Proxy support for service discovery using mDNS/DNS-SD in low power networks

Milosh Stolikj, Richard Verhoeven, Pieter J. L. Cuijpers, and Johan J. Lukkien,
Dept. of Mathematics and Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

***Abstract*—We present a solution for service discovery of resource constrained devices based on mDNS/DNS-SD. We extend the mDNS/DNS-SD service discovery protocol with support for proxy servers. Proxy servers temporarily store information about services offered on resource constrained devices and respond on their behalf while they are not available. We analyze two protocols for the delegation mechanism between a service provider and a proxy server: an active proxy protocol, as used in the mDNS/DNS-SD implementation by Apple, and a new, passive proxy protocol. We implement and simulate both approaches. Based on the delay and energy usage, we show that the second approach converges faster, thus saving more energy by allowing the resource constrained device to be turned off earlier.**

I. Introduction

The Internet of Things (IoT) is a vision of a global, dynamic network of heterogeneous devices, connected via standards based protocols. The IoT will connect virtually every electronic device, from desktop machines to ubiquitous sensors and actuators. Therefore, significant effort has been made towards the specification and adaptation of software protocols which can be supported by all devices. For instance, since the IPv6 standard for end to end connectivity is not applicable for low power networks, a specific adaptation layer (6LowPAN) has been developed and standardized. Such protocols are intended to be run even on the low end spectrum of the IoT, where devices have little memory, low cost/low data rate radio chipsets and limited battery life.

Service oriented computing has been seen as a potential programming paradigm for the IoT [1]. At the simplest level, resources on low power devices can be exposed to the local or global network in the form of services: ex. services for reading sensor measurements or actuating switches. These services can be further composed in more complex services, to provide new functionality. The successful realization of this vision relies on the availability of a common service discovery protocol. Furthermore, using a standards-based service discovery protocol will ease the burden of integration of heterogeneous devices. Previous research [5] [6] has identified the Multicast Domain Name System (mDNS) [2] with DNS based Service Discovery (DNS-SD) [3] as a potential protocol for service discovery in the IoT, and this protocol is the focus of this paper.

mDNS is an extension of the DNS protocol for local area networks. The main differences with the original DNS protocols are: 1) naming information is stored locally, on each node in the network; 2) queries are sent multicast instead of unicast; and 3) each node directly responds to queries. Similarly, DNS-SD is an extension of DNS which uses DNS resource records to describe services in a domain-name like fashion. The service description can further specify the service protocol and additional context information.

This work is supported in part by the Dutch P08 SenSafety Project, as part of the COMMIT program.

Resource constrained, battery powered devices can not efficiently participate in the mDNS/DNS-SD protocol due to their large off-line periods. Battery powered devices use low radio duty cycles to increase their life time, while mDNS/DNS-SD assumes devices to be constantly online. Therefore, if a query for a service hosted at a battery powered device comes while the device is off-line, the service will not be detected.

In this paper, we present an extension to the mDNS/DNS-SD protocol for resource constrained devices. We enable devices to be unavailable for longer periods of time, yet still remain discoverable, by delegating their role in the service discovery protocol to proxy servers. We describe two protocols for the delegation: an active proxy protocol, as used in the mDNS/DNS-SD implementation by Apple, and a new, passive proxy protocol. In the active proxy protocol, the service provider searches and selects the proxy server for delegation. In the passive proxy protocol, the service provider only signals its intention that it requires a proxy service: the proxy server picks up and processes the intention on its own. We implement both protocols in the Contiki [4] operating system. Finally, we compare the two protocols in terms of code size, memory requirements, energy usage and delay.

The paper is structured as follows. Section II covers related work on service discovery. Section III introduces the mDNS/DNS-SD protocol, while the protocol extensions are described in section IV. Section V reports simulation results of the performance of both extensions in a single-hop network. Finally, section VI gives a summary of the presented work.

II. Related work

Related research in service discovery can be divided into two groups: generic service discovery protocols and their application to resource constrained devices, and service discovery protocols directly targeted at low power networks.

mDNS/DNS-SD is a widely used standard for service discovery in local area networks. It has been implemented in all major operating systems for high capacity devices, as well as in resource constrained devices [5] [6]. Further optimizations of the protocol have been proposed [7] to reduce message sizes in order to support low power networks better. These changes do not capture the always-on requirements of the protocol, which is investigated in this paper.

The Service Location Protocol (SLP) [8] is another standardized service discovery protocol for local area networks. SLP has been designed to scale from small, decentralized networks to large corporate networks. A 6LowPAN adaptation of SLP has been proposed [9], but it now seems abandoned. Furthermore, the protocol has been extended with proxy agents [10], for connecting high and low power networks, and context support such as proximity services [11].

The Simple Service Discovery Protocol (SSDP) [12] is part of the Universal Plug and Play (UPnP) specification, which is a commonly used software stack for consumer electronics devices. The protocol itself is rather heavy in terms of message sizes. Therefore, like with SLP, the ports of SSDP to low power networks rely on application level gateways [13], which translate messages between the high and low power networks.

Many service discovery protocols have been designed with resource constrained devices in mind. These protocols can operate in single-hop [14], centralized [16], hierarchical [15] or 6LowPAN [17] networks. Even though such protocols optimize the performance on low power networks, the interpretability with high power networks becomes reliant on application level gateways. In order to provide seamless integration, reduce the translation overhead and preserve end-to-end transparency, such gateways should be minimized in the IoT, which limits the usability of such specific protocols.

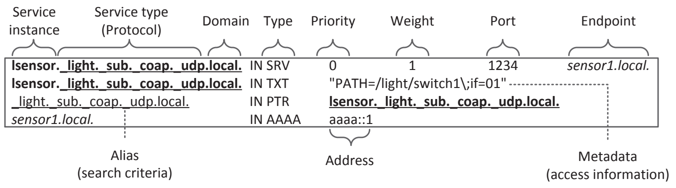
III. mDNS/DNS-SD SERVICE DISCOVERY

mDNS and DNS-SD form the core service discovery protocol in Bonjour, a zero-configuration implementation by Apple. The protocol consists of two components: a communication protocol defined by mDNS, and a service discovery and service description protocol defined by DNS-SD.

mDNS is an extension over the unicast Domain Name System (DNS [18]) for name resolution in local area networks. Names can refer to addresses, as in classic DNS, or to services, using DNS-SD. Therefore, mDNS exclusively resolves host names ending with the *.local* top level domain. The packet structure in mDNS is similar to the one defined in the DNS protocol. The main difference between the two comes in the message exchange protocols: mDNS foresees a fully distributed environment, where resolution information is stored locally on each device within a small network, and each device directly answers to incoming name resolution queries. In that sense, every participating device acts both as a server and a resolver. As a result, multicast messaging is used to effectively distribute both queries and responses.

DNS-SD is a standard which enables DNS clients to discover named instances of a given service using DNS resource records (RR). A service instance in DNS-SD is described using SRV, TXT, PTR and A/AAAA RRs (Figure 1), connected by the service instance name. The SRV RR holds information of the name of the service instance, service type (protocol), time to live, priority, weight, port number, and the endpoint. The priority and weight parameters give preference when the same service is hosted by multiple instances. The endpoint parameter is in the form of a host name, which is resolved to an IP address by an A/AAAA RR. The TXT RR contains the service metadata in the form of *[key]:[value]* pairs. The exact content depends on the protocol used, and can include an URI path for a specific resource, invocation parameters, more specific service description etc. Finally, PTR RRs provide the mapping between service types and service instances.

In mDNS/DNS-SD, all RRs describing a service instance are stored on the node providing the service. The discovery of a service is shown in Figure 2.



```
lsensor._light._sub._coap._udp.local.  IN SRV    0    1    1234    sensor1.local.
lsensor._light._sub._coap._udp.local.  IN TXT    "PATH=/light/switch1\;if=01"
_light._sub._coap._udp.local.          IN PTR    lsensor._light._sub._coap._udp.local.
sensor1.local.                         IN AAAA   aaaa::1
```

Fig. 1. DNS-SD description of a light sensor service. The four resource records are connected through the name of the service.

mDNS in combination with DNS-SD enables easier management and deployment of dynamic networks. Since there is no need to configure separate DNS servers, auto-configuration is easier. Furthermore, adding new devices to the network is trivial, since the new devices can discover and advertise network services independently.

Since mDNS/DNS-SD had been designed for service discovery in local area networks of high capacity devices, it is not directly applicable to networks of resource constrained devices. For example, due to memory limitations alone, it is impossible to reuse existing implementations of the protocols. Table I shows the memory profile of several implementations of mDNS/DNS-SD, including our own optimised implementation, for different device types. From the table, it is clear that the original Bonjour implementation cannot fit on resource constrained devices. Even the Arduino port, which is already a feature-limited implementation of mDNS/DNS-SD, is too large to fit on small factor devices.

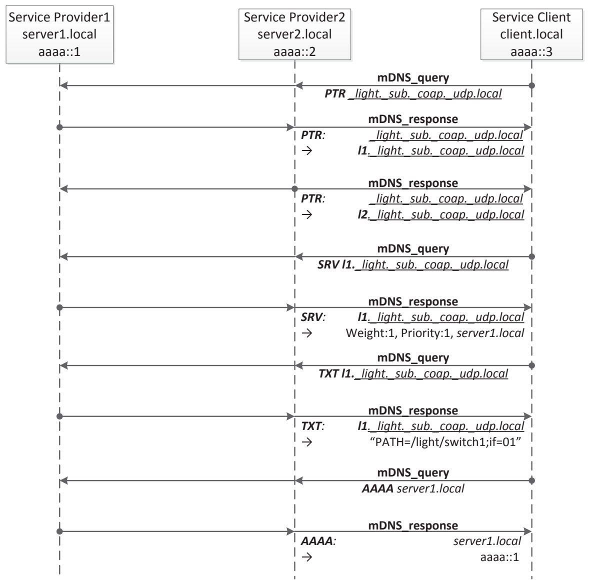
TABLE I
CODE AND MEMORY FOOTPRINT OF DIFFERENT mDNS/DNS-SD IMPLEMENTATIONS

Implementation	Code	Memory
Bonjour by Apple	500KB [1]	/
Ethernet Bonjour for Arduino	14KB	/
uBonjour for Contiki [5]	7.69KB	0.4KB
mDNS/DNS-SD for Contiki [2]	6.51KB	0.7KB

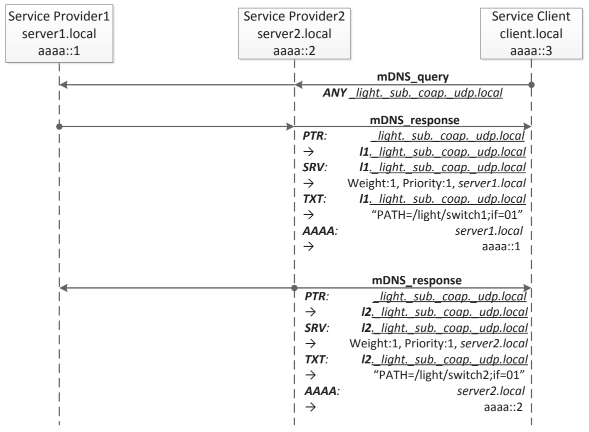
[1] Based on the size of mDNSResponser.exe on 64 bit platforms. Memory information is unavailable.
[2] Available at https://github.com/mstolikj/contiki

IV. PROXY SUPPORT FOR SLEEPING NODES

One of the drawbacks of practical implementations of mDNS/DNS-SD is that devices are assumed to be constantly


Service Provider1
server1.local
aaaa::1
Service Provider2
server2.local
aaaa::2
Service Client
client.local
aaaa::3
mDNS_query
PTR _light._sub._coap._udp.local
mDNS_response
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
mDNS_response
PTR: _light._sub._coap._udp.local
→ l2._light._sub._coap._udp.local
mDNS_query
SRV l1._light._sub._coap._udp.local
mDNS_response
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
mDNS_query
TXT l1._light._sub._coap._udp.local
mDNS_response
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01"
mDNS_query
AAAA server1.local
mDNS_response
AAAA: server1.local
→ aaaa::1

(a) Service discovery without packet fragmentation


Service Provider1
server1.local
aaaa::1
Service Provider2
server2.local
aaaa::2
Service Client
client.local
aaaa::3
mDNS_query
ANY _light._sub._coap._udp.local
mDNS_response
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01"
AAAA: server1.local
→ aaaa::1
mDNS_response
PTR: _light._sub._coap._udp.local
→ l2._light._sub._coap._udp.local
SRV: l2._light._sub._coap._udp.local
→ Weight:1, Priority:1, server2.local
TXT: l2._light._sub._coap._udp.local
→ "PATH=/light/switch2;if=01"
AAAA: server2.local
→ aaaa::2

(b) Service discovery with packet fragmentation

Fig. 2. Resolving a service using mDNS/DNS-SD. First, the resolver needs to find service instances of the requested service, provided by PTR RR. Then, the actual service is resolved, through SRV and TXT RRs. Finally, the host providing the service is resolved via A/AAAA RRs. Depending on the implementation, the four RRs can be distributed independently, in separate packets (a), or can be packed into one larger packet (b). The former is easier to implement, but the latter results in smaller payloads. Using DNS name compression, the four RRs fit in 158 bytes. On a 128 byte frame as, for example, used in 802.15.4, the four compressed RR's would be fragmented in two frames. Note: all messages are multicast and all nodes belong to the same broadcast domain.

online. This comes from the distributed nature of the protocol: if a device is not online when a query for one of its services arrives, it will not be able to respond to it, thus its services will be undetectable. Additionally, in order to be able to quickly adapt to network changes, such as devices leaving from the network, services are advertised with relatively short time-to-live intervals (2 minutes on LANs). Frequent messaging is an unwanted feature for low power networks due to the limited battery life of the participating devices. As illustrated in [19], this behaviour introduces a trade-off between signaling frequency, i.e. increased traffic in the network, and the risk of discovering non-existing services.

In order to use mDNS/DNS-SD for service discovery in sensor networks, the protocol has been adapted to accommodate devices with low duty cycles, or so called sleeping service providers (SSP). One approach to facilitate SSPs is to introduce proxy servers on high capacity (non battery powered) devices. Proxy servers are responsible for taking some of the workload from the SSPs which they serve. In this paper, we focus only on proxy support for service advertisements. We consider two mechanisms for the delegation protocol between the SSP and the proxy server: active and passive proxy protocol. The distinction is based on the role of the sleeping device in the proxy selection phase. Both approaches are explained in the forthcoming sections.

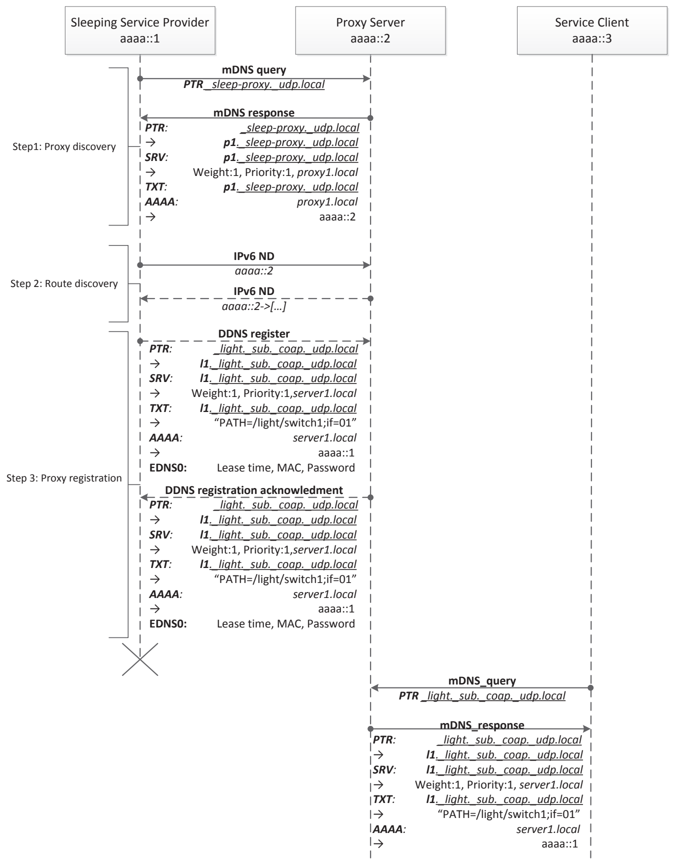
A. Active proxy

Using an active proxy protocol, the SSP is the entity which selects the proxy server to be used. The protocol is described in Figure 3. The SSP initiates the protocol by first searching for a proxy server. After the SSP has selected a suitable proxy server, it first finds a route to it it, and then registers with it. Depending on the response received from the proxy server, the SSP begins its sleep cycle or tries to register to a different proxy server. The registration protocol itself is outside of the mDNS/DNS-SD specification and varies between implementations.

We follow the Bonjour implementation by Apple as an active proxy protocol. The Bonjour active proxy is available as another service in the network, advertised as a *_sleep-proxy._udp* service. By reverse engineering the Bojour protocol we discovered that the SSP registers with the proxy server by sending an unicast dynamic DNS update packet (DDNS) [20], which contains all RRs which should be hosted at the proxy, and an EDNS0 RR [21] which specifies the lease time of the proxy hand-off [22], and ownership information of the SSP [23]. The ownership information is used to transfer the MAC addresses to the proxy server. This address is used to both intercept messages destined for the service provider while it is not available, as well as for waking up the service provider. The proxy server always returns a unicast response, which informs the SSP whether the proxy request was accepted.

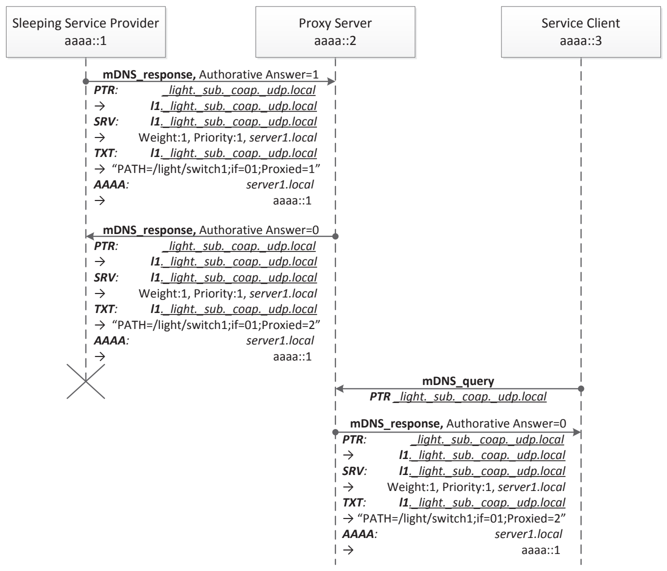
B. Passive proxy

The new passive proxy protocol reduces the number of messages that need to be sent by embedding the service advertisement with the proxy registration. As shown in Figure 4, the SSP adds a parameter in the TXT RR of its service which signals the intent that the service advertisement should be processed by a proxy server. The server that decides to serve



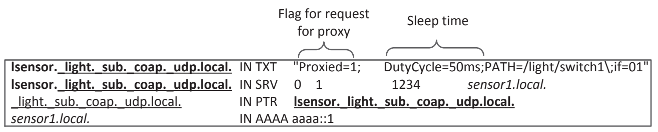
Sleeping Service Provider aaaa::1
Proxy Server aaaa::2
Service Client aaaa::3
Step1: Proxy discovery
mDNS query
PTR _sleep-proxy._udp.local
mDNS response
PTR: _sleep-proxy._udp.local
→ p1._sleep-proxy._udp.local
SRV: p1._sleep-proxy._udp.local
→ Weight:1, Priority:1, proxy1.local
TXT: p1._sleep-proxy._udp.local
AAAA: proxy1.local
→ aaaa::2
Step 2: Route discovery
IPv6 ND
aaaa::2
IPv6 ND
aaaa::2->[...]
Step 3: Proxy registration
DDNS register
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1,server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01"
AAAA: server1.local
→ aaaa::1
EDNS0: Lease time, MAC, Password
DDNS registration acknowledment
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1,server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01"
AAAA: server1.local
→ aaaa::1
EDNS0: Lease time, MAC, Password
mDNS_query
PTR _light._sub._coap._udp.local
mDNS_response
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01"
AAAA: server1.local
→ aaaa::1

Fig. 3. Active proxy protocol in mDNS/DNS-SD. The service provider selects a proxy service and registers to it. Afterwards, the proxy server responds on behalf of the service provider. Note: full lines portray multicast messages; dashed lines portray unicast messages.



Sleeping Service Provider aaaa::1
Proxy Server aaaa::2
Service Client aaaa::3
mDNS_response, Authorative Answer=1
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01;Proxied=1"
AAAA: server1.local
→ aaaa::1
mDNS_response, Authorative Answer=0
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01;Proxied=2"
AAAA: server1.local
→ aaaa::1
mDNS_query
PTR _light._sub._coap._udp.local
mDNS_response, Authorative Answer=0
PTR: _light._sub._coap._udp.local
→ l1._light._sub._coap._udp.local
SRV: l1._light._sub._coap._udp.local
→ Weight:1, Priority:1, server1.local
TXT: l1._light._sub._coap._udp.local
→ "PATH=/light/switch1;if=01;Proxied=2"
AAAA: server1.local
→ aaaa::1

Fig. 4. Passive proxy protocol in mDNS/DNS-SD. The service provider indicates that it wants to be served by a proxy server in the service advertisement. This request is processed by the proxy server, and from there on, it starts responding on behalf of the service provider. The request is acknowledged by re-sending the advertisement. All messages are multicast.



```
lsensor._light._sub._coap._udp.local.  IN TXT   "Proxied=1;  DutyCycle=50ms;PATH=/light/switch1\;if=01"
lsensor._light._sub._coap._udp.local.  IN SRV   0  1         1234        sensor1.local.
_light._sub._coap._udp.local.          IN PTR   lsensor._light._sub._coap._udp.local.
sensor1.local.                         IN AAAA  aaaa::1
```

Fig. 5. Embedded registration request for the passive proxy protocol.

the request acknowledges the delegation by re-sending the service advertisement. The distinction between advertisements originating from the service provider and cached advertisements from the proxy server is done using the Authoritative Answer bit (AA). The purpose of the re-transmission is two fold: 1) the SSP knows that someone has handled its request and can go to sleep; and 2) other proxy servers know that they need not process that advertisement. The protocol can be further optimized by re-transmitting only the first SRV RR in order avoid unnecessary distribution of large (fragmented) messages. The SRV RR is unique to the SSP and can be undoubtedly interpreted by the SSP and by other proxy servers.

The proxy server needs to know the duration of the sleep cycle of the SSP. With the active proxy protocol, this information is sent directly to the proxy server within the registration message, in the form of the lease time. With the passive proxy protocol, it has to be added within the advertisement message. This information, along with the request to be proxied, can be added as additional parameters of the service that are added within the TXT description of the service advertisement (Figure 5). Similarly, other required parameters, such as the MAC address, can be transferred. The compact nature of these two descriptions is of paramount importance since large service advertisements can lead to packet fragmentation.

V. Evaluation

We compare the performance of the active and passive proxy protocols by simulating a set of nodes. We are interested in memory footprint, the delay and energy consumption for registering a service with a proxy server.

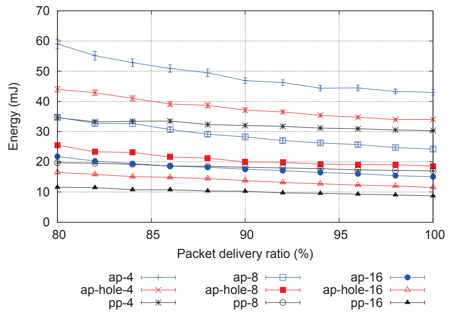
A. Memory footprint

We implemented both protocols for proxy registration in the Contiki operating system [4]. The size of the modules, compiled using the msp430-gcc 4.5.3 compiler for the Tmote Sky sensor motes [24], are shown in Table II.

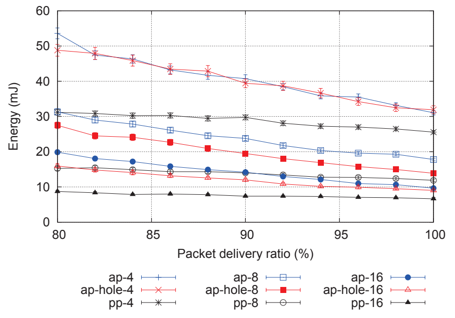
Both the passive proxy client and the proxy server are smaller than the corresponding active proxy components. This difference is due to the additional complexity required for implementing the DDNS protocol for the proxy registration. Even though the packet format is similar in DDNS and mDNS, additional resources are used for establishing the connection for the DDNS update protocol.

B. Simulation

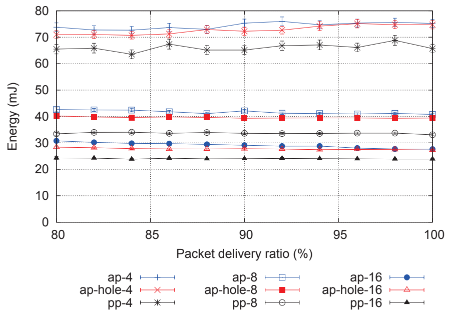
We simulated a single hop scenario in Cooja [26], a cross-level simulator for the Contiki operating system. Cooja internally uses the MSPsim device emulator for cycle accurate Tmote Sky emulation, as well as a symbol accurate emulation of the CC2420 radio chip. The test network consisted of three Tmote Sky nodes - a sleeping service provider, a proxy server



(a) Sleeping Service Provider



(b) Proxy Server



(c) Overhearing Node

Fig. 6. Energy usage for completing the active and passive proxy protocol. *ap* and *pp* refer to the active/passive proxy protocol, with 4, 8 and 16Hz wake-up intervals for the ContikiMAC radio duty cycling protocol. *ap-hole* refers to the optimized version of the active proxy protocol. The energy usage of the overhearing node (c) is measured for the entire simulation duration.

TABLE II
CODE AND MEMORY FOOTPRINT OF DIFFERENT COMPONENTS FOR PROXY REGISTRATION, IN ADDITION TO THE MDNS/DNS-SD IMPLEMENTATION (BYTES)

Component	Proxy protocol	Code	Memory
Service provider	active	1.484	66
Proxy server	active	1.268	452
Service provider	passive	1.136	90
Proxy server	passive	902	432

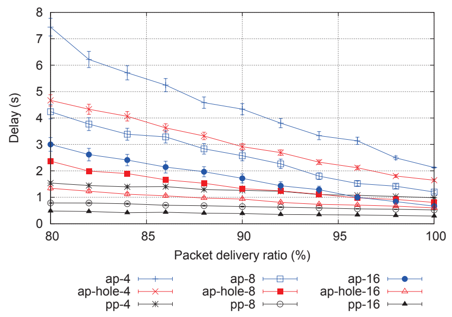


Fig. 7. Required time (delay) for completing the active and passive proxy protocol, as measured by the service provider. *ap* and *pp* refer to the active/passive proxy protocol, with 4, 8 and 16Hz wake-up intervals for the ContikiMAC radio duty cycling protocol. *ap-hole* refers to the optimized version of the active proxy protocol.

(PS), and a dummy node, all located in the same single-hop 6LoWPAN network. The SSP advertises one service, described using four RRs: PTR, SRV, TXT and AAAA. All four RRs are stored in a single DNS packet, which is then fragmented into two 802.15.4 frames for transport. All nodes use the CSMA-CA MAC protocol together with the ContikiMAC [25] radio duty cycling protocol (RDC) with wake-up frequencies of 4, 8 and 16 Hz, which results in wake-up intervals of 250ms, 125ms and 62.5ms, accordingly.

The simulation starts with all nodes being online. At second 5, the SSP advertises its service. At second 7, the SSP starts registering with the PS using one of the previously described protocols. After the registration finishes, the SSP turns off its radio. The simulation is stopped when the proxy registration completes. The dummy node does not participate in the proxy protocol, but it overhears all traffic in the network. We use it to show the impact of the protocols to nodes in the vicinity.

Due to packet loss, the proxy registration may not succeed in its first iteration. Therefore, we implemented repetitions of individual stages of the proxy protocols based on the expiry of fixed timeouts. The timeout for competition of step 1 from the active proxy protocol and the entire passive proxy protocol is set at 5 times the RDC wake-up interval. The repetition of step 2 of the active proxy protocol is dictated by the neighbour discovery protocol, and the timeout is fixed at 10 seconds. Finally, the timeout of the entire step 3 is set at 2 seconds. This should be enough to capture any retransmissions of unicast frames by the CSMA-CA protocol.

The simulations were executed using a constant loss rate model for radio propagation, with varying packet delivery ratio between 80 and 100%, at 2% increments. The charts show the mean values of 1.000 runs, and the error bars correspond to the 95% confidence interval of the mean.

We compared the performance of the active (**ap**) and passive proxy (**pp**) registration protocol. To verify the impact of the neighbour discovery protocol on the active proxy protocol, we also implemented an optimized version of the active proxy protocol (**ap-hole**), where the neighbour discovery stage (step 2 from Figure 3) is skipped. In this optimised version, the link layer address of the proxy server is generated from the last 8 bytes of the IPv6 address, present in the AAAA RR of the *_sleep-proxy* service description.

Figure 7 shows the delay, i.e. the time elapsed from starting the proxy registration protocol until its completion. As expected, increasing the wake-up interval results in much higher delays. With all three different wake-up parameters, due to the smaller number of messages, the passive proxy protocol finishes much faster than the active proxy protocol. The improvements vary depending on the channel check interval rate and the radio error rate, from 2 to 6 fold. The optimized active proxy protocol also converges faster, up to 2 fold. The difference between the two versions grows as the packet delivery ratio drops. Still, even the optimized version requires more time to complete than the passive proxy protocol.

Figure 6 shows the energy usage of the SSP and the proxy server during the proxy registration. These measurements were profiled using the software power profiler Powertrace [27]. Contrarily to expectations, increasing the wake-up interval actually increases the energy usage for the proxy server and the SSP. This behaviour is due to the radio duty cycling protocol: ContikiMAC makes senders do more work than receivers.

The energy usage of the passive proxy protocol is lower compared to the active proxy protocol, though the differences are not as high as with the delay. We attribute this behaviour to the efficiency of the ContikiMAC protocol. Furthermore, in the active proxy protocol, if a message loss occurs during the neighbour discovery phase, the sender will be silent for the entire timeout period, which introduces a large delay, but not much energy usage. This is visible in the energy usage of the active proxy protocol and the optimized version with a 4Hz wake-up interval. Even though the difference between the two in terms of delay is large, they consume similar amounts of energy at the proxy server side.

Finally, Figure 6c shows the average energy usage of the dummy node in the network. The energy usage is measured during the entire simulation, and includes the mDNS initialization stage, where every node advertises its host name and address. The passive proxy protocol requires less energy due to the smaller number of messages during both the initialization phase and registration phase. The passive proxy is silent during the initialization phase, while the active proxy advertises the *_sleep-proxy* service. Surprisingly, the neighbour discovery does not significantly impact the active proxy protocol in this aspect. Namely, the energy footprint of the active proxy protocol is close to the optimized version.

VI. Conclusion

In this paper we proposed an extension to the mDNS/DNS-SD protocol for service discovery for low power networks of resource constrained devices with the help of proxy servers. We presented two protocols for the delegation mechanism between the service provider and the proxy server. In the first, active proxy protocol, the service provider actively searches and selects the proxy server for delegation. In the second, passive proxy protocol, the service provider only signals its intention that it requires a proxy service: the proxy server picks up and processes the intention as a registration request. Both protocols were implemented in the Contiki operating system.

The simulations show that in a single hop network, the passive proxy protocol converges faster, requires less energy and is smaller in code size when compared to the active proxy protocol. The improvements vary depending on the MAC protocol behaviour and the noise in the wireless medium. With a sender-initiated radio duty cycling layer, the passive proxy protocol can finish on average up to six times faster than the active proxy protocol, consuming half of the energy.

For future work, we plan to evaluate the behaviour of both protocols in a multi-hop network. The underlying multicast protocol then plays an important role, and emphasizes the trade-off between multicast and unicast traffic.

References

[1] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, “Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services,” *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 223–235, Jul. 2010.

[2] S. Cheshire and M. Krochmal, “Multicast DNS,” RFC 6762, IETF, 2013.

[3] S. Cheshire and M. Krochmal, “DNS-Based Service Discovery,” RFC 6763, IETF, 2013.

[4] A. Dunkels, B. Grnvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Work. Embedded Networked Sensors*, ser. Emnets-I, 2004.

[5] R. Klauck and M. Kirsche, “Bonjour contiki: a case study of a dns-based discovery service for the internet of things,” in *Conf. Ad-hoc, Mobile, and Wireless Networks*, ser. ADHOC-NOW, 2012, pp. 316–329.

[6] J. Schoonwalder, T. Tsou, and B. Sarikaya, “Protocol profiles for constrained devices,” in *Work. Interconnecting Smart Objects with the Internet*, 2011.

[7] R. Klauck and M. Kirsche, “Enhanced DNS message compression - optimizing mDNS/DNS-SD for the use in 6LoWPANs,” in *Work. Sensor Networks and Systems for Pervasive Computing*, ser. PerSeNS, 2013.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service Location Protocol, Version 2,” RFC 2608, IETF, 1999.

[9] K. H. Kim, W. A. Baig, S. W. Yoo, S. D. Park, and H. Mukhtar, “Simple service location protocol (sslp) for 6lowpan,” IETF, 2010.

[10] S. A. Chaudhry, W. D. Jung, A. H. Akbar, and K.-H. Kim, “Proxy-based service discovery and network selection in 6lowpan,” in *High Performance Computing and Communications*, ser. LNCS, 2006, vol. 4208, pp. 525–534.

[11] S. H. Chauhdary, M. Cui, J. H. Kim, A. K. Bashir, and M.-S. Park, “A context-aware service discovery consideration in 6lowpan,” in *Conf. Convergence and Hybrid Information Technology*, 2008, pp. 21–26.

[12] A. Presser and et al, “Upnp device architecture 1.1,” 2008.

[13] R. Bosman, J. J. Lukkien, and R. Verhoeven, “Gateway architectures for service oriented application-level gateways,” *IEEE Tran. Consumer Electronics*, vol. 57, no. 2, pp. 453–461, 2011.

[14] M. Nidd, “Service discovery in deapspace,” *Personal Communications*, vol. 8, no. 4, pp. 39–45, 2001.

[15] R. Marin-Perianu, H. Scholten, P. Havinga, and P. Hartel, “Energy-efficient cluster-based service discovery in wireless sensor networks,” in *Conf. Local Computer Networks*, 2006, pp. 931–938.

[16] T. Ozcelebi, J. J. Lukkien, R. Bosman, and O. Uzun, “Discovery, monitoring and management in smart spaces composed of low capacity nodes,” *IEEE Trans. Consumer Electronics*, vol. 56, no. 2, pp. 570–578, 2010.

[17] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, “Trendy: An adaptive and context-aware service discovery protocol for 6lowpans,” in *Work. Web of Things*, ser. WoT, 2012.

[18] P. Mockapetris, “Domain names - concepts and facilities,” RFC 1034, IETF, 1987.

[19] M. Tjiong and J. J. Lukkien, “On the false-positive and false-negative behavior of a soft-state signaling protocol,” in *Conf. Advanced Information Networking and Applications*, ser. AINA, 2009, pp. 971–979.

[20] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, “Dynamic Updates in the Domain Name System (DNS UPDATE),” RFC 2136, IETF, 1997.

[21] J. Damas, M. Graff, and P. Vixie, “Extension Mechanisms for DNS (EDNS(0)),” RFC 6891, IETF, 2013.

[22] S. Cheshire and M. Krochmal, “Dynamic dns update leases,” IETF, 2006.

[23] S. Cheshire and M. Krochmal, “Edns0 owner option,” IETF, 2009.

[24] J. Polastre, R. Szewczyk, and D. Culler, “Telos: enabling ultra-low power wireless research,” in *Symp. Information processing in sensor networks*, ser. IPSN, 2005.

[25] A. Dunkels, “The contikimac radio duty cycling protocol,” SICS T2011:13, Tech. Rep., 2011.

[26] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-level sensor network simulation with COOJA,” in *Conf. Local Computer Networks*, 2006, pp. 641–648.

[27] A. Dunkels, “Powertrace: Network-level power profiling for low-power wireless networks,” SICS T2011:15, Tech. Rep., 2011.