Regulating User Arrivals at a Mobile IP Home Agent

Maulik Desai‡ **Thyaga Nandagopal**†
‡ Columbia University, New York, USA. † Bell-Labs, Alcatel-Lucent, NJ, USA.

Abstract—**With the increasing levels of data usage in mobile cellular networks, Mobile IP and its variants serve as the de facto standard for mobility management. At high user loads, the Mobile IP Home Agent is unable to support registration requests within the required delay limit, leading to registration failures during peak user periods. The typical way of addressing this problem currently is to over-provision the Home Agent or to add multiple Home Agents with Load-Balancing in order to handle peak loads. In this paper, we propose an algorithm that regulates user arrivals by manipulating their registration lifetimes, thereby smoothing the arrival process at the Home Agent. Our proposed algorithm controls the variance in server loads very effectively with negligible computational and storage overhead. It also does not penalize users to register more often in order to satisfy the requirement of uniform load at the HA. Simulations with various user arrival and lifetime distributions show that our scheme is highly effective at the regulation of user registration load at the Home Agent.**

I. Introduction

Mobile devices are increasingly becoming ubiquitous and the preferred way for accessing the Internet, either by means of a 3G cellular or a wireless LAN connection. Data throughputs of up to 40Mbps are already realizable in a cellular data network deploying HSPA+, with greater speeds promised by LTE [1]. Coupled with the wide-spread availability of WLAN hot-spots, this has increased the demand for seamless mobility services for users, including support for vertical hand-offs across network types [2].

Seamless mobility services are provided by the Mobile IP protocol [3] or its numerous variants standardized by the IETF or the 3GPP working group. The basic premise is as follows. A Home Agent (HA) serves as the anchor node that tracks the network connection point (location) of a user as the user moves. Note that the location can be in space or it could be simply the types of network that a user is connected to, for e.g., a simultaneous WLAN and 3G connection. Periodically, or whenever there is a change in the user location, the user registers with the HA informing the user's present location. The HA performs a variety of services as part of this registration, namely request verification, user authentication, authorization, accounting, address assignment (for new users), data path forwarding on behalf of the user, etc. Some of these functions could be off-loaded to an external AAA server [7] as well, with the Home Agent serving a re-direction agent for these functions.

The Home Agent is the critical part of the system since it is on the critical path of both signaling and data for mobile users. As the user population grows, the load on the Home Agent also grows correspondingly. Typical responses to increased load are: (a) increase the capacity of the Home Agent [4] to handle larger loads, (b) increase the number of servers and use load-balancing in a transparent manner [5], and (c) deploy Home Agents at multiple points in the network and use any-cast mechanisms to handle mobility functions on behalf of users[6].

All of the above responses are valid solutions to the problem, but they also suffer from one significant drawback. They require new hardware, which increases the capex and opex of the network operators, in addition to management complexity. This motivates us to take a look at the basic problem of increased user load to see if we could avoid deploying new hardware as much as we can.

The HA performs multiple functions, all of them directly proportional to the number of users registering with the HA at any given time. Therefore, the aggregate load on the HA is dependent on the *user registration load*, i.e., the number of users attempting to register with the HA. This load has two components: (a) *average load*, and (b) *variance in load*. If the average load is very high, then the only ways to handle it are to either eliminate/divert users or to add hardware to off-set the load. For the latter component, when the variance in load causes the load to be momentarily high enough to over-load the HA server, then we could either queue the user registration request, thus delaying a response, or off-set the temporary spike to a different server. The former leads to increased congestion at the Home Agent (as will be explained later), while the adding new hardware is less desirable as well.

In this paper, we address the problem of *reducing the variance in user-registration load at a Home Agent*, thereby improving the hardware utilization as well as reducing capex/opex costs for the operator. Our approach is to eliminate the key reason behind the variance in server load, namely assigning improper lifetimes to users that cause them to show up at the inopportune moment (when the server faces high loads). To the best of our knowledge, this is the first paper that addresses this problem at a user-registration server, such as a Mobile IP Home Agent. The key features of our proposed algorithm are as follows.

1) It is fully compatible with current Mobile IP and other seamless mobility standards.
2) We cut down the variance in server loads by a factor of three or more.
3) The algorithm does not cause noticeable additional user registration load at the Home Agent.
4) The algorithm runs in $O(1)$ time and has constant memory overhead.

The organization of the paper is as follows: In Section II, we describe the system model and some related work in this space. In Section III, we discuss some baseline methods to address this problem. In Section IV, we describe the CDF algorithm motivated by mapping of probability distributions. We then validate and compare the performance of our proposed approach via simulations in Section V and conclude in Section VI.

II. System Model and Related Work

We consider a set of mobile users who are registered with a Home Agent. The communication between the users and the Home Agent is based on the Mobile IP protocol [3]. For an overview of the Mobile IP protocol and its applications, see [8] and the references therein.

In Mobile IP, users register with a HA for a specified lifetime (L_i). This lifetime cannot exceed 65535 seconds [3]. The registration request (RRQ) from the user contains a requested lifetime from the user ($L_{i,U}$), and the HA can be configured to assign a maximum lifetime for that user as well, based on policy ($L_{i,S}$). Thus, the actual lifetime assigned to the user for a given RRQ is given by

$$L \leq L_{i,max} = \min(L_{i,S}, L_{i,U}, 65535) \qquad (1)$$

We call $L_{i,max}$ as the *user epoch* for user i.

The user registration is valid until the expiry of this lifetime. As the expiry time approaches, the user re-registers with the HA for an additional period as in the original RRQ. This process continues until the user leaves or de-registers voluntarily. In addition, whenever the user changes location to a different network location, it will re-register itself to indicate a hand-off.

The one measure of control the HA has is the ability to control when a user returns to re-register. It can do this by carefully assigning the lifetime (LT) for the user in response to the RRQ, thereby controlling the re-registration load at a future time. This is an *online problem*, since future arrivals of new users and hand-offs are not known to the HA in apriori.

A related problem is that of multiple-server load re-balancing [9]. However, unlike the problem in [9], we cannot preempt an existing user or move a registered user to another time-slot until the user contacts the server near its expiration time. This is because the HA can send a response only when the user contacts it with a RRQ. It cannot signal the user directly unless it wants to revoke its registration.

The other type of problem that is related is load balancing in P2P systems [10], where objects are re-distributed among peers to distribute the load on the peers. Unlike this problem, we consider only a single server and users cannot be shifted anywhere. Using multiple servers as in [5] is a complementary problem, since we can deploy our solution on each of these servers. In fact, using our solution will reduce the need for multiple servers since we will be able to utilize the resources of each server to the fullest all the time.

Predicting the number of hand-offs and which users are likely to hand-off [11], [12] is similar to predicting the number of new users in the system and is normally based on some form of profiling. If we have this information, we can use this, but we cannot control the occurrence of hand-offs or arrival of new users. Therefore, handoffs are similar to new users in terms of the user-registration load on the HA and are part of an orthogonal problem[1] to the problem under consideration.

The *user registration re-distribution problem* can be formally stated as follows.

Statement 2.1: Let the maximum lifetime that can be assigned by a HA for user i be denoted by $L_{i,S}$, and the maximum lifetime requested by user i be given by $L_{i,U}$. Given a set of N users arriving over some period T_{arr}, and departing over period T_{dep}, determine the online assignment of lifetimes to users arriving at any time slot t such that the variance in the user registration load of the HA is minimized over $\{t, t+T\}$ for any $T > 0$, assuming no changes in the user set in the future, where $T_{arr}, T_{dep} \leq t$.

[1]In a typical cellular network provider, the number of hand-offs is typically around 20-30% of the total registrations.

A. Design Goals

Any solution to the problem above should satisfy three requirements.

1) *Low load variance*: During any given period where there are no changes in the user set, the variance in the number of users registering at the HA at any point in the period should be very small.
2) *Low resource consumption*: Any scheme to reduce the variance should not consume additional resources at the HA, i.e., the run-time as well as storage requirements should be small for such a scheme.
3) *Negligible overhead for users*: Any solution to this problem should not cause undue burden on the users by asking them to register more times than desired. This adds signaling burden on the wireless network, which also increases the total number of registration requests that needs to be processed by the HA.

Recall that we are interested in modifying the behavior of the existing set of users in the system. If we have a prediction algorithm that can determine the number of new users/hand-offs that are about to occur in the future, then this number can be taken into consideration by any solution. However, the prediction algorithm is not the focus of this paper.

III. Basic Schemes

We first consider some vanilla approaches to the user registration re-distribution problem.

NoBalance: This is the default do-nothing algorithm. The HA assigns lifetime to user i based on

$$L_i = L_{i,max} \tag{2}$$

where $L_{i,max}$ is the user epoch given by Equation 1.

RandomBalance: A simple heuristic is to assign lifetimes at random in the hope that the load will balance out in the long run. In other words, the lifetime for user i is given by

$$L_i = \mathrm{random}(1, L_{i,max}) \tag{3}$$

This algorithm could be hoped to reduce the variance in registration load, and is very simple to implement, and does not require any additional memory. However, it will lead to increased registrations at the HA, since the lifetimes can be arbitrarily small.

Hole: When a user i arrives at time t, we consider the average expected load during the interval $(t, t+L_{i,max})$ based on the history of registered users up to time t. This algorithm starts down from $(t + L_{i,max})$ and finds the first slot where the load is below the average.

This algorithm could be hoped to cut down the variance since it attempts to fill the ‘holes’ in the load vector compared to the average load over the user epoch. Since it starts down from the maximum possible lifetime that can be assigned to the user, it should result in a small number of additional registrations at the HA.

LeastLoad: This heuristic finds the least loaded slot in the interval $(t, t+L_{i,max})$, say, t_l, and assigns the lifetime to user i as $L_i = t_l - t$.

This algorithm should be the best in terms of reducing the variance in load, though its run-time and storage requirements are fairly high due to the need to identify the least loaded slot for each user registration. If different users have non-identical user epochs, then identifying the least loaded slot has to be done individually and cannot be optimized across users in the worst case.

As is evident, the vanilla heuristics described above do not satisfy all of the design requirements outlined in Section II-A. Therefore, we aim to design an algorithm that can accomplish all of the design requirements.

IV. The CDF Algorithm

If we think of the user registration load as a random variable X, we want its pdf to be a Gaussian with very low variance over any time window for the set of existing users. However, since the actual distribution might is very different, we want to change the distribution, if we can, to a near-Gaussian distribution to achieve our desired goal. If the distribution function is known in closed form, then we could apply the well-known *Probability Integral Transformation* as applied to discrete random variables [13]. However, the pdf of the user registration load is not in closed form, and we cannot apply these techniques to distribute the user load.

Ideally, we want to smooth out all the load spikes that occur at any time. We accomplish this using a novel heuristic that essentially disperses users who arrive at the same time. This dispersion is not random in nature, but takes into account the weight of the load spike in comparison with the total load. The CDF algorithm is shown in Algorithm 1.

The CDF algorithm works by keeping track of the total number of users expected over a given time window of interest, T. This window could be an hour or a few hours. The performance is fairly robust to the choice of T as long as it is not too small (few minutes) or too large (tens of hours).

In a given time slot t, if a very large number of users arrive at the HA, then we consider the users in order and disperse them using a scatter function (Lines 5 – 7). Once we determine a slot for the user, we search around a constant window R of

Algorithm 1 CDF Algorithm for assigning lifetimes

```
1. INPUT: time slot $t$, numUsers[i], $i = 0, 1, 2, \ldots, t, \ldots$,
          time window T, load window R
2. count = 0
3. while users in queue in slot $t$
4.     Select next user $k$ from queue
5.     count = count + 1
6.     if $k$ is a renewing user
7.         $L_{tmp} = L_{k,max}\left(1 - \frac{\text{count}}{\sum_{i=t}^{t+T} \text{numUsers}[i]}\right)$
8.     else  ($k$ is a new/hand-off user)
9.         $L_{tmp} = L_{k,max}\left(1 - \frac{\text{count}}{1+\sum_{i=t}^{t+T} \text{numUsers}[i]}\right)$
10.    $S_{min} = t + L_{tmp} - R$
11.    $S_{max} = t + \min(L_{tmp} + R, L_{k,max})$
12.    $j = \arg\min_{s \in \{S_{min}, S_{max}\}} \text{numUsers}[s]$
13.    $L_k = j - t$
14.    $\text{numUsers}[j]$ ++
15. end while
```

this slot to determine the least loaded slot (Lines 10 – 12), and give this as the lifetime for the user (Line 13). We keep track of the expected number of users who are supposed to show up in a time slot in the numUsers variable and update it whenever a registration request is granted (Line 14). Since new users are not accounted for by this variable, we modify the scatter function slightly when we see new users (Line 8 – 9).

When a user leaves the system without notifying the HA, then the HA cannot do anything since it does not know until the user's registration expiry time when the expected renewal request does not arrive. A hand-off at time t therefore can be treated as a combination of an existing user not showing up at the expiry time and as a new user registering at time t.

The run-time of this algorithm is $O(1)$, since there is only a constant overhead to searching up to $2R$ time-slots for each user.

V. Performance Validation and Comparison

In this section, we look at the performance of the CDF algorithm when compared to the algorithms discussed in Section III, and see how well the CDF algorithm is able to meet the design goals outlined in Section II-A.

We consider up to 10 million users arriving over a period of 1-4 hours. This is the upper bound of the capacity of some of the leading Home Agents in the market [4]. The CDF algorithm parameters T and R are set to 1 hour and 10 seconds respectively, unless specified otherwise. We evaluate the performance over uniform, Poisson and impulse user arrivals, and consider exponential, Gaussian, log-normal, and impulse distributions for user-requested lifetimes[2]. The results are averaged over multiple runs, unless specified otherwise.

In our evaluation, each time slot is one second long. The performance metrics of interest for all these algorithms are:

- Standard deviation of the user registration load for time window T, immediately after all the users have arrived at least once.
- Overhead of the re-distribution algorithm in terms of additional run-time per time slot, Ψ.
- Percentage of additional registrations, Φ, caused as a result of the re-distribution algorithm.

We first look at the effectiveness of the proposed algorithms. We consider Poisson arrivals with the user-requested lifetimes ($L_{k,max}$) exponentially distributed with a mean of 2 hours. The arrival pattern is shown in Figure V, where the y-axis indicates the number of new arrivals per second at the HA. The results are shown in Table I.

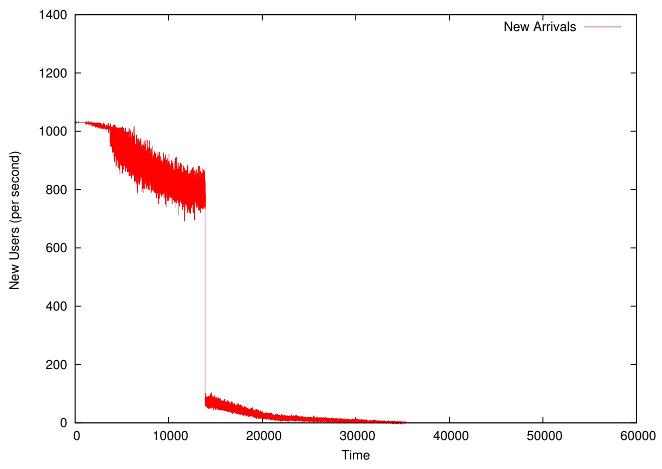

Fig 1. Poisson Arrivals with Exponential Lifetimes

As we can see from the table, a naive randomization heuristic as in RandomBalance fails in all the criteria under consideration. The Hole and LeastHole heuristics reduce the variance significantly, and as can be evidenced by the ratio of the maximum load to the mean load. One can also see this from Figure 2 where we show the number of registered

[2]In this section, any reference to lifetime implies a user-requested lifetime, and is considered to be the same as $L_{k,max}$ in Algorithm 1

Algorithm	Standard Deviation σ	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Extra Run-time per sec Ψ	Extra Reg Φ
NoBalance	40.09	0.38	1.124	0 ms	0
RandomBalance	42.93	0.025	1.088	0.16 ms	56.8 %
Hole	10.25	0.009	1.007	0.84 ms	4 %
LeastHole	10.19	0.009	1.007	1.26 ms	4 %
CDF	23.2	0.022	1.046	0.04 ms	0.3 %

TABLE I
PERFORMANCE: POISSON ARRIVALS AND EXPONENTIAL LIFETIME DISTRIBUTIONS

users per second versus time for a duration of 1 hour after nearly all new users have arrived. Compared to the sample for the same time period using the NoBalance scheme shown in Figure V, the LeastHole algorithm is almost perfect. However, the main issue with Hole and the LeastHole heuristics is that the number of additional registrations is significantly higher, in this example, 4 % more than when compared to the default NoBalance scheme. In addition, the Hole and LeastHole heuristics take much longer to run as well.

The CDF algorithm on the other hand, provides a reasonable trade-off among all of these criteria. It adds negligible amount of additional registration load, and runs the fastest among all the re-distribution heuristics. It's variance is higher than that of the LeastHole algorithm, however, comparing Figures V and 3, we see that the CDF algorithm does a very good job of reducing the variance by a factor of almost 4.

Since the RandomBalance heuristic performs the worst in all categories, we no longer discuss it in the rest of the evaluation.

A. *Arrival and Lifetime Distributions*

We now look at the performance of CDF algorithm for various user lifetime distributions and compare them against the Hole and LeastHole heuristics.

Tables II and III show the results of the performance evaluation for CDF under Poisson user arrivals, where the user lifetimes follow an Gaussian distribution and log-normal distribution respectively. The Gaussian lifetime distribution has a mean of 7200 seconds and a standard deviation of 60 seconds. The parameters of the log-normal distribution are also the same.

The performance under these two distributions is different for all of the algorithms. The Hole and LeastHole heuristics take much longer to run and generate significantly greater number of additional registrations in the Gaussian lifetime distribution, while the CDF has higher variance but performs consistently in terms of run-time and user registration load. Note that, when compared to the NoBalance scheme, the CDF algorithm still does very well, reducing variance by significant fraction. The log-normal lifetime distribution results in good performance for all of the algorithms.

Algorithm	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance	0.597	3.09	0 ms	0 %
Hole	0	1.00	6.05 ms	1.9 %
LeastHole	0	1.00	9.14 ms	10.9 %
CDF	0.132	1.790	0.05 ms	0.2 %

TABLE II
PERFORMANCE: POISSON ARRIVALS, GAUSSIAN LIFETIMES

Algorithm	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance	0.087	1.83	0 ms	0 %
Hole	0.014	1.04	0.78 ms	3.6 %
LeastHole	0.014	1.03	1.08 ms	3.6 %
CDF	0.023	1.043	0.12 ms	0.02 %

TABLE III
PERFORMANCE: POISSON ARRIVALS, LOG-NORMAL LIFETIMES

We also look at another type of setting, where there are only a discrete number of lifetimes requested by the users. This typically can happen when there is a policy-based setting for all devices, such as those belonging to an enterprise provider. We allow 70% of users to request a maximum lifetime of 1 hour, while the other users request a maximum lifetime of 12 hours. All users arrive exponentially at a mean rate of 1000 users/second. The results for this experiment are shown in Table IV. Again, the CDF algorithm gives consistent performance across all the parameters, reducing variance by more than a factor of 4.

One might wonder whether the Poisson arrival patterns have any specific impact on the CDF algorithm. To verify this, we use a uniform arrival pattern for all users, where new users arrive at constant load but with exponentially distributed lifetimes. The results are shown in Table V demonstrating that the CDF algorithm performs consistently well as before.

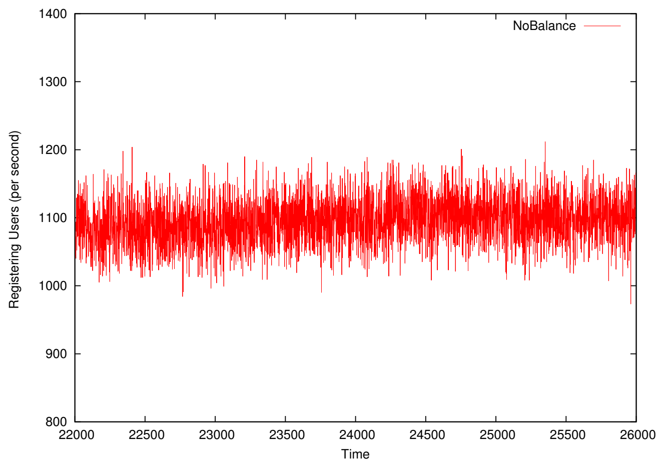

Fig 2. NoBalance: User Distribution with Time

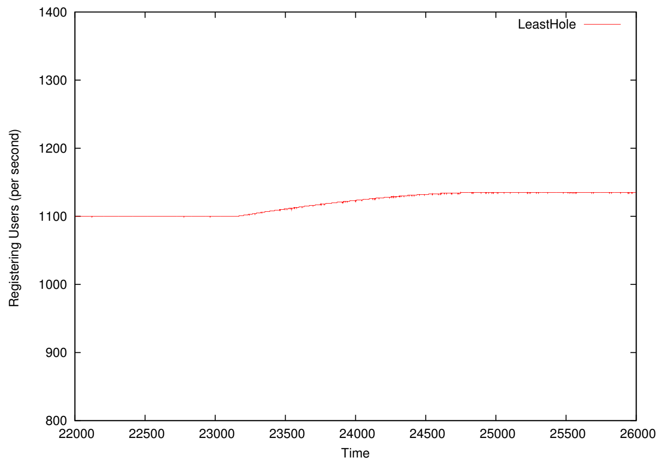

Fig 3. LeastHole: User Distribution with Time

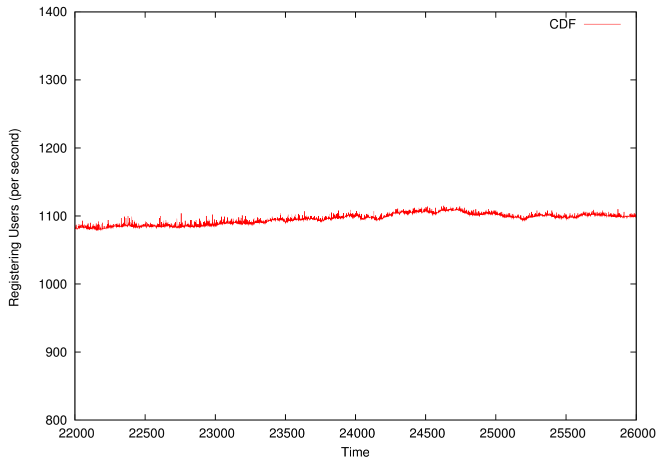

Fig 4. CDF: User Distribution with Time

Algorithm	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance	0.397	1.88	0 ms	0 %
Hole	0	1.00	0.92 ms	3.1 %
LeastHole	0	1.00	1.20 ms	3.8 %
CDF	0.167	1.192	0.02 ms	0.8 %

TABLE IV
PERFORMANCE: POISSON ARRIVALS, BI-VARIATE LIFETIMES

Algorithm	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance	0.085	1.55	0 ms	0 %
Hole	0.010	1.01	0.95 ms	4.0 %
LeastHole	0.01	1.01	1.32 ms	4.1 %
CDF	0.021	1.049	0.17 ms	0.03 %

TABLE V
PERFORMANCE: UNIFORM ARRIVALS, EXPONENTIAL LIFETIMES

B. *Bursty Arrivals*

The worst case behavior for any user registration load re-distribution algorithm occurs when new users arrive in bursts. We let nearly $600,000$ users arrive in a bunch every 15 minutes, for 4 hours. The lifetimes of these users are exponentially distributed. We show the number of registering users per second at the HA versus time in Figures 5 – 8 for the NoBalance, Hole, LeastHole and CDF algorithm respectively. As is clearly evident, all three heuristics achieve significant reduction in variance of user registration load, with the LeastHole heuristic acting in an ideal manner. The CDF algorithm ensures a minimum load on the HA that is very close to the average load, and thus ensures that the hardware is utilized well most of the time.

Table VI provides the extra run-time and extra registration load caused by all these algorithms. The results are consistent with our observations so far, in that, CDF results in negligible added run-time and user registration load on the HA, unlike the Hole and LeastHole algorithms.

Algorithm	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance	0.533	2.51	0 ms	0 %
Hole	0.017	1.16	2.25 ms	3.2 %
LeastHole	0.004	1.002	1.22 ms	4.4 %
CDF	0.030	1.198	0.005 ms	0.5 %

TABLE VI
PERFORMANCE: UNIFORM ARRIVALS, EXPONENTIAL LIFETIMES

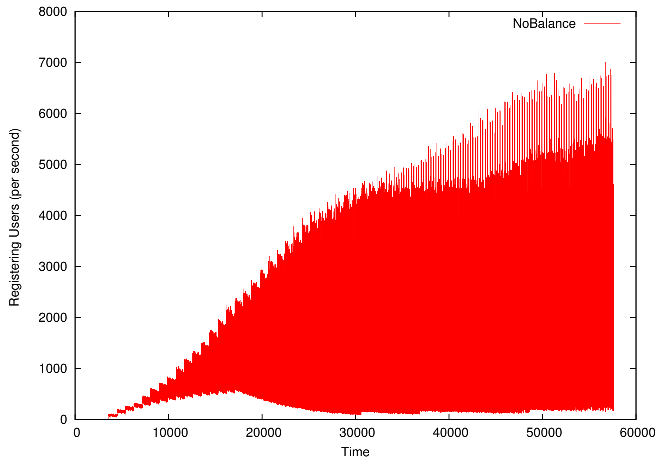

Fig 5. NoBalance: Bursty Arrivals

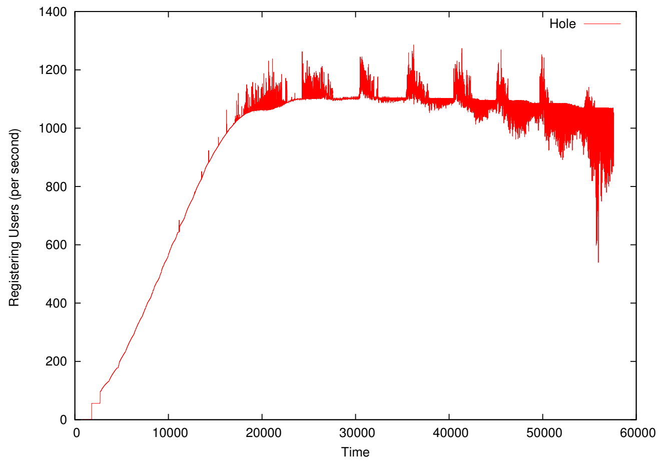

Fig 6. Hole: Bursty Arrivals

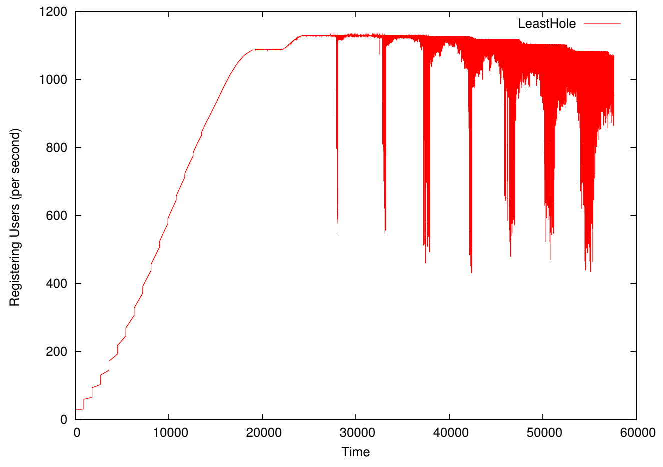

Fig 7. LeastHole: Bursty Arrivals

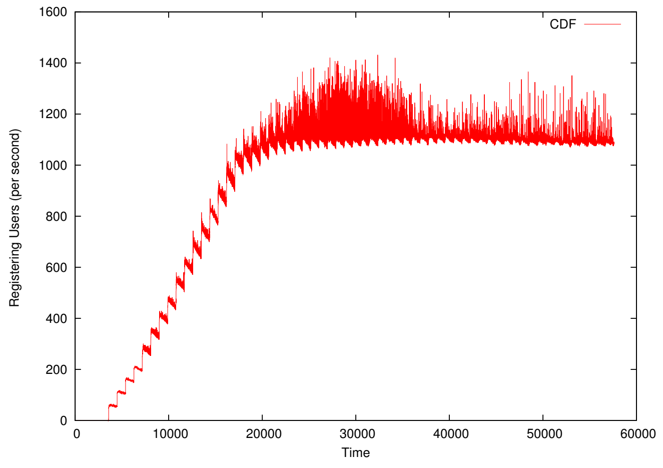

Fig 8. CDF: Bursty Arrivals

C. *Scaling behavior*

We want to evaluate how well these algorithms scale to the number of users. This is critical due to the increasing demand for seamless mobility provisioning both in the cellular space as well as the WLAN space.

We evaluate the CDF algorithm for variable number of users, and find that the ratio of the standard deviation to the mean is a constant for the CDF (as well as Hole and LeastHole) algorithm. This holds true for the ratio of the maximum load to average load in any time window. In addition, the increased user registration load (as a ratio compared to the NoBalance load) is also a constant for all three heuristics.

The main difference is in the additional run-time per time slot as a result of the algorithm. Figure 9 shows the variation of this metric with the number of users. We can see that the CDF algorithm has negligible overhead even as the number of users increases by a factor of 10. However, the Hole and LeastHole heuristics have increasing run-time overheads that are, while linear, increasing at a faster pace than the default NoBalance scheme or the CDF scheme.

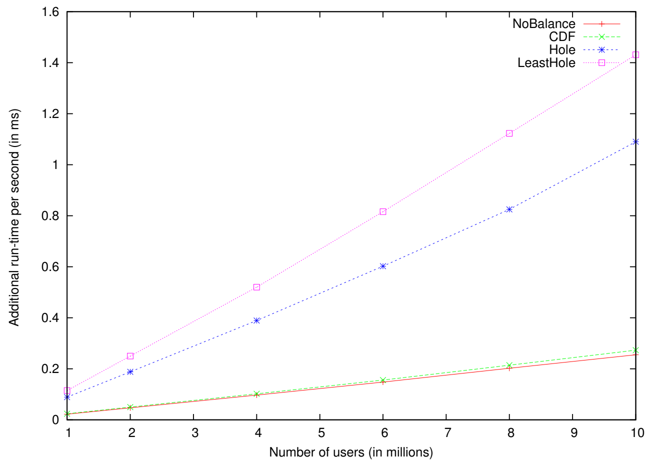

Fig 9. Scaling with Users

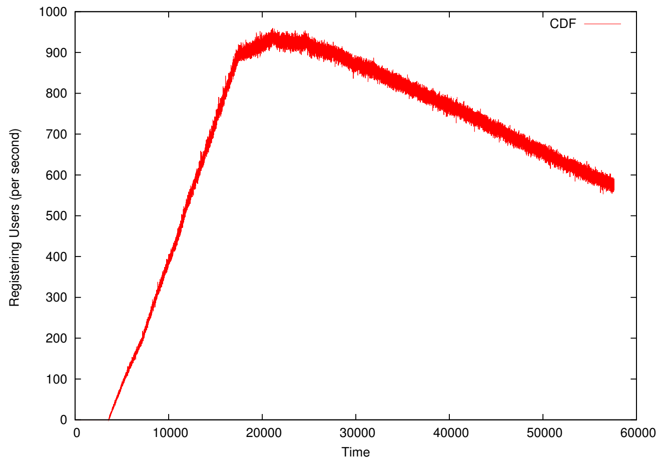

Fig 10. CDF: Arrivals and Departures

D. *Robustness of CDF*

There are two tuning parameters to the CDF algorithm: the time-window T and the load-window R. The time-window, T, for computing the average load is common to the three algorithms being discussed, however, the load-window R is used only for the CDF algorithm. We vary the value of R and show the corresponding results in Table VII. Increasing R does not improve our performance by much, but it also leads to a huge increase in run-time. Therefore, it is desirable to have a value of $R = 10$.

Algorithm	R	$\frac{\sigma}{\text{Mean}}$	$\frac{\text{Max}}{\text{Mean}}$	Run-time Ψ	Extra Reg Φ
NoBalance		0.043	1.12	0 ms	0 %
CDF	10	0.022	1.046	0.05 ms	0.13 %
CDF	100	0.021	1.045	0.17 ms	0.18 %
CDF	1000	0.019	1.039	1.40 ms	0.5 %

TABLE VII
CHANGE IN LOAD WINDOW FOR CDF: POISSON ARRIVALS, EXPONENTIAL LIFETIMES

CDF is also robust to the choice of T. None of the performance metrics varies for CDF by noticeable margins for T in the range of 30 minutes to 10 hours. On the other hand, the Hole and LeastHole heuristics end up consuming a lot of CPU cycles as T increases, since they have to check for the load across the entire time window in the worst case. This also results in a large number of additional registrations. In the average case, the run-time scales linearly with T for Hole and LeastHole.

E. *Arrivals and Departures*

We have so far considered only users arriving at the HA. We now show how the re-distribution works when users start leaving the HA as well. We allow users to stay registered for a variable time (uniformly chosen between 1 and 20 hours), and see how the load changes for the CDF algorithm. We show the results in Figure 10 for Poisson arrival process and exponentially distributed lifetimes. As we can see, the system adjusts to the departing users very well, without any dips or spikes in the load. We have tested this for different departure processes, and the results are pretty much the same.

To summarize the evaluation, the CDF algorithm offers very good, but not perfect, performance in reducing the variance of user registration load at the Home Agent. It does not require any additional resources at the CPU and it also does not cause users to register more than they have to. It should also be noted again that the run-time and storage complexity for the CDF algorithm is $O(1)$ and does not vary with the number of users.

VI. CONCLUSIONS

Mobility is becoming an increasing part of networked lifestyle, and seamless mobility requires users to be anchored at all times at a Home Agent or its equivalent device. In addition to scaling the number of HAs based on demand, it is imperative that we make better use of the deployed HAs by ensuring that the user arrival process is a smooth one. Regardless of how users first enter the system, we present the CDF algorithm that ensures that the load is nearly constant at

HA. The proposed algorithm, if deployed, will reduce capital and operational expenditures at network providers by reducing the need for additional server capacity in order to meet the vagaries of user arrival patterns. To the best of our knowledge, this is the first such attempt to address such a problem, and we demonstrate the efficacy of our solution through extensive simulations.

References

[1] GSM Consortium, “High Speed Packet Access (HSPA)”, *http://hspa.gsmworld.com*, 2009.
[2] A.Calvagna, and G.D. Modica, “A User-centric Analysis of Vertical Handovers”, *ACM WMASH*, 2004.
[3] C. Perkins, Ed. “IP Mobility Support for IPv4”, *IETF RFC 3344*, August 2002.
[4] Starent Networks, “ST40 Multimedia Core Platform”, *http://www.starentnetworks.com/en/st40.php*, 2009.
[5] Cisco Networks, “Cisco Home Agent Redundancy and Load Balancing”, *Cisco IOS Mobile IP White Paper*, 2005.
[6] R. Wakikawa, G. Valadon, and J. Murai, “Migrating home agents towards internet-scale mobility deployments”, *ACM CoNEXT*, 2006.
[7] The FreeRadius Server Project, “ freeRadius”, *http://freeradius.org*.
[8] Wikipedia, “Mobile IP”, *http://en.wikipedia.org/wiki/Mobile_IP*, 2009.
[9] G. Aggarwal, R. Motwani and A. Zhu, “The Load Rebalancing Problem”, *ACM SPAA*, 2003.
[10] B. Godfrey et. al., “Load Balancing in Dynamic Structured P2P Systems”, *IEEE Infocom*, 2004.
[11] R. Hsieh, et.al., “S-MIP: A Seamless Handoff Architecture for Mobile IP”, *IEEE Infocom*, 2003.
[12] S. Sharma, I. Baek, T. Chiueh, “OmniCon: a Mobile IP-based vertical handoff system for wireless LAN and GPRS links”, *Software: Practice and Experience*, Wiley, 37(7), Nov. 2006.
[13] F.N. David, “The transformation of discrete variables”, *Annals of Human Genetics*, 19 (3), Sep 2007.