Overlay Enhanced Mobility for the Internet of Things
Kardeby, Victor, Jennehag, Ulf, Gidlund, Mikael
Department of Information and Communications Systems
Mid Sweden University, SE-851 70 Sundsvall, Sweden
Email: {victor.kardeby, ulf.jennehag, mikael.gidlund}@miun.se

Abstract—One of the major challenges to realize the Internet of Things is to support IP mobility for the large amount of connected entities when they move between different locations and access methods. Current solutions for mobility are host centric, requiring support from the infrastructure, or breaks backwards compatibility, which will take a long time or high economic motivation to implement. Solutions for context information exchange are created for specific, small, or localized scenarios with centralized coordination that do not scale well. There is therefore a need for a solution which both scales well, and support IP mobility, without additional demands on current or future Internet infrastructure. We propose the use of a dual-overlay network structure for both information dissemination and as an alternative to current IP mobility technologies. It separates identities from location by introducing a second overlay network where the identity-to-location association is stored. We show analytically that the proposed solution provides logarithmic latency for localization and reduces the overall workload when the number of sensors per host increases beyond seven, with a workload reduction of 15 percentage points at fifteen sensors per host.

Index Terms—Internet of Things, Mobility, Overlay Network, Performance Measurement, Analytical Approach

I. INTRODUCTION

THE vision for the Internet of Things (IoT) [1] is that all devices should be interconnected and share information with other devices, people, places, virtual entities, and systems; all of these could jointly be termed as endpoints, which exchange information on the Internet. The combination of these different types of endpoints is sometimes denominated as the Internet of Everything [2]. The number of endpoints in the Internet of Everything is rapidly increasing, following the increase of devices projected to populate the IoT, the proliferation of all types of personal digital assistants, and an improved acceptance towards and adaptation to connected devices by consumers. To achieve the vision of the IoT there are several challenges that must be solved, [1] lists key features that the IoT must support, such as support for heterogeneous devices, self-organization, semantic interoperability, scalability of naming, communication, information management and service provisioning. This paper focus on scalability due to the projected size of the IoT. There are several different projections of the increase in the size of the IoT. In [3] the authors state fifty or one hundred billion devices by 2020; while [4] are more modest, projecting twenty four billion devices in 2020. In the other end, [5] speculate that there may be trillions of communicating devices in the future. An observed common denominator is that the projections estimate that devices, and thus machine to machine (M2M) communication, will outnumber machine to person communication with several magnitudes [6] in the near future.

The basic mission of the IoT is to enable remote access to sensor data produced at the end devices, and when a device is not capable of Internet access there will be an intermediate gateway or host that acquire and maintain the sensor data as resources for remote endpoints to access. The sensor data is then used to provide context to the current situation of whatever the sensor is associated with. Current approaches [7] focuses on consumer products where there is a clear, and fixed, association between a sensor and its owner and location. For instance, a smart wristband is associated with a person and utilizes a smartphone for internet access, and an apartment’s wireless equipment is associated with the family living there. But, as sensors start to saturate not only the consumer market, but the public environment, measuring many different aspects of things and places, it is less clear as to how these sensors are associated with devices and people. For example, a person using a taxi might want to be associated with the location of the vehicle, a building apartment might associate itself with averages of all environmental sensors currently occupying the premises, or a dance club might use sensor fusion between the guests accelerometers, their public profiles, and the evening sales to select what music to play on the dance floor.

The availability and utilization of sensor data and the increased separation between sensors, hosts, and locations require a new approach to sensor access. This far, hosts represents information as file resources and are given either a static or a dynamic address based on both their location on the Internet. This has been working well for the current usage of the Internet. But another type of online presence is required when it is possible for sensors that have no distinct host to migrate between different hosts while they provide information to peers on the Internet. Resource based access couples the sensor intimately with its current host. Promoting sensors to endpoints would

Manuscript received April 21, 2015; revised July 15, 2015; accepted July 21, 2015.

© 2015 ACADEMY PUBLISHER
doi:10.4304/jnw.10.7.420-430
decouple the sensor from its host, making them mobile and enable sensor owners to more freely control different aspects of their sensors. For example, a car rental service could give customers a better price for renting their cars if the customers allow the sensors in the car to share information with the rental service while in motion. This elevation would thus present all sensors in the rental car as individual endpoints available for communication, but more simplistic and resource constrained sensors would require assistance by a more capable central computer in the car. This would, enable a tire-manufacturer to gain access to the air pressure sensor in the tires of the car to receive statistical data of wear and tear. The manufacturer would access the tire-sensor as an isolated endpoint even though it may be provided by the computer in the car, and if the rental service switched tires between cars there would be no noticeable difference for the tire-manufacturer even though the tires are on another car, since their only concern is to retrieve data related to the tires. Elevating sensors to endpoints would allow more fine grained access control while and sensor specific process isolation that increase security and reduce the impact if the security is breached.

There are several approaches [8], [9] to sensor mobility for wireless sensor networks, for the Internet of Things, where 6LoWPAN is suggested as a carrier for mobility approaches. But sensor mobility will not solve anything without access to the mobile endpoint. One approach is to give these endpoints write access on a service hosted by larger centralized computers, such as in [10] or [11]. Two disadvantages of centralized solutions are that they are a central point of failure and are expensive to scale. A decentralized solution, where endpoints share information themselves have a finer control of access rights and better scalability, but require that all participating devices are accessible and locatable. This is where a dynamic address, which is updated as the device changes location, becomes a vital problem for context aware applications that require the use of remote sensors.

The problem is that while context aware applications which only generate and publish context information, they simply need a predetermined static information broker-location to publish their information to, but as the applications begin to provide services themselves, they must be accessible in order to receive incoming requests.

Devices that are to remain accessible to peers have a few different strategies available to them. The first strategy is to maintain an open connection to an intermediate service through which the device can be contacted, the device will be required to maintain this open connection to the service at all times, consuming resources. The second strategy is to immediately inform an intermediate service whenever their IP changes so that peers may connect directly, such as [12]. The third strategy is to use a third party support for IP mobility such that peers contact the device on the same location all the time. This mobility support can furthermore be either explicitly invoked [13] or implicitly maintained [14] in the underlying network architecture. The disadvantage is that device communication performance depends on the performance of the intermediate service.

Future context aware applications on the Internet of Things require a solution that enables sensors and mobile devices to be continuously available, so they can act as an actuator in the real world, and respond to changing context of their users. Additionally, the solution must support near instantaneous information sharing and scale to accommodate the projected 50 billion of devices in order to support future time critical large scale scenarios.

This paper proposes a strategy that elevates sensors as endpoints in order to support both sensor and host mobility individually of each other. The strategy is realized with an algorithm that is implemented as a fully distributed peer-to-peer overlay network, capable of quick re-establishment of communication sessions between peers in transit and capable of exchanging information in near real-time. The peer-to-peer network maintains an online presence for the mobile devices during their temporary absence, which enables authenticated access to sensors over non trusted hosts. We compare the implementation with a host based solution in terms of overlay workload and latency for location transition. The workload analysis will provide the number of peers required to maintain the proposed overlay without saturating the processing capacity of the devices. The latency evaluation will give an upper bound on the size of the overlay network to maintain a desired scenario specific operation delay.

Sequence diagrams will be created for the proposed solution and analytically examined to find the number of messages required to complete the different operations. The number of messages are then evaluated with respect to average Internet latencies and compared to a host based solution. Additionally, the workload subjected to a participating peer will be calculated by assuming an algorithm execution frequency and divide the number of involved messages in the whole system by an exaggerated processing time per message. This will give the minimum number of needed overlay peers for a desired number of users and average workload per device given a sensor per device ratio.

The analytical results will be used in a simple scenario where millions of mobile peers M register several sensors, actuators or context information each S_p, in an overlay of N peers, and then frequently request to communicate with other sensors, actuators or context information. The results will show how workload and latencies changes as the size of the overlay increases in order to show the scalability of the proposed solution in the scenario. Additionally, the analysis will find the point where the increased maintenance signaling in the proposed overlay will support more sensors, actuators and context information than the compared solution for a given overlay size.

Section II gives an overview of related mobility solutions requirements for the Internet of Things and how their scalability can be evaluated. Section III details our
proposed solution for overlay-assisted mobility. Section IV examines the proposed solution and provides analytical estimations of workload and latency. Section V evaluate and compare the solution to related work. Finally, section VI presents our conclusions.

II. RELATED WORK

In section II-A we examine current mobility solutions, and in section II-B a related overlay network used for secure communication is presented.

A. Mobility

Devices that are to remain accessible to peers while they change their location have different strategies available to them. The strategies can be divided into separate categories, in [15] the strategies are separated by the OSI model where it is suggested that different scenarios may benefit from mobility on different layers. In [16] the support is divided into different paradigm methods. In this paper, we choose to separate IP mobility support by their paradigms and the following sections present the approach and representative solutions.

1) Dumb Terminal: Dumb terminal-based mobility uses a fixed location to which the participating mobile devices remains connected to at all times. The fixed location, acts as a routing agent; delivering data to the location of the mobile device. Whenever a client moves to a new location it simply reconnects to its fixed location to resume all ongoing sessions. This approach is either centralized, such as [10], [11] or distributed similarly to Mobile IPv6 [17]. Both alternatives are points of possible failures, since either parts or the whole system collapses if a proxy goes offline.

2) Identity / Locator Split: Identity / locator split-based mobility assign a new identity to the device, either humanly readable, i.e. DNS [18], or not, i.e. LISP [19] or Hi3 [20], [21]. The identity is then automatically translated into the current location of the device by the network when needed. This approach either has a centralized directory where the identity-to-location information is kept, or requires devices to be updated to communicate using the new protocol. In either case there will be an additional lookup time in the directory before communication is reestablished.

3) Indirect Routing: Indirect routing-based mobility use intermediate points in the network to assist an intelligent mobile device by providing a static address to the device to be used when communicating with peers, for example Mobile IPv6 [17]. Alternatively the intermediate point can fool a legacy device that it has the same address regardless of access location, as in PMIPv6 [22]. In both alternatives the device is contacted by its static location which is routed through the network to the device’s actual location. Indirect routing approaches thus require extensive support in the Internet infrastructure and do not scale well.

B. Overlay networks

Based on the several paradigms for solving mobility on the Internet that is presented in the previous section, we select a distributed identity / locator split approach based on Chord [23]. In a chord-based DHT all peers participates and the relative location of a peer is determined by its current IP address. The location determines the responsibility of a subset of the fixed hash-based key-space for the peer. To join the network, a peer identifies its intended location in the circle. To assume the responsibilities for that partition of the key-space the peer communicate with anyone that is connected, which will be forward the join request to the peer currently responsible for the key-space.

When a mobile peer change its location, its key-space responsibility changes as well, which results in a need to graciously leave the network at the current location and rejoin the network at the new location. Since changes in IP-address happen without advance notification it results in that the peer makes an ungraceful leave of the network, requiring the network to recover from an undesired state. Recovery from ungraceful disappearance of peers in chord is managed through replication of data to immediate successors of the peer who is responsible for an entry giving the circle time to heal. A peer can therefore receive the same requested data by asking the successor of a missing peer. When a peer reconnects it have to retrieve the entries associated with its new area of responsibility.

The Hi3 architecture [20], [21] uses the Identity / Locator split in HIP together with the overlay network in i3 as a control plane. I3 is in turn based on chord, which make Hi3 a good architecture to compare with a chord based implementation.

1) Scalability Estimations in Hi3: In [20] the authors perform latency and workload estimation of the control plane, an Identity / Locator split-based mobility solution that is based on HIP and i3. In short, i3 is a fixed infrastructure of devices that maintain name to location resolution information within a Chord-based DHT and provide indirect routing of data, and HIP is a transport layer extension that separates identity from location. Combined they are used to provide secure communication setup and maintenance through i3 and data transfer using HIP to enable scalable and secure communication between clients and servers.

The Hi3 architecture is evaluated in three scenarios in which they examine latencies and workload. The latency is analytically investigated and defined as the number of transmitted messages in an operation multiplied with the average delay of transmitting all messages in the operation. The authors focus on one operation that scale logarithmically and show that the other operations differ by a small factor. A transmission latency of \( \tau = [50, 100, 150] \) ms is examined and the most frequently used operation, optimized association setup, and is shown to be performed within 1000 ms with a control plane of 1000 nodes.

The article [20] continues by defining the workload for
one node in the control plane doing one operation as: 

\[ W = \frac{M\lambda H}{\mu} \]

where \( H \) is the number of nodes using the control plane, \( \lambda \) is a uniform request per time unit, and \( N \) is the size of the control plane. The utilization of a node is 

\[ U_{CPU} = \frac{W_{\mu}}{\mu} \]

where \( \mu \) is the time it takes to process one message. They examine a scenario where a number, \( M \), of mobile devices are considered, another scenario where there are \( M \) mobile devices and \( L \) servers. And a third scenario where there are a number of malicious nodes introducing interfering data in the system. They show that \( M = 5 \times 10^6 \) devices can be serviced using a supporting overlay network of \( N > 200 \) i3 nodes with a utilization below 30%. The latency and throughput is verified in [21] where they show that Hi3 is an improvement over i3 that compares well to regular TCP communication given the provided security and DoS resilience.

2) Adapting Hi3 to the Internet of Things: In order to use the Hi3 architecture in an Internet of Things scenario, where sensors, actuators and context information is made available independently of their hosts, we promote said sensors, actuators and context information into endpoints in the system. This would provide independent access control and individual security for them all. The impact on the Hi3 architecture would be that the \( H \) in workload is multiplied by the number of sensors per host \( S_p \) for a generic workload of \( W = \frac{M\lambda H S_p}{\mu} \) per operation.

III. OVERLAY ASSISTED MOBILITY

In this paper we propose that two individual overlay networks with different responsibilities can be used in order for the solution to support mobility in the Internet of Things. In essence, one overlay maintains associations between sensors and their current host, while the other overlay maintain the associations between hosts and their location on the Internet, i.e. their IP addresses. In practice, one overlay will be maintained inside the other, therefore we denote them as inner and outer, where the inner maintains sensor identities to host identities, while the outer maintains the same host identities to Internet location. The inner overlay may maintain additional state regarding the hosts, such as the latest value, public cryptographical key, or meta information. Since overlays are peer-to-peer we choose to denote the hosts in them as peers. The overlay will be maintained by the hosts due to the limited capacity and power of some sensors. One example of a host would be a Raspberry Pi\(^1\).

The inner overlay associate sensor endpoints to peer identities, while the outer overlay links peer identities with peer locations. We denote the sensor identity as \( S_p \# \) where \( p \) is the peer and \# is an enumeration of the sensors, \( S_p/2 \) thus denotes the second sensor of peer 2. The identity of a peer is denoted as \( ID_p \) while its current location is denoted \( IP_p \). Both sensor identities and the peer identity is selected, either manually, randomly or based on some algorithm, with the requirement that is must not depend on the current location of the peer and be unique among all participating peers. The outer overlay stores the association between \( ID_p \) and \( IP_p \). The inner overlay stores the associations between \( S_p\# \) and \( ID_p \). This gives different desirable properties for each overlay. The outer overlay should support low latency lookups and high churn rates, while the inner overlay should allow for delay tolerant lookups and recursive communication. The inner overlay must use the identities, \( ID_p \), as an addressing scheme for its communication. This strategy reduces the amount of transferred data during a location change by a ratio proportional to the ratio between sensors and peers

\[ \frac{1}{\text{count}(S_p\#)} \]

A. Communication Algorithm

The algorithm to communicate between overlays is described in this section. A peer require that the outer overlay is connected and operational or that a cache of locations is in use and accurate in order to communicate within the inner overlay. Peer \( p \) has an IP address \( IP_p \) and a preselected identity \( ID_p \). The peer additionally has a number \( S_p = \text{count}(S_p1...S_pN) \) of sensors that the peer are to share. Each sensor represents either physical or logical information about a peer and has an identity \( S_p\# \) which follows the UCI scheme described in [24], for example: victor@miun.se/body/temperature or victor@miun.se/calendar/today.

Communication in the outer overlay uses the native methods available, while the inner overlay should use the outer overlay for location lookup. To communicate with peer \( p' \) in the inner overlay a peer must retrieve the \( ID_{p'} \rightarrow IP_{p'} \) association from the outer circle if not already stored in a local cache. To be concrete, the algorithm for sending a message \( m \) between two peers in the inner overlay is:

**Require:** input \( p' \) message \( m \)

**Ensure:** \( p' \) not null

if cache contains \( p' \) and cache-entry timestamp is ok

send \( m \) to cached \( IP_{p'} \)

if Successful then

return

end if

end if

request entry corresponding to \( ID_{p'} \) from the outer overlay

store \( ID_{p'} \rightarrow IP_{p'} \) in local cache

send \( m \) to cached \( IP_{p'} \)

return

B. Implementation

A single type of overlay structure is implemented for both the inner and outer overlays, where a circular DHT [23] is selected. The implemented overlay structures peers

\(^1\)http://www.raspberrypi.org
in two circular numbered spaces each one with a fixed preselected size. To visualize this, one circle is drawn outside the other. The area of responsibility is determined differently in the circles. The location of a peer, \( IP_1 \), determine the responsibility in the outer circle, same as chord, while the identity of a peer, \( ID_1 \), determines the responsibility in the inner circle.

Fig. 1 show a simple example where four peers are connected in the two circles. The key-space is \( 2^{12} \) values which for simplicity are labeled as 0-15 in the outer circle, and A-Q in the inner. Each peer has an identity (ID) in both circles separated by a slash. Each peer additionally has its immediate successor (\( F[0] \)) and predecessor (P) in the routing table both separated by a slash as well. Finally they store the coupling between identity and location in the outer circle, and in this example each node has two sensors connected in the network stored in the inner circle. Peer I in figure 1 thus store that the key 14 is equal to the value IP_6 in its outer storage, and that the key P equals G and so on in its inner storage. In figure 2 peer II have changed location in the outer circle, while maintaining the location in the inner.

In order for Peer III in Fig. 2 to find and communicate with Sensor \( S_{B1} \) it first calculates a hash (\( # \)) of the string representation of the sensor name: \( #(S_{B1}) \) which in this example is equal to the value \( P \). The hash is denoted as the key of \( S_{B1} \). Then it have to find the successor of \( P \), (which is C, peer I) by following the circle clockwise. By hashing its inner finger \( F[0] = C \), its successor, it get that the value associated with the hash of C, \( #(C) \), is equal to 2. Peer III finds that the successor of 2 in the outer circle know the location of C. Peer III forwards the request around the outer circle to find the successor of 2 via it’s outer finger 10, which forwards the request to 0 who in turn forwards the request to Peer II at IP_4. Peer II answers Peer III that the successor of 2 is IP_0. Peer III then contacts Peer I at IP_0 and requests the identity of \( S_{B1} \), which is G. Finally Peer III hashes \( #(G) = 6 \) and forwards the request for the IP address in the outer circle. Therefore, all peers should employ a caching mechanism such that the identity of a peer in the inner circle and its corresponding address remains in a local storage for a reasonable time.

C. Algorithms

The algorithms to join, leave, maintain and update entries in the DHT is described in the following section; which follows the algorithms for Chord presented in [23]. Whenever a peer \( p \) is to join the overlay it require knowledge of a peer \( p’ \) that already is a member of the overlay. The discovery of \( p’ \) is out of the scope of this paper, but could either be a preselected rendezvous point, a cache of peers from previous sessions or some kind of multicast/broadcast discovery. Peer \( p \) have an IP address as its current location \( IP_p \) and a preselected identity \( ID_p \).

1) Locally stored data: A peer in the overlay maintains two similar sets of state information, one for each of the two circles. Both states contain a local identity, a finger table, a predecessor, a list of immediate successors \( IS \), and all data that is the local identity’s responsibility just as Chord [23] but with one difference as to what is stored in each entry. The inner circle’s predecessor, successor list and finger table entries store the preselected identity \( ID_n \) of the recorded peer while the outer circle store the IP address \( IP_n \). Finally the outer circle maintains a cache of identity to IP mappings that the peer has been communicating with. A small number of immediate successors are maintained to increase the robustness of the overlay. The number of immediate successors \( IS \) to be maintained is \( \log(N) \) where \( N \) denote an estimation of the number of peers maintaining the system. Estimation for a peer \( p \) is currently done by calculating \( IS = \frac{1}{2}((KeyID_p - KeyID_{p-1}) + (KeyID_{p+1} - KeyID_p)) + (KeyID_p - KeyID_{p-1}) + (KeyID_{p+1} - KeyID_p) \), where KeyID is the hash of the ID and IP is the hash of the IP. Improvements are deferred for future work.

2) Join: To join an overlay, \( p \) contacts a peer \( p’ \) which is a member of the overlay. The join is performed recursively for both circles simultaneously in accordance with the Chord [23] algorithm.
3) Rejoin: When a peer changes its location it only perform a join to the outer circle as no state need to be changed in the inner circle. The initial peer \( p' \) is in this case selected based on the previous closest finger in the \( p \)'s finger table but otherwise follows the algorithm for Chord [23].

4) Leave: To leave the overlay gracefully, a peer communicates with their predecessor and successor to hand over the key’s it is responsible for.

5) Find successor: The find successor operation is performed as in [23] and is used to perform insertion and retrieval of data entries in the overlay network.

6) Maintenance: Periodical maintenance perform four tasks, of which three corresponds to those performed by the chord algorithm. Same as chord, a peer verifies its successor, predecessor and one finger, with each iteration, in both circles. A peer additionally removes old cached entries from the cache of \( 1D_{p'} \rightarrow IP_{p'} \) entries. The algorithm is:

\[
\text{for all circles do} \\
\quad \text{Perform chord:} \\
\quad \quad \text{stabilization} \\
\quad \quad \text{fix fingers} \\
\quad \quad \text{check predecessor} \\
\text{end for} \\
\text{for all entries in peer } \text{ID}_{p} \rightarrow \text{IP}_{p} \text{ cache do} \\
\quad \text{if entry is old then} \\
\quad \quad \text{remove entry} \\
\quad \text{end if} \\
\text{end for} \\
\text{return}
\]

D. Signaling

Implementing the algorithms recursively results in the following signaling for each action.

1) Join: Fig. 3 show the signaling involved when a join is performed. Peer \( p \) joins the overlay by sending a join request to peer \( p' \). Peer \( p' \) forwards the join request to the successor of \( KeyID_{p} \) and \( KeyIP_{p} \) concurrently by asking the finger who precedes each key by the smallest amount possible. Finally the successors of \( KeyID_{p} \) and \( KeyIP_{p} \) will reply to \( p \) with their identity. When \( p \) receive the reply, it begins inserting itself in the overlay by assuming the responsibilities of all keys between the predecessor of the responding node and the \( p \)'s key. Finally it will register the sensors and actuators with their respective successors in the inner overlay, and register its location in the outer overlay.

2) Rejoin: When rejoining the overlay after a shorter interruption, such as the one made after a change of IP address, a shorter optimized join can be made. Peer \( p \) will have its previous state when it rejoins. It can therefore use its outer finger table to find its new successor for \( KeyIP_{p} \) while the inner overlay remains the same. The entry for \( KeyIP_{p} \) is moved when the new outer successor is found. Fig. 4 details a rejoin.

3) Leave: To leave the overlay network, peer \( p \) notifies its successors in both the outer and inner overlay that it is leaving.

4) Find successor: To find the owner of a sensor or actuator with key \( KeySi \), peer \( p \) forwards a find_successor message to the peer \( p_i \) whose \( KeyID_i \) in the inner finger table barely precedes \( KeySi \). Recursive forwarding will result in a find_successor_response from the successor of \( KeySi \) containing the stored information associated with that key. Peer \( p \) can then find the successor of the associated \( KeyID_i \) using a find_successor for the outer overlay. A response contains the IP address of the peer who owns the sensor. The entry is stored locally in a cache which is refreshed whenever that peer do not respond within a selected timeout. Fig. 5 details the find successor signaling.

To insert an entry of a sensor or actuator into the inner overlay, peer \( p \) first find the successor of the key \( KeySi \) for the entry in the inner circle, and transmits an insert_entry message to that successor containing the key \( KeyID_i \) and the value \( ID_p \). For the first insertion, an additional insertion have to be made in the outer circle at the successor of \( KeyID_p \), where the key \( KeyID_p \) is associated with the IP address of peer \( p \). If the entry already exists there is a possibility to include a cryptographical authentication method in the entry to allow updates. Fig. 6.
6 details an insert.

5) Update finger: A finger table entry is updated by periodically finding the successor for the ideal finger, which is calculated using the finger index i, and keyIDp + (2^i−1) or keyIPp + (2^i−1) for either the inner or outer overlay, both are calculated modulo 2^keysize. Fig. 7 details a finger update.

6) Maintenance: A peer p asks its successor for its predecessor, and if the predecessor’s key is a better successor it will be replaced as p’s successor and an update_predecessor message will be sent to the new successor. Fig. 8 details regular maintenance.

IV. ANALYTICAL EVALUATION

In the first subsection we perform an analysis of the internal and external latencies in the proposed overlay. In subsection IV-B we examine the workload of the overlay network. In this paper we use the same variable definitions as [21]. The following list is a short summary of them.

N: The number of peers in the overlay network.

A. Latency estimation

There are several algorithms presented in Section III, in this section we intend to find an analytical expression of an upper bound of the latency for them. Each request is divided into two parts, internal and external latency. Internal latency is defined as the delays within the overlay network, while external delays are latencies for communication to and from the overlay. This separation is performed in the case that the overlay use a supernode structure where the latency between the overlay peers is different from the other users of the overlay. The total latency for an operation is therefore the sum of both latencies L = k(τX + τint) = T^X + T^int where k is the number of messages involved in the operation and τ is the average delay for all messages. A special situation arise when all participating users also maintain the overlay network, then all latency would be internal latency. Viewing the peer as an external user outside the system yield a higher latency and may therefore show a higher bound for the latency.

1) Find successor: The find successor algorithm is performed in nearly all overlay operations whenever the successor is not previously known; it is named an i3 lookup in Hi3. In [23] the find successor algorithm communicates with an average of αLog(N) peers. Using recursive communication this gives an internal latency T^int = α(τ + μ)Log(N). The external latency for a find successor operation is T^fs = 2τp. 

2) Join: To join the overlay a peer must find the correct successor for each circle and then insert a KeyID > IP record in the outer circle. This gives an internal latency of T^join = 3T^fs + 3α(τ + μ)Log(N). By transmitting a join message routed through the overlay to peer p’, we get an external latency T^join = 3τp’ + 2μp’ + μ + 3μp’ + 3τp’.

3) Rejoin: To rejoin the overlay a single KeyID > IP record update is performed after joining the outer circle. This gives an internal latency of T^rejoin = T^fs + α(τ + μ)Log(N). And an external latency T^rejoin = τp’ + μp’ + μ + τp’ + 2τp’. As the rejoin use the existing finger table to select the best finger for joining should give
\[ T_{\text{rejoin}}^{\text{Int}} < \alpha(\tau + \mu)\log(N) \] but this is ignored since the expression should represent a worst case latency.

4) **Insert:** To insert a sensor record in the overlay the internal latency is \( T_{\text{insert}}^{\text{Int}} = T_f + \mu \alpha(\tau + \mu)\log(N) \), and external latency is \( T_{\text{insert}} = T_f + 2\mu + T_{\text{r}}^{\text{p}}.p \).

5) **Retrieve:** To retrieve a single entry require two internal lookups initially, after the KeyID \( \sim \text{IP} \) record is cached it only need to be reacquired after a move. If the probability that a peer have moved is \( P_{\text{us}} \) (same as that of a double jump in Hi3) we get an average latency of \( T_{\text{retrieve}}^{\text{Int}} = T_f + P_{\text{us}} \cdot T_f = (1 + P_{\text{us}})\alpha(\tau + \mu)\log(N) \), and an external latency: \( T_{\text{retrieve}} = T_f + 2\mu + T_{\text{r}}^{\text{p}}.p \).

6) **Update:** An update is performed through a direct transmission to the responsive successor, unless it have moved, then there is an additional internal lookup performed in the outer ring. The internal latency of an update is:

\[ T_{\text{update}}^{\text{Int}} = P_{\text{us}} \cdot T_f = (P_{\text{us}})\alpha(\tau + \mu)\log(N) \]

and the external latency is \( T_{\text{update}} = T_f + 2\mu + T_{\text{r}}^{\text{p}}.p \).

### B. Workload estimations

This sections presents the estimated workload of an operation, the workload is defined as the number of messages a peer have to process per second. If the workload \( W \) is multiplied with the average processing time per message \( \mu \) we find the cpu utilization of the peer, \( U = W \cdot \mu \). The workload is calculated as \( W = \frac{\alpha M \lambda \log(N)}{N} \) where \( \lambda \) is the message frequency, \( H \) is the number of peers using the system, \( N \) is the number of peers that compose the overlay, and \( r \) is the number of nodes involved in the request, either \( r = c \) or \( r = c \log(N) \).

1) **Find successor:** The find successor algorithm is performed in nearly all overlay operations whenever the successor is not previously known, and is the main operation which depends on the size of the overlay. Using recursive communication this gives a workload of:

\[ W_f(\lambda) = \frac{\alpha \lambda \log(N)}{N} \]

2) **Join:** To join the overlay a peer must find the correct successor for each circle and transmit a join message. This gives a workload of:

\[ W_{\text{join}} = \frac{\alpha \lambda_{\text{join}} H(6 + 3\log(N))}{N} \]

3) **Rejoin:** To rejoin the overlay a single KeyID \( \sim \text{IP} \) record update is performed after joining the outer circle. The workload become:

\[ W_{\text{rejoin}} = \frac{\alpha \lambda_{\text{rejoin}} H(4 + \log(N))}{N} \]

4) **Insert:** To insert a sensor record in the overlay the workload is:

\[ W_{\text{insert}} = \frac{\alpha \lambda_{\text{insert}} H(2 + \log(N))}{N} \]

5) **Retrieve:** To retrieve a single entry require either two internal lookups in a full retrieve, which after the KeyID \( \sim \text{IP} \) record is cached only need to be reacquired after a move. Otherwise a simpler retrieve is performed. If the probability that a peer have moved is \( P_{\text{us}} \) (same as that of a double jump in Hi3) we get a workload of:

\[ W_{\text{retrieve}} = \frac{\alpha \lambda_{\text{retrieve}} H(2 + (1 + P_{\text{us}})\log(N))}{N} \]

6) **Update:** An update is performed through a direct transmission to the responsive successor, unless it have moved, then there is an additional internal lookup performed in the outer ring. Additionally, the entry in the outer circle must be refreshed periodically, but since the inner circle is more stable there are different refresh rates \( \lambda_{\text{update}} \) and \( \lambda_{\text{update}} \). The workload of an update is:

\[ W_{\text{update}} = \frac{\alpha (\lambda_{\text{inner}} + \lambda_{\text{outer}}) H(2 + (P_{\text{us}})\log(N))}{N} \]

### V. Numerical Results

This section compares the latencies from IV-A and workload IV-B with Hi3 using the data found in [20], [21]. The comparison with Hi3 is done due to similarities between Hi3 and the proposed algorithm. A summary of relevant variables is presented in Table I which is copied from the first scenario presented in [20]. In Table II and III we show the formulae used in the comparison. The following sections compare the Hi3 architecture to the proposed overlay using the Hi3 adaptation in Section II-B.2.

### A. Latency

To compare the proposed overlay with a similar approach we map the different functions in the overlay to those in Hi3. The most used operation in the overlay is the retrieve operation, responsible for finding the IP address of the desired sensor, which corresponds to the association setup in Hi3, whose main purpose is to initiate new connections between peers. In this comparison we map a full retrieve with the pure association setup and the more frequent used retrieve with the optimized association setup. HiT Insertion is mapped to the Insert operation, HiT refreshment & removal with update, and rejoin with the double jump operation. Table II summarizes these mappings and displays the corresponding functions. The comparison is performed by assuming that all links in the system are ideal for both the proposed solution and Hi3 with an average latency of \( \tau = 100\text{ms} \). The different functions in Table II are then plotted with respect to \( N \). Fig. 9 represents the logarithmic functions and show the internal latency of Hi3 overlay for the optimized

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( \frac{1}{7} )</td>
<td>( \lambda_{\text{so}} )</td>
<td>(30 mm)(^{-1} )</td>
</tr>
<tr>
<td>( \tau )</td>
<td>100 ms</td>
<td>( \lambda_{\text{r}} )</td>
<td>(30 mm)(^{-1} )</td>
</tr>
<tr>
<td>( \mu )</td>
<td>1.0 ms</td>
<td>( \lambda_{\text{s}} )</td>
<td>(1 h)(^{-1} )</td>
</tr>
<tr>
<td>( \mu_{\text{pr}} )</td>
<td>100-200 ms</td>
<td>( \lambda_{\text{p}} )</td>
<td>(5 mm)(^{-1} )</td>
</tr>
<tr>
<td>( \tau_{\text{Hi3}} )</td>
<td>100 ms</td>
<td>( \lambda_{\text{p}} )</td>
<td>(30 s)(^{-1} )</td>
</tr>
<tr>
<td>( \tau_{\text{so}} )</td>
<td>100 ms</td>
<td>( P_{\text{us}} )</td>
<td>10(^{-2} )</td>
</tr>
<tr>
<td>( \tau_{\text{sc}} )</td>
<td>100 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
association setup and the proposed overlay for a retrieve. This show that the proposed overlay network can initiate a connection between a peer and a sensor faster than Hi3 create an association between two hosts. The main influencing factor for the difference between these latencies are the factors 2 and \((1 + P_{us})\) in that Hi3 always perform two lookups as a part of the association setup, one for the public trigger, and one for the private, while the proposed overlay finds the location of a sensor with one lookup and only require a second lookup in the event that the peer hosting the sensor have changed its location with a probability \(P_{us}\). The latency of the initial join is not compared since it is performed once per peer, do not have a corresponding function in Hi3 since the overlay network is not maintained by the participating mobile devices, and will not impact the latency of running operations significantly.

B. Workload

The workload frequencies of the different operations is matched against operations in Hi3 similarly to the latency evaluation above and then the total workload is evaluated where \(W_{\text{Overlay}} = W_{\text{join}} + W_{\text{rejoin}} + W_{\text{insert}} + W_{\text{retrieve}} + W_{\text{update}}\). For Hi3, in [20], the workload is

\[
W_{\text{Hi3}}^{\text{tot}} = \frac{\lambda_{mob} M}{N} \log(N) + \frac{\lambda_u M}{N}
\]

where \(\lambda_{mob} = 6\lambda_s + 2\lambda_{so} + \lambda_u P_{us}\). The formulae used are summarized in Table III.

The \(\lambda\) rates in [20] for the mobility scenario are used in the comparison and mapped to the proposed overlay as: \(\lambda_s = \lambda_{so} = \lambda_{ret} = (30 \text{ min})^{-1}\), \(\lambda_i = \lambda_{ins} = \lambda_{join} = \lambda_{in}^{\text{update}} = (1 \text{ h})^{-1}\), \(\lambda_u = \lambda_{rejoin} = (5 \text{ min})^{-1}\), and \(\lambda_e = \lambda_{out}^{\text{update}} = (30 \text{ s})^{-1}\). Both association setups are matched with the retrieve operation. The HiT insertion is mapped to both the insertion and join operations. The HiT update operation is matched to the update in the inner circle. The HiT Refresh/Removal is mapped to the update in the outer circle. The comparison is performed by assuming that all devices in the system take an identical time to process each message for both the proposed solution and Hi3 with an average processing time \(\mu = 1.0\text{ms}\). The different functions in Table III are then plotted with respect to \(N\). In Fig. 10 we examine the workload per peer that maintains the overlay when the system has \(M = 5 \times 10^6\) peers and \(S_p = 15\) sensors registered per peer. We see a performance benefit of 15% when \(N = 5000\) in the figure. Varying the number of sensors per host showed that there is a performance benefit in the proposed overlay when there are at least seven sensors registered per peer.

C. Scalable mobility

In order for the proposed overlay assisted mobility to scale well, there is a need to balance low latency and low workload to reduce the performance impact on the devices using the system while providing an adequate latency for the intended usage scenario. By changing update rates in the maintenance cycles and measuring the join and retrieve frequencies it is possible to predict a latency for a given workload, overlay size and supported number of users. For example, in order to support 50 billion sensors with 1 billion devices \((S_p = 50)\) using the same numbers as in Table I and a 25% utilization will require \(1.8 \times 10^6\) peers maintaining the overlay network, with a resulting internal latency of 1.1 s for the retrieve operation. Hi3 would require \(7.8 \times 10^6\) peers in the overlay, resulting in a latency of 2.4 s.

VI. Conclusions

A major challenge for the Internet of Things is the influx of new devices with sensor information that they share while frequently changing locations. Current solutions focus on host centric mobility requiring either support from the infrastructure or breaks backwards compatibility. In this paper we present a strategy to elevate sensors and actuators to endpoints on the internet in order to support both host and sensor mobility. The strategy is implemented using a circular DHT based on Chord. It is used to store the mappings between identity and location.
TABLE II.
Comparison of latency, HI3 and the proposed solution.

<table>
<thead>
<tr>
<th>Request type</th>
<th>HI3 Internal latency, $\tau$</th>
<th>External latency, $\tau_{HI3}$</th>
<th>Double circle overlay External latency, $\tau_{Overlay}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure association setup / full retrieve</td>
<td>$4\alpha(\tau + \mu)\log(N)$</td>
<td>$4\alpha + 2\mu_{pr}$</td>
<td>$\tau_{pr} + \mu + \tau_{pr}'$</td>
</tr>
<tr>
<td>Optimised association setup retrieve</td>
<td>$2\alpha(\tau + \mu)\log(N)$</td>
<td>$2\alpha + \mu_{pr} + \tau_{SC}$</td>
<td>$\tau_{pr} + \mu + \tau_{pr}'$</td>
</tr>
<tr>
<td>HiT Insertion / Insert</td>
<td>$2\alpha(\tau + \mu)\log(N)$</td>
<td>$2\alpha + \mu_{pr}$</td>
<td>$\tau_{pr} + \mu + \tau_{pr}'$</td>
</tr>
<tr>
<td>HiT refreshment &amp; removal update</td>
<td>$2(\tau + \mu)\log(N)$</td>
<td>$2\alpha + \mu_{pr}$</td>
<td>$\tau_{pr} + \mu + \tau_{pr}'$</td>
</tr>
</tbody>
</table>

TABLE III.
Comparison of workload, HI3 and the proposed solution.

<table>
<thead>
<tr>
<th>Request type</th>
<th>HI3 Workload, $W$</th>
<th>Double circle overlay Workload, $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure association setup / Retrieve</td>
<td>$\lambda_s \lambda_s \log(N)$</td>
<td>$\lambda_{ret} 2(1 + P_{us}) \log(N)$</td>
</tr>
<tr>
<td>Optimised association setup / Retrieve</td>
<td>$\lambda_{so} \log(N)$</td>
<td>$\lambda_{ret} 2(1 + P_{us}) \log(N)$</td>
</tr>
<tr>
<td>HiT Insertion / Insert</td>
<td>$\lambda_i \log(N)$</td>
<td>$\lambda_{ms} 2(spr) \alpha \log(N)$</td>
</tr>
<tr>
<td>Location update / Insert</td>
<td>$\lambda_u 1$</td>
<td>$\lambda_{us} \log(N)$</td>
</tr>
<tr>
<td>HiT refreshment &amp; removal / Update</td>
<td>$\lambda_r 2$</td>
<td>$\lambda_{up} \log(N)$</td>
</tr>
<tr>
<td>Double Jump</td>
<td>$\lambda_s P_{us} \alpha \log(N)$</td>
<td>$\lambda_{rejoin} \log(N) + 2$</td>
</tr>
</tbody>
</table>

Context-, sensor- and actuator-information are stored in a separate inner DHT, based only on the identities stored in the outer. The main benefits are: (1) a single entry is changed whenever a peer changes location, even though a peer may have multiple active sensors, actuators and contact information entries. And (2) the data that is stored in the inner ring require less maintenance to accommodate the majority of stored information due to its more static nature. Together these benefits support the need for a support to be accessible regardless of their current location. We thus enable sensor, actuator and context information mobility in conjunction with peer mobility and enable a more fine grained security and access control to the user. This paper show that in a scenario with only 15 sensors, actuators or context information per peer, and $5 \times 10^6$ peers, the overlay reduces the workload on the overlay network by 15 percentage points.

ACKNOWLEDGMENT

This work has been supported by grant 2010-00681 of VINNOVA the Swedish Governmental Agency for Innovation Systems, and by grant 00163383 of the EU European Regional Development Fund, Mellersta Norrland, Sweden.

REFERENCES


[8] Z. Zinonos and V. Vassiliou, “Inter-mobility support in...


