Free-Riding the BitTorrent DHT to Improve DTN Connectivity

Sebastian Schildt Till Lorentzen

Johannes Morgenroth Wolf-Bastian Pöttner Lars Wolf

Institute for Operating Systems and Computer Networks
Technische Universität Braunschweig
Braunschweig, Germany
schildt|lorentze|morgenroth|poettner|wolf@ibr.cs.tu-bs.de

ABSTRACT

Until now there exists no standardized or widely-used name resolution mechanism for Bundle Protocol based DTNs. In local IP based networks the IP Neighbor Discovery (IPND) protocol provides link-local discovery of DTN neighbors, however in the Internet there is no mechanism that maps Bundle Protocol addresses (EIDs) to Convergence Layer addresses. This seriously hampers connectivity in the DTNBone.

In this paper we argue, that a DHT-based naming mechanism is a good fit for DTN networks. We show the advantages of using the existing DHT infrastructure from the BitTorrent file sharing network as a basis instead of rolling out a custom solution. We will present experiences and performance measures with a lightweight implementation of the proposed system that we began to distribute as integrated component of the IBR-DTN Bundle Protocol implementation.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Store and forward networks*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*

Keywords

Bundle Protocol, DTN, DHT, BitTorrent, Naming Service, Routing

1. INTRODUCTION

The Bundle Protocol (BP), specified in RFC5050 [10] is a widely used DTN protocol. It realizes the Store-Carry-and-Forward mechanism inherent to DTNs and additionally supports end-to-end acknowledgements on top of the hop-by-hop approach. In fact, the BP can be seen as a superset of IP and TCP, as it includes elements from both the networking and the transport layer: In a continuously connected network it works much like the classical TCP/IP stack, while additionally it is able to deal with disruptions of the network. This leads to the use of the BP in networks with IP enabled mobile devices such as smartphones, which are regularly connected to the Internet [9][3].

However, the BP ecosystem has a major shortcoming when it comes to operating in large-scale fully interconnected networks such as the Internet: There is no standard mechanism to find a node or the next hop for a specific DTN EID. Compared to the standard IP architecture there is no standardized naming system such as DNS and there is no usable routing protocol to find the next hop to a destination if that hop is located in a so far unknown network across the Internet. The various proposed routing protocols for DTNs normally assume an ad-hoc scenario relying on different forms of flooding and local neighbor discovery, both of which are not applicable in the Internet.

This leads to the situation that currently the DTNBone, a public internet-based DTN network operated by interested parties, largely consists of a Wiki page[1]. An aspiring DTN node operator who wishes to "join" the DTNBone needs to add some static routes to the IP addresses of existing DTNBone nodes found on the Wiki, and possibly add the connectivity information of his new node to the Wiki, if he intends to be contacted by others. This combined with the fact, that different DTNBone members use different routing protocols leads to the situation that the DTNBone is far more fragmented then it should be. What is fundamentally missing for BP based DTNs, is the ability to connect to a node when only its EID is known.

As DTN EIDs have no structure imposed on them, apart from the requirement of being valid URIs, a highly structured and tiered approach such as DNS is not a good fit for BP based DTNs. In fact an EID can have arbitrary information coded into it. For example it might describe a node, a group or address some content. In [9], it has been suggested that DHTs might be a feasible way to tackle the naming problem in DTNs. DHTs are a robust way to store data in a distributed fashion. A DHT is a key-value store which is distributing the load evenly across participant nodes while

[1] `http://www.dtnrg.org/wiki/DtnBone`

CHANTS'12, August 22, 2012, Istanbul, Turkey.

still providing good lookup performance. Generally DHTs are resilient against node failure, have excellent scalability and support a flat name space, which is a good fit for the BP.

In [12] Waldhorst sketches Arriba, a generic DHT-based architecture for routing in overlay networks spanning heterogeneous technologies. That work focusses on routing, but it is not tied to the BP and does not specify how to create the necessary unique node ids and assign them to device names or underlay addresses. In [8] a comprehensive custom DHT-based naming and notification for heterogeneous BP based DTNs has been presented and was implemented as a prototype for IBR DTN.

The contribution of this paper is to solve this shortcoming in the BP ecosystem. Building on the experiences from [8] we will present a naming system for BP based DTNs that uses the BitTorrent (BT) DHT to provide the mapping from DTN EIDs to convergence layer information. Using an existing DHT has various advantages: Many stable and proven implementations are available, and there are always thousands of usable DHT members online, making the system more reliable even when the DTN Naming service is only deployed on a few nodes.

The remainder of this paper is structured as follows: In section 2 we present the concept of our naming system and the needed compatible extensions to the BitTorrent DHT. We present technical details of our implementation in section 3 and conduct an extensive evaluation of the systems performance and reliability in section 4. Finally, in section 5 we finish with some concluding remarks.

2. ARCHITECTURE

In this section we describe the general architecture and operation of our DTN DHT approach as depicted in figure 1. The system is composed of three parts: We use a fully compatible standard compliant BT DHT implementation as basis. On top of this standard BT DHT we implement some extra functionalities that realize the DTN naming service. These extensions do not compromise compatibility with existing BT DHT implementations. This extended DHT service can then be used by applications such as DTN daemons to resolve and announce Bundle Protocol EIDs.

2.1 The BitTorrent DHT

The standard BT DHT protocol [1] is a Kademlia[7] based DHT which is modified to the requirements of the BT protocol. A Kademlia DHT is organized as a tree using the XOR operation as distance metric. Due to Kademlia's properties, nodes automatically learn more about the DHT's structures while routing and forwarding DHT messages. As the BT Kademlia variants are widely deployed, their performance has been studied in detail and their properties are widely understood [4].

The BT protocol uses SHA-1 [5] hashes to identify files, which will be shared by a user's BT application on a specific port. A user of the DHT stores his local BT port to the BT DHT using the SHA-1 hash of the file as key. The nodes storing the port information also store the IP address from the node where the UDP store message originated from and puts it together with the port in a linked list belonging to that key. On a lookup all, or a subset, of the stored IP and port combinations should be returned. There is no method to delete an entry from the DHT. All entries should time out after 30 minutes. Therefore the store method has to be executed regularly to have a stable entry in the DHT.

The BT DHT uses a lightweight RPC protocol. The RPC messages in the BT DHT are bencoded [2] ASCII messages sent through UDP. This encoding is also used by the BT protocol. There are several open source BT clients with DHT support available. A widespread, platform independent open source client is Transmission[2]. It utilizes a very lightweight BT DHT implementation written in ANSI C[3].

2.2 BitTorrent DHT Protocol Extension

For a BP naming service there are several kinds of information that a user might want to store and retrieve: The most important information is the convergence layer address for a given EID. For the mostly used convergence layers this includes the IP address and a port for the TCP and the UDP convergence layer. A node might provide information about more than one convergence layer. Additionally, information about a node's neighbors and information about groups is of interest. However, the BT DHT does not allow much flexibility when storing data: The only data a user can reliably store in the BT DHT is a port number (in BT this is the port, the BT daemon is listening on). Even an arbitrary IP address can not be stored directly in the BT DHT, instead it is automatically taken from sender of a DHT Store RPC.

Changing the normal BT DHT routing operations or augmenting the Store RPCs was not an option, as we want to remain compatible with the BT DHT to be able to leverage the services of all BT DHT members. Therefore we choose to extend the BT DHT protocol with another RPC call in a way that retains compatibility with BT clients: When storing information to the DHT, DTN nodes will use the EID of the node or group they want to store information about as hash and store the port of their DHT implementation (opposed to the port of the BT daemon as BT clients do). This structure is depicted in figure 1: The rightmost part represents an unmodified BT DHT implementation: It uses the standard *get_peers* RPC of the BT DHT, which either returns new nodes running the DHT implementation, which are nearer to the queried key in the DHT topology, or it returns the actual IP, port combinations that have been stored for the queried hash. The middle part of figure 1 contains the two added RPCs and the left side represents the application using our DTN DHT implementation, usually a DTN daemon.

Upon querying the DHT for an existing and announced EID, the DTN-DHT implementation should receive the IP and port information of one or more DTN-DHT implementations that stored this entry (the received *values* in figure 1). Until this point our implementation behaves exactly like vanilla BT DHT implementations. However, in the next step the querying node sends a new DHT RPC, a DTN Information Query (*DTNIQ*), to all the nodes from the query's result set. In accordance with the structure in [1] a *DTNIQ* looks like this

bencoded `"t":"aa", "y":"q", "q":"dtn", "a":{"eid" : "dtn://my_hostname"}`

where t is the transaction id, that needs to be repeated in the response, y denotes that this is a query of the newly de-

[2] `http://www.transmissionbt.com/`

[3] `http://www.pps.univ-paris-diderot.fr/~jch/software/bittorrent/`

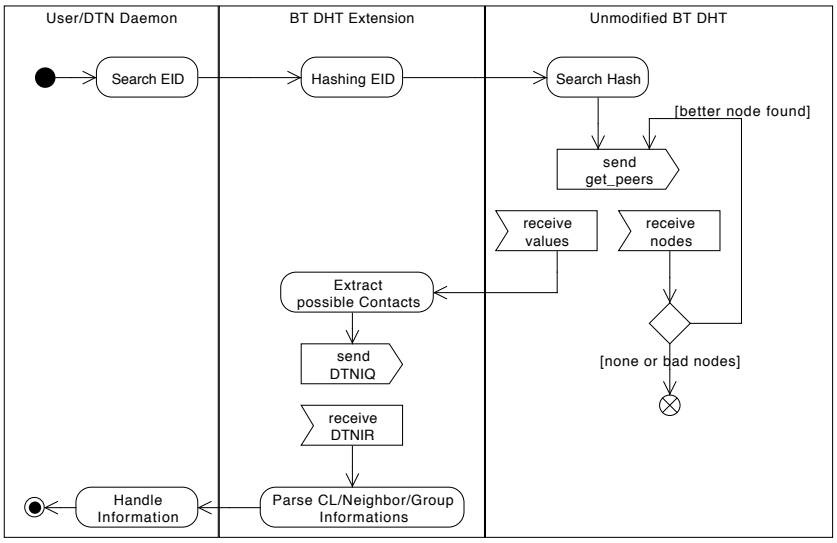


Figure 1: DTN DHT components

fined type (q) "dtn". The dictionary of arguments a contains the key eid which stores the EID of the querying node.

If this query is sent to a node, which in fact is not a DTN DHT node (because there is an invalid entry in the result set), that node will just ignore the UDP RPC query. Since most BT nodes advertise a TCP port for their BT service, a UDP message will never reach the other user's BT application. Should the other party run a UDP variant of the BT protocol on the advertised port, it will abort parsing the unknown message and silently drop it (in fact, since there is a lot of questionable software around in the BT cloud, most implementations are hardened to deal with all kinds of broken, garbled or unexpected messages).

If the query is received by a DTN-DHT member, it will answer with a DTN Information Response (*DTNIR*) containing information about itself, including

- EID
- List of group EIDs it is a member of
- List containing EIDs of its direct neighbors
- List of the provided convergence layers

In accordance with the structure in [1] a *DTNIR* looks as follows:

bencoded: `"t":"aa", "y":"r",`
`"r": {"eid":"dtn://my_hostname" ,`
`"cl":["name=TCP;port=4556","name=UDP;port=4556"],`
`"nb":["eid1","eid2", ... ],`
`"gr": ["eid1", "eid2", ...]}`

where t needs to reflect the transaction id of the matching query and y denotes that this message is a response. The r dicitonary contains the EID of the answering node. Because nodes can also announce their direct neighbors and groups, this is not necessarily the same EID as the key that was originally queried in the DHT. The cl array contains a list of convergence layer addresses, the nb array lists all neighbors of this node and the gr array contains all group EIDs this node is a member of. Currently the implementation supports the TCP and UDP convergence layers in the cl array, however conventions for encoding the information of other convergence layers can be added easily.

The *DTNIQ*/*DTNIR* handshake also makes sure, that only connectivity information of valid DTN nodes are given to the DTN daemon. This is important, because we noticed that answers to DHT searches frequently contain, among the correct data, also bogus values. We analyze this in further detail in section 4.5. As those non-DTN nodes will not answer the *DTNIQ* handshake, they will not be given to the DTN daemon, preventing it from making futile connection attempts to invalid nodes.

2.3 Routing and Neighbor announcement

To use the DHT-based naming service more effectively, we allow a node to not only announce its own EIDs to the DHT, but also publish EIDs of neighboring nodes. There are several reasons why this might be desired:

- A node might use a DTN implementation that is not DHT aware. If such a node is discovered by a DHT aware neighbor using a mechanism such as IPND [6], it will improve connectivity, if that node announces its neighbor in the DHT.

- A node might be behind a NAT or firewall, which precludes it from being reached directly or announce itself in the DHT. In this case its EID might be announced by another node in the same network, which is reachable from the internet.

- A DTN node might be part of a non-IP network such as a IEEE 802.15.4 based wireless sensor network running uDTN or an AX.25 based DTN2 network. In this case an IP-capable gateway might want to announce the non-IP based EIDs into the DHT, which allows for technologically heterogeneous DTN networks to be connected through the internet.

When adding neighbor information to the DHT, the proposed naming service can also be seen as a routing. If DHT nodes announce each other as neighbors reciprocally, a participant can get a good idea of the network structure by recursively querying the DHT for all nodes that some node announces as neighbor. Sometimes it may not be desired that a single DHT node exposes a whole network by announcing all members. Therefore, apart from disabling the neighbor announcement completely, in IBR-DTN we support a new IPND service field, which allows a node to opt-out from the announcement process.

It can be argued, whether the naming approach described here augments or replaces common DTN routing protocols in the internet. The version shipped with IBR-DTN can be used in both ways: Without using other routing protocols, the implementation can be configured to add static routes to not-yet known neighbors of found nodes on the fly, e.g. once a node has been returned by the DHT and information queried, the node and its neighbors are immediately reachable. When used in combination with some sort of routing such as PRoPHET, a node returned by the DHT will be announced as new DTN neighbor using IBR-DTN's messaging system, which will trigger the routing module to exchange routing messages with the newly found node.

2.4 Security Considerations

As explained in section 2.1, in case of conflicting entries the DHT will store all entries. This makes spoofing a node very easy. However, as the BP does not specify any structure for the EIDs and since we do not want to introduce an unnecessary central point to the system, the uniqueness and authenticity of entries can not be guaranteed. In fact, according to the BP it is quite valid (and to be expected) that many nodes with the EID *dtn://test.dtn* exist.

This is however, not a problem. For applications where tighter security is needed the BP Security Extensions [11] can be used to verify the authenticity of the communication partner when establishing the link. In this case spoofing nodes can only lead to longer delays, while a node is checking all prospective communication partners. This might be abused for a DoS attack, by injecting many conflicting entries into the DHT, however in an open public system there are always suitable vectors for a DoS attack. Also, using the DHT does not preclude the daemon's administrator from configuring static routes to important communication partners.

3. IMPLEMENTATION

We implemented the extended BT DHT as a self-contained open sourced library. The library is written in ANSI C and is based on the lightweight, platform independent, widespread DHT implementation of the Transmission BT client. The library contains utility functions for bootstrapping and extended logging. Example programs for the announcement and querying of keys are provided with the library. An IBR-DTN module is used to interface between the DHT library and the DTN daemon. It should be trivial to integrate the DHT library with other BP implementations such as DTN2.

When a DHT node starts up it has to connect itself to the DHT. To do this, it needs to know a few initial DHT nodes to connect to. Once connected to any DHT member, a node can gradually learn about the structure of the DHT by forwarding messages and populating its neighbor table.

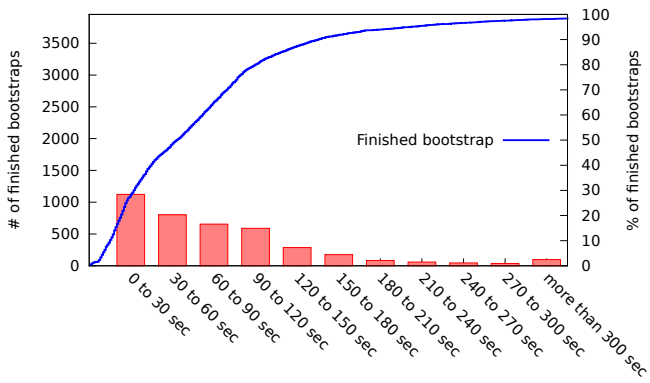


Figure 2: Bootstrap times

The dtndht library provides several ways to learn about a few initial nodes to connect to. Firstly, the addresses of a few well-known DHT member nodes can be configured statically. Additionally we extended IBR-DTN IPND discovery to include a special service field indicating that a node is DHT capable. Such locally detected DHT member nodes can be given to the library as new DHT contacts. The simplest way to bootstrap the DHT is using DNS: The dtndht library allows bootstrapping by making a DNS query to dtndht.ibr.cs.tu-bs.de, which will provide one or more IP and port pairs of DHT member nodes. Additionally the library can be instructed to reload a file containing neighbors stored from the last run. However, if a node wishes to keep on using its neighbor tables upon restart, it is crucial, that it will not change its node id, as this determines its position in the DHT according to the employed XOR distance metric. Also, if the stored neighborhood is very old, it is to be expected that many nodes will already be offline, and thus slowing down the bootstrap process.

To fill its routing table faster, upon restart the library has the ability to perform several searches for random hashes. As these queries are routed through the DHT, the node will learn more about the DHT's structure. The DNS query method is the most easiest and common method to bootstrap DHT nodes. However, the other methods may be faster by providing more initial nodes.

4. EVALUATION

In the evaluation we looked at the performance and reliability of storing information in the DHT. For the evaluation we always assume the worst case scenarios: Each test has been done with a new random DHT id and a new UDP port to prevent contacting to or being contacted by previously known DHT nodes. Also the boot strapping method is a basic DNS request, which means for each run it will take some time populating the routing tables.

4.1 Boot Strapping

We measured the DNS bootstrapping method to show how long the process takes without any prior knowledge of the existing DHT. The duration of the bootstrapping process is the time between initializing the library and knowing at least 8 DHT nodes. After performing 3954 test runs (figure 2) we see that in more than 95% of the cases, the bootstrap process is finished after at most 4 minutes. For nearly 50%

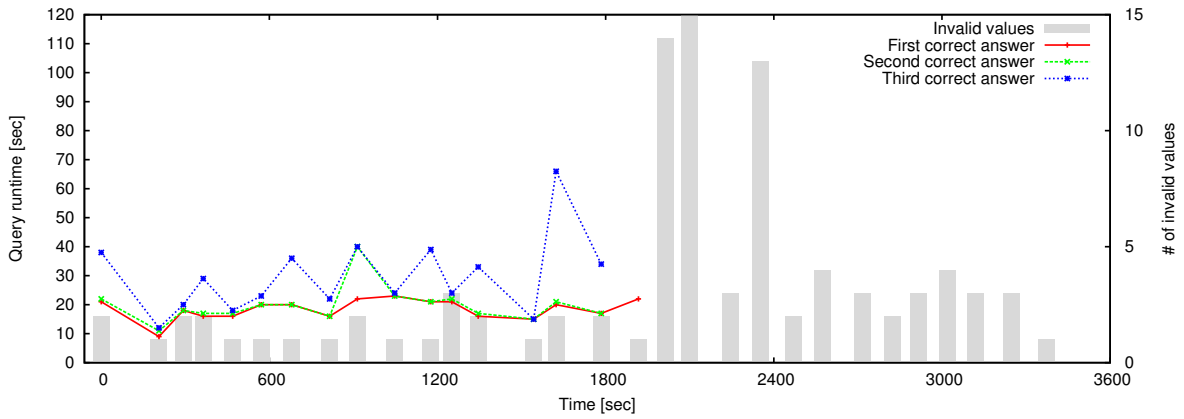


Figure 3: Fast lookup, correct lifetime

Correct Answers	Occurrences	Ratio
8	3	0.6%
7	29	5.8%
6	45	9%
5	69	13.8%
4	94	18.8%
3	102	20.4%
2	93	18.6%
1	42	8.4%
0	23	4.6%

Table 1: Number of correct answers in 500 lookups

of the test cases, the bootstrap process has finished after 60 seconds. The median duration was 63 seconds and the maximum observed duration was 1797 seconds.

4.2 Lookup Success

In this test we want to determine how reliably we can retrieve announced entries from the DHT. To make the different tests comparable, for each lookup we performed the following sequence

1. Generate a random EID
2. Start a new DHT member node (using a new id and port)
3. After the bootstrap is finished, announce the EID
4. After the announcement finished, shut down the DHT node
5. Start a new DHT node (using a different id and UDP port)
6. After the bootstrap finishes, start a lookup for the EID generated in step 1

The DHT nodes are bootstrapped with a DNS query and with differing ids in steps 2 and 5. This test setup prevents DHT nodes to benefit from a good neighborhood from earlier runs. The announcement will store the SHA-1 of the given EID on 8 DHT nodes. Ideally, the same 8 DHT nodes should be found by the new DHT node making the lookup. But in reality, most of the time not all replicas are located.

We executed the test sequence lined out above 500 times and listed the occurrence of correct answers on table 1. Only in 0.6% of all runs all 8 replicas are returned. In more than 95% of the runs, at least one entry is found. In nearly 5% of the lookups, the announced EID hasn't been found. However, in a real deployment this result is not very problematic: A DHT node that is announcing an EID will periodically do so to refresh the entries. This will for example generate new replicas at other nodes, if some member nodes already left the network. Also, a longer running node is usually better connected in the DHT and as such might find better neighbors to store the data. The same applies to the querying node: As long as it has some bundle for a EID for which it has not yet located an appropriate DTN neighbor, it will repeat the lookup.

So in real deployments we observed, that for a small fractions of queries it takes a few minutes longer to find a published entry in the DHT, but eventually the correct entry will be found. Figures 3 and 4 give some insight into the general duration of lookups. The graphs show, how long it takes until a query for a previously published key yields the first, second and third correct answer. Usually most queries can be answered in less than a minute, but in some situations it is harder to find an entry, and especially finding a second or third correct answer can take a considerably longer time. The figures also show the number of invalid results returned. Invalid results are entries which we get for the queried key from the DHT which are different from the values we published. We take a closer look at the wrong results in section 4.5.

4.3 Entry Lifetime

As per [1] all information published in the DHT should be deleted after 30 minutes. This prevents nodes from providing outdated information. Whether this actually happens depends on the individual DHT implementations of the participating nodes.

To measure the lifetime of an entry, we stored a random EID to the DHT as explained in section 4.2 and then performed continuous lookups. For most of the entries we found a lifetime of 30 minutes as can be seen in the example from figure 3. In some of our experiments we found, there seem to be some DHT implementations out there which are saving entries for a longer period of time. This can be seen in the test results depicted in figure 4, where even 47 minutes after the last announcement the entry is returned.

To get a better understanding of the average lifetime of

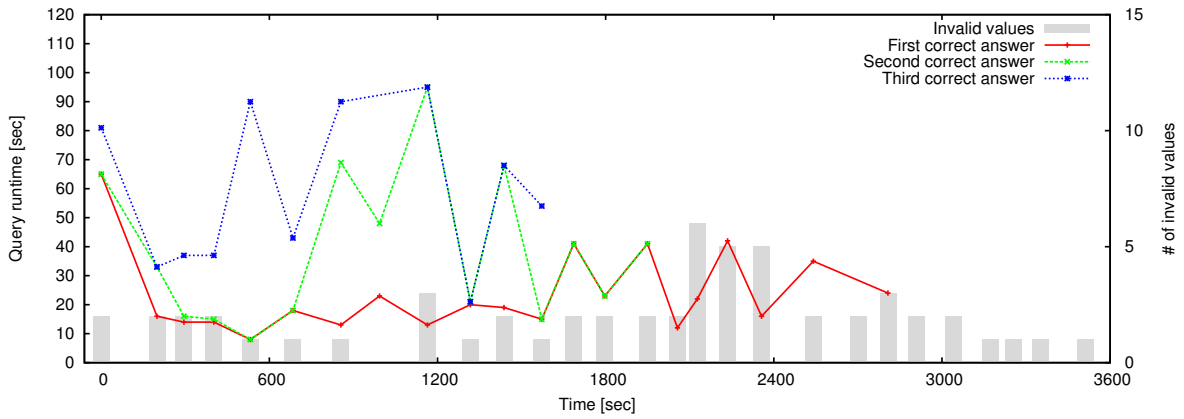


Figure 4: Slower lookup, extended lifetime

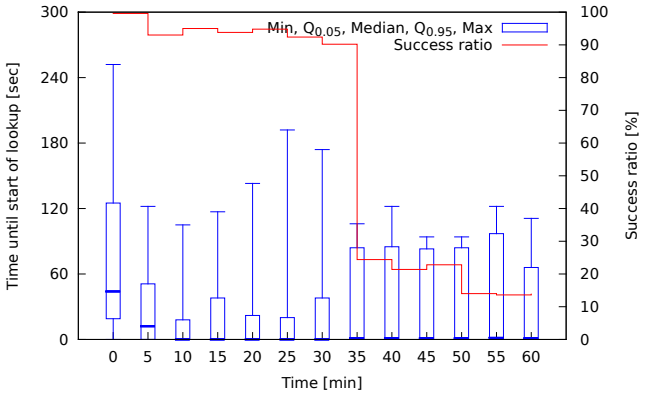


Figure 5: Average entry lifetimes for 500 random EIDs

Duration	Occurrences	Ratio
0 to 155.65 sec	4	0.40%
155.65 to 191.3 sec	20	2.00%
191.3 to 226.95 sec	55	5.50%
226.95 to 262.6 sec	151	15.10%
262.6 to 298.25 sec	282	28.20%
298.25 to 333.9 sec	269	26.90%
333.9 to 369.55 sec	147	14.70%
369.55 to 405.2 sec	54	5.40%
405.2 to 440.85 sec	14	1.40%
440.85 to 476.5 sec	1	0.10%
476.5 to 512.15 sec	1	0.10%
512.15 to 547.8 sec	1	0.10%
more than 547.8 sec	1	0.10%

Table 2: Duration of 1000 announcements

entries in the BT DHT we announced 500 EIDs and performed regular lookups every 5 minutes. The results of this tests can be seen in figure 5. The success ratio shows, how many lookups have been successful. We define successful as returning the correct entry at least once. The blue candle sticks show the median time until the first result for a successful query arrives, the interval in which 90% of the successful queries get their first results as well as the minimum and maximum duration until receiving the first result. It can be seen, that in accordance with the BT DHT specification, after about 30 minutes, the success ratio is in sharp decline and falls to about 25%. This demonstrates that the majority of DHT nodes honour the requirement to drop data 30 min after the last refresh, but there are also a lot of implementations out there which will store values for a longer time.

4.4 Announcement

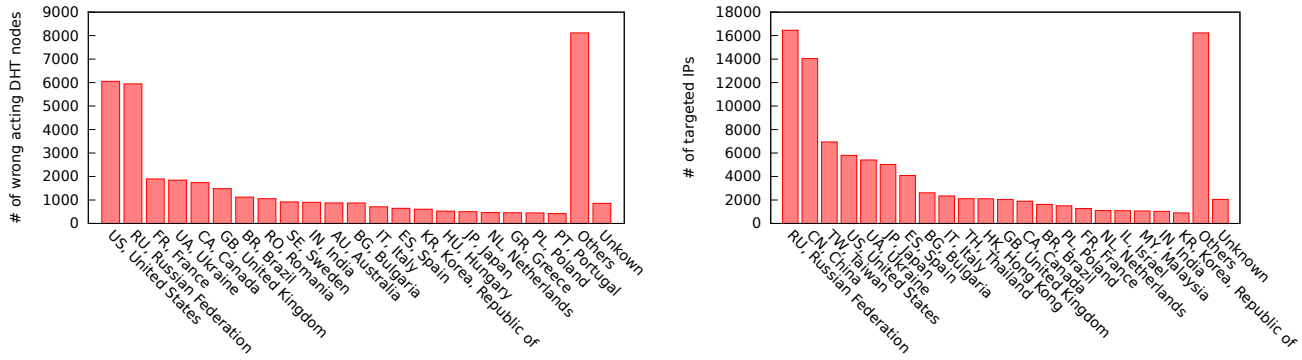
An announcement has two steps: First there is a lookup (see 4.2) to get in contact with the DHT nodes which are responsible for storing the key that should be announced. After the lookup phase the 8 nodes which are nearest to the key will be contacted by a RPC and asked to store the key.

We evaluated how long it takes to announce a key. We always measured the first announcement of a given key, which means the time to lookup the 8 responsible nodes will be high. Upon subsequent refreshes, because the relevant nodes are already in the neighbor tables, it is to be expected that the lookup phase will be much shorter just as demonstrated in the lookup evaluation in section 4.2. We measured announce times for a duration of 8 hours on 2 different days. We found the announce time to be independent of the day or time of day. Therefore table 2 shows the distribution of announcement times from both days. The results show, that the first announcement of an EID takes usually about 5 minutes, with a variance of 2 minutes.

4.5 Invalid Contact Informations

The BT DHT contains a lot of nodes, which are not acting like they should. We observed many nodes, which are sending wrong answers on lookups. We define a wrong answer, as receiving IP addresses for an announced hash, that are different from the one we published. Due to SHA-1's big keyspace it is highly unlikely that this is caused by a random hash collision. This can be seen in figure 3 and 4 where we also plotted the number of invalid results. It seems, many nodes answer with invalid contact information even when querying non-existing SHA-1 keys.

To provoke this behaviour, we generated over 10000 random SHA-1 keys and searched for them on the DHT. If all DHT members are working correctly, it is to be expected that we do not receive any answers, as hitting a specific ex-



(a) Location of the responding DHT node

(b) Location of returned results

Figure 6: Bogus answers in the BitTorrent DHT

isting SHA-1 key by randomly generating it is highly unlikely (a chance of 1 to 2^{160}). However, on average 3.51 nodes answered our searches, resulting in about 9.79 invalid contacts per lookup. We used a geoip database to get an overview from which countries the wrongly acting nodes are replying (figure 6(a)), and to which country the returned IPs point (figure 6(b)). We can not determine, whether these wrong answers stem from broken clients, or if they are intentional. As BT clients will try to establish a BT connection to returned IPs, a rationale for returning bogus values when receiving any query can be the intention of misusing the BT swarm to perform a DDoS attack against a specific target.

5. CONCLUSIONS

We presented and evaluated a concept and implementation to utilize the BT DHT service to provide a naming service and routing assistance to BP based DTNs. This overcomes a significant shortcoming in current BP specifications and implementations and enables global DTN connectivity based on DTN EIDs without the need for further configuration. Using an existing DHT instead of rolling out an application specific one has the advantage that the DHT network will be much more available and less affected by network churn than a small custom DHT only composed of DTN nodes. Because our DHT implementation is fully compatible with the BT DHT it will not negatively affect the operation of the BT DHT but strengthen it as a valid member. We have shown that, even though the BT DHT contains a significant amount of bad nodes and bogus data, our system can retrieve correct information from the DHT in a reliable and timely manner. As of IBR-DTN version 0.8 the DHT service ships as standard part of the release. The implementation of the DHT functionality in a self contained library makes it easy to port this mechanism to other DTN implementations such as DTN2.

Acknowledgments

This work has been supported by the NTH School for IT Ecosystems.

6. REFERENCES

[1] Andrew Loewenstern. BEP 5: DHT Protocol. `http://bittorrent.org/beps/bep_0005.html`, 2008.

[2] Bram Cohen. BEP 3: The BitTorrent Protocol Specification. `http://bittorrent.org/beps/bep_0003.html`, 2008.

[3] C. Caini, P. Cornice, R. Firrincieli, M. Livini, and D. Lacamera. DTN meets smartphones: Future prospects and tests. In *5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC), Modena, Italy*, 2010.

[4] S. A. Crosby and D. S. Wallach. An Analysis of Bittorrent's Two Kademlia-based DHTs. *Department of Computer Science, Rice University*, 2007.

[5] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), Sept. 2001. Updated by RFCs 4634, 6234.

[6] D. Ellard and D. Brown. DTN IP Neighbor Discovery (IPND). *Internet-Draft*, 2010.

[7] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin / Heidelberg, 2002.

[8] O. Ohneiser. DHT basiertes Naming und Service Discovery für DTNs. Master's thesis, Institute for Operating Systems and Computer Networks, Technische Universität Braunschweig, 2011.

[9] J. Ott. Application Protocol Design Considerations for a Mobile Internet. In *Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture (MobiArch), San Francisco, USA*, 2006.

[10] K. Scott and S. Burleigh. RFC5050 - Bundle Protocol Specification. *RFC*, 2007.

[11] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle Security Protocol Specification. RFC 6257 (Experimental), May 2011.

[12] O. P. Waldhorst. On Overlay-Based Addressing and Routing in Heterogeneous Future Networks. In *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on, Zurich, Switzerland*, pages 1–8, 2010.