ADDER: Probabilistic, Application Layer Service Discovery for MANETs and Hybrid Wired-Wireless Networks

George Oikonomou, Iain Phillips, Lin Guan
Computer Science
Loughborough University
Loughborough, UK
LE11 3TU
Email: {G.Oikonomou, I.W.Phillips, L.Guang}@lboro.ac.uk

Alan Grigg
Systems Engineering Innovation Centre
Loughborough University
Loughborough, UK
LE11 3TU
Email: A.Grigg@lboro.ac.uk

Abstract—Over the past years, Mobile ad-hoc networks (MANETs) have attracted a considerable degree of research attention, with service discovery, selection and invocation being among the topics of interest of previous efforts. In this paper we introduce ADDER, a probabilistic, hybrid, directory-less service discovery mechanism. It has been designed for military IPv6-based MANETs but will work in any hybrid wired-wireless deployment. It achieves very low service acquisition time through the exchange of a very small number of short messages. Propagation of service descriptions is based on a distance vector algorithm, achieving loop and starvation freedom through a feasibility condition, which has been adopted from established and well-tested routing protocols. This paper also presents evaluation results, obtained by actual execution of the ADDER daemon on two different test beds. The experiments aim to demonstrate that the mechanism achieves good scalability with increasing number of services and network size.

Keywords—MANETs; Hybrid Wired - Wireless Networks; Service Discovery;

I. INTRODUCTION

In this paper we introduce ADDER, a novel, lightweight and efficient service discovery (SD) mechanism. It is the outcome of a project on military ad-hoc networks. Those networks are composed of nodes with high variance in mobility as well as power supply. For example, foot soldiers carry equipment with limited power supply, while it can be assumed that ground vehicles and aircraft have an on-board energy source which can be used to power communication equipment. However, vehicles and aircraft are highly mobile and often follow unpredictable movement patterns. With that in mind, we have designed ADDER to be energy efficient by limiting the size and number of messages exchanged between nodes.

Additionally, in order to cope with high speeds and mobility, the mechanism has low service acquisition time. Military deployments often span across multiple ad-hoc partitions (e.g. foot squads) interconnected over a wired infrastructure network. The resulting diversity in underlying technologies (routing protocols, layer 2 mechanisms) makes cross-layer approaches unsuitable. In order to cope with this diversity, ADDER has been designed as an application layer service. Lastly, ADDER adopts a loop-free, starvation-free forwarding algorithm in order to propagate service information throughout the network.

In this context, the paper’s main contributions are the following:

• The design and implementation (as a linux daemon) of a hybrid, directory-less service discovery scheme for wired-wireless hybrid networks and MANETS. ADDER combines lightweight design with high efficiency and high service discoverability.
• Evaluation, on two different test beds, demonstrating the scheme’s efficiency in terms of response time, network traffic and scalability. We also observe the performance differences caused by changes in the underlying physical layer.

The remainder of this paper is organised as follows: We investigate previous research in service discovery in sec. II. Section III discloses ADDER’s design and implementation details. Section IV starts with a message size comparison between ADDER and three other discovery schemes. Subsequently, we present field evaluation results. Section V concludes this work with a discussion on future research directions.

II. RELATED WORK

Traditional service discovery mechanisms, such as the Service Location Protocol (SLP) and Universal Description, Discovery and Integration (UDDI), rely on the existence of a centralised directory (or lookup server or mediator) to store information about all services available in the network. While such approaches are viable in an infrastructure environment, adopting them for a MANET deployment is not tenable; lookup servers are demanding in terms of resources and energy consumption, while no single node can be assumed to be always reachable. Thus, previous research efforts have mainly focused on distributed directory and directory-less (also direct or peer-based) approaches.
Distributed directory approaches employ clustering and peer-to-peer algorithms, aiming to form a distributed backbone of mediators (CASD [1] and SANDMAN [2]). In [3], the authors present results showing that due to increased synchronisation overhead, clustering algorithms are best suited for dense networks with limited dynamics.

In direct approaches there are no directories; requesting nodes flood the network with lookup queries, while providers respond to queries and may proactively flood the network with advertisements. Nodes receiving an advertisement or overhearing a response may store the information in a cache and respond to subsequent requests. Since broadcast-based flooding is costly, in order to further reduce network traffic, direct approaches often employ techniques such as discovery scoping and selective forwarding. With global schemes, all services are discoverable from any node, anywhere in the network. With limited scope, services are only available to a subset of the network nodes (a zone). In selective and probabilistic forwarding schemes, instead of forwarding an advertisement (or request) to all its neighbours, each node uses an algorithm to select an optimal subset (based on a set of well-defined criteria in the former case or random chance in the latter). GSD [4], PCPGSD [5], Konark [6], Allia [7], CARD [8] and Service* [9] are examples of directory-less service discovery schemes.

Every discovery mechanism is paired with a service description method, aiming to define the network location of service instances as well as information about particular details of each instance. For example, the definition of a video stream may contain information about the stream’s frame rate, resolution and codec. Researchers have so far adopted three methods for service definitions: i) Simple key-value pairs [10], ii) XML documents (Konark [6], OSDA [11]) and iii) Ontologies (GSD [4] and later PCPGSD [5]). Ontological and XML solutions cater for semantically richer service descriptions which aid in resolving more sophisticated queries, at a cost of efficiency due to larger and more complex messages.

Based on the temporal relation between service lookups and the dissemination of service information, discovery mechanisms can be classified into four categories: i) proactive, ii) reactive, iii) hybrid, iv) adaptive. In proactive mode, providers periodically advertise their services to other nodes (or directories) in the network. When a node issues a service request, the information on the location of potential matches is already known, thus service acquisition happens very fast. An example of such an approach is CARD [8]. It has been shown that for a highly dynamic network (in mobility and membership terms) proactive mode does not yield satisfactory results due to the amount of traffic necessary in order to maintain up-to-date service information across the network. In reactive mode, requests are formed and issued on demand when a node tries to invoke a service. Upon reception of a request, a provider issues a reply with information on how to access the service. Reactive service discovery is less efficient in latency terms but more suitable for highly dynamic environments. Hybrid and adaptive schemes aim to combine the benefits of both aforementioned approaches (GSD [4], ABSN [12]).

Cross-layer (or integrated) service discovery mechanisms aim to reduce discovery-related network traffic by integrating service information with routing messages. The reasoning behind this approach is that application layer discovery techniques would cause the injection of routing control messages into the network. In contrast, while cross-layer approaches increase the size of routing messages, the total overhead is reduced since service requests do not introduce any further traffic [12], [13]. In proactive mode, service provision information will be available in the network when a node needs it whereas in reactive mode, service requests will be transported inside route requests.

Integrating service messages with routing comes at a cost. The routing protocol itself needs significant modifications, often rendering the service-enabled version incompatible with the original one. Furthermore, the service discovery mechanism becomes bound to a particular routing protocol. Thus, cross-layer approaches will not operate on networks spanning multiple routing domains, when each domain uses a different routing mechanism (unless all of them integrate service messages). Therefore, integrated SD mechanisms require significant porting effort before they can operate in a hybrid wired-wireless network. On the other hand, an application layer solution is applicable regardless of lower-layer infrastructure.

III. DESIGN

ADDER provides a service discovery mechanism targeting military ad-hoc networks and has been optimised for message size and fast service acquisition. ADDER is made up of three components: i) a service description mechanism (Section III-A), ii) a discovery protocol over UDP - IPv6 (Sections III-B, III-C) and iii) a daemon (Section III-D). ADDER is hybrid proactive-reactive, it runs on the application layer and is fully distributed (directory-less).

In reactive mode, a node will form a service request message in order to discover services matching specific criteria. This message is broadcast to the node’s single-hop neighbours. Nodes receiving a service request will in turn forward it to their own single-hop neighbours. Upon reception of a request matching one of the services it has to offer, a node will generate a service response and send it to the originating node in a unicast IPv6 datagram. Intermediate nodes overhearing the response will cache the information in order to answer subsequent requests for the same service.

In proactive mode, nodes will periodically send service advertisement messages to their single-hop neighbours. Nodes receiving an advertisement store the information in
their service cache and include it in their own advertisements. Thus, information about a service slowly propagates through the network. Proactive mode can be enabled and disabled per node. Any number of nodes can have proactive mode enabled. The single-hop, "beacon"-based forwarding scheme for requests and proactive advertisements does not rely on a routing protocol in order to propagate information across the network. On the other hand, the unicast nature of responses implies that the network needs to be able to somehow establish a path between the responding and requesting node. However, if a path does exist, responses will reach their destination regardless of what routing protocol was used to establish it.

ADDER is inspired by distance vector routing protocols; service advertisements are tagged with a metric (distance) between the advertising node and the provider. To prevent loops and count-to-infinity, we use a feasibility condition similar to the one used by DSDV, AODV, babel [14], [15] and by the EIGRP routing protocol. This algorithm, which has been formally proven to be loop-free, relies on a sequence number which accompanies each service advertisement (destination sequence). This number is initialised and periodically incremented by the service providing node, while intermediate nodes forward advertisements without modifying it. The same sequence numbers are used to prevent multiple transmissions of the same message. This reduces message overhead and contributes to preserving node energy. Lastly, sequence numbers are used as part of ADDER’s state maintenance mechanism.

A. Service Description and State Maintenance

In ADDER, each service is uniquely identified across a deployment by a UUID. Bluetooth has been using UUIDs as service identifiers since the early versions of the specification. The same approach has also been adopted by other service discovery schemes for ad-hoc environments [7], [10], [13], even though the scheme documented in [7] uses short length (16-bit) UUIDs instead of the full 16 bytes.

A service description has two sections. The generic section contains information about attributes common to all services, such as the IPv6 address and port over which the service is accessible. It also contains the distance between the provider and the advertising node, the entry’s sequence number and a Time-To-Live (TTL) value. The second section is service-specific. It is an arbitrary list of key-value pairs aiming to define the details of each particular service. For the attributes section we have adopted plain ASCII key-value pairs (as opposed to the more verbose and computationally intensive XML documents or ontological service descriptions). While this means that ADDER lacks the ability to resolve sophisticated queries (e.g. keyword-based search), it gains a performance increase via reduced message size as we demonstrate in sec. IV-A.

ADDER uses a hybrid state maintenance mechanism, aiming to prevent erroneous (stale) advertisements of services when they are no longer available. It relies on soft Time-To-Live values for service entries while at the same time offering hard service de-registration functionality. In proactive mode, it is the provider’s responsibility to periodically re-advertise its services, alongside a new sequence number. If a cached service entry does not get refreshed within the specified TTL value, it gets flagged as stale. Optionally, a provider may send a service unavailable message to its single-hop neighbours, thus notifying that a specific entry should be deleted. This message will get forwarded across the network based on the forwarding mechanism discussed in a later section.

B. Protocol and Flooding Control

Existing service discovery solutions rely on flooding mechanisms in order to push information across the network. ADDER adopts a hybrid proactive-reactive approach and follows the same principle. In Figure 1, solid lines represent periodic (proactive) service advertisements and dotted lines represent service requests (reactive). The thick dashed line represents a unicast service response. Node A is the request source.

ADDER uses two message types: Service Advertisements and Service Requests. Both message types share a common header, which carries information about the message type and protocol version. The payload portion of a Protocol Data Unit (PDU) is either a list of service entries or a single request. A service entry belongs to one of three types: i) short, ii) delete or iii) full. Delete and short entries are always fixed size whereas full service entries are variable size. Table I outlines ADDER PDU sizes. A service is characterized by a set of attributes, all of which are listed in a full entry for that service. A full entry’s actual length is dependent on the exact number and length of attributes. Full advertisements include the entire list of attributes for a service, whereas short entries only include the generic section. This is done in order to achieve better scalability when the number and complexity of services in the network increases. For example, a short
entry can be used in proactive mode in order to refresh the TTL of a previously fully advertised service.

C. Probabilistic Forwarding and Discovery Scope

ADDER uses two separate mechanisms for flooding control, one for each mode of operation. The object of the forwarding mechanism is a different entity for each mode; in reactive mode, nodes forward requests. In proactive mode, nodes forward service entries.

Thus, in reactive mode, a service request will never get forwarded more than once per network interface. In proactive mode, the daemon investigates the feasibility of received service entries based on the destination-sequencing algorithm discussed at the start of this section. Each advertisement for a service $s$ with metric (distance) $m$ is decorated with a sequence number $f$. This sequence number is originally set and may be modified only by the provider. Sequence numbers are strictly monotonic and persistent; when a daemon is restarted by the user or due to an error, it must resume counting from the last sequence number it used before it was stopped. Thus, service advertisements are essentially a vector with three components $(s, m, f)$. Upon reception of a service advertisement, a node examines the metric and sequence number and decides if the advertisement is feasible.

Consider a node with information about a service $s$ in its cache. $s$ is tagged with metric $m$ and sequence $f$. A different service advertisement for the same service $s$ with metric $m'$ and sequence $f'$ is feasible if and only if:

$$ f' > f \text{ or } f' = f \text{ and } m' < m $$

In other words, an update for a service is feasible if the sequence number is greater (thus the advertisement is newer than the one in the cache) or if the sequence number is equal to the one in the cache but the service is advertised as being closer. If a service update is not feasible then it might be part of a loop and is discarded. It is important to note that the feasibility condition is sufficient but not necessary for loop-freedom. In other words, a feasible update is definitely not part of a loop. However, there can be loop-free updates which do not satisfy the condition.

The daemon stores feasible entries in the cache with i) the identifier of the interface from which each entry was received and ii) a reception time stamp. In the next advertisement interval the daemon forms outgoing messages according to the following rules:

- Services flagged as unavailable are included in the next message as delete entries.
- Services flagged as new or active may be included either as full or short entries. This is further discussed below.
- Services flagged as stale are not included.

In an ADDER deployment, there is an implicit limit on the maximum hops that queries and advertisements can travel. It is imposed by the hops field in service entries. Currently this field is a 4 bit unsigned integer, thus taking values in the range $[0 \ldots 2^4 - 1]$. The reasoning behind this design decision is two-fold; i) in a MANET environment, matches beyond this limit are going to be of limited use due to the distance between the requesting node and the service provider ii) in a mobile environment, by the time the advertisement has traversed its full distance, the provider might have changed location rendering the advertisement obsolete. Depending on the nature of the requested service, the originator may specify an even more limited scope for the lookup query. For instance, a user may issue a request for a service within 5 hops.

In proactive mode, nodes periodically exchange service information in the form of entries. A straightforward way to achieve this would be to include all entries in all advertisement messages. However, this approach would suffer from poor scalability with an increasing number of providers in the network; cache size would grow very fast and message sizes would rapidly increase beyond the network’s MTU. Therefore, ADDER adopts a probabilistic forwarding scheme. Entries eligible for forwarding (status new or active) have a probability $p$ of getting included in the advertisement. While probabilistic forwarding has positive impact on the mechanism’s scalability, it has negative impact on network convergence time. However, it does not have any impact whatsoever on query hit:miss ratio; if a service exists within the query’s hop limit and if a path exists between the provider and the querying node, any query for this service will result in a match. What changes is the identity of the node responding to the query; in the worst case scenario, if the entry has not been forwarded at all, the response will originate from the provider itself.

In the current implementation, each node uses a fixed $p$ value for all known entries (different nodes can use different $p$ values). However, using an adaptive $p$ on a per-service basis can greatly influence the mechanism’s performance. For instance, it can be argued that new entries should have
higher forwarding priority (higher $p$) than active ones, since they are not known to the network. It can also be argued that the $p$ for a service should decrease over time, since entries which have not been refreshed for a long period are an indication that the provider may have moved or left the network altogether. However, in scenarios with high mobility and high node churn rate, the aforementioned rationale would be flawed: High churn rate would result in many new services, very frequently. This would in turn result in older service entries not getting refreshed often enough and eventually becoming stale. Therefore, it can also be argued that very old services (nearing their TTL expiry) should be forwarded at higher priority (which is in direct contrast to the “decrease priority over time” approach). Those simple examples show us that it is not straightforward to select a $p$ value. As this is of increased importance, we are currently investigating various alternative algorithms to control the value of $p$ over time on a per-service basis. Theoretical analysis of possible solutions and an evaluation of their impact on ADDER’s performance is part of our future plans.

D. Daemon

The ADDER daemon has been implemented as a single-threaded process. During runtime, the daemon listens for and processes incoming advertisements and requests. If proactive mode is enabled, it will periodically form and transmit an advertisement. The daemon has been implemented in C for Linux. Original development took place on CentOS release 5.3 (Final). We also tested the code on Ubuntu 9.04 and 10.04. In addition, we are working on a client library which will allow developers to issue service requests and receive responses from within their applications. Intercepting unicast responses in intermediate nodes in order to cache the information contained therein has not been developed yet. We are planning to use the packet capture library (libpcap) in order to implement this.

IV. Evaluation

In this section we compare ADDER’s service description and message sizes to those used by other approaches. Subsequently, we present some performance evaluation results. Measurements have been obtained by actual execution of the daemon on two different test beds and are used as an indication of the mechanisms viability and scalability. We investigate scalability in terms of both network size (number of nodes) as well as number of services available in the network. We also investigate the potential performance gain when proactive mode is enabled.

We conducted our experiments on two different test beds: i) 8 personal computers interconnected through a gigabit ethernet switch. We configured virtual LANs on the switch and equipped the nodes with multiple NICs in order to avoid link contention. This configuration was mainly used in order to avoid wireless link errors during development, debugging and first stage evaluation. ii) A hybrid wired-wireless network formed by four Notebooks with 802.11g in ad-hoc mode. One of the notebooks was connected to a fifth node over fast ethernet. We use those two diverse test beds to observe the performance differences caused by the change of the underlying physical layer.

In both scenarios, nodes were positioned forming a line. By adjusting the transmit power on the notebooks and the distance between them, we made sure that the radio signal from each one could only reach its single-hop layer 3 neighbours. In all experiments conducted on the hybrid wired-wireless test bed, we observed discovery misses at a rate lower than 1%, caused by wireless packet losses.

A. Comparative Analysis of Message Sizes

In sec. III-A we argued that using key=value pairs for service attributes results in more compact messages than those produced by XML descriptions. In order to validate this, we expressed two services in the formats used by ADDER, OSDA [11], Konark [6] and SLP. The first service, called ‘simple’, is an imaginary service with zero attributes. The second is the ‘lightpath’ service, used as an example by Limam et al. in [11]. By using one service without attributes and another with multiple ones (8), we aim to demonstrate that the difference in message sizes diverges with an increasing number of attributes.

We chose to compare ADDER message sizes with OSDA, Konark and SLP for multiple reasons: i) The service descriptions in all those approaches can be broken down into a generic section, followed by multiple service-specific attribute entries ii) The works discussing OSDA and Konark disclose thorough details of service descriptions, with examples. It was thus feasible to obtain accurate information and express the exact same services in all four description formats iii) OSDA and Konark both use XML-based descriptions, SLP uses key=value pairs iv) All approaches are application layer v) Similar to ADDER, having been specifically designed for cross-domain discovery, OSDA is suitable for hybrid wired-wireless deployments.

Table II summarizes the sizes of the resulting descriptions and service advertisements for both services. The column labelled $Hdr$ refers to the header size of service advertisement messages. In Konark, service advertisements are carried inside HTTP POST messages, thus header length is highly
variable, ranging from tens to a few hundred bytes. The column labelled Gen lists the size of the generic sections. For Konark and SLP, this field’s length varies per service, since identifiers are arbitrary text fields and URLs respectively. For all cases, the numbers listed are the absolute minimum requirement for a meaningful description (e.g. in the case of SLP we have omitted all optional fields as well as security extensions).

In order to better understand the relation between the number of attributes and message sizes, we plot indicative trend lines in Figure 2. For simplicity and clarity, the lines have been drawn assuming that each additional attribute would change the message size by a fixed increment. However, since attribute lengths are variable in all four mechanisms, the exact increase in message size for each additional attribute will also vary between occasions. More in-depth analysis of the mechanics of the four approaches reveals that, if \( N \) and \( V \) denote the number of bytes required to encode an attribute’s name and value respectively, then: Konark message sizes increase by \( N + V + 65 \) bytes per additional attribute, where 65 bytes is the necessary XML markup. Similarly, ADDER adds \( N + V + 2 \) bytes per attribute, SLP adds \( N + V + 6 \) bytes and OSDA adds \( 4N + 2V + 12 \) bytes. Additionally, it is trivial to prove that the increase in size is always smaller for ADDER, for any value of \( N \) and \( V \) (under the limitation \( N + V + 1 \leq 255 \), which is ADDER’s maximum length per attribute). Thus, in a real scenario the trends depicted in Figure 2 would not be perfectly linear and the break-even point between Konark and OSDA would be at different co-ordinates. However, the relative positions of the four lines would remain unchanged, making Figure 2 a good depiction of reality.

The results show that the overhead imposed by ADDER’s key=value attribute pairs is lower than that of SLP’s, which in turn is lower than that of XML-based descriptions (OSDA and Konark). While adding more attributes increases the semantic richness of service information, the resulting description sizes diverge even further. Additionally, using URLs as service identifiers also results in longer service descriptions, compared to those generated by mechanisms using fixed-length UUIDs.

### B. Impact of Hop Count on Acquisition Time

With proactive mode turned off on all nodes, we injected service queries into the network and measured service acquisition time. We started with the provider being a single hop away from the requesting node. We gradually increased the distance by 1 hop and performed the same experiment up to a maximum of 7 hops (4 hops in the hybrid testbed).

The dashed line with crosses in Figure 3 is a simple linear regression line based on 35,000 samples, displaying the impact of hop count on acquisition time in reactive mode. The main observation of interest in this experiment is the behavior of response time as a result of increasing query diameter. The estimated line demonstrates very good fit to the sample; the \( R^2 \) (squared correlation coefficient) value was 0.9854, meaning that the linear model explains approximately 98.5% of the variance in the dependent variable (acquisition time). Thus, with increasing distance between the querying node and the provider, service acquisition time increases linearly. This is an indication of good scalability and was observed on the wired testbed at the very early evaluation stage.

Subsequently, we enabled proactive mode on all nodes and waited for the network to converge. We then repeated the same experiment and obtained a new sample. The triangles in Figure 3 represent average response times per hop. As anticipated, with proactive mode enabled, response times average around the same value regardless of the distance between the querying node and the service provider.

The results from the same experiments in the hybrid topology are plotted in the same figure with solid lines. Notice how service acquisition time is, again, distance-independent (circles) with proactive mode. Lastly, we observe that each WiFi hop increases acquisition time considerably in reactive
C. Number of Services and Forwarding

When proactive mode is enabled, nodes try to advertise entries in their cache to their neighbours. They do so by forming periodic messages, each one containing multiple (full or short) service entries. As the number of services known to the network increases, so will the size of advertisements. Including every entry in every message would eventually lead to scalability issues; with an increasing number of services in the network, messages would keep growing in size until they got capped by the physical medium’s maximum packet size.

As discussed in section III-C, ADDER adopts a probabilistic forwarding scheme, in an attempt to reduce message size and achieve better scalability in networks with a high number of service providers. Even with probabilistic forwarding, the network’s MTU is still a limiting factor. However, it takes a considerably higher number of services before packet sizes start approaching this threshold. Furthermore, overall traffic injected to the network per node decreases (Figure 4).

With probabilistic forwarding, each entry has probability $p$ to get included in the body of a service advertisement. In this experiment, we set the value of $p$ to 0.2. We gradually increased the number of services present in the network and measured advertisement message sizes. Each circle in Figure 4 corresponds to the average observed message size (100 samples per X-axis point). The line labelled “0.2 (Estimated)” represents the average message size which we anticipated before running the experiment. The average observed values (circles) turned out to be indeed very close to our anticipations, almost entirely covering the line in the chart. The lines labelled “0.5 (Estimated)” and “Full” indicate how the slope would increase if one chose a higher value for $p$. By choosing $p$ values 0.2, 0.5 and 1 we aim to cover a wide spectrum of configurations. The results presented here will be extended once we have concluded our investigation on adaptive, per-service $p$ values (sec III-C).

D. Propagation and Discovery of New Services

When a new provider joins the network, information about its services slowly becomes available to other nodes. Queries can initially only be answered by the provider itself. Thus, they have to traverse a potentially long path between the two nodes. With proactive mode enabled, this information will slowly become known to other nodes throughout the network. Subsequent queries will thus return a match before reaching the provider.

We positioned the provider and the querying node 7 hops away from each other and we started the experiment with proactive, probabilistic forwarding and with a value of 0.2 for $p$ (4 hops on the hybrid test bed). Initially, queries would traverse the entire network, thus service acquisition was slow. Over the course of time (increasing values in the X axis in the plot), information about services was propagated throughout the network and was stored in the cache of intermediate nodes. Responses started arriving faster. Each point in Figure 5 marks the round trip time for each query-response (measured on the requesting node) over the course of time.

Notice that in proactive mode, the requesting node might already have information in its cache, in which case the PDU would not leave the node at all, resulting in almost instantaneous, zero-hop acquisition. Since this does not add significant value, in this figure we only plot acquisition times for requests which actually entered the network.
Figure 5. Service acquisition times during network convergence.

V. CONCLUSIONS - FUTURE WORK

We are planning to extend our evaluation in a larger scale wireless test bed. Additionally, comparisons with other approaches are going to be conducted in a simulated environment, where we can also evaluate ADDER in very large scale deployments and validate its behaviour in mobile scenarios.

ADDER is currently hybrid; it always operates in reactive mode, while proactive mode can be turned on optionally on all or part of the nodes. As part of our future work, we are planning to make it adaptive. Proactive mode will be turned on and off automatically at runtime, depending on node speed, mobility and energy. For example, on fixed nodes with “unlimited” supply of energy, proactive mode will be automatically enabled. On the other hand, low power or high mobility nodes will be able to automatically switch off proactive mode without user intervention.

Lastly, we are currently investigating various alternative algorithms for the adaptation of $p$ over time on a per-service basis and evaluating their impact on ADDER’s performance.

REFERENCES


