users are left behind. The resulting lost rate is 0. The herding process is sufficiently robust such that when the actual buffer size is between 80 to 120, the lost rate remains high,

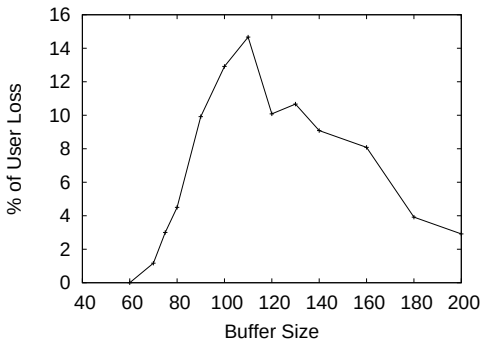

Fig. 12. Impact of buffer size

between 10% to 14%. However, if the buffer size is beyond 120, attack efficiency drops. When the buffer size reaches 200, the loss rate drops to 3%.

7 Results on Test-Bed

In order to validate that herding can indeed be carried out in practice, we repeat the experiments done in Section 5.3 using PlanetLab (http://www.planet-lab.org). 10 nodes from U.S.A, Canada, Spain, Italy and Singapore were used to make the experiments as realistic as possible. Attackers and users are emulated on PlanetLab nodes and each node emulates a total of about 30 to 50 users and attackers. Our server is represented by a Java program that places arrived requests in a First-In-First-Out queue. For each request, the server provides some dummy calculations as service.

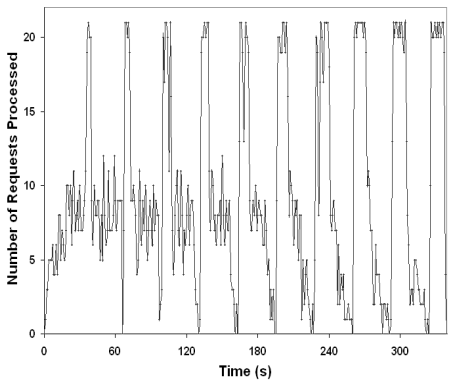

Fig. 13. Number of requests served per second

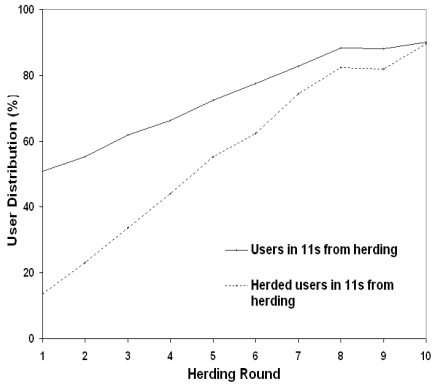

Fig. 14. The effect of herding

In the first set of experiments, we observe the temporal pattern of user requests during herding. Figure 13 illustrates the serving rate at the server as herding progresses. Observe that after each herding round, herded users are pushed forward. Also notice that the service rate in between peaks is non-zero. This is due to new users entering the system periodically. Figure 14 shows the effect of herding by measuring the percentage of users herded and that arrives within 11s from a herding round. Note that after 10 rounds of herding, almost all users within 11s are herded.

Next, we repeat the experiment with different herding offsets and compare the values after 10 herding rounds. Figure 15 shows the effectiveness of the different offsets for herded users and users arriving within 11s. Note that the optimal herding offset is approximately $\mu N_A = 2.5$s which corresponds to the optimal offset analyzed from the model.

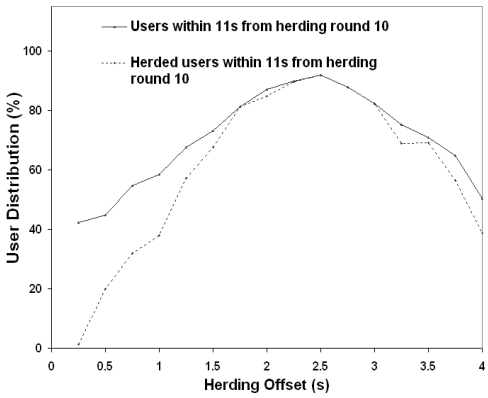

Fig. 15. The effect of herding offset

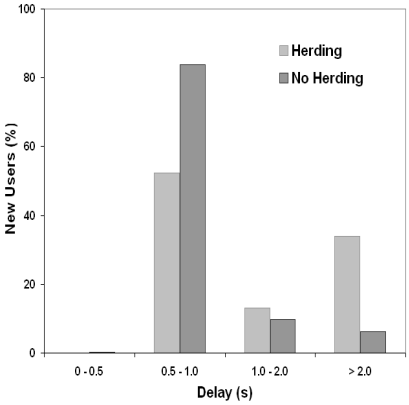

Fig. 16. Delay experienced by new users

Finally, we conduct another set of experiments to illustrate the effect of herding on new users. As before, in every period, 5% of new users join the system and 5% of existing users leave the update cycle. Compared to the existing users, a new user is more sensitive to the delay because he initiates the first request. We first conduct herding with 50 attackers for 10 rounds by which the system behavior has stabilized. Next we measure the delay experienced by new users over 30 update periods. We also consider the case in which no herding is performed. Figure 16 shows that with herding, 40% of new users are likely to experience a delay of more than 2s. Compared to Figure 6, measurements from the test-bed display larger delays. This is because the network nodes employed for test-bed experiments have longer links and larger network delay variation to reach the server.

8 A Prevention Approach

In this section, we present an approach to negate the effect of the proposed herd-burst attack. Though the herd-burst attack is shown effective and robust, it depends on the constant period of updates. In particular, herding relies on the periodicity to work correctly. Therefore, one simple way to prevent herding is to add a small random component with mean 0 to the length of the update interval.

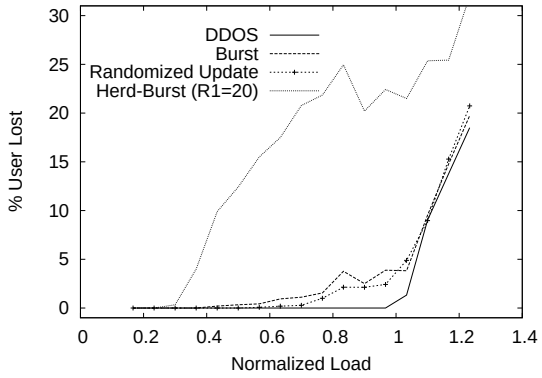

Fig. 17. Effect of adding a randomization component of 3s

In Figure 17, the same parameter setting used to generate Figure 9 is used. The experiment on herd-burst attack without pre-herding is modified. It is executed over an update period uniformly distributed between 27 and 33 seconds. With such randomization added to the update period, the herding process fails and the user lost rate drops slightly lower than the burst attack. This is because with $R_2 = 2$, the herd-burst attack only burst once every three rounds.

9 Conclusion

Periodic updates can be viewed as a feedback queue whereby the served requests are further delayed before rejoining the queue. Due to the growing popularity of the use of automatic refreshment of web services, it is interesting to investigate such model. In this paper, we first study the advantages of relative update verse absolute update. We show that relative update gives better performance in term of average service delay. More interestingly, we found that relative updates indirectly provides a self-correcting mechanism, and can be stable even when the system is overloaded. Although relative updates provides good performance, potentially it can be manipulated by a small number of attackers. We give a herding strategy whereby a small number of attackers can herd a significant portion of the users to arrive in a small time interval. Such herding can be used as a way to

"condition" the request arrival distribution so that subsequent burst attacks can be effectively carried out. It can also be employed to discourage new users from joining the system. The herding can be performed by adhering to the normal application semantics, and thus it is difficult to identify individual attackers. Fortunately, herding can be easily prevented by introducing randomness to the length of the update interval.

References

1. Mirkovic, J.: D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks. PhD thesis, UCLA (2003)
2. Kuzmanovic, A., Knightly, E.: Low-Rate TCP-Targeted Denial of Service Attacks. Proc. ACM SIGCOMM (2003) 75–86
3. Guirguis, M., Bestavros, A., Matta, I.: Explaining the Transients of Adaptation for RoQ Attacks on Internet Resources. Proc. Int. Conf. Network Protocols (2004) 184–195
4. Sun, H., Lui, John C.S, Yau, David K.Y.: Defending against Low-Rate TCP Attacks: Dynamic Detection and Protection. Proc. Int. Conf. Network Protocols (2004) 196–205
5. Floyd, S., Jacobson, V.: The Synchronization of Periodic Routing Messages. IEEE/ACM Trans. Networking, Vol. 2. (1994) 122–136
6. Mogul, J.: Observing TCP Dynamics in Real Networks. Proc. ACM SIGCOMM (1992) 305–317
7. Plonka, D.: Flawed Routers Flood University of Wisconsin Internet Time Server Netgear Cooperating with University on a Resolution. http://www.cs.wisc.edu/∼plonka/netgear-sntp (2003)
8. Mills, David L.: Survivable, Real Time Network Services. DARPA Report (2001)
9. Law, Y., Hartel, P., Hartog, J. den, Havinga, P.: Link-Layer Jamming Attacks on S-MAC. Proc. IEEE 2nd European Workshop on Wireless Sensor Networks (EWSN) (2005) 217–225
10. Law, Y., van Hoesel, L., Doumen, J., Hartel, P., Havinga, P.: Energy-Efficient Link-Layer Jamming Attacks against Wireless Sensor Network MAC Protocols. Proc. 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SANS) (2005) 76–88
11. Lindgaard, G., Dudek, C., Fernandes, G., Brown J.: Attention web designers: you have 50 milliseconds to make a good first impression. J. Behaviour & Information Technology, Vol. 25 (2005) 115–126