CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Using a DHT in a Peer to Peer architecture for the Internet of Things
Authors	Tracey, David;Sreenan, Cormac J.
Publication date	2019-04
Original Citation	Tracey, D. and Sreenan, C. (2019) 'Using a DHT in a Peer to Peer Architecture for the Internet of Things', IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland 15-18 April, pp. 560-565. doi: 10.1109/WF-IoT.2019.8767261
Type of publication	Conference item
Link to publisher's version	https://ieeexplore.ieee.org/document/8767261 - 10.1109/WF-IoT.2019.8767261
Rights	© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-10-30 03:43:30
Item downloaded from	https://hdl.handle.net/10468/9653

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Using a DHT in a Peer to Peer Architecture for the Internet of Things

David Tracey, Dept. Of Computer Science,
University College Cork, Cork, Ireland

Cormac Sreenan, Dept. Of Computer Science,
University College Cork, Cork, Ireland

***Abstract*—A challenging aspect of The Internet of Things (IoT) is to provide an architecture that can handle the range of IoT elements ranging from Cloud-based applications to constrained nodes in Wireless Sensor Networks (WSNs). Such an architecture must be scalable, allow seamless operation across networks and devices with little human intervention. This paper describes a set of abstractions and an architecture for the flow of data from sensors to applications supported by a Distributed Hash Table (DHT) and our novel Holistic Peer to Peer (HPP) Application Layer protocol to handle node ids, capabilities, services and sensor data. We show that this architecture can operate in a constrained node by presenting a 'C' implementation running on the Contiki3.0 OS and consider the effectiveness of its use of a DHT and its abstractions.**

Index Terms—Wireless Sensor Networks, IOT, Tuple Space, DHT, Data Model, OMA LWM2M.,

I. INTRODUCTION

IoT can be described as allowing the easier integration of the physical world with the Internet's virtual world [1]. IoT is a distributed system comprised of individually addressed nodes, including constrained nodes with sensing or actuation capabilities in a Wireless Sensor Network (WSN). The use of IoT is expected to grow in a range of applications, such as environmental monitoring and healthcare. The potential of new applications to take advantage of IoT is limited by the difficulties caused by the heterogeneous nature, constrained computing and memory capabilities of nodes, exacerbated by limited development environments. Also, deployments may be in challenging environments for wireless [2] and may be dedicated to a particular use with proprietary or specialized software/protocols to optimize an aspect like lifetime.

By making sensor data available over the Internet, IoT allows Cloud services and Big Data approaches to store and analyze it in a scalable manner, supported by Cloud provider tools and Fog/Edge Computing [3]. A key consideration is how to seamlessly find, store and analyze increasing amounts and variety of IoT data on such Cloud services and on constrained devices, so that a range of application software can be developed. One approach is to allow software to understand data from sensors/actuators in the way people using browsers understand information on the Web [4] and use defined data models for sensors/actuators, e.g. IPSO Smart Objects [5], accessible using a Client/Server approach as in the Constrained Application Protocol (CoAP) or the publish/subscribe model of MQTT.

This requires being able to scale the technology down to resource-constrained devices and to scale it up to billions of devices [6]. This will require seamless interoperability and sets of abstractions to support that. In this context, Peer-to-Peer (P2P) approaches offer a number of potential benefits, such as scalability, a low barrier to entry, greater autonomy, and robustness. These features have been demonstrated in systems such as BitTorrent [7].

We previously presented an architecture that uses a set of service-based abstractions and a tuple space based data store for local and remote data [8], with the novel CacheL algorithm using leases [9]. We termed this architecture holistic as it considers the varied roles in an IoT system, from constrained devices to Cloud services.

This paper presents the detail of a Holistic Peer-to-Peer (HPP) application layer protocol we have added and its support for the data-centric approach in our Architecture. This paper also considers our contribution of an application overlay that can span the WSN and services over the Internet using a Distributed Hash Table DHT, based on Kademlia [10] to find nodes and allow new nodes to join by knowing only the address of a node in the overlay. This DHT is also used to allow an innovative use of forming groups of data or nodes with an associated identifier, similarly to an info-hash in BitTorrent. We believe this P2P approach will allow IoT to move beyond isolated islands of data to nodes and services that are more easily deployed, developed and integrated, e.g for the healthcare scenario outlined in [11]. This also applies at the edges of the Internet, making it suitable for Fog Computing. A prototype implementation is presented on the Contiki3.0 OS [12] and Linux servers that demonstrate the overlay across the WSN to external services. Its shared codebase and abstractions also helped to make development and testing easier.

The rest of this paper is organized as follows. Section II presents prior work on P2P and DHTs and Section III gives an overview of our architecture. Sections IV and V present and review a prototype implementation of the Holistic Peer to Peer (HPP) protocol and DHT. It concludes in section VI.

II. P2P OVERVIEW

P2P systems are used for communication, collaboration, computation, distributed storage/databases and file sharing. primarily for music file sharing, e.g. BitTorrent. Freenet [13] is an example of a purely decentralized, self-organizing P2P network designed to hide the origin or destination of files. File sharing P2P systems were driven by advances in hard-disk capacity, processing power and bandwidth availability.

One view considers that a system is P2P if it meets the test "Does it give the nodes at the edges of the network significant autonomy?" [14]. Such a definition including edge nodes makes P2P relevant in the Fog Computing scenario, e.g. Figure 1 from the OpenFog Consortium [15] illustrates the diverse range of devices, services and roles from the edge to the Cloud.

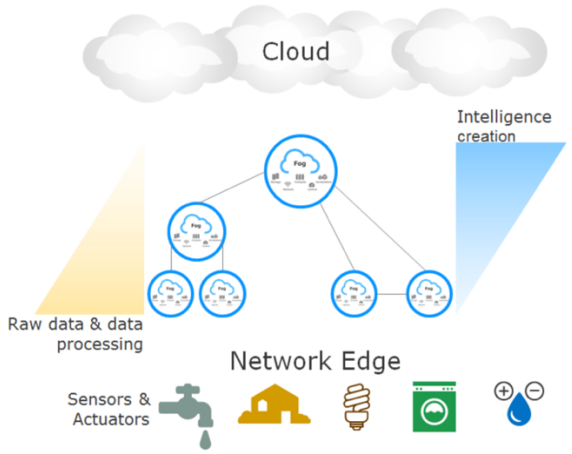


Figure 1 OpenFog Architecture Scenario [15]

A. *Peer to Peer (P2P) in WSNs*

The file-sharing use case is different to the constrained WSN node environment, with its limited storage and bandwidth, but there are characteristics that make P2P suitable for WSN and IoT nodes and services, i.e. scalability, decentralized control, robustness and self-organizing nodes..

One approach to using P2P in a WSN is for it to interact between the WSN and the gateway to an external network. An example [16] views the sensor network as one peer in the P2P network where the gateway represents it in the wider network and also included a Sensor Network Abstraction Layer with P2P protocols to publish available services, to collaborate on tasks, to query all sensor nodes and to search for services using a service discovery protocol. Another approach is to use a P2P Overlay Network [17], which includes the WSN nodes. A P2P overlay network allows applications to identify and send to peers, without requiring knowledge of the underlying network implementation. The P2P overlay topology can also be mapped with the physical topology so that the P2P neighbor is the physically closest node. As pointed out by [17], real deployments often require assigning nodes a globally unique identifier anyway, e.g. to support network management, and so this can be provided by a DHT and not considered an overhead of a DHT. The identifier size can also be reduced in some cases, e.g. by assigning dynamically smaller locally-unique identifiers for use locally within a sensor network. DHT computation is within the capabilities of simple node platforms.

B. *Distributed Hash Tables*

Distributed Hash Tables (DHT) are used in P2P systems to provide efficient routing, without centralized control. DHT's "appear to provide a general-purpose interface for location-independent naming upon which a variety of applications can be built. Furthermore, distributed applications that make use of such an infrastructure inherit robustness, ease of operation, and scaling properties" [18].

A hash-table is suitable for distributed lookup as it only requires that data is identified using unique numeric keys. A data item is inserted into a DHT and found by specifying a unique key for it. Nodes store information about neighboring nodes, forming an overlay network to store and retrieve keys. Given that the purpose of sensor networks is to collect data, the lookup times achievable by DHT's and their scalability suggest that their use in sensor networks is appropriate.

A DHT algorithm must map which node is responsible for storing the data associated with any given key, probably using a hash function. It must also build routing tables holding their node identifiers and forward a lookup(key) to a node (maybe the destination) with a "closer" identifier to that key [18]. The key could be the result of applying a hash function to a file name if storing files and a user retrieves the file using lookup(key) and is returned the node, e.g. its IP address, holding that key's data.

Examples of P2P systems using DHTs include Chord [19] and Pastry [20], which differ in how they build and maintain their routing tables as nodes join and leave. They rely on a somewhat fixed topology to assign data to peers and subsequently look up, e.g. Chord uses a one-dimensional space to assign Ids for both keys and nodes. BitTorrent [7] uses a DHT based on Kademlia [10].

1) *Kademlia*

Kademlia [10] is a P2P system to store and lookup key-value pairs, using 160-bit keys. Each node uses a key for its id. Kademlia defines the distance between two keys as their bitwise exclusive or (XOR). It uses XOR to find the closest peer nodes (those with more common bits in their prefix) and to route queries. Its use of a single routing algorithm differs from Chord or Pastry, where one algorithm is used to get close to the desired identifier and a different one for the final message hops. The symmetric property of XOR allows Kademlia to use information in the queries it receives.

Kademlia nodes keep a list of ⟨IP address, UDP port, Node Id⟩ triples for nodes of distance between 2^i and 2^{i+1} from itself for $0 \leq i < 160$. These lists are termed k-buckets as they grow up to a defined size of k and they are sorted by time last seen. On receiving a message, a node identifier already in the bucket is moved to the list's tail and its times updated and a node identifier not in the bucket is inserted if the bucket is not full. Kademlia uses a set of messages to manage these buckets, i.e. PING, FIND_NODE, FIND VALUE (for a target key identifier) and STORE (a key-value pair). In particular "node lookup" finds the k closest nodes to an identifier using the FIND_NODE in a defined manner, beginning with the initiating node picks α nodes (from its closest non-empty k-bucket) and sending them a FIND_NODE. The lookup finishes when it has received replies from the k closest nodes.

2) *BitTorrent*

BitTorrent [7] is a protocol for distributing static data, primarily files broken up using a SHA-1 hash. A metadata file (torrent) is distributed to peers with a tracker reference, the SHA-1 hashes of all pieces of files and their mapping to files. A swarm is a set of peers distributing the same files. A peer joins a swarm by asking the tracker for a peer list and then it connects to those peers. The tracker can be a central server, holding a list of all peers in the swarm, but such a tracker is a single point of failure and may be a bottleneck for publishers. Trackerless torrents are an alternative, e.g. BitTorrent peers may use a DHT based on Kademlia, which holds the location of peers to download from. The key is the

info-hash (the hash of the metadata), which uniquely identifies a torrent and the value is a peer list of the contact information for peers in the swarm.

BitTorrent extended the messages in Kademlia, i.e. PING, FIND_NODE, GET_PEERS and ANNOUNCE_PEER. It retained the essentials of the node lookup and buckets. BitTorrent keeps only "good" nodes in the routing tables, using a 15 minute period to determine the last seen recency and refreshing buckets unchanged in 15 minutes. A peer becomes part of the distributed tracker by looking up the 8 nodes closest to the info-hash of the torrent and sending them an announce message. Those 8 nodes add the announcing peer to the peer list stored at that info-hash. k is set to 8 as this was considered sufficient to reduce the probability of that number of nodes disappearing between refreshes. This also reduces the overhead of the routing tables and the number of messages exchanged.

III. Holistic Architecture Overview

Our HPP architecture [8] is a decentralised P2P architecture where peers act according to their role without central coordination. This fits the vision of a seamless IoT of nodes that share data, collaborate and act autonomously. A node abstracts the low-level device details and a peer abstracts the connectivity and P2P aspects, e.g. finding peers. A Peer runs on a node and has a set of capabilities to handle HPP messages. A Service has a set of roles (*sink, source, forwarder, store, aggregator, matcher, bootstrap*) and it runs on a peer. The HPP Architecture consists of the layers in Figure 2 and it is flexible enough to run on nodes of different capability.

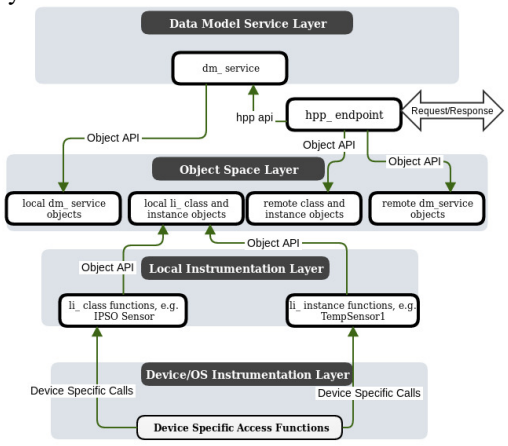


Figure 2 Node Architecture

The Data Model Service Layer provides a high-level abstraction for data to decouple the application from the network and node hardware. It is independent of the data model and has a simple API, supported by the Object Space, which is a data store modeled as a tuple space with leases associated with objects and a simple API. In this context, an object is described by a template provided by the caller and the Object Space is non-prescriptive in what objects it stores, e.g. they could be a set of key value pairs and methods to represent a sensor. It holds the node's data or data it has cached from remote nodes. It includes the CacheL algorithm [9], designed for constrained nodes, which uses leases in its cache replacement policy. Objects can be added to several nodes/groups and do not require explicit removal as they are removed on expiry of their lease. The Local Instrumentation Layer provides methods to map the node's hardware devices such as sensors or the node's OS specific functions, e.g. its OS version, to templates and objects to be stored in the Object Space layer.

Remote healthcare monitoring is an example of the scenario in [11],where health sensors connect to a home gateway, which stores and forwards data to cloud-based services. This contrasts with the approach of separate components and abstractions for the constrained device and the more capable systems, e.g. the Eclipse IoT Stack [21] in Figure 4, and which require mapping those different abstractions.

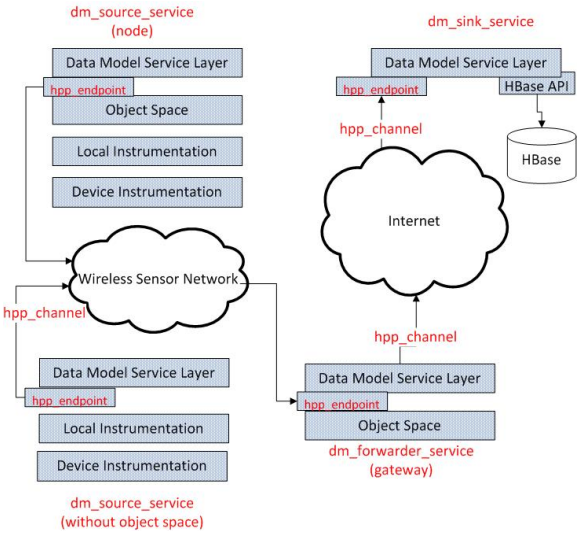


Figure 3 Interaction of Node Services

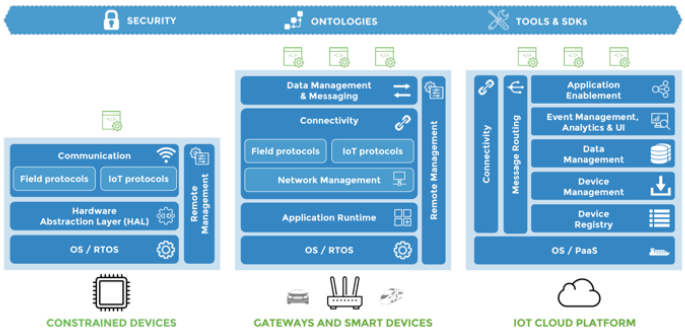


Figure 4 Eclipse IoT Stacks

IV. HPP Protocol Design

The Holistic Peer to Peer Protocol (HPP) is sufficiently simple for low capability devices to use and provides a consistent means to exchange information independently of the underlying network. Two key abstractions are the *hpp channel* (the link between peers hiding the network specifics) and the *hpp endpoint* (represents a communication endpoint consisting of hpp channels). Using *hpp channel* and *hpp*

endpoint means applications do not need to be re-coded for different networks. The key principles in our use of P2P are:

- no fixed placement of data.
- consistent handling of local and remote data with a small set of messages, aligned with the object space and easy to map to RESTFul APIs.
- all peers use the same P2P overlay network according to their capabilities.
- leases per class and instance, set by the source, to allow nodes to cache data and aid data consistency

A DHT with Kademlia k-buckets was chosen to be the basis for the P2P overlay in HPP for the following reasons:

- Kademlia has proven scalability and robustness in its use in BitTorrent.
- Kademlia reduces the number of configuration messages as this information is also carried in messages used to lookup keys.
- Its single XOR based routing algorithm is relatively easy to implement, i.e. no secondary routing tables.
- Kademlia's symmetric routing algorithm facilitates the use of caching.
- Kademlia nodes can use metrics to route queries through low-latency paths.
- Kademlia's use of parallel queries to k nodes to avoid timeout delays from failed nodes.

The key principles and novelty in our DHT are:

- use of Kademlia DHT buckets for node-identifier and xor based routing, initialized with 1 bucket as in BitTorrent (not 160 as in Kademlia) to reduce the memory required.
- use of Kademlia buckets to dynamically group peers, e.g. to group nodes with a sensor type. This uses the DHT to not just hold identifiers to peers, but to hold identifiers to groups that can be joined and retrieved using the same HPP messages that a peer uses to join or be found in an overlay.
- peer longevity in the cache uses a lease set by the source and HPP messages are used to reduce the overhead of co-ordination and information exchange in updating leases and buckets.

B. HPP Messages

Every HPP message consists of a command, message header and an object. Command is one of the allowed commands *Hello*, *Bye, Get, Add, Take, Notify*. The message header consists of defined key value pairs and the object is an encoding of attributes and values, e.g. as key-value pairs. Responses are similar, with the addition of status, to allow shared message handling code and reduce memory use, e.g. caching the data in a get reply uses the same code as an *Add*. There is no *action*, as this is done with an *Add* message with the method arguments specific to an object, e.g. a LED will have a method to set its state.

The message header must contain a *msgId* (unique to sender), a *senderId* and may optionally contain *originatorId*, *hppVersion*, *capabilities, name*, *objectHandle* or *lease*. The *originatorId* and *msgId* do not change as a message is forwarded or replied to, so the original sender can be sent the reply from any node and the reply does not have to use the same path as the request. HPP messages consist of distinct blocks, e.g. for the header. The string encoding uses delimiters, but a binary encoding has lengths in each block.

HPP shares peer capability using a *Hello* message, which can be considered a richer form of Kademlia Ping. *Bye* removes a peer's information ahead of lease expiry. *Get* uses an object handle or can match using specified keys or attributes. A Get can also specify an info-hash or group Id if the object was added to that group id. HPP Get is like Kademlia's find for a node and like its query for data. On getting a reply to a Get Peer message, a peer must check the peer ids as in a Kademlia "find round".

Add adds new classes or instances to a node using its DHT identifier or to an info-hash and an object handle (id) will be returned in the reply. It can also update values. It contains the object to create or update, which may be a template (class) object or an instance. Templates can be referenced by later adds, e.g. to avoid including all attributes. Lease renewal uses an *Add* message with the objectHandle and a new lease value. *Take* removes an object from a node or info-hash using the object handle or a full description of the object. It is not simply a Delete as it returns the object, so it can be added back, simplifying concurrency issues.

Notify has been added for the actuator and alert functionality of devices, similar to observe in CoAP. It tells a peer we are interested in updates to an object for a lease period. That peer will send on any add/take message for that object, maybe piggybacked in the next reply to the interested peer. HPP operates as follows:

- Every Peer must support *Hello* and respond with its identifier (if known) and its capabilities. Hello is deliberately simple to run on very limited nodes.
- Every peer should handle at least a Get for its Peer Instance, containing up to 8 closest node-ids
- Nodes may support any of the other HPP messages, which are the capabilities in its Hello reply.
- A new node joins a HPP overlay network by sending a Hello message to a known peer.
- HPP Get requests for keys are passed from node to node. If a node has the requested data, then it sends that back to the requester, otherwise it forwards the request to the node with the "closest" identifier in its routing table.
- A peer may not accept connections for security reasons, e.g. a source may only connect out.
- A peer will get its closest peers and send a Get to peers of interest to discover the classes and instances on that peer, avoiding the need for a centralised Resource Directory as in CoAP.

A. HPP Message Flows

1) Initialisation - Hello Exchange

A peer sends a *Hello* message to at least one known peer on starting. It contains its encoded capabilities and may contain a senderId. If it does not have a senderId, a receiver with the bootsrap role will check any message credentials and reply with a DHT identifier for that node's identifier.

2) Closest Peer Information

Once the Hello reply has been received, a Node can send a *Get* for the peer's object (identifier shown as zzz) and its k neighbors will be in the reply, as below:

Command	Message Header	Object
Get	msgId=2 senderId=XXX name=peers/peer Lease=60	peerId=zzz

Command	Message Header	Object
reply=Get status=Ok	msgId=2 senderId=XXX name =peers/peer objectHandle=1001	closePeers =yyy, xxx closeAddresses=a.b.c.d:7014, e.f.g.h:7014

3) Data Transfer

This node exchanges HPP messages with known peers, e.g. to *Get* or *Add* classes/instances with sensor readings.

4) Lease Renewal

Lease renewal is required for peer objects and for objects added to a node. The peer object's lease replaces the republishing period in Kademlia. A peer requests a lease and the bootstrap grants it per its policy. If the lease is not renewed and no message is seen within the lease, then the bootstrap tries to refresh the lease by sending a Get to the peer and removes the peer object if it does not reply.

V. Implementation and Evaluation

The implementation was coded in C on Linux and ported to the Contiki 3.0 OS. The Linux implementation allowed the use of advanced debugging and testing tools. Integration with the Contiki erbium-REST implementation [6] allowed accessing objects using CoAP via the Data Model Service layer. The code was run in Contiki's Cooja simulation environment as a WiSMote [22], using an MSP-430 processor with 128KB of Flash Memory and 6KB of SRAM.

The same codebase was able to run on the constrained nodes and more capable Linux nodes, as per the design goal for the architecture. The value of the *hpp_endpoint* and *hpp_channel* abstractions can be seen in the simplicity of the code below, with the endpoint handling the channel initialization, socket listen and message fragmentation. Other functions use *hpp_endpoint* and *hpp_channel* for message exchange and update peer bucket statistics for each message:

```
Rv = hpp_endpoint_check(endpoint_ptr);
if (rv == 0) {
   channel_ptr = hpp_endpoint_accept(endpoint_ptr);
} else if (rv > 0 ) {
   hpp_endpoint_get_messages(endpoint_ptr);
} // timed out with no data, so loop again
```

Contiki and Linux required different implementations for communication handling due to different underlying TCP/IP stacks, socket APIs and event handling. Dynamic memory allocation was easy to use on Linux, but the limited RAM on the WisMote required static memory allocation to avoid runtime heap and stack issues. In spite of these issues, the abstractions for channels, endpoints and the Data Model layer allowed the higher layer HPP message handling and services code to be unchanged in both environments.

Table 1 shows the memory (bytes) used by a node using only HPP and a node that also included the OMA Lightweight Machine to Machine (LWM2M) and CoAP engines. Statically defined structures were used for connections, channels (8), messages (1 per channel), objects (20), peers and buckets. It shows that our architecture meets the requirement to run on resource constrained nodes, with memory use equivalent to CoAP and LWM2M. The node in the table included local LED and Temperature Sensor IPSO instances.

	HPP Only (Text/Data)	HPP +IPSO/LWM2M (Text/Data)
HPP Libraries	16961/5363	16961/5363
Service layer	2828/204	2828/204
DM layer	2143/214	2143/214
Object layer	3055/1261	3055/1261
Li layer	2570/828	2570/828
DHT	3387/16	3387/16
uIP Stack	26361/4765	26361/4765
RPL	10865/250	10865/250
CoAP	-	9289/761
LWM2M	-	9641/543
IPSO	-	2671/275
TOTAL	107240/23917	130727/26291

Table 1 Memory Use of Nodes of Different Capability

The DHT implementation used Kademlia approaches to create and compare dht ids, for the selection of closest nodes and to create and manage buckets functions, e.g to place closest peer ids in the correct k-bucket. The information for peers in a bucket was held in a Peer object in the Object Space store, with a lease, like any other object.

Several test scenarios were run up to 10 times each, with example networks shown in Figure 5. These scenarios used an edge router running RPL, allowing a CoAP or LWM2M server or external HPP Peer to access WSN nodes. The HPP nodes are located in a simple overlay network and given the address of a *bootstrap* peer. The *source* and *sink* node ran a series of tests to send hello to get an identifier and join the overlay, add new objects remotely and to get objects such as node readings and peer information. Stack use was monitored to ensure it did not exceed the allocated 1KB (its largest size was 750bytes).

Table 2 shows the time to process the different types of messages on the server and for the reply to be received on those nodes a single hop away (to focus on hpp processing time rather than routing). The server processing times did not depend on the number of objects being searched or added, albeit there were at most 20 objects. These results suggest that hpp is feasible on constrained nodes even using a string format. The robustness of hpp was demonstrated by stopping

nodes during tests and their peer information (and any added objects) was deleted when their leases expired.

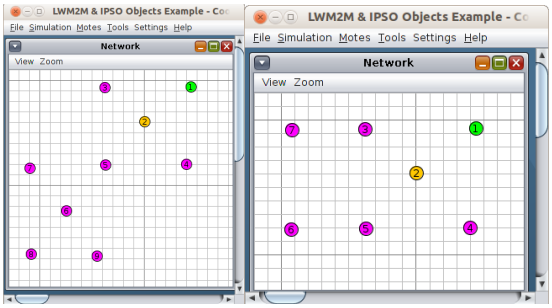


Figure 5 Simulated Networks

Server Hello (ms)	Server Get (ms)	Server Add (ms)	Hello Reply (ms)	Get Reply (ms)	Add Reply (ms)
31	7.8	20.8	166	208	173.8

Table 2 Message Times

VI. Conclusion

This paper has outlined the use of a DHT based on Kademlia and its successful implementation in our HPP protocol and architecture. The architecture's abstractions allowed code re-use on constrained nodes and Linux servers, with code changes only in the lower layer implementation, making testing and development easier and providing consistent concepts for programmers.

Our use of DHT in HPP also demonstrated the benefits we believe that P2P approaches can bring to WSNs, such as robustness, scalability and easier deployment, as a node only needed the address of one node to join the network and discover other peers, and decentralization, e.g. the decision to allow a node to join is made by the node it contacts, which may also provide a DHT identifier. HPP also supported distribution, with peers physically and logically distributed reflecting the distributed IoT environment and providing data to be stored or processed in more than one node.

Future work will consider storing metrics when refreshing and processing replies, which can be used to select low-latency paths for subsequent requests. Also, associating a prefix of the 160 bit identifier with a Bootstrap peer (and an edge router) would allow a shorter identifier to be used within a local physical network and use the 160 bit identifier externally. Furthermore, a binary encoding of HPP would be straightforward to implement and reduce message size.

References

[1] S. Haller, "The Things in the Internet of Things," in *Internet of Things Conference (iot2010)*, 2010.

[2] S. Nawaz, X. Xu, D. Rodenas-Herr'aiz, P. Fidler, K. Soga and C. Mascolo, "Monitoring a Large Construction Site Using Wireless Sensor Networks," in *RealWSN*, 2015.

[3] F. Bonomi, R. Milito, J. Zhu and S. Addepalli , "Fog Computing and Its Role in the Internet of Things," in *MCC Workshop on Mobile Cloud Computing*, 2013.

[4] M. Koster, "Information Models for an Interoperable Web of Things," in *Position paper for W3C Workshop on the Web of Things – Enablers and Services for an Open Web of Devices*, 2014.

[5] IPSO, "IP for Smart Objects (IPSO) Alliance," 2014. [Online]. Available: http://www.ipso-alliance.org. [Accessed 2018].

[6] M. Kovatsch, "Scalable Web Technology for the Internet of Things (PhD Thesis)," ETH Zurich, 2015.

[7] B. Cohen, "The BitTorrent Protocol Specification," 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html. [Accessed May 2017].

[8] D. Tracey and C. Sreenan, "A Holistic Architecture for the Internet of Things, Sensing Services and Big Data," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2013.

[9] D. Tracey and C. Sreenan, "CacheL - A Cache Algorithm using Leases for Node Data in the Internet of Things," in *IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2016.

[10] P. Maymounkov and D. MaziIIeres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," in *First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[11] X. Le et Al, "Secured WSN-integrated Cloud Computing for u-Life Care," in *IEEE Consumer Communications and Networking (CCNC)*, 2010.

[12] "Contiki: The Open Source OS for the Internet of Things," [Online]. Available: http://www.contiki-os.org/. [Accessed May 2018].

[13] I. Clarke, O. Sandberg, B. Wiley and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science,* vol. 2009, 2001.

[14] C. Shirky, "What is P2P....And What Isn't?," 2000. [Online]. Available: http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html. [Accessed May 2018].

[15] "Openfog Consortium," [Online]. Available: https://www.openfogconsortium.org . [Accessed May 2018].

[16] S. Krco, D. Cleary and D. Parker, "P2P Mobile Sensor Networks," in *IEEE Conference on System Sciences*, 2005.

[17] M. Ali and K. Langendoen, "A Case for Peer-to-Peer Network Overlays in Sensor Networks," in *Proceedings of WWSNA with 6th IPSN*, 2007.

[18] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris and I. Stoica, "Looking Up Data in P2P Systems," *Communications of the ACM,* vol. 46, no. 2, pp. 43-48, 2003.

[19] I. Stoica, "Chord: A scalable peer-to-peer lookup service for Internet applications," *IEEE/ACM Transactions on Networking,* no. February, 2003.

[20] A. Rowstron and P. Druschel, "Pastry : Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *18th IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[21] "Eclipse IoT," [Online]. Available: https://iot.eclipse.org/. [Accessed October 2018].

[22] Arago Systems, "WIsmote," [Online]. Available: http://www.aragosystems.com/produits/wisnet/wismote/. [Accessed 2018].