Nested Buddy System: A New Block Address Allocation Scheme for ISPs and IaaS Providers

Michael B. Crouse, H.T. Kung
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
Email: mcrouse@seas.harvard.edu, kung@harvard.edu

***Abstract*—We propose a novel block address allocation method, called the nested buddy system, which can make use of wasted areas in the classical buddy system due to internal fragmentation. While achieving high utilization of address space, our new scheme supports efficient address matching for routers in packet forwarding and for network middleboxes in packet filtering. Specifically, the scheme uses just one prefix rule for each allocated address block in a packet routing/filtering table. We show by analysis and simulation that the increased address utilization can lead to significant reduction in the probability of a denial-of-service under bursty address allocation requests. In contrast, the classical buddy system requires the aggregation of many requests over time to smooth out demand, resulting in service delays undesirable to end users. Our solution is applicable to ISPs in serving mobile users carrying many network connected IoT devices and IasS providers in the cloud in serving tenants with dynamically varying demands for network addresses.**

***Keywords*—Networking, IP allocation, Cloud, ISP, IaaS, IoT**

I. Introduction

We address a fundamental conflict in large-scale resource allocation, namely, efficient allocation of shared resources versus ease in managing the allocated resources. Specifically, we propose a novel block address allocation scheme, the nested buddy system, which exhibits desirable trade-offs between these two goals. Compared to previous solutions based on the standard buddy system, our solution can increase address-space utilization by more than 50% while requiring only one prefix rule per allocated address block for routing and firewall table entries.

We show by analysis and simulation that the increased utilization of address space can lead to a significant reduction in the possibility of denial-of-service due to depletion of address resources resulting from bursty demands. We illustrate how our solution can benefit Internet Service Providers (ISPs) in serving mobile users carrying many network connected Internet of Things (IoT) devices and Infrastructure-as-a-Service (IasS) providers in the cloud for serving tenants with dynamically varying demands on network addresses.

The contributions of the paper are as follows:

1) The proposed nested buddy system for block address allocation (Section IV-C).
2) Analysis on the utilization of address space under the classical buddy system (Section IV-B) and how nested buddy system can improve upon it.
3) An overlapping allocation scheme which allows use of only one prefix rule per allocated address block, and an efficient Ternary Content Addressable Memory (TCAM) implementation for resolving the allocation ambiguity (Section IV-C2).
4) Validation by numerical simulations and analysis for arrivals and frees of block address requests with sizes under various distributions (Sections V,VI and VII).
5) Formulation of address block allocation problems for achieving high address-space utilization while minimizing the required number of rules per allocated block in packet routing and filtering tables for both ISPs serving mobile users with many IoT devices (Section VII) and IaaS service providers serving tenants with dynamic requests of network addresses (Section VI).

II. Related Work

Providing optimizations for high-speed packet matching and forwarding consists of many different levels of abstraction. In order to perform filtering and routing at a line rate of multiple gigabits per second, optimizations are crucial; a survey of several approaches can be found here [1], among them are rule compression, re-ordering and rule-representation structuring [2], [3], [4]. Others have focused on designing overlapping and searching schemes for prefix specification used in hardware accelerators such as TCAMs [5].

The set of optimizations most similar to our work are efficient representations of packet matching rule sets for hardware implementation. Srinivasan et. al. showed that many network level packet filters that rely on source and destination addresses can be easily represented as a set of prefix entries that are an ideal fit for TCAM implementations [6]. Another optimization is in the representation of the rules (entries) themselves using encoding schemes to compress the rules within the limited budget of a Content Address Memory block [4].

Our work focuses on block allocation schemes, specifically the buddy system allocation scheme which was described by Knuth [7]. A survey of allocation schemes as well as workload simulation specifically for memory systems provides a complete treatment of different approaches and types of allocators [8]. Our nested structure leverages the basic premise of the buddy scheme but alters it to suit address block allocations identifiable by address prefixes for fast packet classification with TCAM. As far as we know, our propsed nested buddy system, which can minimize waste in block address allocation due to fragmentation and the number of required entries for their representation, is new.

III. Motivating Application Scenarios

We consider major trends associated with the increase of demand, and exhaustion, of public IPv4 addresses, in regards to the IoT and IaaS. The increase in number of devices and their requirement of routable addresses introduces the need for highly optimized design of the allocation and management of public addresses in order to increase address space utilization while minimizing the cost of packet routing and filtering.

A. *IaaS Providers in the Cloud*

Consider IasS providers which provide tenants' applications with external-facing public IP addresses. Given the scarcity of IPv4 addresses, today's service providers usually do not have a single, large contiguous address space [9]. Instead, they use an aggregation of address blocks of various sizes, accumulated over the entire IPv4 address space [10], [11]. When a particular address block is exhausted, the service provider will need to allocate tenants to other blocks, possibly even from other registration regions which can introduce poor performance as well as unseen cybersecurity. It is therefore critical that address allocation schemes have high utilization to limit the chance of address-space exhaustion.[1]

Address-space utilization must be balanced with ease of management of allocated addresses, especially in the number of required rules for routing and filtering tables in routers and firewall middleboxes. For high-speed, line-rate packet processing, hardware acceleration (e.g., TCAM) is often utilized which entails expensive circuitry and high power consumption for which it is crucial to minimize the number of installed rules. For example, use of just one TCAM entry for each allocated address block (i.e., one rule per block) is highly desirable. Use of entries logarithmic to the block size would be considered as prohibitively expensive [5], [6]. For these reasons, addresses should be allocated in blocks rather than individually to allow a single rule to represent a block of addresses.

Furthermore, for efficient address matching, it is preferable to route or filter based on address prefixes as in conventional IP routing. The classical buddy system is a standard block address allocation scheme that supports prefix matching, however, it generally incurs substantial waste due to fragmentation, inducing a degradation in terms of address-space utilization. The nested buddy system introduced in this paper aims at improving address-space utilization while requiring a minimum number of prefix rules for router and firewall tables.

B. *ISPs for Mobile Users with IoT Devices*

A main technology trend in the past several years has been the increased commercial availability of a wide variety of wearables, which together with other sensing and control devices, form the Internet of Things (IoT). We consider a scenario where a mobile user may carry many such network connected wearables while moving from location to location. When a user arrives at a foreign network, the network will need to provide routing and firewall services for his or her devices, in the style of mobile IP [12]. For easy management, it would be desirable to manage the devices associated with each user as a single address block subject to the same forwarding and filtering rules associated with the user. Therefore, it is important to allocate such address blocks efficiently to avoid an unnecessary denial-of-service due to depletion of available addresses, especially when users arrive in bursts.

[1]Note that for IPv6, which has abundant IP addresses, efficient address allocation remains critical, for line-rate hardware-assisted packet filtering/matching; see Section III-B.

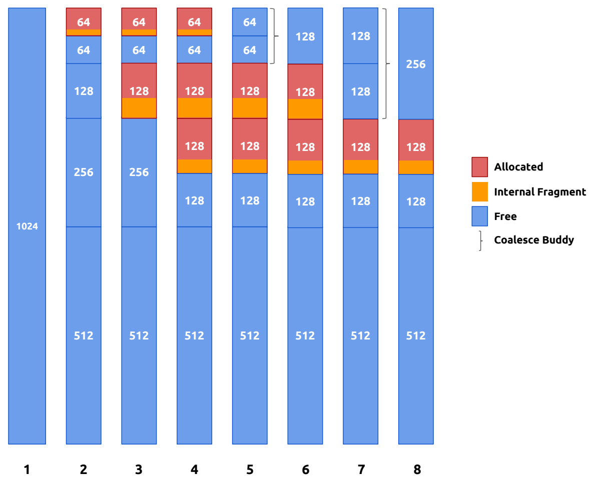


Fig. 1: Illustration of the classical buddy allocation system managing a resource of 1024 items. The orange areas within red allocations represent internal fragmentation. Buddies can only be recombined with both are free as in Step 5 and 7.

IoT devices will benefit from the emergence of IPv6 which has a substantial 128-bit address space, where every device can literally be assigned to a unique address as opposed to IPv4's 32-bit addresses. Unfortunately, IPv6 does not mean that address allocation is no longer a challenge. In fact, in order to route/filter packets at high speeds via hardware circuits such as TCAM, we still need to associate a logical group of devices such as those carried by a mobile user with a prefix address string of only a relatively small number of bits (e.g., 16 or 24). We can in general only expect gradual improvements in the implementation cost and energy consumption of hardware accelerators such as TCAM. These incremental improvements will likely be overwhelmed by new demands such as those imposed by IoT devices. We project the problem of efficient block address allocation addressed in this paper will only gain importance over coming years.

IV. Block Address Allocation Schemes

To be self contained, this section briefly reviews the basics of block address allocation schemes. An efficient allocation scheme must have low management cost while minimizing the amount of address space waste introduced when serving allocation requests. We adopt the terminology from the memory management literature for this discussion.

The wasted space introduced by an allocation scheme is referred to as fragmentation of which there are two types: **internal** and **external**. Internal fragmentation is the result of over-provisioning when performing an allocation for a request generally in order to provide easier management of the allocated block. External fragmentation refers to wasted blocks resulting from a sequence of allocations and frees yielding "islands" of small, unallocated blocks, of sizes too small for use by the application.

In addition, we are concerned with the cost of using the blocks once they are allocated as in the case of packet routing and filtering. A well structured, hierarchical allocation of IP addresses can greatly reduce the number of entries in the routing/filtering table necessary for specifying the desired network behavior, e.g., packet routing or filtering. We are

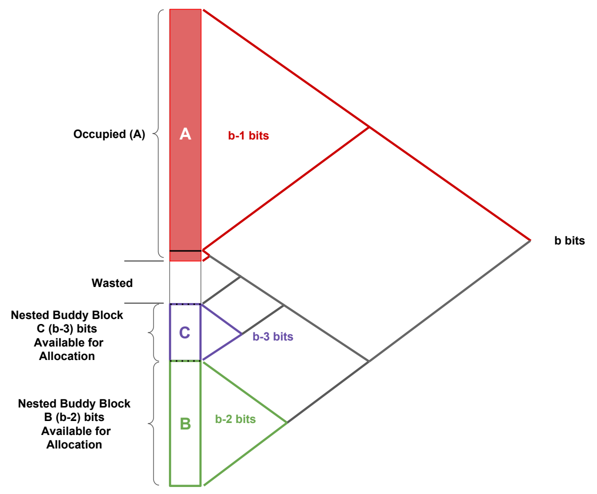

Occupied (A)
A
b-1 bits
b bits
Wasted
Nested Buddy Block C (b-3) bits Available for Allocation
C
b-3 bits
Nested Buddy Block B (b-2) bits Available for Allocation
B
b-2 bits

Fig. 2: Nested buddy system allocation utilizes the unoccupied portion of a buddy block allocation (block for A) from the bottom up. Each nested buddy block is the largest power of two block beginning from the bottom that can be allocated without overlapping with occupied region in red.

interested in minimizing the the cost in terms of number of entries necessary for compact representation of each allocation block. Thus the goal of an allocation scheme is to minimize the amount of fragmentation of the resource while providing minimal cost in representing the allocations made.

A. Buddy System Allocation

The address allocation scheme proposed by this paper builds on top of the classical buddy system. The general buddy system allocation handles dynamic requests by recursively dividing the resource until an appropriate size block is reached. The system was originally specified by Knuth in [7] and a full treatment of block allocation schemes, including the buddy system, is found in the literature (e.g., [8], [13]). The buddy system is particularly relevant to the subject of this paper because it embeds the basics of prefix-based address schemes widely used in Internet routing by only allocating address-prefix identifiable blocks.

The most common form of the buddy system divides the resource into two equal parts at each recursive step, always yielding block sizes in powers of two. Each pair of blocks at a given size are referred to as “buddies”; when both are free, they are recombined to form a single block of twice the size. Figure 1 depicts a sequence of requests and frees using the buddy system. The buddy system’s requirement of allocated blocks being identifiable by address prefixes results in single prefix entries per allocation, i.e., each block allocation can be represented by a single rule or, in TCAM implementations, a single TCAM entry.

Unfortunately, the cost of allocating blocks residing on prefix boundaries the potential of over-allocating a block could waste up to 50% of the addresses. For many application domains, such as IP address allocation for IaaS providers and ISPs considered in this paper, this amount of wasted address space due to internal fragmentation is too costly.

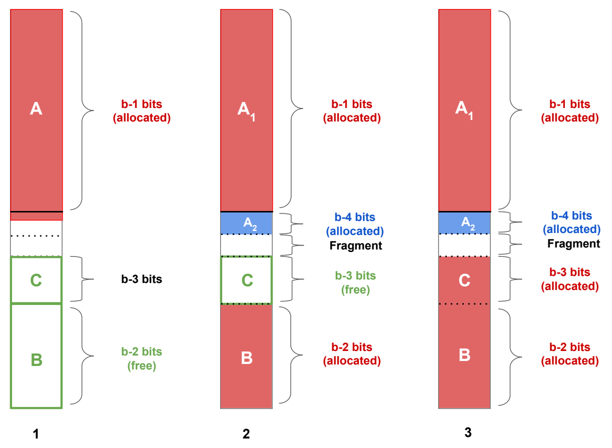


Fig. 3: Nested buddy system with two allocations of nested blocks B and C. At step 3, a fragment remains that is smaller than the minimum allowed on the nested free list. Red blocks (A_1, B, C) as well as the blue block (A_2) require just a single entry per block to refer to the entire address block.

B. Expected Fragmentation in Buddy System Allocation

The internal fragmentation rate for single allocation block is defined by

$$w = \frac{2^{\lceil log_2(r) \rceil} - r}{2^{\lceil log_2(r) \rceil}}. \tag{1}$$

where r is the request size, simply the allocated block size minus the size of the request, normalized by the size of allocated block.

Let the range of possible request sizes be $[m, n]$ where $m, n > 0$ and X be a random variable representing the allocation request size. Then $Pr(X)$ is the probability distribution over the range of allocation requests. The expected fragmentation rate is:

$$\frac{\sum_{r=m}^{n} Pr(X = r)(2^{\lceil log_2(r) \rceil} - r)}{\sum_{r=m}^{n} Pr(X = r) 2^{\lceil log_2(r) \rceil}}. \tag{2}$$

When using a continuous distribution, such as a Gaussian, the density which lies outside the range, $[m, n]$, is ignored for simplicity. We save a complete analysis for the simulation and empirical results.

C. Nested Buddy System Allocation

To resolve the waste due to internal fragmentation, we introduce the nested buddy system which allocates buddy blocks within the wasted internal fragments of the buddy system. The basic concept can be seen in Figure 2 which illustrates the underlying goal of utilizing the internal fragments created by buddy system allocations.

To use the unoccupied portion of the block allocated for A, we will allow a portion of the unused area to be allocated by future requests. Figure 2 shows the entire buddy block of size b bits allocated for request A. When allocating from the unoccupied region, we allocate from the bottom of the block up so that each nested block allocation is identifiable by a single prefix address. When A is allocated, the green and purple blocks, B and C respectively, can be made available for allocation.

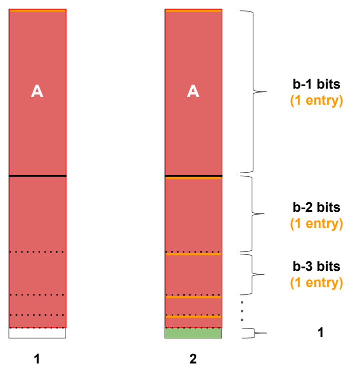


Fig. 4: Example of the potential cost of allocating a nested block for a minimum size, assuming that allocated blocks do not overlap. Allowing a nested allocation B, in green, to be arbitrarily small can introduce a high cost in the number of prefixes for representing A exactly.

Figure 3 illustrates the allocation of nested blocks and their allocation over a sequence of three requests. If a request is made which is larger than all available blocks, we split the request into multiple block allocations, each block is still identifiable by a single prefix entry.

We study the performance of our nested buddy system by first assuming that allocated blocks do not overlap. We then describe an overlapping scheme which can achieve high address-space utilization while still requiring only one entry per allocated block.

1) Non-overlapping Scheme: Figure 4 illustrates the worse case in terms of total entries for allocation of A and B within the same root block. When A is initially allocated, it only requires a single entry to represent the whole root block, shown by the orange block. In step 2, when a request is filled with the nested block in green it will require the prefix entries for A to be expanded, shown by the $b-1$ orange lines.

Based on the tree representation of Figure 2, we can prove the following claims. Their proofs are straightforward but we omit them due to space constraints.

Claim 1 *Suppose that allocation blocks cannot overlap. Then the number of entries in the worst case for allocating A and B into a root block of size b bits is $\mathcal{O}(b)$.*

That is, the number of entries required to represent two blocks within a root block in the worst case is logarithmic in the size of the block in order to specify A and B without the prefix entries correspond to overlapping address blocks. Scaling the number of entries logarithmically with the size of the block is considered to be prohibitively expensive as noted in Section I-A.

Let A_2 correspond to the part of A's occupancy that expands over the $b-1$ bit boundary which is shown in Figure 3 in blue. Let b_{A_2} and b_B be the size in bits for A and B, respectively. We state the following:

Claim 2 *A root block with allocations A and B can each be represented exactly with at most two entries if $(b-1)-(b_{A_2}-b_B) \geq 0$.*

This is simply to state that if A_2 and B can each be represented by a single power of two block without overlapping then the number of entries is equal to the number of blocks, two in this case.

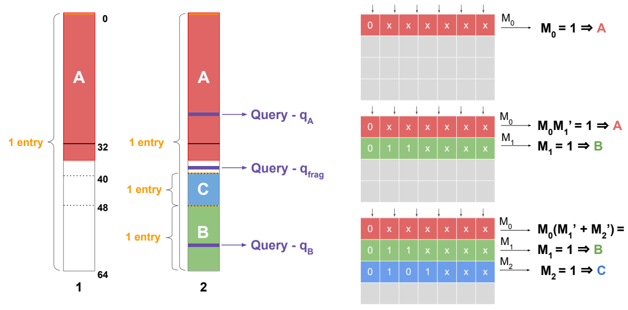


Fig. 5: Overlapping allocated blocks (e.g., A, B and C) allows for each allocation block to be representing by a single TCAM entry, as illustrated by color-coded TCAM entries on the right. Bit-string inputs and matchline outputs for the three instances of TCAM shown are denoted by downward and rightward arrows, respectively. The three possible query regions can be easily distinguished based on the nature of the overlaps and implemented efficiently via simple comparison after TCAM lookup. M' means M's complement. For example, for a TCAM lookup, if $M_1 = M_2 = 0$ and $M_0 = 1$, then $M_0(M_1' + M_2') = 1$, so the looked up entry is considered to be in A.

2) Overlapping Scheme: In order to provide maximum utilization of the internal fragments, we can relax the assumption of Claim 2. Rather than increasing the number of TCAM entries to specify A when allocating the nested block B, we simply maintain A's single entry that specifies the entire root block.

Figure 5 shows how the entries will be configured for each allocation within the root block. A and B will each have a single entry but regions covered by these entries overlap. There are three cases to resolve, as demonstrated by the queries in purple. If the packet address lies in A, shown as q_A, it will match only the corresponding TCAM entry for block A. If the packet address is in the unoccupied fragment, q_{frag}, then it will match the TCAM entry for A; however, there is nothing at that specific address so applying the policy is of no consequence. For the last case, if the packet address is in the block for B, q_B, it will match both entries corresponding to A and B.

We can easily handle the matchline conflict by realizing any match on M_1 must lie within the nested block B. The same is true of nested block C and so on. The right part of Figure 5 illustrates the TCAM entries for a buddy block with increased nested blocks entries added. If matchline M_0 is activated, before we can conclude that A was queried we must be certain that neither B nor C were activated, i.e., both are not matched. This overlapping entries scheme allows us to represent every address block with just a single TCAM entry.

V. Empirical Evaluation of Allocation Schemes

In the next four sections we will empirically evaluate the buddy and nested buddy systems using various application scenarios. In this section, we provide a baseline evaluation using synthetic workloads generated over a large parameter space using discrete Gaussian distributions for generating the allocation request sizes. We also consider the impact on fragmentation when requests and frees of blocks are intermixed.

In the next section, we present an IaaS provider scenario for allocating their public IP address space among their many tenants and the impact of auto-scaling [11], [10]. Finally, in Sections VII and VIII, we will present a personal IoT scenario in which users, along with their connected devices, move between ISPs.

To evaluate the the buddy and proposed nested buddy systems for block IP address allocation, we have constructed a simulator for serving requests and frees for a large IP address space. The simulator is implemented in Python and runs on an Intel Core-i7 3.6 GHz processor with 4 hyper-threaded cores and 16 GB RAM under Ubuntu 14.04 LTS. Each simulation configuration was run 500 times and results were averaged. The size of the address space to be allocated is a $/14$ network (16K addresses) for all presented results. We have found that larger address spaces have similar trends but were slower to evaluate. [2]

A. Baseline Evaluation of Allocation Fragmentation

We now present an empirical comparison with the theoretical model described in Section IV-B for the classical buddy system and the improved performance of our proposed nested buddy system. We consider a discrete Gaussian distribution for our baseline analysis with increasing variance. We will set the mean of the distribution to be 16 for the figures presented in this section.[3] The first set of experiments are performed without any de-allocations, referred to as freeing, in order to focus on the fragmentation in a simple case as modeled in Section IV-B.

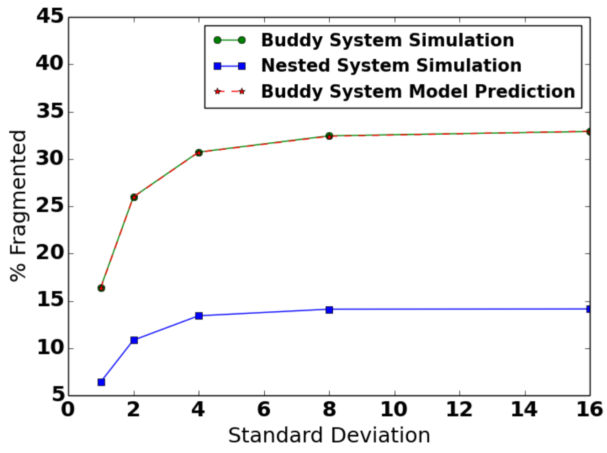


Fig. 6: A comparison of fragmentation predicted by our theoretical model compared with simulation results for buddy and nested buddy systems using a discrete Gaussian distributions for request sizes. Note that for the buddy system, simulation results match model predictions perfectly.

Figure 6 shows that the buddy system fragmentation predicted by the theoretical model correctly matches the simulation results under varying Gaussian distributions. The figure illustrates that the nested buddy system has a large advantage over the buddy system, in most cases greater than 50% improvement in terms of fragmentation. The buddy system also decays at a slightly faster rate as the variance increases.

[2]We chose 16K addresses for simulation in order to reduce run time without introducing noise which occurs when the address space is smaller.

[3]Plots confirming this have been omitted due to space constraints.

B. Analysis with Freeing

Now we consider the impact of mixing of allocation requests and frees on both the buddy and nested buddy systems. We present empirical results regarding the average number of allocated blocks per allocation request and the number of allocation requests fulfilled by both schemes. [4]

For a baseline evaluation of the impact on the allocation performance in regards to freeing previously allocated blocks, we introduce a Bernoulli distribution for expressing the ratio of the number of frees to the number of allocation requests. We refer to this parameter as the “free probability”, where a 10% free probability (corresponding to probability $p = .1$ for the Bernoulli distribution) means that one average one free will be made for every nine allocation requests. This an simplistic model of the interplay between requests and frees but provides a simple mechanism to test the breaking points of each system in regards to fragmentation and number of prefix entries for each request.

At each iteration of the simulation, a draw is taken from a Bernoulli distribution with a probability p to determine if an request or freeing action will occur. If the event is an address allocation request, the size will be determined according either to a uniform or discrete Gaussian. If the event is a free, one of the previously allocated requests is chosen at random to be freed.

Figure 7a and 7b are simulations using uniform and Gaussian distributions for the size of the allocation requests respectively. As the free probability approaches 50% (or $p = .5$), with high probability the allocation systems will never exhaust the address space. In this case the nested buddy system will behave similarly to the buddy system.

VI. IP Address Allocation for IaaS Providers

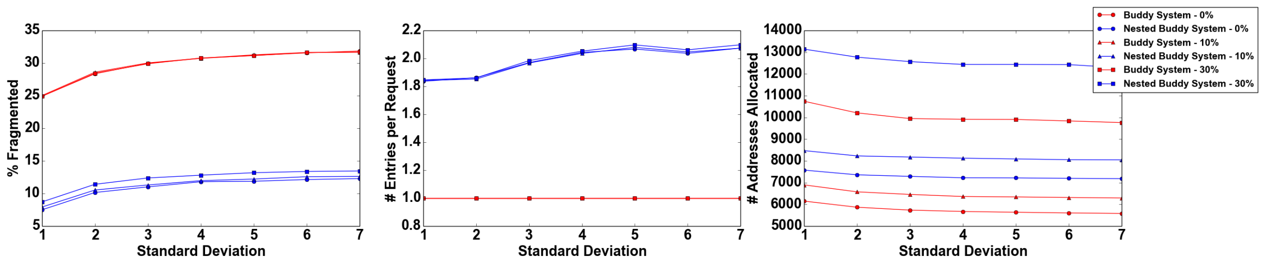
We now consider the scenario of an IaaS provider with a set of tenants, or clients, who want to outsource their web application and database system deployments to the cloud. Each tenant will request resources in an on-demand fashion which we will equate with the requests of public IP addresses from the IaaS provider's address space. To accommodate the fluctuations in the size and number of tenants, the IaaS provider will dynamically allocate blocks based on the tenant requests.

We will evaluate the allocation schemes described above in the context of the IaaS provider and tenant model for public IP address allocation, the cost in terms of expected fragmentation, the number of rules (entries) per allocation request and the number of allocations before having to deny service due to resource depletion.

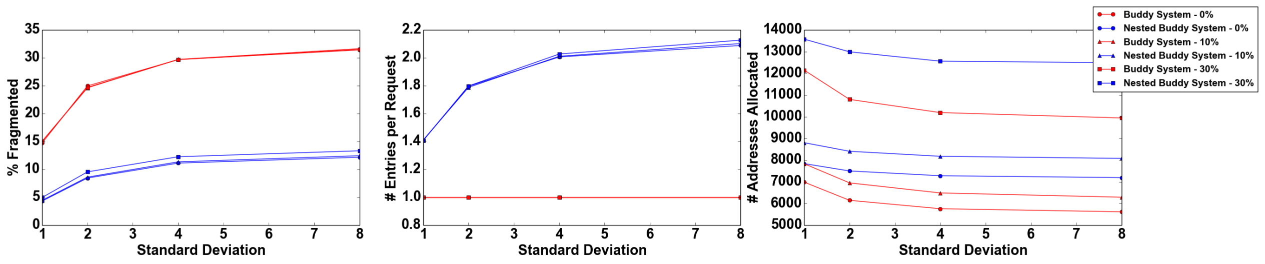
For generating tenant sizes and requests that more closely resemble IaaS usage, we assume a tenant's initial deployment size can be modeled as a discrete Gaussian distribution. Generally tenants will determine initial deployment sizes based on the average application response time and average number of requests for their application using Little's Law [14].

To study these assumptions and their corresponding parameters, we inspected several large IaaS providers stated quotas

[4]The number of blocks per request for the buddy system is always one. For the nested buddy system, a single request can be allocated to multiple blocks, requiring more than one entry for an allocation request.



(a) Uniform Distribution for Request Sizes



(b) Gaussian Distribution for Request Sizes

Fig. 7: Fragmentation, average number of entries and number of allocation requests by the buddy and nested buddy systems with increasing free probabilities (0%, 10%, and 30%) under a uniform and discrete Gaussian distributions. The nested buddy system has a slight increase in fragmentation and number of entries per allocation request but improves significantly over the buddy system.

and best practice guides for using their platforms. For example, most IaaS providers have instance limits for new customers, on the order of 10s of machines each with their own public IP addresses [15]. There are also statistics from companies of various sizes regarding the number of instances needed for peak or average traffic for running their web applications or services [16], [17]. These vary in size from 40 instances up to 100s of virtual instances to run their services. This gives us general ranges regarding the initial sizes for several types of tenants.

A. *Auto-scaling Allocation Performance*

Most of the IaaS providers have best practice guides for cloud tenants that want automatic scaling of their resources based on the demand for their applications [15], [18]. The general guidelines for auto-scaling involves increasing resources by 50% once a specified utilization metric is met for a sustained period of time. Resources are reduced by a constant number when utilization drops back below that threshold.

We construct a simulation where the tenants request initial blocks and then additional requests based on the auto-scale guidelines. We utilize a discrete Gaussian distribution for the initial tenant size (i.e., the size of the tenant's allocation request) and use that value as the base for auto-scaling. Figure 8 shows the same evaluation criteria but over the average initial tenant size configured for the discrete Gaussian distribution. As the initial size of tenant grows, the nested buddy system lowers the amount of internal fragmentation while the buddy system becomes more fragmented. This is due to the buddy system not being able to handle large and small allocation blocks as well as the nested buddy system which can fit the small blocks in the wasted space of the larger, earlier allocation requests.

VII. Personal IoT

One of the major trends in the IoT is a drastic increase in the number of connected devices such as wearables, sensing and control devices. We consider an IoT scenario in which users move between locations carrying many connected devices which require routable IP addresses. When the user arrives at a location, the local ISP will need to provide routing and firewall services for all the devices for each user. In order to limit the number of entries in the routing and filtering tables, it is beneficial to group the user's devices into a block representable with a single prefix entry so that all the devices in the same block can follow the same rules associated with the user.

To simulate a mobile IoT scenario for Internet providers we simulate Poisson arrival of users with a discrete Gaussian number of devices per user. We also model the number of users leaving per interval of time with a Poisson distribution. Specifically, at each time interval, we simulate the number of arrivals each with a Gaussian distributed number of devices. The simulator also then generates a number of departures which results in freeing that number of blocks from the Internet provider's allocated blocks. The arrival and departure distributions are parametrized each with their own λ_a and λ_d, which we consider the ratio of departure to arrival:

$$\lambda_r = \frac{\lambda_d}{\lambda_a}. \tag{3}$$

Figure 9 is the result of λ_r between .2 and .4 in order to ensure that the ISP's address block will eventually be exhausted in order to analyze both allocation systems under a full load. We selected the departure rate, λ_d, to be 8 and vary

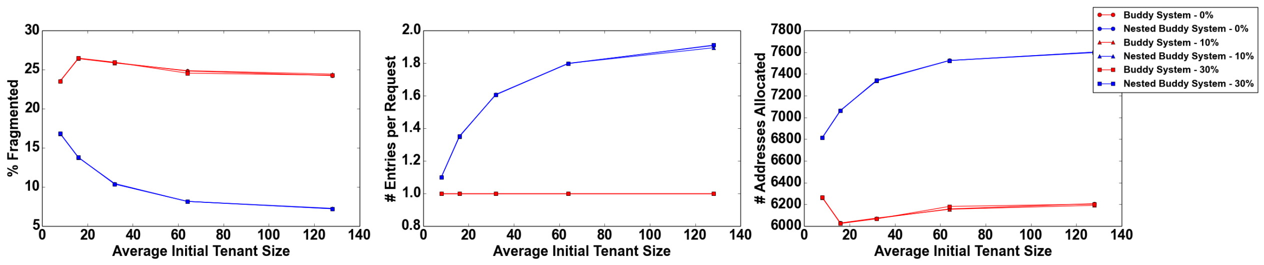


Fig. 8: Empirical results under an IaaS provider scenario with auto-scaling fluctuations using the 50% scaling rate with a discrete Gaussian distributed initial tenant request size.

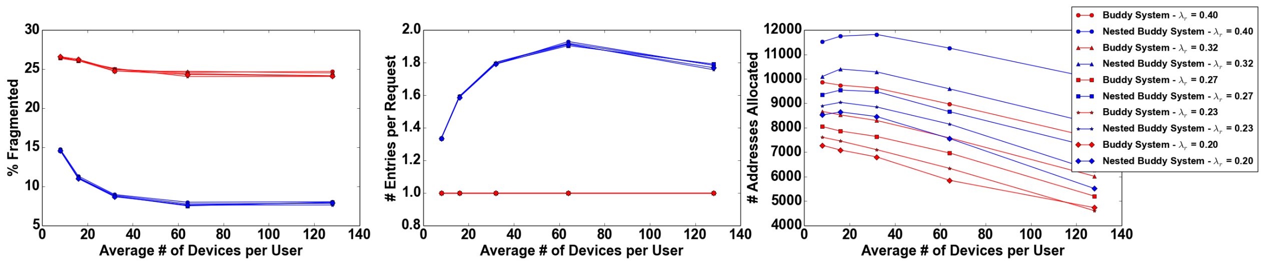


Fig. 9: Personal IoT scenario using Poisson arrival and departures for users with increasing number of devices.

the arrival rate, λ_a, between 20 and 40. The trend remains the same for other arrival and departure rates if the same ratio range ($\lambda_r = .2$ to $.4$.) is considered. As the number of devices for each user grows, the buddy system's fragmentation increases slightly while the nested buddy system maintains the same level of fragmentation, similarly to the baseline arrival model presented in the previous section. The cost comes in terms of the average number of table entries for each request but is still relatively low, less than two entries.

Even when the arrival Poisson distribution of users has a large variance (associated with bursty arrivals), and a more constant departure distribution (small λ), the nested buddy system has a 50% improvement in wasted addresses and 20% improvement in number of requests completed before a denial-of-service.

A. *Impact of Increased Tolerance to Bursty Arrivals*

We now consider the impact of increased tolerance to bursty arrivals of requests of the nested buddy system relative to the traditional buddy system. One methodology for improving the performance of the buddy system is to limit the burstiness of allocation requests which can cause it to exhaust the address space leading to a denial-of-service for new requests.

In general, to limit the amount of bursts in request arrivals one can simply buffer the requests together over a longer interval of time which will smooth out the distribution of request sizes. Simply increasing the delay between request and allocation is relatively easy to implement for the service provider; however, it may place an unacceptable penalty on the end user as waiting for address allocation prevents Internet access to their devices. The nested buddy system has increased tolerance to bursty requests; therefore the increased capability to handle larger variation in allocation requests, which eliminates the need to introduce system delays. That is, we can afford to provide lower latency for allocation requests while maintaining a small likelihood of exhausting the address space due to a large burst in allocation requests.

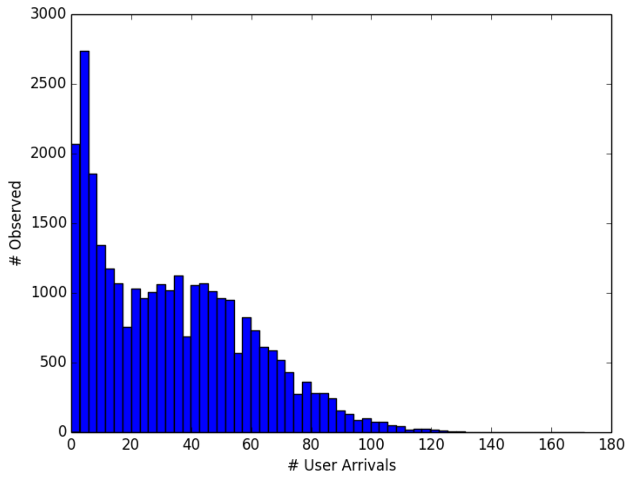


Fig. 11: Histogram of the number of arrivals during thirty minute intervals for entire real-world traffic dataset collected from fifteen wireless APs.

VIII. Personal IoT with Real-world Arrivals

Our final evaluation of the buddy and nested buddy allocation systems is with real arrival data rather than the simplifying distributions used in the previous two sections. We use a publicly available collection of wireless APs logs collected over a six year period [19], [20]. Each log records session

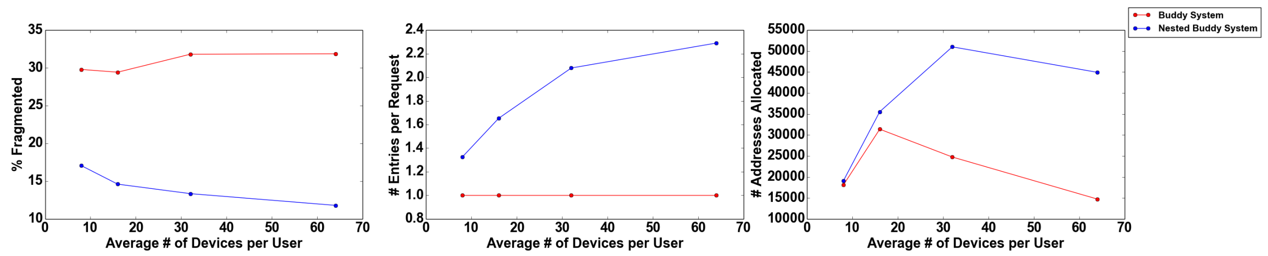


Fig. 10: Empirical results using real arrival data from wireless APs at a college campus with simulated number of devices for each arrival.

data for each unique user with an arrival and departure time. From this, we can consider any time window and count the number of new users that arrive within the interval. Figure 11 is a histogram of the number of arrivals in intervals of thirty minutes which illustrates that arrivals are, in fact, long tailed.

For the last simulation, we utilize the arrival data from the wireless AP logs as the input to the simulator for the number allocation requests. For each arrival, we generate the number of devices, or the size of the request, using a discrete Gaussian to be the request size. Figure 10 is the fragmentation, number of entries per request and number of successful allocations as the average number of devices per user increases according to a discrete Gaussian with low variance. The figure illustrates that under real-world user arrivals with many IoT devices, the nested buddy system provides an improvement of more than 50% in fragmentation while requiring less than two entries on average per request. This increase in capacity allows the allocation system to handle significantly larger bursts of users with many devices without exhausting the address space.

IX. Conclusion

The nested buddy system proposed in this paper mitigates two costs in block address allocation for network addresses: (1) low address-space utilization due to fragmentation and (2) use of multiple TCAM entries in servicing a user allocation request. Our solution can minimize both costs to be near their lowest-possible levels. In contrast, previous solutions can only minimize one of these two costs, not both. Block address allocation empowered by our solution allows the same policies to be applied to a group of network addresses.

In particular, we note that our solution has high tolerance against bursty requests (see, e.g., Figure 9) in avoiding the depletion of address space. A service provider could therefore offer the same low-latency service under bursty conditions, without having to aggregate requests over time in order to smooth out demand. We have provided analysis and extensive simulation to validate these results under synthetic request and free distributions and those modeled in the context of two example service scenarios: IaaS providers and ISPs for mobile users. Given the expected very large and highly dynamic demands on network addresses in such services in the future, we project that the proposed nested buddy system will become increasingly important over the coming years (see, e.g. discussion in Section III-B related to IPv6).

Acknowledgment

This work is supported in part by gifts from the Intel Corporation and in part by the Naval Supply Systems Command award under the Naval Postgraduate School Agreements No. N00244-15-0050 and No. N00244-16-1-0018.

References

[1] P. Gupta and N. McKeown, “Algorithms for packet classification,” *Network, iEEE*, vol. 15, no. 2, pp. 24–32, 2001.

[2] E. W. Fulp and S. J. Tarsa, “Trie-based policy representations for network firewalls,” in *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, pp. 434–441, IEEE, 2005.

[3] D. Rovniagin and A. Wool, “The geometric efficient matching algorithm for firewalls,” in *Electrical and Electronics Engineers in Israel, 2004. Proceedings. 2004 23rd IEEE Convention of*, pp. 153–156, IEEE, 2004.

[4] H. Liu, “Efficient mapping of range classifier into ternary-cam,” in *High Performance Interconnects, 2002. Proceedings. 10th Symposium on*, pp. 95–100, IEEE, 2002.

[5] R. Panigrahy and S. Sharma, “Sorting and searching using ternary cams,” *IEEE Micro*, no. 1, pp. 44–53, 2003.

[6] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, *Fast and scalable layer four switching*, vol. 28. ACM, 1998.

[7] D. E. Knuth, *Art of Computer Programming Volume 1: Fundamanetal Algorithms*. Addison-Wesley Publishing Company, 1972.

[8] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, “Dynamic storage allocation: A survey and critical review,” in *Memory Management*, pp. 1–116, Springer, 1995.

[9] G. Srinivasan, “Microsoft azure’s use of non-us ipv4 address space in us regions.” https://azure.microsoft.com/en-us/blog/, 2014. [Online; accessed 2016-03-31].

[10] “Microsoft azure documentation.” https://azure.microsoft.com/en-us/documentation/, 2016. [Online; accessed 2016-03-31].

[11] “Aws ip address ranges.” https://ip-ranges.amazonaws.com/ip-ranges.json, 2016. [Online; accessed 2016-03-31].

[12] C. Perkins, D. Johnson, and J. Arkko, “Mobility support in ipv6,” tech. rep., 2011.

[13] P. R. Wilson, “Some issues and strategies in heap management and memory hierarchies,” *ACM SIGPLAN Notices*, vol. 26, no. 3, pp. 45–52, 1991.

[14] J. D. Little, “A proof for the queuing formula: L= λ w,” *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[15] “Aws-ec2 documentation.” https://aws.amazon.com/ec2/, 2016. [Online; accessed 2016-03-31].

[16] “Scaling twitter.” http://www.slideshare.net/Blaine/scaling-twitter, 2007. [Online; accessed 2016-04-01].

[17] “How shopify scales rails.” http://www.slideshare.net/jduff/how-shopify-scales-rails-20443485, 2013. [Online; accessed 2016-04-01].

[18] “Google cloud platform documentation.” https://cloud.google.com/docs/, 2016. [Online; access 2016-03-31].

[19] M. Lenczner and A. G. Hoen, “CRAWDAD dataset ilesansfil/wifidog (v. 2015-11-06).” Downloaded from http://crawdad.org/ilesansfil/wifidog/20151106, Nov. 2015.

[20] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, “CRAWDAD dataset dartmouth/campus (v. 2009-09-09).” Downloaded from http://crawdad.org/dartmouth/campus/20090909, Sept. 2009.