NASDI - Naming and Service Discovery for DTNs in Internet Backbones

Sebastian Schildt, Wolf-Bastian Pöttner, Oliver Ohneiser, and Lars Wolf

Institute of Operating Systems and Computer Networks,
Technische Universität Braunschweig, Germany
[schildt|poettner|ohneiser|wolf]@ibr.cs.tu-bs.de

Summary. Delay Tolerant Networking (DTN) approaches based on the Bundle Protocol are commonly used within mobile IP based networks. Instead of being isolated applications, the Internet is often used to provide additional services or to route through other parts of the DTN network. A major drawback is that current DTN routing and discovery protocols are not generally applicable in the Internet as there is no common protocol to resolve DTN node names to convergence layer addresses outside a local network.
We present NASDI, an approach based on Distributed Hash Tables which can support naming, routing, notifications and service discovery in a heterogeneous DTN linked by the Internet. We present the architecture and initial evaluations of a NASDI prototype system we built for the IBR-DTN software.

1 Introduction

Delay Tolerant Networking (DTN) approaches replace the end-to-end semantics of common protocols such as IP with a hop-by-hop store and forward architecture [1]. Originally devised for interplanetary networks where nodes might see each other only occasionally, this approach has also been widely applied for ad-hoc networks with high mobility such as vehicular networks [2]. The Bundle Protocol [3] is a standardized widely used DTN protocol. It supports optional end-to-end acknowledgements on top of the hop-by-hop approach. In fact, the Bundle Protocol 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 former, while in addition it is able to deal with disruptions of the network. This leads to the use of the Bundle Protocol in networks with mobile IP devices such as smartphones, which are regularly connected to the Internet [4] [5].

However, the Bundle Protocol 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 nodename. 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 network discovery, both of which are not applicable in the Internet.

In fact the "DTNBone" [6], which is a collection of DTN nodes operated in the Internet by different institutions for DTN testing purposes, consists mainly of a webpage. This page contains (often inaccurate) information about which node can be reached at which IP address using which transport protocols. A DTN administrator who wants to connect with the DTNBone takes this information in order to configure a static route within his DTN server.

Therefore, we propose NASDI, a mechanism that allows for naming, service discovery and routing between DTN nodes operated in a large backbone network such as the Internet. NASDI is able to integrate peripheral networks and nodes which are only intermittently connected to the backbone and it allows nodes which are unaware of the approach to take advantage of the benefits. As DTNs contain intermittently connected nodes that may enter or leave the network at any time, it is our goal to be notified about such events even if the node in question is not located in our direct network neighborhood. To facilitate this NASDI offers an asynchronous notification mechanism. NASDI is based on Distributed Hash Tables (DHT) and is specifically adapted to the needs of DTNs.

In Section 2 we present some related work. Section 3 describes our system architecture followed by Section 4 describing the proposed algorithms. Section 5 introduces our implementation. Finally, in section 6 we wrap up our experiences with NASDI and line out future work.

2 Related work

For LAN or small-scale ad-hoc DTNs there exists a standard mechanism to detect neighboring nodes and services: IP Neighbor Discovery (IPND) [7]. IPND works by regularly broadcasting beacons and thus it does not scale to the Internet. In [8] Waldhorst sketches Arriba, a general architecture for routing in overlay networks spanning heterogeneous technologies based on generic node ids. That work focusses on routing, but it does not specify how to create unique node ids and assign them to device names or underlay addresses. Closely related to the problem of node discovery is the problem of routing: Most general routing protocols proposed for DTNs are designed for ad-hoc type scenarios and as such are often variants of flooding like Epidemic routing [9] or PROPHET [10]. Other approaches exploit domain specific knowledge [11] and are thus not generally applicable.

Earlier DTN specifications included the concept of "regions". DTN regions were a hierarchical naming concept for DTN nodes based on their network affiliation [12]. Current DTN specifications have abandoned this concept in favour of a more flexible flat URI based namespace. The idea is, that different networks can be identified by different URI schemes, but generally the usage of URIs is meant to be much more open. The specification suggests things such as "expressions of interest" based URIs [1].

To allow hosts to find out the network layer address for a host name in the Internet the Domain Name System [13] is used. In addition to a number of shortcomings of traditional DNS in the Internet [14], it is also not optimal in a DTN. DNS systems are partitioned assuming a hierarchy of hostnames, which does not exist in the flat URI based namespace of DTNs. Furthermore DNS is not self-organizing but instead it involves significant organizational overhead. Also, DNS assumes that the network of servers is static and rather stable - a property, which can not necessarily be found in DTN. To overcome some of these problems, DDNS [15] has been proposed. It is based on a DHT as data storage and embeds a hierarchical namespace in the flat key space of a hash table. DDNS tries to mimic the behavior of DNS and is destined to be a DNS replacement. Although it is also based on a DHT, it has some significant differences to our approach. It just transfers DNS semantics to a distributed DHT store and thus lacks an asynchronous notification mechanism and support for groups.

As suggested in [4], 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 still providing good lookup performance. Generally DHTs are resilient against node failure, have excellent scalability and support a flat name space. DHT implementations differ in the topology in which they organize participant nodes, and in the metric used to determine which key belongs to which node. One of the earliest well-known DHT is Chord introduced in the seminal paper by Stoica et al. [16]. Chord uses a ring topology. In addition to successor information, each node has a routing table which enables $O(log(n))$ communication complexity while searching for a key. Among other well known DHT variants are Pastry [17], which tries to exploit local neighborhood information and CAN [18] which organizes data in a d-dimensional grid. A more recent DHT is Kademlia [19] which is widely used in the EMule and BitTorrent P2P networks. Thus, from all DHTs Kademlia has best proven the DHT's alleged scalability and performance in real world scenarios [20]. For a more thorough DHT introduction see [21].

As lined out above, one of NASDI's goals is to provide an asynchronous event notification mechanism. Several papers about publish-subscribe solutions based on DHTs have been published, such as the SCRIBE [22] system. It is based on Pastry [17] and allows topic-based subscriptions. The XEvent [23] publish-subscribe system is based on Bamboo [24] and allows either topic-based or event-content-based subscription using XPath expressions as filter. Both approaches target networks with a large number of subscribers and events and are therefore focused on strategies to efficiently and reliably deliver these events. Our use case has only a limited number of events that does not need such sophisticated structure. In addition, neither SCRIBE nor XEvent support the notion of asynchronous, cached events that will be delivered whenever a node rejoins the network.

3 System architecture

We propose Nasdi, a distributed system that can provide naming, routing and service discovery for Internet connected DTNs. Nasdi's goal is to help connecting to other DTN nodes from which only the name is known. As we will show, Nasdi does not necessarily find the destination itself, but possibly only a suitable next hop that can be used to route to the destination. Due to the various, sometimes very application specific, routing protocols available for DTNs, unlike [8], Nasdi does not try to replace former routing protocols in DTNs. It rather augments them by bridging the gap between separate DTN networks connected through the Internet that cannot discover each offer by the conventional link local discovery mechanisms. For example, to reach a so far unknown node in another network, Nasdi provides the connectivity information of a suitable node. This node can then be contacted and sent the bundle. Additionally routing packets for a mechanism such as PRoPHET might be exchanged with the newly discovered node to learn about the network behind it.

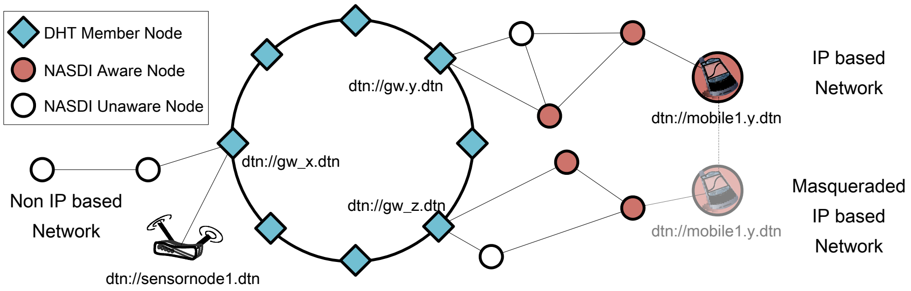


Fig. 1. Scenario Overview.

An overview of a Nasdi System is depicted in Figure 1. Apart from providing naming services to well connected IP capable DTN nodes, the Nasdi architecture also allows to integrate peripheral networks that are not directly reachable or which use a non-IP transport layer for the Bundle protocol. These networks can be transparently proxied by gateway routers as explained in section 3.2. Additionally Nasdi can also improve the connectivity of legacy nodes not supporting Nasdi.

3.1 Assumptions

The following standard DTN terminology is required to understand the information managed by Nasdi: Each DTN Node has one or more EIDs (Endpoint Identifier). EIDs identify a node or a service and have the form of URIs [1]. To connect to an EID within a DTN it is necessary to know which convergence layer, and which convergence layer address can be used to connect with the node. Common convergence layers are TCP [25] and UDP [26] or the Licklider Transport Protocol [27].

Generally, any DHT structure can be used. For the notification we do however assume, that the location of a node in the DHT topology is deterministic, and that this position can be determined by any DHT member. For example a node's location within the DHT's topology can be determined by that nodes name. More specifically, for the notification to work we assume that each DHT member is responsible for a range in keyspace containing its own node id.

As mentioned before the EIDs used in Bundle Protocol [3] have the form of an URI. Commonly the *dtn://* schema is used. An example EID would be *dtn://node1/echo* which identifies the application "echo" on the node "node1". For the remainder of the text we assume that we use the EID's scheme and authority parts [28] as keys for the DHT, i.e. *dtn://node1* represents a node id and is used as key for the DHT. Other schemes might use a different mechanism to derive the DHT keys depending on the scheme's semantics.

3.2 DTN Node roles

When Nasdi is deployed each node can assume different roles in the Nasdi system. The different roles are depicted in figure 1. First, it is to be expected that not all nodes will support Nasdi:

- **Nasdi aware nodes:** Nodes, which implement Nasdi (filled circles and diamonds in figure 1). These nodes can be DHT members, i.e. storing information for the DHT or query the DHT.
- **Nasdi unaware nodes:** Nodes, which do not know about the Nasdi mechanism (empty circles in figure 1). Lots of unaware nodes are to be expected before this approach is widely adopted within the DTN community. Nasdi unaware nodes can still be announced or proxied within the DHT by a DHT member. They cannot, however, query the DHT for information. They can benefit from Nasdi by routing through a DHT member.

Nasdi aware nodes can choose to become a **DHT member** (diamonds in figure 1). A DHT member is responsible for storing data which is assigned to it by the DHT implementation. Additionally it is responsible for regularly refreshing the DHT information of nodes it proxies or announces in the DHT. Therefore a node with high mobility or insufficient network connection might choose not to join the DHT. For nodes which have contact information in the DHT, two kinds of information can be stored in the DHT:

- **Announced nodes:** Announced nodes are nodes, whose convergence layer information is stored within the DHT. The convergence layer information stored in the DHT points to the node itself, i.e. contains its current IP address. A node can announce itself in the DHT if it is a DHT member, or it can ask a DHT member to announce it.
- **Proxied node:** Nodes in networks that are not accessible via IP from the Internet. They need a DTN router in order to participate in the global network.

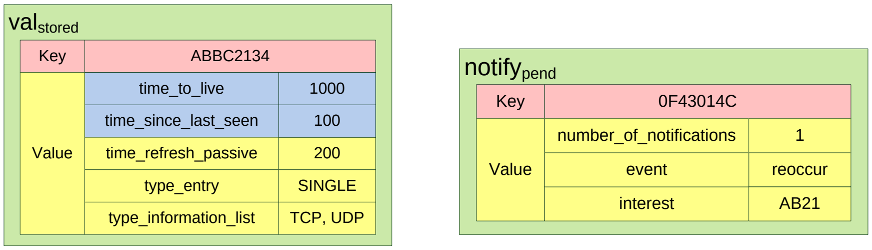
Reasons for unreachable nodes might be firewall or NAT router. Proxied nodes are nodes which have the convergence layer information of another DTN node stored in the DHT. The proxy can also be used as a gateway between different underlying network technologies such as IP and ZigBee. Another rationale behind proxying nodes is, that a node might only be intermittently available which leads to frequent DHT updates and inaccurate information. Instead, storing the convergence layer information of a node that is more likely to be online, allows other DTN members to route bundles in the correct direction while the proxy node is in a good position to relay the information to the target as soon as it is available.

Please note, that a node may be announced directly and at the same is being proxied by others.

4 DHT Information Management

This section details the information stored in the DHT and the steps needed to maintain and query the DHT. We assume that the DHT provides a method DHT_ROUTE(MSG, KEY, VALUE) which delivers a message of type *msg* with content *value* to the node(s) responsible for the partition of key space containing *key*.

4.1 Information stored in the DHT



$\text{val}_{\text{stored}}$

Key	ABBC2134	
Value	time_to_live	1000
	time_since_last_seen	100
	time_refresh_passive	200
	type_entry	SINGLE
	type_information_list	TCP, UDP

$\text{notify}_{\text{pend}}$

Key	0F43014C	
Value	number_of_notifications	1
	event	reoccur
	interest	AB21

(a) *val_stored* single entry

(b) $notify_{pend}$ entry

Fig. 2. Data stored in the DHT

The storage at DHT member nodes is assumed to be a set val_{stored} of $(key, value)$ tuples. For a given key a number of values can be concatenated, which is needed for group management and is also a way to deal with duplicate names: As the Bundle Protocol forces no structure on the EID naming space, it is valid and to be expected that for example multiple `dtn://test` nodes will join the network. Replacing is not an option because we do not want malicious or similarly named nodes to expunge valid entries from the DHT. Security critical applications which need to certify the identity of the other node, can use the

Bundle Protocol Security extension [29]. Therefore, spoofing other nodes in the DHT does not pose an additional risk.

Figures 2a shows an example entry for a stored node. The *key* is the key used as address in the DHT and it is derived from a node's id by SHA-1 hashing it. The *type_entry field* denotes whether this is a group or single node entry (the Bundle Protocol allows the same form of id to be used for either a group or a node). The *type_information_list* contains the IP address and port numbers of the TCP and/or UDP convergence layer for a single node entry, or a list of hashed node ids for a group entry.

Different timers are used to determine when to expunge an entry and to assess the freshness of the data:

- time_to_live (*ttl*): This timer determines how long this entry is considered to be valid. The initial value is determined by the node publishing the key into the DHT. The node that stores this entry decrements it. This entry is measure how long the contact information in this entry is assumed to be valid
- time_since_last_seen (*tls*): Even if the the *ttl* is very high (e.g. for announcing a stationary node), the entry should be refreshed periodically. The *tls* counts how many seconds have past since the entry was last updated.
- time_refresh_passive (*trp*): The *trp* value is constant and indicates how often the publishing nodes intends to refresh the entry. If $tls > trp$ this means that an entry was not refreshed within the expected time. This can indicate that the publishing node has connectivity problems.

4.2 DHT RPCs

On top of the DHT the following message types have to be implemented by DHT members. We assume that a *key* (which is used for DHT routing) and a *value* is associated with each message.

- GET: Standard DHT operation. Returns all entries for *key*. The *value* parameter is ignored for this call.
- STORE: Standard DHT store. Stores *value* associated with *key*. Existing entries for *key* are extended.
- JOIN_GROUP: Allows to augment information stored for a *key* describing a group. Creates a new val_{stored} group entry if the group does not exist so far.
- LEAVE_GROUP: Deletes the node in *value* from the group entry designated by *key*. Does not touch other entries for *key*.
- NOTIFY_REQUEST: Indicates that the node id contained in *value* wants to be notified when a modification is done to key *key*. A user should be able to specify whether this should be a “one-shot” notify, i.e. whether the notification request should be cleared after the notification is fired the first time or whether this should be a permanent notification request. See also section 4.3.
- NOTIFICATION: This message contains a notification for the key *key*. If the current node's id is *key*, the notification is forwarded to the application layer. Otherwise the $(key, value)$ tuple is stored.

The processing of these messages is shown in algorithm 1.

Algorithm 1 Process messages

```
procedure processMessage($msg, key, value$)
    if $msg$ = GET then
        return $\{(k,v) \mid (k,v) \in val_{stored} \wedge k = key\}$
    else if $msg$ = STORE then
        $entry \leftarrow (key, v) \mid (key, v) \in val_{stored}$
        if $entry == \emptyset$ then
            $entry \leftarrow \{(key, value)\}$
        else
            merge_single($entry$, $value$)
        end if
        $val_{stored} \leftarrow val_{stored} \setminus \{(k,v) \mid k = key\}$
        $val_{stored} \leftarrow val_{stored} \cup entry$
    else if $msg$ = JOIN_GROUP then
        $entry \leftarrow (key, v) \mid (key, v) \in val_{stored}$
        if $entry == \emptyset$ then
            $entry \leftarrow \{(key, value)\}$
        else
            merge_group($entry$, $value$)
        end if
        $val_{stored} \leftarrow val_{stored} \setminus \{(k,v) \mid k = key\}$
        $val_{stored} \leftarrow val_{stored} \cup entry$
    else if $msg$ = LEAVE_GROUP then
        $entry \leftarrow (key, v) \mid (key, v) \in val_{stored}$
        if $entry \neq \emptyset$ then
            $entry \leftarrow$ remove_from_group($entry$,$value$)
            $val_{stored} \leftarrow val_{stored} \setminus \{(k,v) \mid k = key\}$
            $val_{stored} \leftarrow val_{stored} \cup entry$
        end if
    else if $msg$ = NOTIFY_REQUEST then
        $notify_{pend} \leftarrow notify_{pend} \cup \{(key, value)\}$
    else if $msg$ = NOTIFICATION then
        if $key = my\_id$ then
            notify_app($value$)
        else                                                  ▷ Indirect notification
            $val_{stored} \leftarrow val_{stored} \cup \{(key, value)\}$
        end if
    end if
    if $msg \neq$ NOTIFY
      and $msg \neq$ NOTIFICATION then
        check_notify($key$)                                   ▷ See algorithm 2
    end if
end procedure
```

4.3 Asynchronous Notification

In a mobility enabled DTN network nodes might not be reachable at all times. This is a standard case in DTN networks and participating nodes keep bundles for an unreachable destination until a suitable next hop becomes available or the bundle expires. However, it is beneficial if the node storing the bundle is notified as soon as the destination becomes available again. This can be implemented using the DHT. To support notifications a DHT member node maintains a second set $notify_{pend}$. A $notify_{pend}$ entry is depicted in figure 2b containing the following items:

- key: This is the DHT key of the node, we want to receive notifications about
- number_of_notifications: How often this notification should fire. Typical values are 1 or ∞. For 1 the event fires once, and afterwards the $notify_{pend}$ entry will be deleted, for ∞ the event will fire every time its triggering conditions are met.
- event: Defines, which kind of event triggers this notification. Possible triggers are the reappearance of a node, the change of any value in the val_stored entry for key, or the change of a specific value.
- interest: The key of the node that wants to receive a notification when this events fires.

To demonstrate the different steps of establishing a notification request and the further processing, we will look at an example from figure 1:

Assume that in Figure 1 the node *dtn://mobile1.y.dtn* is proxied by *dtn://gw.y.dtn*. When *mobile1.y.dtn* becomes unavailable this will be detected by *gw.y.dtn* and the corresponding entry will expire in the DHT. However, foreign nodes might still be sending bundles for *mobile1.y.dtn* to *gw.y.dtn* because they used a cached older entry with a higher *ttl*, or the bundles have been send before the DHT entry expired. Thus *gw.y.dtn* stores a NOTIFICATION_REQUEST for *mobile1.y.dtn* into the DHT, issuing the DHT command

```
DHT_ROUTE(NOTIFY_REQUEST, dtn://mobile1.y.dtn, dtn://gw.y.dtn)
```

This request will be routed to the same node that is responsible for the key *dtn://mobile1.y.dtn* in the DHT keyspace. Whenever a DHT member receives a call that creates or modifies tuples in its val_{stored} it will check whether there are any notification requests pending for the modified key (see algorithm 2). In our example *mobile1.y.dtn* joins another network and gets itself proxied by *dtn://gw_z.dtn*. This means *dtn://gw_z.dtn* will issue a STORE to the DHT:

```
DHT_ROUTE(STORE, dtn://mobile1.y.dtn, conv_layer(dtn://gw_z.dtn))
```

The node responsible for the key *dtn://mobile1.y.dtn* will check its pending notifications for this key and finds *gw.y.dtn*. Instead of directly trying to contact *gw.y.dtn* it will use the DHT:

```
DHT_ROUTE(NOTIFICATION, dtn://gw.y.dtn, conv_layer(dtn://gw_z.dtn))
```

If *gw.y.dtn* is online, this has the same effect as contacting *gw.y.dtn* directly, because we assumed that we use a DHT where each node is responsible for a range in keyspace containing its own node id (see sect. 3.1). If *gw.y.dtn* is currently not available, the notification is stored on another node currently responsible for the key *dtn://gw.y.dtn*. Once *gw.y.dtn* becomes available again and rejoins the DHT, depending on the DHT, the mechanism of the underlying DHT will hand over the data for its chunk of the keyspace, including the notification. This ensures, that receivers are notified as early as possible.

Algorithm 2 Check and transmit pending notifications

Check whether there are pending notifications for EID

```
procedure CHECK_NOTIFY($EID$)
    $targets \leftarrow \{target \mid (EID, target) \in notify_{pend}\}$
    $data \leftarrow \{(k, v) \mid (k, v) \in val_{stored} \wedge k = EID\}$
    for all $target$ in $targets$ do
        DHT_ROUTE(NOTIFICATION,$target$,$(EID, data)$)
    end for
end procedure
```

5 Implementation

While Nasdi is not a routing protocol in a strict sense, it could be implemented as such using the routing module interface of DTN2 [31]. For IBR-DTN [30] a new discovery module is the best choice for integration. IBR-DTN allows different submodules to plug into its event-based core. DTN2 offers an XML based interface for external routing implementations. In our implementation we opted to use the Maidsafe library[1] which provides a Kademlia implementation including NAT traversal capabilities. The Nasdi implementation is an external program using Maidsafe which communicates via TCP IP with a new IBR DTN discovery module that acts as a wrapper for the external Nasdi implementation. This setup allows for great flexibility while developing Nasdi and should make it relatively easy to connect Nasdi to DTN2's external interfaces later.

5.1 DHT functionality tests

While the performance of large scale Kademlia deployments has already been examined, e.g. in [32] we performed some small scale tests, to verify that the Nasdi implementation is performing as expected. We used virtual machines running instances of Nasdi and IBR-DTN. The first Nasdi instance is always started standalone, while the following instance get one of the running instances as booststrap partner.

Lookup Test Figure 3 shows the time to lookup a value in the DHT. The diagram includes the min max and median values as well as the $Q_{0.25}$ and $Q_{0.75}$ quantiles. We modifed the number of Nasdi nodes and the amount of tuples stored in the DHT. When the number of stored elements is increased from 1 to 1000 the average response time goes up from ~ 70 ms to ~ 90 ms, which shows the additional processing overhead in the Nasdi instances. The variance for each measurement is due to the fact, that the DHT structure is different between runs, so that the responding Maidsafe instance might be nearer or further away. The rightmost plot shows the situation, when a node can answer the query from its local storage without the need to contact other DHT members.

[1] `http:/code.google.com/p/maidsafe-dht`

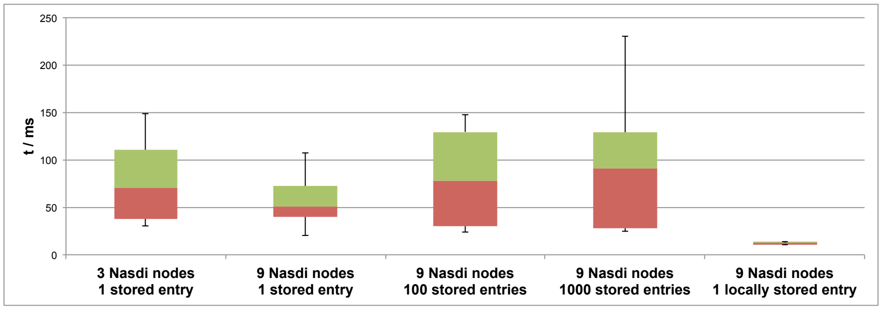


Fig. 3. Lookup times

Notification Delay Test For this test we used 9 NASDI instances. The IBR-DTN node *node1* was started sending a bundle to *node2*, which was currently not available. This leads NASDI to store a notification request. Subsequently we started IBR DTN node *node2*. The NASDI instance for *node2* announces its contact information in the DHT, which leads to a notification being dispatched to *node1*, which in turn connects to *node2*, delivering the stored bundle. We measured the time between starting of *node2* and the reception of the bundle. This took around 3 seconds. A breakdown of the used time can be seen in figure 4. As can be seen in this case the notification itself is nearly instant, while the biggest amount of time is spent in the IBR-DTN daemon getting the cached bundle from storage and preparing it for transmission.

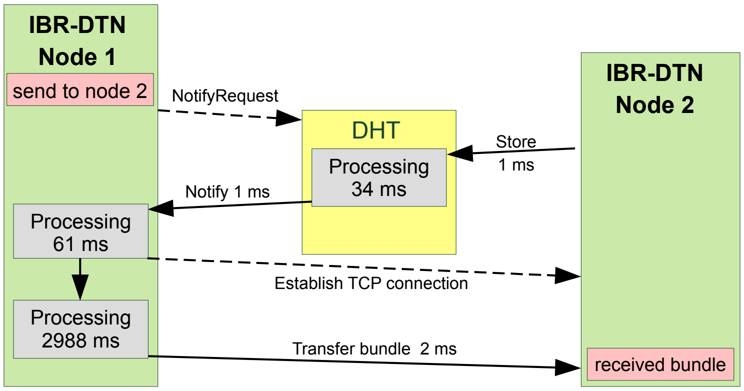


Fig. 4. Notification latencies

6 Conclusions and Future Work

We presented NASDI, an approach that allows DTN nodes connected to the Internet to efficiently store and retrieve convergence layer information of other DTN nodes in a distributed manner. Nasdi supports building groups and integrates an asynchronous event notification mechanism. While Nasdi's DHT member nodes should be located in the Internet and should be chosen in such a way that a long uptime can be expected, peripheral, possibly non-IP, intermittently connected networks can benefit from Nasdi whenever they have a connection to the backbone. Those peripheral networks can be transparently bridged through the Internet. Nasdi can be easily implemented as a routing module for widely used DTN implementations. Deploying Nasdi is simple, because it can coexist with Nasdi unaware nodes and still provide its benefits.

During the implementation and evaluation of Nasdi we identified some areas for further improvement: We think that the services offered by Nasdi can be very beneficial when they are used on a large scale in DTN implementations. To reach this goal, Nasdi functionality should be shipped and enabled by default in Bundle Protocol implementations. We choose Maidsafe as basis for our DHT, which proved to be a very versatile library. However, its huge size and various dependencies may be a negative point when trying to integrate it with IBR-DTN or DTN2. This is especially a problem for IBR-DTN which is optimized for embedded devices. Therefore, we are currently looking into more lightweight DHT implementations.

We have only been able to perform tests with 9 nodes. While this shows the systems works as advertised, it makes it hard to predict how the system would perform with thousands of nodes. This also reveals a problem when the system is deployed initially: If there are only a few nodes operating in the Nasdi DHT the overall resilience and reliability of the system might be less than in our controlled lab experiments. To solve this, we are currently looking into the possibility of leveraging the DHT subsystems of popular filesharing applications such as BitTorrent or the eMule network for Nasdi. This means having less control over the DHT implementation, which could mean that some Nasdi functionalities can not be implemented as efficiently or fully featured as outlined in this paper. On the other hand at any given time thousands of nodes will be online and available in the DHT, which should make the overall system very robust.

Acknowledgments

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

References

1. S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "RFC4838 - Delay-Tolerant Networking Architecture," *RFC*, 2007. [Online]. Available: http://tools.ietf.org/pdf/rfc4838.pdf
2. S. Lahde, M. Doering, W.-B. Pöttner, G. Lammert, and L. Wolf, "A practical analysis of communication characteristics for mobile and distributed pollution measurements on the road," *Wireless Communications and Mobile Computing*, vol. 7, no. 10, pp. 1209–1218, Jan 2007.

3. K. Scott and S. Burleigh, "RFC5050 - Bundle Protocol Specification," *RFC*, 2007. [Online]. Available: http://tools.ietf.org/pdf/rfc5050.pdf
4. 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)*, 2006.
5. 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)*, 2010.
6. Delay Tolerant Networking Research Group, "DTN-Bone." [Online]. Available: http://www.dtnrg.org/wiki/DtnBone
7. D. Ellard and D. Brown, "DTN IP Neighbor Discovery (IPND)," *Internet-Draft*, 2010. [Online]. Available: http://tools.ietf.org/pdf/draft-irtf-dtnrg-ipnd-01.pdf
8. 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*, 2010, pp. 1–8.
9. A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Duke University, Tech. Rep. CS-200006, May 2000.
10. L. Anders, D. Avri, and S. Olov, "Probabilistic Routing in Intermittently Connected Networks," *SIGMOBILE Mobile Computing and Communication Review*, vol. 7, no. 3, pp. 19–20, Jan 2004.
11. M. Doering, T. Pögel, and L. C. Wolf, "DTN Routing in Urban Public Transport Systems," in *ACM MobiCom 2010 Workshop on Challenged Networks (CHANTS)*, Chicago, USA, Sep 2010.
12. K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, Aug 2003.
13. P. Mockapetris and K. J. Dunlap, "Development of the domain name system," *SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 123–133, 1988.
14. H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, Aug 2004.
15. R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 155–165.
16. I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 17–32, Feb 2003.
17. A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (Middleware)*, Nov 2001.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, Aug 2001.
19. P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin / Heidelberg, 2002, vol. 2429, pp. 53–65.

20. C. Wang, N. Yang, and H. Chen;, “Improving Lookup Performance Based on Kademlia,” in *Proceedings of Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC)*, vol. 1, 2010.
21. R. Steinmetz and K. Wehrle, *Peer-to-peer systems and applications*, R. Steinmetz and K. Wehrle, Eds. Secaucus, NJ, USA: Springer-Verlag New York, Inc., Jan 2005.
22. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, “SCRIBE: A large-scale and decentralized application-level multicast infrastructure,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489 – 1499, 2002.
23. R. Wang, W. Rao, and C. Zhang, “XEvent: An Event Notification System over Distributed Hash Table (DHT) Networks,” *IEEE Intelligent Informatics Bulletin*, vol. 6, no. 2, pp. 19–25, 2006.
24. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling churn in a DHT,” in *Proceedings of the USENIX Annual Technical Conference (USENIX)*, Jun 2004.
25. M. Demmer, U. Berkeley, and J. Ott, “Delay Tolerant Networking TCP Convergence Layer Protocol,” *IETF Draft*, 2008. [Online]. Available: http://tools.ietf.org/pdf/draft-irtf-dtnrg-tcp-clayer-02.pdf
26. H. Kruse and S. Ostermann, “UDP Convergence Layers for the DTN Bundle and LTP Protocols,” *IETF Draft*, 2008. [Online]. Available: http://tools.ietf.org/pdf/draft-irtf-dtnrg-udp-clayer-00.pdf
27. M. Ramadas, S. Burleigh, and S. Farrell, “Licklider Transmission Protocol - Specification,” *Experimental RFC*, 2008. [Online]. Available: http://tools.ietf.org/pdf/rfc5326.pdf
28. T. Berners-Lee, R. Fielding, and L. Masinter, “RFC3986 - Uniform Resource Identifier (URI): Generic Syntax,” *RFC*, 2005. [Online]. Available: http://tools.ietf.org/pdf/rfc3986.pdf
29. S. Symington, S. Farrell, H. Weiss, and P. Lovell, “Bundle Security Protocol Specification,” *IETF Draft*, 2010. [Online]. Available: http://tools.ietf.org/pdf/draft-irtf-dtnrg-bundle-security-17.pdf
30. S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, “IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation,” *Electronic Communications of the EASST*, vol. 37, pp. 1–11, Jan. 2011.
31. “DTN2 implementation.” [Online]. Available: http://sourceforge.net/projects/dtn/
32. Z. Ou, E. Harjula, O. Kassinen, and M. Ylianttila, “Performance evaluation of a Kademlia-based communication-oriented P2P system under churn,” *Computer Networks*, vol. 54, no. 5, pp. 689–705, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.09.022