A Denial-of-Service Resistant DHT

Baruch Awerbuch*[1] and Christian Scheideler**[2]

[1] Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA
[2] Institut für Informatik, Technische Universität München, 85748 Garching, Germany

Abstract. We consider the problem of designing scalable and robust information systems based on multiple servers that can survive even massive denial-of-service (DoS) attacks. More precisely, we are focusing on designing a scalable distributed hash table (DHT) that is robust against so-called past insider attacks. In a past insider attack, an adversary knows *everything* about the system up to some time point t_0 not known to the system. After t_0, the adversary can attack the system with a massive DoS attack in which it can block a constant fraction of the servers of its choice. Yet, the system should be able to survive such an attack in a sense that for *any* set of lookup requests, one per non-blocked (i.e., non-DoS attacked) server, every lookup request to a data item that was last updated *after* t_0 can be served by the system, and processing all the requests just needs polylogarithmic time and work at every server. We show that such a system can be designed.

1 Introduction

On Feb 6 of this year, hackers launched a distributed denial-of-service (DoS) attack on the root servers of the Domain Name System (DNS) [10]. DoS attacks can overwhelm servers with hacker-generated traffic and make them unavailable for legitimate communications. While the attacks significantly slowed the operations of some of the servers, they caused no problems for the overall DNS system because the system shifts work to other root servers if it has trouble with the first ones it tries to reach. This is possible because information is replicated among all root servers, and the root servers together have sufficient bandwidth to handle even major DoS attacks.

In this paper, we consider the problem of designing distributed information systems that are highly resilient against DoS attacks even if every piece of information is not replicated everywhere but only among a small subset of the servers. For distributed information systems that are connected to the Internet, like the DNS system, the servers may be known and therefore open for DoS attacks. There are various forms of DoS attacks. Application-layer DoS attacks, that try to abuse the protocols of the system in order to prevent it from functioning correctly, or network-layer DoS attacks that just aim at overloading servers

* Email: baruch@cs.jhu.edu. Partially supported by NSF CCF 0515080, ANIR-0240551, CT-0716676, CCR-0311795, and CNS-0617883.
** Email: scheideler@in.tum.de

with junk or faked messages in order to prevent them from processing legal ones. We are interested in designing a scalable information system (i.e., a system in which data and requests are handled in a scalable way) that can withstand even massive application-layer and network-layer DoS attacks (i.e., the attacker is powerful enough to generate requests or junk that can affect a constant fraction of the servers). Certainly, if the attacker has *complete* knowledge of the information system, then scalability and robustness against massive DoS attacks cannot be achieved at the same time. But what if the attacker only has complete knowledge up to some time step t_0 (that may not be known to the system)? Would it at least be possible to protect anything that was inserted into the system or updated *after* time step t_0? To answer this question, let us first formally define the attack model we will be focusing on in this paper.

1.1 The past insider attack model

The past insider attack model is motivated by the fact that a large percentage of the security breaches in corporate systems have internal reasons, many of them being caused by human error or negligence or insider attacks. In these cases, the system may be temporarily exposed, with potentially severe consequences for its functionality.

In the past insider attack model, we assume that an attacker has complete knowledge of the system up to some time step t_0 that is not known to the system. It can use this knowledge to attack the system at any time point after t_0. Given n servers, we allow the attacker to generate any collection of lookup requests it likes, one per non-blocked server, including lookup requests to blocked or non-existing data, and to block any set of ϵn servers for some sufficiently small constant $0 < \epsilon < 1$. The goal is to design a storage strategy for the data and a lookup protocol so that the following conditions are met:

For every data item d, the total space for storing d in the system is by at most a polylogarithmic factor larger than the size of d, and for any set of lookup requests with at most one request per non-blocked server, the following holds:

- *Scalability:* Every server in the system spends at most a polylogarithmic amount of work and time on the requests.
- *Robustness:* Every lookup request to a data item inserted after t_0 (or a non-existing data item) is served correctly.
- *Correctness:* Every lookup request to a data item is served correctly whenever the system is not under a DoS attack.

By "served correctly" we mean that the latest version of the data item is returned to the server requesting it. (We assume that there is a unique way of identifying the latest version such as the version number or a time stamp.)

Note that our model is different from proactive security models in which the adversary can never learn too much about the system within a certain time frame. Approaches for this model aim at protecting *everything* in the system, but this comes at a high price because this means that all the information in

the system has to be continuously refreshed, which may not be feasible in practice. We can show that one can protect nearly everything without continuous refreshing, and most importantly, everything that was updated after the security breach.

1.2 Towards robustness against past insider attacks

Let us have a quick look at the basic approaches for storing data in a distributed system.

- *An explicit data structure such as a distributed search tree or skip graph:* This approach has major problems with correctness since it is difficult to update the structure at attacked parts.
- *An implicit data structure like a hash table:* The hash table is structureless and therefore has no problems with correctness. It is also scalable, but it is not robust because the adversary knows exactly where the copies of a data item are located and can therefore block these.
- *The random placement of data copies among the servers:* This is not scalable but certainly robust.

Is there a way of combining these approaches in order to achieve scalability, robustness and correctness at the same time? Our main contribution in this paper is to show that a certain hybrid version of a hash table and random placement can achieve this task. More precisely, we will prove the following result.

Theorem 1. *Given n servers, our storage strategy just needs $O(\log^2 n)$ copies per data item so that our lookup protocol can serve any set of lookup requests, one per non-blocked server, in a scalable, robust and correct way, w.h.p. The robustness holds for any DoS attack in which at most ϵn servers are blocked, where $\epsilon > 0$ is a sufficiently small constant.*

In the proof of the theorem, we assume that the servers are completely interconnected since we are only focusing on reliable servers, so there are no scalability problems w.r.t. connectivity.

Although we consider only problems where all lookup requests are given at the beginning, we note that our lookup protocol can also be applied in a scenario where continuously new requests are generated. Furthermore, the correctness condition can be strengthened in a sense that beyond $O(n + D/n^k)$ data items, where D is the total number of data items in the system and k can be an arbitrary constant, all of the data inserted before t_0 can still be accessed by our lookup protocol under a DoS attack, but it can obviously not be guaranteed that everything is still accessible. Using coding strategies (like Reed-Solomon codes), the storage overhead for the data items can be reduced to $O(\log n)$ in Theorem 1. The constant ϵ that we need in our proofs is $\epsilon < 1/144$, but we did not try to optimize constants in this paper.

The beauty of our approach is that, even though it uses much more sophisticated concepts, it is still based on the well-known consistent hashing principle

[5], i.e., the servers are assigned to points in the $[0,1)$-interval and a data copy mapped to point $x \in [0,1)$ is stored at the server that is the closest predecessor of x in $[0,1)$. Thus, it could in principle be used on top of existing DHTs based on consistent hashing such as Chord in order to turn these into highly DoS-resilient DHTs. However, notice that in the scenarios considered in this paper, we can afford a completely interconnected network though most DHT implementations are based on bounded degree overlay networks, which would create an additional vulnerability.

Finally, we remark that we do not address the problem of handling insert requests in this paper but only how to store data in the system in a scalable way so that it can be retrieved despite massive DoS attacks. Managing insert requests is a tricky issue when application- and network-layer DoS attacks are allowed, and we discuss some of the reasons behind that in Section 2.1. Taking this restriction into account, our strategies would work best for archival systems or systems for information retrieval like Google, CiteSeer or Akamai.

1.3 Related work

The most prominent approach for a scalable information system is to implement a distributed hash table, or DHT. Well-known examples of DHTs are Chord [22], CAN [18], Pastry [3], and Tapestry [24]. Most of the DHT-based systems are based on concepts proposed in two influential papers: a paper by Plaxton et al. on locality-preserving data management in distributed environments [17] and a paper by Karger et al. on consistent hashing and web caching [5]. However, since in both cases the data management is based on hashing, none of these approaches is robust against past-insider attacks.

Various attacks on the data management layer of DHTs have been considered in the past. Most of the work considers the flash crowd scenario in which many peers want to access the same information at the same time. When using a pure DHT design, this can lead to severe bottlenecks. To remove these bottlenecks, various caching strategies have been proposed. Among them are CoopNet [16], Backslash [19], PROOFS [20] in the systems community and [14] in the theory community. However, being able to handle flash crowds is not sufficient to handle arbitrary collections of lookup and insert requests in a scalable way because much worse than having many requests to the *same* data item is to have many requests to *different* data items at the *same* location. Standard combining or caching strategies do not work here, but work on deterministic simulations of CRCW PRAMs (e.g., [11]) turned out come to the rescue here. These concepts allow the design of insert and lookup protocols that are guaranteed to handle *any* set of requests with at most one request per server that can be chosen by an adversary knowing *everything* about the system [2]. Thus, application-layer DoS attacks can be handled but not network-layer attacks since the protocols in [2] are purely hash-based.

There is a vast amount of literature on network-layer DoS attacks (see, e.g. [4, 12] for a taxonomy of these DoS attacks). Several authors have explored the use of DHTs to prevent DoS attacks from outsiders (e.g., [8, 6, 13]). Secure

Overlay Services (SOS) [8], for example, uses a proxy approach based on the Chord network to protect applications against flooding DoS attacks. WebSOS [21] is an implementation of SOS for web servers that makes use of graphical Turing tests, web proxies and client authentication. Mayday [1] generalizes the SOS architecture and analyzes the implications of choosing different filtering techniques and overlay routing mechanisms. Internet Indirection Infrastructure (i3) [9] also uses the Chord overlay to protect applications from direct DoS attacks. Other DoS limiting overlay network architectures have been explored in, e.g., [15, 23]. Most of the approaches above use traffic analysis or indirection approaches to make DoS attacks hard, but none of these would be able to survive the attackers considered in this paper since they essentially rely on the ability to protect servers from direct hits of adversarial traffic.

2 A DoS-resistant DHT

In this section we describe how to store information in a scalable and robust way in a DHT of completely interconnected servers. The DHT is based on the consistent hashing principle in a sense that the servers (also called *nodes* henceforth) are given points in the $[0,1)$-ring and any data copy that is mapped to a point $x \in [0,1)$ is stored at the node that is the closest predecessor of x in $[0,1)$.

First, we present our data storage strategy. Afterwards, we present and analyze our lookup protocol. For simplicity, we make the following assumptions:

- The number of nodes in the DHT is fixed to n, and n is a power of 2.
- The nodes are numbered from 0 to $n-1$, and node i is responsible for the interval $[i/n, (i+1)/n)$ in $[0,1)$.

Both assumptions can be relaxed (one can imagine, for example, that the nodes are randomly spread in $[0,1)$ so that the DHT does not need central coordination), but we use them here since they will keep our proofs simple.

2.1 The storage strategy

Like in [2], we use $c = \Theta(\log m)$ hash functions, denoted by $h_1, \ldots, h_c$, that map data names to points in the $[0,1)$ interval, where m represents the size of the universe of all data names, but this is the only feature the approach in this paper has in common with [2].

First, we introduce some notation. We assume that the points in $[0,1)$ are given in binary form, i.e., point $x \in [0,1)$ is given as $(x_1, x_2, \ldots) \in \{0,1\}^*$ with $x = \sum_{i \geq 1} x_i/2^i$. For any two bit sequences $x, y \in \{0,1\}^*$, $x \circ y$ is the unique point $z \in [0,1)$ with $(z_1, z_2, \ldots) = (x_1, \ldots, x_{|x|}, y_1, \ldots, y_{|y|})$. For any point $x \in [0,1)$ and $\ell \in \mathbb{N}$, we call set $T_\ell(x) = \{z \in [0,1) \mid z = y \circ x \text{ for some } y \in \{0,1\}^\ell\}$ the set of all points *at distance* ℓ from x. A *route* to x of length ℓ is any sequence of points $R = (z_\ell, z_{\ell-1}, \ldots, z_0)$ with the property that $z_0 = x$ and for every $i > 0$, $z_{i+1} = b \circ z_i$ for any bit $b \in \{0,1\}$ (which implies that $z_i \in T_i(x)$ for every i). Let $\mathcal{R}_\ell(x)$ be the set of all possible routes of length ℓ to x. A *random* route to

x is a route R chosen uniformly and independently at random from $\mathcal{R}_\ell(x)$ (i.e., z_ℓ is chosen uniformly and independently at random from $T_\ell(x)$).

When a data item d is inserted or updated in the system, we select a random route $R_i = (z_{i,\log n}, z_{i,\log n-1}, \ldots, z_{i,0}) \in \mathcal{R}_{\log n}(h_i(d))$ for every $i \in \{1, \ldots, c\}$. For each distance $j \in \{0, \ldots, \log n\}$, we store $\gamma \log n$ copies of d, for some constant γ that will be determined later. For each of these copies, we select an $i \in \{1, \ldots, c\}$ uniformly and independently at random and store the copy in point $z_{i,j}$ (resp. the node owning that point according to the consistent hashing scheme). Hence, altogether, we store $O(\log^2 n)$ copies of each data item in the system. It would be sufficient for our lookup protocol if instead of storing $O(\log n)$ copies for each data item for each distance j, we use Reed-Solomon or other codes to store each data item in $O(\log n)$ encoded pieces for each distance. That would reduce the overall storage overhead to $O(\log n)$ in Theorem 1, but for simplicity we will just assume that copies of d are stored.

Notice that as long as the set of DoS-attacked nodes is static, our storage strategy could be transformed into an efficient insert protocol together with techniques in [2] to avoid congestion problems. However, for a dynamically changing set of DoS-attacked nodes this is tricky since some non-DoS-attacked nodes are now missing information that is necessary for our lookup protocol to work correctly. A potential countermeasure here could be to delay the execution of the insert protocol at DoS-attacked nodes until the DoS-attack goes away. This requires extra management overhead and complicates the design of the insert protocol, which is why we left it out here.

For the rest of this paper, we will assume that the binary representations of all points are rounded to $\log n$ bits (i.e., the points are multiples of $1/n$, so there is one point for each node). For any $\ell \in \mathbb{N}$ let $\mathcal{T}_\ell = \{T_\ell(x) \mid x \in [0, 1)\}$, where we consider the points in $T_\ell(x)$ to be rounded to $\log n$ bits. That is, $|\mathcal{T}_\ell| = n/2^\ell$ and each set in $\mathcal{T}_\ell$ has a size of 2^ℓ.

2.2 The lookup protocol

We assume that we are given any collection of lookup requests, one per non-blocked server. The lookup protocol consists of two stages. The first stage is the contraction stage and the second stage is the expansion stage. During the contraction stage, the lookup requests are forwarded along random routes towards the hash values of the requested data items. Each lookup request encountering too many blocked or congested nodes stops and waits for the expansion stage to be executed. During the expansion stage, lookup requests are woken up in a controlled manner, and node sets of exponentially increasing size are explored in order to search for copies of the requested data items until sufficiently many copies have been found or the protocol decides that the item does not exist in the system. We start with the formal description of the contraction stage.

The contraction stage Each lookup request for some data item d chooses, for each $i \in \{1, \ldots, c\}$ and $j \in \{1, \ldots, \alpha \log n\}$, a random route $R_{i,j} \in \mathcal{R}_{\log n}(h_i(d))$

of length $\log n$ to $h_i(d)$, where α is a sufficiently large constant. Hence, altogether there are $\alpha c \log n$ random routes. For every i, let $Q_{\ell,i}$ be the set of all nodes in the routes $R_{i,j}$ to $h_i(d)$ that belong to $T_\ell(h_i(d))$, and let $Q'_{\ell,i}$ be the non-blocked nodes in $Q_{\ell,i}$.

Initially, all lookup requests are active, and all $i \in \{1, \ldots, c\}$ are active for all lookup requests. Then the contraction stage proceeds in rounds, executed from $\log n$ down to 0. In round r, every active request for some data item d sends a message to all nodes in its set $Q_{r,i}$ for all active i. Each of the nodes $v \in Q'_{r,i}$ replies back to that request. The reply contains the number $m_{v,i}(d)$ of messages it has received from requests to data item d for index i, which is called the *multiplicity* of d at v, and the number $C_{v,i}$ of different data items for which it was contacted by requests for index i, which is called the *congestion* at v. Afterwards, each lookup request checks the following rules:

1. For each active $i \in \{1, \ldots, c\}$ with $|Q'_{r,i}| < (\alpha \log n)/2$, the request deactivates i. If the total number of deactivated i's is at least $c/2$, the request becomes inactive.
2. For each active $i \in \{1, \ldots, c\}$ with $|\{v \in Q'_{r,i} \mid |C_{v,i}| \geq 2\alpha c \log n\}| \geq |Q_{r,i}|/2$, the request deactivates i. If the total number of deactivated i's is at least $c/2$, the request becomes inactive.
3. If there is an active i for which there is a node v in $Q'_{r,i}$ with $m_{v,i}(d) \geq 2\alpha \log n$, the request becomes inactive.

Inactive lookup requests do not participate any further in the contraction stage.

The expansion stage Each lookup request for some data item d that was active till the end of the previous stage, gets the most up-to-date copy of d from every non-blocked $h_i(d)$, returns the most up-to-date copy among these and finishes.

For the other requests, the expansion stage proceeds in rounds, this time numbered from 1 to $\log n$. In round r, every lookup request for some data item d that got deactivated in a round $r' < r$ and is not finished yet sends a message of the form $(d, r, i, -)$ (where "$-$" is an empty placeholder for a copy of d) to a random node in $Q'_{i,r}$ for each i that was active at the end of that round in the contraction stage. Each node v stores the IDs of the nodes that sent messages to it in S_v and stores the messages it received from them into its active pool of messages A_v, one copy for each $(d, r, i, -)$. If $|A_v| > 3c/\delta$, then any set of messages is discarded from A_v to get down to $|A_v| = 3c/\delta$, where the constant δ is chosen as in Lemma 1 below. For any remaining $(d, r, i, -)$ in A_v for which v stores a copy b of d (due to the data storage strategy defined above), $(d, r, i, -)$ is replaced by (d, r, i, b). Afterwards, A_v is managed as a FIFO queue. Every node v in the system executes the following push strategy $O(c \log n)$ many times:

- v dequeues one message (d, r, i, b) from A_v, enqueues it back to A_v and sends a copy of it to a random node in $T_r(h_i(d))$.
- For each message (d, r, i, b) received by v, v first checks whether A_v contains some message (d, r, i, b') in which copy b' is older than b (or empty). If so,

v replaces b' by b. Otherwise, v checks if $|A_v| = 3c/\delta$. If so, v discards the message. Otherwise, it checks whether it stores a copy b' of d that is younger than b. If so, v inserts (d, r, i, b') into A_v, and otherwise it inserts (d, r, i, b) into A_v.

If after these steps $|A_v| = 3c/\delta$, then v sends for each node $w \in S_v$ with original message $(d, r, i, -)$ the message $(d, r, i, *)$ back to w, where the "$*$" indicates that v was too congested. Otherwise, v sends (d, r, i, b) in A_v back to w.

Each lookup request that receives at most $c/4$ many $(d, r, i, *)$ messages returns the message (d, r, i, b) with the most up-to-date b (which may also be "$-$" if no copy was found) to whoever generated the request and is finished. Otherwise, it continues to participate in round $r + 1$.

2.3 Robust hash functions

In this section, we specify a central property the c hash functions $h_1, \ldots, h_c$ have to satisfy for the lookup protocol to work correctly and efficiently.

Given a set S of data items and a $k \in \mathbb{N}$, we call $F \subseteq S \times \{1, \ldots, c\}$ a *k-bundle* of S if every $d \in S$ has exactly k many tuples (d, i) in F. Given $h_1, \ldots, h_c$ and a distance ℓ, let $\Gamma_{F,\ell}(S) = \bigcup_{(d,i)\in F} T_\ell(h_i(d))$. Let U be the set of all possible (names of the) data items and $\mathcal{H}$ be the collection of hash functions $h_1, \ldots, h_c$, and let $m = |U|$. Given a $0 < \sigma < 1$, we call $\mathcal{H}$ a *(k, σ)-expander* if for any $\ell \leq \log n$, any $S \subseteq U$ with $|S| \leq \sigma n/2^\ell$, and any k-bundle F of S it holds that $|\Gamma_{F,\ell}(S)| \geq 2^\ell |S|$.

Lemma 1. *Let $0 < \lambda < 1$ be any constant. Then it holds for any $c \geq 8 \log m$ and $\sigma \leq 1/24$ that if the functions $h_1, \ldots, h_c$ are chosen uniformly and independently at random, then $\mathcal{H}$ is a $(c/4, \sigma)$-expander with high probability.*

Proof. Suppose that, for randomly chosen functions $h_1, \ldots, h_{2c-1}$, $\mathcal{H}$ is not a $(c/4, \sigma)$-expander. Then there exists an $i \leq \log n$ and a set $S \subset U$ with $|S| \leq \sigma n/2^i$ and a $c/4$-bundle F of S with $|\Gamma_{F,i}(S)| < 2^i |S|$. We claim that the probability $p_{s,i}$ that such a set S of size s exists is at most

$$\binom{m}{s}\binom{cs}{cs/4}\binom{n/2^i}{s} \cdot \left(\frac{s}{n/2^i}\right)^{cs/4}$$

This holds because there are $\binom{m}{s}$ ways of choosing a subset $S \subset U$. Furthermore, there are $\binom{cs}{cs/4}$ ways of choosing $cs/4$ pairs (d, j) for F and at most $\binom{n/2^i}{s}$ ways of choosing a set W of s sets in $\mathcal{T}_i$ witnessing a bad expansion of the pairs in F. The fraction of collections $\mathcal{H}$ for which the selected pairs (d, j) indeed have the property that $T_i(h_j(d)) \subseteq W$ is equal to $(\frac{s}{n/2^i})^{cs/4}$ because the hash functions $h_1, \ldots, h_c$ are chosen independently and uniformly at random.

Next we simplify $p_{s,i}$. Using the conditions on c and σ in the lemma it holds that

$$\begin{aligned}
&\binom{m}{s}\binom{cs}{cs/4}\binom{n/2^i}{s}\cdot\left(\frac{s}{n/2^i}\right)^{cs/4}\\
&\leq \left(\frac{em}{s}\right)^s (4e)^{cs/4}\left(\frac{en}{s2^i}\right)^s\left(\frac{s2^i}{n}\right)^{cs/4} = \left[\frac{em}{s}\cdot\left(4e^{1+4/c}\cdot\left(\frac{s2^i}{n}\right)^{1-4/c}\right)^{c/4}\right]^s\\
&\leq \left[m\cdot\left(4e^{1+4/c}\cdot\sigma^{1-4/c}\right)^{c/4}\right]^s \leq \left[m\cdot\left(\frac{1}{2}\right)^{c/4}\right]^s \leq \frac{1}{m^s}
\end{aligned}$$

if $c \geq 8\log m$ and m is sufficiently large. Hence, summing up over all possible values of s and i, we obtain a probability of having a bad $c/4$-bundle of at most $(2\log n)/m$, which proves the lemma. □

We remark that the hash functions have to form a $(c/4,\sigma)$-expander for some constant σ for our lookup protocol to work, but they do not have to be chosen at random. The proof above just illustrates that if they are chosen at random, they will form a $(c/4,\sigma)$-expander w.h.p.

2.4 Analysis of the lookup protocol

Next we show that the lookup protocol is correct, robust and efficient, i.e., for every lookup request for some data item d inserted after time t_0 (the threshold in the past insider model), a correct answer will be delivered for any DoS attack under our model, and every node spends only polylogarithmic time and work on the requests in the system. The correctness condition for all other data items is implied by our proofs. First, we prove the correctness of a lookup request given that it finishes in the expansion stage.

Lemma 2. *If a lookup request for some data item d finishes in round r of the expansion stage and d was inserted after t_0, then it returns the most up-to-date version of d.*

Proof. Consider any lookup request for some data item d that finishes in round r of the expansion stage and d was inserted after t_0. This means that it got at most $c/4$ many messages of the form $(d,r,i,*)$. The request only participates in round r of the expansion stage if it was still active at the beginning of round $r-1$ in the contraction stage, which it only the case if it had at least $c/2$ active indices i at the beginning of round $r-1$. These indices were all used in round r of the expansion stage, which means that at the end of that round the request got at least $c/4$ messages back of the form (d,r,i,b) where b is the most up-to-date copy of d that the node contacted by the request in $T_r(h_i(b))$ found. Let I be the set of these indices. From the fact that each $i \in I$ was active at the beginning of round $r-1$ in the contraction stage it follows that in round r of that stage, $Q'_{r,i} \geq (\alpha\log n)/2$ (because otherwise i would have

been deactivated in that round). Hence, at least half of the nodes in $Q_{r,i}$ that were sampled from $T_r(h_i(b))$ are non-blocked nodes. Since the nodes in $Q_{r,i}$ were chosen independently at random, it follows from the Chernoff bounds that the total number of blocked nodes in $T_r(h_i(b))$ is at most $2|T_r(h_i(b))|/3$, w.h.p., if α is a sufficiently large constant. We call a $T_r(h_i(b))$ satisfying such a property *non-blocked* in this proof. We can show the following claim.

Claim 1 *For any node v in a non-blocked set $T_r(h_i(b))$ it holds that if after $\phi c \log n$ executions of the push strategy in the expansion stage, where ϕ is a sufficiently large constant, $|A_v| < 3c/\delta$, then there are lookup requests for less than $3c/\delta$ data items that sent messages to nodes in $T_r(h_i(b))$ and every entry (d, r, i, b) in A_v stores the most up-to-date copy of d among the non-blocked nodes in $T_r(h_i(b))$ at the end.*

Proof. Can be shown along the lines of existing proofs on random, push-based broadcasting in complete networks (e.g., [7]). □

Hence, the lookup request obtains at least $c/4$ many replies (d, r, i, b) with most up-to-date copies in the respective sets $T_r(h_i(b))$. Since for each $i \in I$ at least a third of the nodes in $T_r(h_i(b))$ are not blocked, w.h.p., the lookup request returns the most up-to-date copy for at least $(c/4) \cdot (2^r/3) = c2^r/12$ of the $c2^r$ nodes in all sets $T_r(h_i(b))$, $i \in \{1, \ldots, c\}$. According to the robust storage strategy, $\gamma \log n$ many most up-to-date copies of d are randomly distributed among these $c2^r$ nodes, and none of these locations is known to the adversary, so the probability that none of these is stored in the at least $c2^r/12$ non-blocked nodes accessed by the request is at most $(1 - 1/12)^{\gamma \log n}$, which is polynomially small if γ is a sufficiently large constant.

We remark that for the data items least recently updated before t_0, at most $f = \max\{n, D/n^k\}$ many of them are bad, w.h.p., in a sense that none of their most up-to-date copies is stored in the at least $c2^r/12$ non-blocked nodes, where D is the number of data items in the system and k can be any constant. This is because there are at most 2^n ways of selecting $c2^r/12$ non-blocked out of at most n considered nodes and $\binom{D}{f}$ ways of selecting bad data items while the probability that these are indeed bad is at most $((1 - 1/12)^{\gamma \log n})^f$. If γ is sufficiently large compared to k, multiplying these terms gives a polynomially small probability. Hence, apart from f data items, the lookup protocol will deliver correct answers also for data items inserted or updated before t_0, w.h.p. □

Next, we look at the robustness and efficiency and will show that within a polylogarithmic time every request finishes, w.h.p. First, we consider the contraction stage. We show that the number of messages sent to any node is polylogarithmic in every round, which implies that every node spends only a polylogarithmic time and work on the contraction stage.

Lemma 3. *For every round r, at most $O(c \log^2 n)$ messages are sent to any node in the system, w.h.p.*

Proof. Consider some fixed node v in some set $T \in \mathcal{T}_r$. First, we bound $m_{v,i}(d)$ for any data item d and index i with $T_r(h_i(d)) = T$. Suppose that $m_{v,i}(d) \geq 8\alpha \log n$. Since every request for d chooses the nodes in $Q_{r,i}$ independently at random from the nodes in T, it follows from the Chernoff bounds that the expected number of messages for d at a node in T is at least $6\alpha \log n$, w.h.p. Hence, the expected number of messages for d in $T_{r+1}(h_i(d))$ was at least $3\alpha \log n$, w.h.p. However, in this case it holds for every node $w \in T_{r+1}(h_i(d))$ that $m_{w,i}(d)$ was at least $2\alpha \log n$, w.h.p. Thus, every request for d that was still active at round $r+1$ must have either deactivated i or became inactive. Hence, it must hold for every node $v \in T$ that $m_{v,i}(d) \leq 8\alpha \log n$, w.h.p., for any data item d and i with $T_r(h_i(d)) = T$.

A congestion bound of $|C_{v,i}| \leq 8\alpha \log n$ can be shown along exactly the same lines. Since there are c different indices i, the total number of messages sent to v is bounded by $c(8\alpha \log n)^2 = O(c \log^2 n)$. □

Hence, the runtime of the contraction stage and work per node is $O(c \log^3 n)$, w.h.p. Next we analyze the number of different data items for which lookup requests become inactive. This will be important to bound the congestion at the nodes in the expansion phase.

Let D_r be the set of all data items for which there are lookup requests that become inactive in round r. Furthermore, let BC_r be the set of data items with requests that become inactive due to too many inactive indices and MC_r be the set of data items with requests that become inactive due to a too high multiplicity. Certainly, $D_r = BC_r \cup MC_r$. First, we bound BC_r.

Lemma 4. *If $\epsilon < 1/144$, then it holds for every round r that $|BC_r| \leq 8\epsilon n/2^r$, w.h.p.*

Proof. For any r and any $T \subseteq \mathcal{T}_r$, we call T *blocked* if the attacker blocks more than a third of its nodes with its DoS attack, and T is called *congested* if more than a third of the nodes in T have a congestion of at least $2\alpha c \log n$. Consider any data item d. We call d *blocked* at round r if at least $c/4$ of its c sets $T_r(h_i(d))$ are blocked, and we call it *weakly blocked* at round r if there are blocked sets $T_{r_1}(h_{i_1}(d)), T_{r_2}(h_{i_2}(d)), \ldots, T_{r_k}(h_{i_k}(d))$ with $r_1, \ldots, r_k \geq r$ and $k = c/4$ and $i_1, \ldots, i_k$ being pairwise different. Similarly, we call d *congested* at round r if at least $c/4$ of its c sets $T_r(h_i(d))$ are congested, and we call it *weakly congested* at round r if there are congested sets $T_{r_1}(h_{i_1}(d)), T_{r_2}(h_{i_2}(d)), \ldots, T_{r_k}(h_{i_k}(d))$ with $r_1, r_2, \ldots, r_k \geq r$ and $k = c/4$ and $i_1, \ldots, i_k$ being pairwise different. In the following, WB_r denotes the set of weakly blocked data items and WC_r the set of weakly congested data items at round r.

Claim 2 *Whenever a request for some data item d deactivates some index i in round r, then $T_r(h_i(d))$ is either blocked or congested, w.h.p.*

Proof. Consider any request for some data item d in round r. Index i is deactivated for that request if

1. there are more than $(\alpha \log n)/2$ nodes in $Q_{r,i}$ that are blocked, or

2. there are more than $(\alpha \log n)/2$ nodes in $Q'_{r,i}$ that are congested.

In the first case, suppose that $T_r(h_i(d))$ is not blocked. Then the probability that $R_{i,j}$ chooses a node in $T_r(h_i(d))$ that is blocked is at most $1/3$ and, hence, the expected number of nodes chosen by the routes $R_{i,1}, \ldots, R_{i,\alpha \log n}$ that are blocked is at most $(\alpha \log n)/3$. Since the nodes are chosen independently at random, it follows from the Chernoff bounds that the probability that at least $(\alpha \log n)/2$ nodes in $Q_{r,i}$ are blocked is polynomially small in n (if the constant α is sufficiently large). Hence, if the request deactivates index i because of at least $(\alpha \log n)/2$ blocked nodes, then $T_r(h_i(d))$ is blocked, w.h.p.

The arguments for the congestion follow along the same lines. □

Now suppose that a request for data item d becomes inactive at round r due to at least $c/2$ deactivated indices. Then there are at least $c/4$ indices for which condition 1 in the contraction stage is true or at least $c/4$ indices for which condition 2 is true. In the first case, it follows from Claim 2 that d is weakly blocked, and in the second case, it follows from Claim 2 that d is weakly congested, w.h.p. For weakly blocked data items, the following claim holds.

Claim 3 *If s blocked nodes can cause a set of b weakly blocked data items at round r, then a set of $2s$ blocked nodes can cause a set of b blocked data items at round r.*

Proof. Consider item d to be weakly blocked, and let $T_{r_1}(h_{i_1}(d))$, $T_{r_2}(h_{i_2}(d))$, $\ldots, T_{r_k}(h_{i_k}(d))$ be the sets witnessing that with $k = c/4$. Any route through a set $T_{r'}(h_{i'}(d))$ with $r' > r$ can have at most $2^{r'-r}$ sets $T \in \mathcal{T}_r$ it can go through, and each of these sets T has a size of $|T_{r'}(h_{i'}(d))|/2^{r'-r}$. Hence, the number of nodes causing $T_{r'}(h_{i'}(d))$ to be blocked is sufficient to block also all $T \in \mathcal{T}_r$ reachable from $T_{r'}(h_{i'}(d))$. Hence, for any set of b weakly blocked data items, we can turn them into blocked data items when moving the blocking of nodes at distance $r' > r$ to nodes at distance r. Since we have to keep those sets $T_{r'}(h_{i'}(b))$ with $r' = r$ blocked, we have to at most double the number of nodes needed to transform weakly blocked data items into blocked data items. □

If the adversary can block at most $2\epsilon n$ nodes, then at most $6\epsilon n/2^r$ of the $n/2^r$ sets in $\mathcal{T}_r$ can be blocked, which covers at most $6\epsilon n$ nodes. Suppose the attacker can block a set S of data items in round r. Then there is a $c/4$-bundle F for S. According to Lemma 1, it holds that $|\Gamma_{F,r}(S)| \geq 2^r|S|$ if $|S| \leq \sigma n/2^r$. Since the largest possible size of $\Gamma_{F,r}(S)$ is $6\epsilon n$, it follows that $|S| \leq 6\epsilon n/2^r$, which is less than $\sigma n/2^r$ (so that Lemma 1 implies an upper bound on $|S|$) if $6\epsilon < 1/24$, or $\epsilon < 1/144$. Hence, if the adversary can block at most $2\epsilon n$ nodes, then it can cause at most $6\epsilon n/2^r$ blocked data items in round r. This implies together with Claim 3 that if the adversary can block at most ϵn nodes, then it can cause at most $6\epsilon n/2^i$ weakly blocked data items in round r. Combining this with Claim 2, it follows that if the adversary can block at most ϵn nodes, then $|WB_r| \leq 6\epsilon n/2^r$, w.h.p.

Using the same arguments for congested data items, it also follows that $|WC_r| \leq 6\epsilon n/2^r$, w.h.p. Hence, $|BC_r| \leq |WB_r| + |WC_r| \leq 12\epsilon n/2^r$, w.h.p. □

Next we bound MC_r.

Lemma 5. *For every round r, $|MC_r| \leq n/2^r$.*

Proof. Suppose that there are ℓ many lookup requests for data item d in round r. Then the expected number of messages per node w.r.t. i in any $T_r(h_i(d))$ is $(\alpha \log n)\ell/2^r$. If $\ell \leq 2^r$, this is at most $\alpha \log n$. Since each message chooses a node independently at random, it follows from the Chernoff bounds that the probability that there is a node in a set $T_r(h_i(d))$ with at least $2\alpha \log n$ messages for d is polynomially small. Hence, if a lookup request for some data item d becomes inactive due to condition 3 of the contraction stage, then d has a multiplicity of at least 2^r, w.h.p. Since there are at most n active requests at round r, it follows that $|MC_r| \leq n/2^r$. □

Combining Lemmas 4 and 5 it follows that $|D_r| \leq 12\epsilon n/2^r + n/2^r \leq 2n/2^r$, w.h.p. Now we are ready to analyze the expansion stage. First, we bound the number of messages sent to a node in each round.

Lemma 6. *For every round r, at most $O(c \log n)$ messages are sent to any node in the system, w.h.p.*

Proof. Only requests that were active in round $r-1$ of the contraction stage will participate in round r of the expansion stage. According to Lemma 3, the number of messages sent to any node in the system in any round of the contraction stage is $O(c \log^2 n)$ w.h.p. Since every lookup request sends out $\alpha c \log n$ messages in the contraction stage but only at most c messages in the expansion stage, it follows that the number of messages sent to any node in the expansion stage is at most $O(c \log n)$, w.h.p. □

The description of the expansion stage and Lemma 6 immediately imply that the runtime and work per node of the expansion stage is at most $O(c \log^3 n)$. Combining this with our bounds for the contraction stage, we get:

Lemma 7. *For any collection of lookup requests, one per non-blocked node, the lookup protocol needs at most $O(c \log^3 n)$ time and work at every node.*

Next, we show that every request will eventually finish in the lookup protocol, w.h.p.

Lemma 8. *For every round r, the number of data items with requests participating in it is less than $3n/2^r$.*

Proof. We prove the lemma by induction on the number of rounds. For round 1, the lemma certainly holds. So consider any round $r \geq 1$ for which the lemma holds. A set $T \in \mathcal{T}_r$ is called *congested* if there are messages for at least $3c/\sigma$ many different data items in T. Since there are requests for less than $3n/2^r$ many data items and the messages for each data item are limited to c sets $T \in \mathcal{T}_r$, there must be less than $\sigma n/2^r$ many sets $T \in \mathcal{T}_r$ that are congested. A data item d is called *congested* if there are at least $c/4$ many indices i for which

$T_r(h_i(d))$ is congested. Let S be the set of congested data items. Then there is a $c/4$-bundle F for S. According to Lemma 1, it holds that $|\Gamma_{F,r}(S)| \geq 2^r|S|$ if $|S| \leq \sigma n/2^r$. Since the largest possible size of $\Gamma_{F,r}(S)$ is less than σn, it follows that $|S| < \sigma n/2^r$. As a worst case, we assume that all requests for congested data items will not finish in round r and therefore have to continue in round $r+1$. These will combine with the requests for at most $n/2^r + 12\epsilon n/2^r$ data items with requests that became inactive in round r of the contraction stage, which gives an upper bound of less than $3n/2^{r+1}$ on the number of data items with requests in round $r+1$ (given that $\epsilon < 1/144$ and $\sigma \leq 1/24$). □

Hence, in round $r = \log n$, there are at most 3 data items left with requests. In this round, all sets $T_r(h_i(d))$ are equal to the entire node set, so there is no congested node set left. Hence, according to the expansion protocol, every request will finish in that round. Combining this with Lemmas 2 and 7 proves Theorem 1.

3 Conclusions

In this paper we showed that a DHT for scalable data storage and retrieval can be designed that is provably robust against massive application- and network-layer DoS attacks. Certainly, low-level protocols still have to be developed for our operations that work well and correctly in an asynchronous environment. Also, it would be interesting to find out whether adaptations of our strategies are possible to bounded degree DHTs so that they can sustain DoS attacks of a similar magnitude as considered in this paper.

References

1. D.G. Andersen. Mayday: Distributed filtering for internet services. In *4th Usenix Symp. on Internet Technologies and Systems*, 2003.
2. B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. In *Proc. of the 18th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2006. See also http://www14.in.tum.de/personen/scheideler.
3. P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.
4. D. Dittrich J. Mirkovic, S. Dietrich and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall PTR, 2005.
5. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahi. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of the 29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, 1997.
6. F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. *World Wide Web*, pages 514–524, 2001.
7. R. Karp, S. Shenker, C. Schindelhauer, and B. Vöcking. Randomized rumor spreading. In *Proc. of the 41st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

8. A.D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proc. of ACM SIGCOMM*, pages 61–72, 2002.
9. K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming ip packet flooding attacks, 2003.
10. G. Lawton. Stronger domain name system thwarts root-server attacks. *IEEE Computer*, pages 14–17, May 2007.
11. K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel mamories. *Acta Informatica*, 21:339–374, 1984.
12. J. Mirkovic and P. Reiher. A taxonomy of ddos attacks and defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2), 2004.
13. W.G. Morein, A. Stavrou, D.L. Cook, A.D. Keromytis, V. Misra, and D. Rubenstein. Using graphic turing tests to counter automated ddos attacks against web servers. In *Proc. of the 10th ACM Int. Conference on Computer and Communications Security (CCS)*, pages 8–19, 2003.
14. M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2003.
15. G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson. A framework for collaborative ddos defense. In *Proc. of ACSAC 2006*, 2006.
16. V.N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
17. G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM '01*, 2001.
19. T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
20. A. Stavron, D. Rubenstein, and S. Sahn. A lightweight robust P2P system to handle flash crowds. In *Proc. of the IEEE Intl. Conf. on Network Protocols (ICNP)*, 2002.
21. A. Stavrou, D.L. Cook, W.G. Morein, A.D. Keromytis, V. Misra, and D. Rubenstein. Websos: An overlay-based system for protecting web servers from denial of service attacks, 2005.
22. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM '01*, 2001. See also `http://www.pdos.lcs.mit.edu/chord/`.
23. X. Yang, D. Wetherall, and T. Anderson. A dos-limiting network architecture. In *Proc. of the ACM SIGCOMM*, 2005.
24. B.Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, UCB/CSD-01-1141, University of California at Berkeley, 2001. See also `http://www.cs.berkeley.edu/~ravenben/tapestry`.