User-Friendly, Versatile, and Efficient Multi-Link DNS Service Discovery

Daniel Kaiser, Marcel Waldvogel, Holger Strittmatter
University of Konstanz
Konstanz, Germany
<first>.<last>@uni-konstanz.de

Oliver Haase
University of Applied Sciences Konstanz
Konstanz, Germany
oliver.haase@htwg-konstanz.de

***Abstract*—When mobile devices at the network edge want to communicate with each other, they too often depend on the availability of faraway resources. Feasible user-friendly service discovery is essential for direct communication. DNS Service Discovery over Multicast DNS (DNS-SD/mDNS) is widely used for configurationless service discovery in local networks; due in no small part to the fact that it is based on the well established DNS, and efficient in small networks.**

In our research, we enhance DNS-SD/mDNS providing versatility, user control, efficiency, and privacy, while maintaining the deployment simplicity and backward compatibility. These enhancements are necessary to make it a solid, flexible foundation for device communication in the edge of the Internet.

In this paper, we focus on providing multi-link capabilities and scalable scopes for DNS-SD while being mindful of both user-friendliness and efficiency. We propose DNS-SD over Stateless DNS (DNS-SD/sDNS), a solution that allows configurationless service discovery in arbitrary self-named scopes – largely independent of the physical network layout – by leveraging our Stateless DNS technique and the Raft consensus algorithm.

***Index Terms*—DNS, Multicast, Multi-Link, Service Discovery.**

I. Introduction

Zero configuration service discovery is omnipresent as it is essential for convenient interconnection and communication of today's variety of devices in the edge of the Internet. A widely used zero configuration service discovery solution is DNS Service Discovery [1] over Multicast DNS [2] (DNS-SD/mDNS). It allows users to detect printers and streaming devices, to share data, and to communicate with others in a very convenient way. A particular benefit is that services can be requested and offered using DNS resource records, leveraging the solid and well established DNS; thus all means of offering and requesting DNS records can also be used for service discovery. DNS based service discovery cannot only be used in local networks leveraging mDNS but also – losing the zero configuration property – in the Internet using standard DNS servers (DNS-SD/DNS).

While the current means of DNS-SD distribution are appropriate for single-link local networks and the Internet, there is no efficient, user-friendly means of DNS-SD distribution for multi-link networks, used e.g. in universities or other institutions. Since multicast packets are not propagated across routers, devices in different subnets cannot exchange service information using mDNS. Even if the routers propagated the messages, multicast based solutions would not scale. For bandwidth conservation, many institutions deactivate multicast in their WiFi network denying mobile users the benefits of local service discovery. On the other hand, DNS-SD/DNS would pose an unacceptable configuration overhead and would not scale if every user was allowed to offer services. Where suited, an institution could use DNS-SD/DNS to offer a fixed number of services to its members.

To enable communication among the myriad of smart end user devices and thereby becoming an enabling technique for edge-centric computing [3], DNS-SD needs user-friendly, decentralized means to distribute resource records in multi-link networks. If not, users are forced to trust central service directory providers as soon as multicast is not enabled or services have to be discovered across links.

The zero configuration community has reached consensus that adding multi-link support to DNS-SD/mDNS is necessary [4]. Apart from the community consensus, a petition [5] expressing popular demand for providing a DNS-SD/mDNS multi-link extension had been published.

This aligns well with our research (Figure 1), in which we provide enhancements for DNS-SD/mDNS that are necessary to make it a solid, flexible foundation for device communication in the edge of the Internet, by

- increasing the versatility by adding multi-link support to DNS-SD/mDNS,
- providing user control through scopes instead of network boundaries within an organization,
- increasing efficiency, especially in large networks, and
- adding easy-to-use privacy,

while maintaining deployment simplicity and backward compatibility.

In this paper we propose DNS-SD over Stateless DNS (DNS-SD/sDNS) facilitating versatile configurationless respectively low configuration modes of DNS-SD operation for multi-link networks. Our Stateless DNS technique [6] allows registrationless provision of DNS resource records via existing DNS cache servers. This technique allows to discover the service directory which is distributed among few hosts within the institution's network, by providing NS resource records that delegate a special service discovery domain to these hosts.

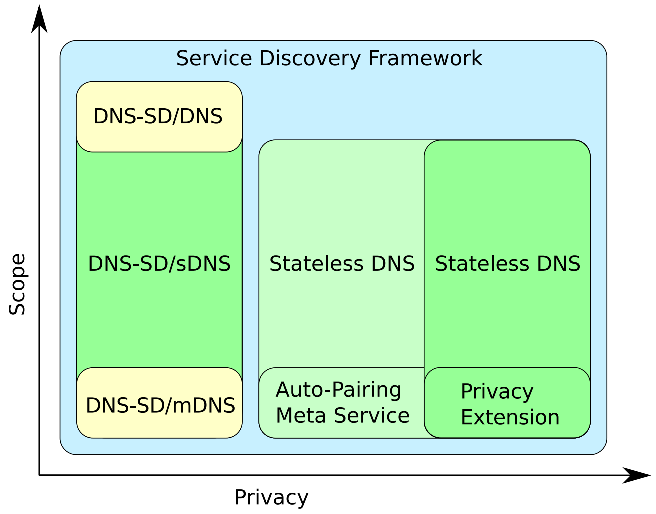


Fig. 1. Service discovery framework supporting scalable privacy and scopes. The solution for the DNS-SD/sDNS area is covered in this paper. The yellow areas are solved by existing RFCs [1], [2]; we provide solutions for the green areas. For the dark green areas we have prototypical implementations.

For synchronizing the service directory among these hosts, we use the Raft consensus algorithm [7].

Using the existing DNS infrastructure, the only additional entity needed is a lightweight reflector – implementable e.g. in a few lines of Perl – that can run within or (publicly) outside the current institution's network.

The contribution of this paper is a new way of DNS-SD resource record distribution that

- is user-friendly as it offers a zero configuration mode of operation in multi-link networks,
- is versatile as it crosses broadcast domains and adapts to arbitrary scopes,
- is efficient as it does not depend on multicast and poses negligible overhead on clients, and
- seamlessly integrates into the DNS query process using well established techniques making it backwards compatible.

In section VII we show the bigger picture integrating DNS-SD/sDNS with our service discovery framework.

II. Related Work

Much research has been done in the field of service discovery, especially for ad hoc networks [8]. In this paper we focus on related work in the field of DNS based service discovery, because these approaches work with existing infrastructure and are either backwards compatible to standard solutions or can at least be easily deployed.

A. Scalable Multicast DNS-SD in Low-Power Networks

Since multicast causes significant network load in wireless networks [9], techniques that make DNS-SD/mDNS more scalable – especially in 6LoWPAN [10] networks – have been developed.

Klauck et al. present and analyze their DNS-SD/mDNS implementation for low-power devices in [11] and propose methods to compress DNS messages to make DNS-SD/mDNS more suitable for 6LoWPAN networks in [12].

EADP [13] is a protocol for scalable service distribution in 6LoWPAN Networks; it has been leveraged as means of distribution for DNS-SD in [14] (DNS-SD/EADP).

Since these solutions are designed for 6LoWPAN and also depend on multicast, they are not applicable to larger multi-link home or institutional networks. But they can be incorporated in a DNS-SD framework offering appropriate means of resource record distribution depending on the network and on the capabilities of the client.

B. Centralized DNS Related Service Discovery Solutions

There are also DNS related service discovery methods, SkyDNS[1] and Consul[2], using multiple central directory servers. For our use-case, they are not suitable because like DNS-SD/DNS they demand setup and maintenance. Nevertheless, they might be the solution of choice for an institution willing to maintain the service directory, because they offer a zero configuration experience for the user, if the directory information is propagated using DHCP.

SkyDNS uses etcd[3] as back end, which is a distributed key-value store also leveraging Raft [7] to maintain consistency. Services are announced by sending the service information with JSON over HTTP to the underlying etcd. Thereafter, these service instances are retrievable using standard DNS queries.

Consul uses another synchronization algorithm and is also suitable for distributed computing centers.

III. Requirements

RFC 7558 [4] defines requirements that should be met by a scalable service discovery solution. It summarizes the requirements, desiring "[...] a mechanism [...] that populates the DNS name space with the appropriate DNS-SD records with less manual administration than is typically needed for a conventional unicast DNS server.“ Our solution (DNS-SD/sDNS) offers precisely that.

In this paper we focus on the requirements for multi-link home and institutional networks. For low-power and lossy networks, we propose to use the solutions mentioned in subsection II-A. For single-link home networks the widely used DNS-SD/mDNS is well suited, because it poses no significant impact on the network load [15]. DNS-SD/DNS or solutions presented in subsection II-B are suitable for global scope service discovery. Our service discovery framework described in section VII covers further requirements stated in RFC 7558 and also protects the user's privacy.

IV. DNS-SD over Stateless DNS

To resolve services in multi-link home or institutional networks (where multicast does not work across links) we

[1] https://github.com/skynetservices/skydns
[2] https://www.consul.io
[3] https://github.com/coreos/etcd

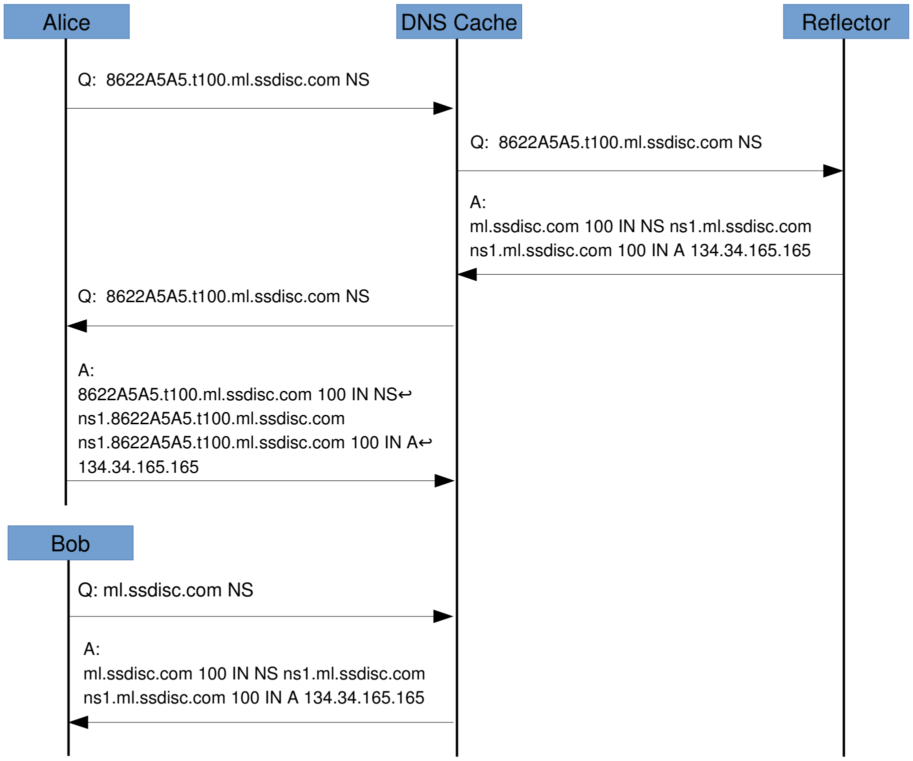

Alice
DNS Cache
Reflector
Q: 8622A5A5.t100.ml.ssdisc.com NS
Q: 8622A5A5.t100.ml.ssdisc.com NS
A:
ml.ssdisc.com 100 IN NS ns1.ml.ssdisc.com
ns1.ml.ssdisc.com 100 IN A 134.34.165.165
Q: 8622A5A5.t100.ml.ssdisc.com NS
A:
8622A5A5.t100.ml.ssdisc.com 100 IN NS↩
ns1.8622A5A5.t100.ml.ssdisc.com
ns1.8622A5A5.t100.ml.ssdisc.com 100 IN A↩
134.34.165.165
Bob
Q: ml.ssdisc.com NS
A:
ml.ssdisc.com 100 IN NS ns1.ml.ssdisc.com
ns1.ml.ssdisc.com 100 IN A 134.34.165.165

Fig. 2. Using Stateless DNS to provide an NS record for the ml-scope of our example service discovery domain with a TTL of 100 seconds. Alice registers her device as SNS and Bob retrieves the registered information. Thereafter, Bob can interact with the SNS running on Alice's device registering and retrieving service instances.

introduce Scope Name Servers (SNS) that act as service directories. Any host in the network can become an SNS following the rules explained later in this section. A group of SNSes is responsible for one service discovery scope that in turn is defined by a *service discovery domain*, e.g. ml.ssdisc.com[4] or floor3.buildingB.ssdisc.com. We integrate the service querying process seamlessly into DNS such that the query process is exactly the same as when using DNS-SD/DNS. This allows client software to be independent of the resource record transmission mechanism.

A very important design goal is to allow a zero configuration mode of operation for both clients and SNSes, including the process of becoming SNS. Further, the administrator of the respective multi-link network has to neither deploy anything nor be aware of our service discovery method.

The questions how to discover SNSes, how to become SNS, how to synchronize a set of SNSes, how to query, and how to register service instances are addressed in the following subsections.

A. Service Discovery Domains

We propose three options[5] for clients to learn about service discovery domains: preset discovery domain, DNS-SD/DNS, and DHCP. The default service discovery domain can be preset, e.g. to ssdisc.com, so that clients work without any additional configuration. Further service discovery domains can be provided via DNS-SD/DNS, e.g. on dom.ssdisc.com. Institutions could also provide their own reflector, e.g. using one of our implementations, and distribute the corresponding service discovery domain via DHCP. This would only pose a minor configuration overhead for an institution compared to maintaining a centralized service directory, because only a lightweight reflector and a one-time DHCP entry are required. Even if it is possible to provide an institution specific reflector, in most cases it is not necessary. Reasons for providing a reflector might be high privacy[6] and availability requirements.

The combination of a service discovery domain and a DNS cache defines a service discovery scope; thus even when using a public reflector (and service discovery domain), service directory information is only available to hosts accessing the same DNS cache server. A user can create named sub-scopes independent of the underlying network structure by establishing itself as SNS for an arbitrary subdomain of a service discovery domain.

B. Providing the NS Records

Since an SNS might leave the network at any time, very dynamic means of providing SNS information is required.

[4] ml → multi-link; ssdisc → scoped service discovery. The service discovery domain ssdisc.com is operational for experiments.

[5] The DNS interface for quering an SNS is indepented of the method used to learn about the discovery domain.

[6] When using a public reflector, the only transmitted data beyond the institutions network are scope names and the (local) IP addresses of the SNSes.

Further, SNSes should only be discovered within the institution they currently sojourn in. To this end, we store name server delegations to SNSes in the current network's DNS cache using our Stateless DNS technique[7]. This allows to have location dependent SNSes for a single service discovery domain; depending on the network a host is currently discovering in, different resource records can be retrieved using the same global service discovery domain.

To make Stateless DNS work, the only additional infrastructure we need is a stateless reflector implemented e.g. in a few lines of Perl. It acts as authoritative name server for the parent zone of the service discovery domain.

Figure 2 shows the process of leveraging Stateless DNS to enter an NS resource record in the current DNS cache. Alice sends a programming query – which is a valid DNS query – by asking for an NS resource record with the label

```
8622A5A5.t100.ml.ssdisc.com IN NS
```

that will be handled by the local DNS server. Because the stateless reflector is authoritative for this query, it will receive the query from the local DNS server. The reflector then generates the following response using only information encoded in this query[8].

```
Question:   8622A5A5.t100.ml.ssdisc.com IN NS
Authority:  ml.ssdisc.com 100 IN NS ns1.ml.ssdisc.com
Additional: ns1.ml.ssdisc.com 100 IN A 134.34.165.165
```

Since it is a delegation to an in-Bailiwick [16] name server the cache will accept the answer if this in-Bailiwick name server – Alice's notebook with the IP address 134.34.165.165 – is able to answer the programming query and NS queries for the service discovery domain.

```
Question:   <LABELS.>ml.ssdisc.com IN NS
Authority:  ml.ssdisc.com ↵
                    100 IN NS ns1.<LABELS.>ml.ssdisc.com
Additional: ns1.<LABELS.>ml.ssdisc.com ↵
                                  100 IN A 134.34.165.165
```

The programming query answer's sole purpose is to make the cache name server store the NS entries. The answer for NS queries of the service discovery domain is needed to be able to retrieve the SNSes' IP addresses. The general form of the programming query is[9]

```
(<hexenc_IPaddr>.){1,4}t<TTL>.<scope>.<sd_domain>
```

allowing to specify up to four name servers in a single programming query. All specified name servers have to be able to answer the programming query and the NS queries for the service discovery domain. Each of them has to return *all* NS entries[10] to make the cache name server ask the next available name server in the case the first one asked is offline.

C. Establishing SNSes

When entering a network, a host supporting our multi-link DNS-SD technique sends a standard DNS query asking for NS records belonging to the service discovery domain to get a list of SNSes (see Figure 3). The query corresponding to our example service discovery domain is

```
ml.ssdisc.com IN NS
```

In the bootstrap phase there will be no SNS and the host will become the first SNS using the Stateless DNS method explained above (Figure 2). Because these NS entries cannot be overwritten, the TTL should be chosen adaptively. The first SNS should choose a low TTL when establishing its first NS entry, e.g. 10 seconds. To mitigate race conditions arising when other SNSes exist whose TTL just expired in the moment the new host asks to become name server, the new host must ask a second time after a random back off.

When the host gets a list of SNSes, it asks one of them whether it should also join the set of SNSes and if requested joins the set. This concludes discovering the current SNSes and the host can now ask for service information or register service instances as described in subsection IV-F. Figure 3 illustrates the process of SNS discovery.

It might happen that none of the returned SNSes answers. This is the case when all of them have gone offline without the TTL expiring and with no new SNSes joining. Since Stateless DNS cannot overwrite existing NS records, the discovery domain is blocked in this situation. For this reason, we need a defined fallback discovery domain and means to restore the standard discovery domain as soon as possible. The fallback discovery domain can be derived from the standard discovery domain by appending -0 to the highest scope defining label; thus the fallback domains for ml.ssdisc.com and floor3.buildingB.ssdisc.com are ml-0.ssdisc.com and floor3.buildingB-0.ssdisc.com, respectively. With increasingly low probability, the fallback domain might be blocked in the same way. The fallback of the fallback is defined to be the domain with the corresponding integer incremented by 1. When a discovery domain is blocked in this way, a host has to check whether the next fallback domain has an SNS that answers and if not, become SNS for this fallback domain (see Figure 3). To recover from this situation as soon as possible, a host that has to become name server for a fallback domain sets the TTL of its NS resource records to the remaining TTL of the NS records belonging to the standard domain. This makes the standard discovery domain's TTL and all fallback domains' TTLs expire at the same time and the host can register for the standard domain with all fallback domains being unblocked. Because of the adaptive TTL, the chain of backup domains is expected to be small. Further this problem will only occur in the bootstrap phase because the system will stabilize as described later.

D. Synchronization of SNS data

To make the system robust, we need several SNSes for each scope. These SNSes have to synchronize so that clients receive consistent information. Further, we want a querying client to be able to abstract away from the existence of several SNSes that need to synchronize, considering each of them as an equal representative of a black box providing the desired information.

[7] In our techreport [6] we also propose methods to store other record types and evaluate the proposed methods. The method used in this paper (to store NS records) works reliably as it behaves like a normal subzone delegation.

[8] The IP address of the new SNS is transmitted in hexadecimal notation.

[9] We use () for grouping and {} for repetition count.

[10] Since the SNSes know each other, this does not pose a problem.

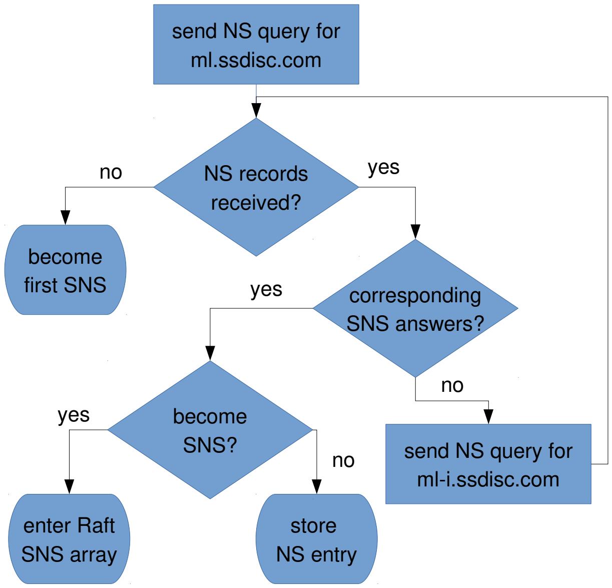


Fig. 3. Process of querying for existing SNSes and becoming SNS, for the ml-scope of our example service discovery domain.

For synchronizing the service directory among the SNSes we use Raft [7], a simple and efficient consensus protocol, that uses heartbeat messages and randomized timers to elect a leader whose log state is copied to the other participants called followers. After each heartbeat there is consensus[11] about who is leader and what the current log state is. In our case the log state contains the resource records corresponding to registered service instances. Changes of membership are also expressed and communicated using the log state.

Raft is very efficient with respect to network load as it only uses $2n$ messages per heartbeat, where n is the number of participants. During each heartbeat phase the leader sends a message to each follower, which can

- be empty if there were no log changes
- contain a log change
- contain a confirmation of a log change

and each follower sends a message to the leader which can

- be empty if the leader's message was empty
- contain a confirmation of a log change if the leader's message contained a log change.

If the leader fails to send a message within the heartbeat before a follower's timer runs out, the follower becomes a candidate and sends a further type of heartbeat message to each participant, asking to vote it as the new leader. Participants that did not get any other heartbeat message within the current interval will send an answer message voting for the candidate to become the new leader. If the candidate gets consensus it becomes the new leader. All information needed to agree on log changes, leader changes and membership is communicated in these $2n$ messages per heartbeat interval.

E. Processing Queries

Our goal is to also incorporate all the additional logic necessary for the SNSes within the heartbeat messages described above. Since we want each SNS to be able to process service information updates, we allow a follower to append an update request within the heartbeat answer. The leader receiving this request will then issue the corresponding update request to all followers. This message will also be received by the issuing follower, serving as a confirmation.

1) Deciding on a new SNS: The number of SNSes for a scope should be chosen dependent on the number of hosts, the number of offered service instances, and the churn rate. The decision whether a querying client should become SNS should be based on a ranking taking the hosts expected time to stay online into consideration. As of yet, we are still assessing which kind of ranking to use. If there is only one or two other SNSes, the new client should definitely be chosen. The closer the number of SNSes gets to the desired SNS set size, the higher the rank of the new client should be in order to be accepted. There should still be the possibility to substitute the new client for an SNS in a full SNS set if the rank is appropriate.

2) Updating the NS Records: When the TTL of the current NS entry in DNS cache runs out, the current leader has to reestablish itself and its followers as name servers. Since the number of name servers that can be provided via the DNS cache is limited by the reflector implementation – 4 in our current implementation – the leader chooses the followers with the highest rankings; we call an SNS that is established as name server *listed SNS*. The TTL grows with the average ranking scores of the current SNSes, but should not exceed a sensible limit. Since the current leader will not change as long as it is online, the system will stabilize. With increasing online time, hosts are more likely to be leader.

3) Relaying Queries: In very large scopes there might be a significant load on the listed SNSes as they are queried by the hosts (see section VI). To cope with this problem, listed SNSes can relay queries to a randomly chosen non-listed SNS.

F. Querying the SNSes

SNSes offer the standard DND-SD/DNS interface to clients which allows asking for

- a listing of all existing service types,
- a listing of service instances of a service type and
- the resolution of a certain service instance.

The process of querying is independent of the structure the SNSes are organized in. A client can query any of the SNSes retrieved from the DNS cache.

Clients that need an up-to-date list of instances of a certain service can request DNS push [17] from the SNSes; this is important e.g. to provide an up-to-date list of online contacts in a chat application.

Resource records provided by SNSes have a TTL of 10 minutes, which seems to be a good compromise between avoiding stale information and efficiency. When hosts leave

[11] There is a negligible possibility for split votes. If a split vote occurs, consensus is very likely to be reached during the next heartbeat interval.

```perl
use warnings;
use strict;
use Net::DNS::Nameserver;

sub reply_handler {
  my ($qname, $qclass, $qtype, $host, $query, $conn) = @_;
  my ($rcode, @ans, @auth, @add, %headermask);

  # parse query
  $qname = lc($qname); # to lower case
  my (@ips, $ttl, $alias, $qbase);
  my $rx_i  = qr/(\w+(?:\.\w+){0,3})/;
  my $rx_t  = qr/t(\d{1,6})/;
  my $rx_a  = qr/((?:\w|-)+)/;
  my $rx_q  = qr/((?:\w+\.)*\w+)/;
  if($qname =~ /^${rx_i}\.${rx_t}\.${rx_a}\.${rx_q}$/) {
    @ips = split('\.', $1);
    ($ttl, $alias, $qbase) = ($2,$3,$4);
    $rcode  = "NOERROR";
  }
  else{
    $rcode = "NXDOMAIN";
    return ($rcode, \@ans, \@auth, \@add, \%headermask);
  }

  # assemble answer
  $headermask{aa} = 1; # set authoritative answer flag
  my $alias_domain = $alias . '.' . $qbase;
  my $ns_num = 1;
  foreach my $ip (@ips){
    my $ns_domain  = 'ns' . $ns_num++ . '.' . $alias_domain;
    #convert from hex to dotted notation
    $ip = join '.', unpack "C*", pack "H*", $ip;
    my $rr_ns = new Net::DNS::RR(name => $alias_domain,
       ttl => $ttl,
       class => "IN",
       type => "NS",
       nsdname => $ns_domain);
    push @auth, $rr_ns;
    my $rr_a = new Net::DNS::RR(name => $rr_ns->nsdname,
       ttl => $ttl,
       class => "IN",
       type => "A",
       address => $ip);
    push @add, $rr_a;
  }
  return ($rcode, \@ans, \@auth, \@add, \%headermask);
}

# create nameserver object
my $ns = new Net::DNS::Nameserver(
  LocalAddr    => "51.254.124.217",
  LocalPort    => 53,
  ReplyHandler => \&reply_handler,
) || die "couldn't create nameserver object\n";

# start nameserver main loop
$ns->main_loop;
```

Fig. 4. Stateless reflector implementation in Perl. This implementation runs as authoritative name server for our example domain ssdisc.com.

the network gracefully, they can send a sign-off message to an SNS allowing the SNS deleting all corresponding resource records and pushing this information to affected hosts. To make a single message containing a hostname sufficient, both hosts and SNSes store a mapping from host to resource records offered by this host.

G. Hierarchical SNSes

SNSes can delegate sub zones; e.g. to create sub scopes or to delegate the resolution of certain service types. This could e.g. be used for load balancing or hiding the existence of certain sub scopes.

H. Security and Privacy Considerations

Like DNS-SD/mDNS [1], [2], our technique currently relies on the fairness of the participating hosts. Neither technique offers privacy, and has the unmitigated risk of malicious modification of resource records.

Using DNS-SD/mDNS, even passive hosts receive all resource records related to service instances as soon as they are requested or offered by anyone in the network [18]. Each user can overwrite existing service instances by violating the protocol. Since every host gets these messages, such a malicious user can be detected by looking at the traffic; mitigating techniques currently do not exist.

Using DNS-SD/sDNS, hosts have to actively ask an SNS to get resource records. An SNS could use filters to only provide selected hosts with the requested records. Only hosts that are currently in the SNS-role are allowed to overwrite existing service instances. Further, SNSes tend to be more trustworthy compared to random nodes because they are expected to be stable nodes that are part of a network for a long time. However, malicious SNSes can silently overwrite or drop service instances. We work on mitigating this problem.

Privacy can be added to either technique using our orthogonal privacy extension [19], [20] that provides means for secure privacy preserving service discovery among hosts sharing a previously exchanged secret.

V. Implementation

We have several implementations of the reflector (Perl, C, Java), which are ready for deployment and work reliably. Figure 4 shows the small but operational Perl implementation of the reflector that is authoritative for our example domain ssdisc.com; it supports the name server delegation method used for DNS-SD/sDNS. Our flexible C implementation[12] supports all Stateless DNS methods described in [6] and can be easily augmented to support new methods by providing a corresponding template file.

Our prototypical implementation of an SNS-capable service discovery daemon[13] demonstrates configurationless service discovery in large multi-link networks without using multicast.

VI. Analysis

In order to get widely accepted, user-friendliness with respect to configuration effort is not sufficient; the solution also has to be efficient. For service discovery this means it's working should be imperceptible to users with respect to both network load and computational overhead. Despite the fact that as of yet we did not thoroughly evaluate the network impact of our solution[14] the following analysis suffices to point out the user-friendliness, suitability and scalability of our solution.

We analyze the influence of our solution on all relevant entities, namely Hosts, SNSes, the SNS-Leader, the Reflector, and the DNS-cache. An SNS is also a host, meaning it has

[12] https://gitlab.com/holst/StatelessDNS-EchoServer

[13] https://gitlab.com/kaiserd/sns

[14] We are going to evaluate our service discovery framework thoroughly leveraging the Omnet++ discrete event simulator (https://omnetpp.org)

to perform host actions and specific SNS actions; the SNS-leader also performs SNS and host actions in addition to SNS-leader specific actions. We group the actions into the following categories

- Raft related,
- messages to and from the reflector,
- messages to and from the DNS cache, and
- host to SNS communication which consists of
  - service type listing,
  - service instance listing,
  - service instance resolution,
  - service registration,
  - service deregistration and
  - queries about joining the SNS cluster.

The unity of our analysis is a single scope because the actions causing the highest impact on the network load – service instance listing and resolution – do not propagate beyond scope boundaries. For actions reaching beyond scope boundaries, e.g. communication with the reflector, we examine an appropriate wider area.

Before going into more detail, we want to shortly address the main efficiency concern, namely the number of service instances a host wants to be listed. With DNS-SD the host requests a service listing for service types it is interested in and then selects the service instances it wants to be resolved. In huge scopes this number can be quite large and the SNS has to send the full list of PTR resource records corresponding to these service instances to the host. This only has to be done once per host joining a network; but a large number of hosts joining a network might cause a significant load on the corresponding SNSes. However, since the user has to choose among the service instances manually, we argue that this number should not be large. In future work, we consider attribute based service instance selection on the SNS to cope with this problem. Nevertheless, our solution in its current state scales very well as we show in this section. We meet the scale requirement of RFC 7558 [4] – ”It must scale to a range of hundreds to thousands of DNS-SD/mDNS-enabled devices in a given environment.” – as shown in the following analysis. Standard multicast DNS service discovery does not scale to scopes that large because the mere number of multicasts would tie the network.

We do not take background traffic into consideration[15] and consider only nodes supporting DNS-SD/sDNS. For ease of calculation we assume that the devices register all their services right when they join the network.

Our analysis uses the following variables and their respective limits.

- n, the number of hosts in the scope. We want scopes to be scalable up to $n = 10000$. Larger scopes can be handled by limiting the number of listed service instances or by subdivided them.
- n_o, number of nodes joining (and leaving) the network per second. We use 10 minutes as minimum for the average time a host is online [9], leading to maximum arrival rate of $n/600 = 16.\overline{6} \approx 17$ users per second.
- s_o, the average number of service instances offered by a host. We regard $s_o = 5$ as sensible default.
- s_l, the average number of service instances a host wants to be listed. In huge scopes, we consider 5% of the service instances offered a reasonable upper bound, leading to $\mathtt{max}(s_l) = ns_o * 0.05 = 2500$.
- s_r, the average number of service instances requested by a host. In huge scopes, we consider 5% of the listed service instances a reasonable upper bound, leading to $\mathtt{max}(s_r) = s_l * 0.05 = 125$.
- k, the total number of SNS in the network. We consider $k = 20$ a sensible maximum as it is very unlikely for 20 SNS to fail at the same time.
- k_l, the number of listed SNS, i.e. SNS stored in the DNS-cache. With our current implementation $\mathtt{max}(k_l) = 4$.
- h, Raft heartbeats per seconds. We consider $h = 3$ sufficient to provide hosts with current information. Increasing the heartbeat frequency to e.g. $h = 10$ would increase the number of transmitted packets, but not the network load because the service related information that has to be synchronized per time interval does not change.
- $\mathtt{size}_{\mathtt{P}}$, average size of a PTR resource record. We use 100 Bytes as upper bound.
- $\mathtt{size}_{\mathtt{ST}}$, average size of SRV and TXT record. We use a single variable because these records are always transmitted together. The upper bound used in the following is 500 Bytes.[16]
- $\mathtt{size}_{\mathtt{PST}}$, $\mathtt{size}_{\mathtt{P}} + \mathtt{size}_{\mathtt{ST}}$, average size of all records associated with a service instance, summing up to an upper bound of 600 Bytes.
- $\mathtt{TTL}$, the TTL of entries in the DNS cache. Like described above, the minimum is 10 s.

Figure 5(a) shows the estimated network load the different entities have to cope with.

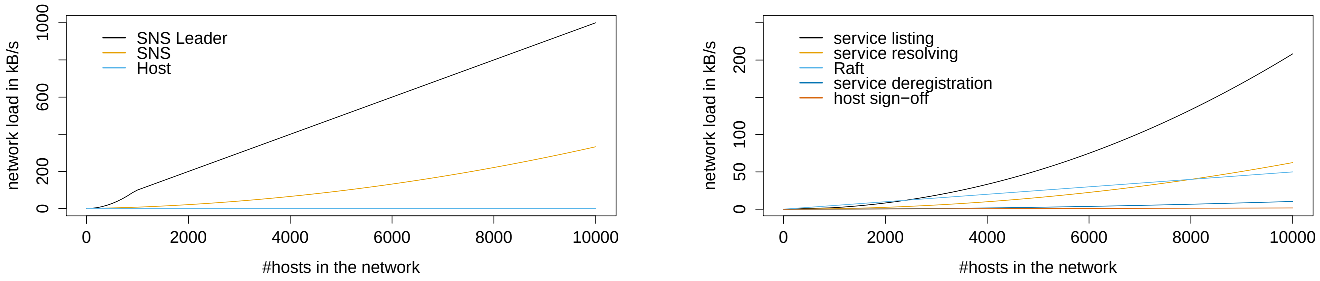
A. Host

A non-SNS host is agnostic to the Raft related actions. Further, it does not communicate with the reflector. The communication with the DNS cache to get the current SNS list is negligible as it is a tiny fraction of the many DNS requests when surfing the web. Queries concerning joining the Raft cluster, and registering and deregistering services can be neglected because these actions only demand a few messages per session. Listing service types is also imperceptible because even in a very large scope the number of offered types is expected to be manageable and the set of available service types is expected to change slowly.

After discovering SNSes, a host asks for a listing of service instances followed by a resolution request for chosen services. Even with $\mathtt{max}(s_l) = 2500$ and $\mathtt{max}(s_r) = 125$, the amount of data received would only be $s_l * \mathtt{size}_{\mathtt{P}} = 250\,\mathrm{kB}$

[15] This will be part of our Omnet++ evaluation

[16] These average resource record sizes are very high; in all load critical situations, many records are transmitted in one packet, which reduces the header overhead significantly.



(a) Estimated network load the different DNS-SD/sDNS entities have to cope with.

(b) Estimated network load of an SNS distributed over the different SNS actions.

Fig. 5. Estimated network load caused by DNS-SD/sDNS, dependent on the number of hosts in the examined scope. We chose 2% of the hosts as SNSes, with a minimum of 4 and a maximum of 20. In scopes with less than 4 hosts, every host is an SNS.

and $s_r * \mathtt{size}_{\mathtt{PST}} = 75\,\mathtt{kB}$, respectively, without taking compression into consideration. The load on the host while sojourning in a network is very low even when using the afore mentioned maximum values: $n_o s_o * 0.05 * \mathtt{size}_{\mathtt{P}} \approx 425\,\mathtt{B/s}$ and $n_o s_o * 0.0025 * \mathtt{size}_{\mathtt{PST}} \approx 125\,\mathtt{B/s}$ for service listing and resolving, respectively; the arrival rate of new users per second is about 17 with each of them offering 5 service instances on average, of which in turn 5% have to be listed and 0.25% have to be resolved. Even in our large example scope the network load a host is exposed to only sums up to a manageable burst of $325\,\mathtt{kB}$ when joining the network and $550\,\mathtt{B/s}$ while sojourning in the network.

Further, the computational overhead on the host devices used for packet processing is imperceptible to the user, both in terms of responsiveness of the system and battery life. The host has far less load compared to mDNS-SD as managing services is the SNSes' task; this makes our solution feasible for low power devices.

B. SNS

Raft handling only needs two messages per heartbeat ($2h$ messages per second), and only messages propagating new service information might be of considerable size. The load caused on an SNS by Raft to synchronize the service information is $n_o s_o * \mathtt{size}_{\mathtt{PST}} = 16.\overline{6} * 5 * 600 = 50\,\mathtt{kB/s}$. This also includes the cost for receiving service registration queries as an SNS receives this query only once; either directly from a host or via a Raft heartbeat message.

Deregistering services is also handled in the heartbeat messages; it only needs a hostname per host signing off, adding just $n_o * \mathtt{size}_{\mathtt{P}} = 16.\overline{6} * 100 \approx 1.7\,\mathtt{kB/s}$.

We don't consider service type listing. Typically there are only a few different service types (roughly $ns_o/s_l = 20$ in our example scope) and since most hosts already know which types they are interested in, they don't need to list service types.

The load caused by service instance listing and service instance resolution corresponds to the load on a host when joining the network multiplied by the arrival rate, divided by the number of SNS: $n_o s_l/k * \mathtt{size}_{\mathtt{P}} = 16.\overline{6} * 2500/20 * 100 \approx 210\,\mathtt{kB}$ per second and $n_o s_r/k * \mathtt{size}_{\mathtt{PST}} = 16.\overline{6} * 125/20 * 600 = 62.5\,\mathtt{kB/s}$ for service instance listing and service resolution, respectively.

Pushing service deregistration information to clients needs $n_o s_r/k * \mathtt{size}_{\mathtt{P}} = 16.\overline{6} * 125/20 * 100 \approx 10.5\,\mathtt{kB/s}$, in the unfavorable case that each service a host resolved was offered by a distinct host.

The memory capacity needed by the SNS to store the service directory amounts to $ns_o * \mathtt{size}_{\mathtt{PST}} = 30\,\mathtt{MB}$. When an SNS goes offline and a new SNS is chosen, the service directory has to be synchronized to the new SNS. Since in small scopes the caused load is insignificant, in large scopes the SNS are expected to stay online for a long time, and only one of the SNSes has to transmit the directory to the new node, we don't consider these occasional bursts in the average network load per second an SNS has to cope with. If every ten minutes one SNS went offline – which is a high frequency of SNS change for such a large scope – an SNS has to cope with such a burst approximately once every 3 hours.

In summary, an SNS has to cope with a network load of $340\,\mathtt{kB/s}$ in our large example scope when estimating the neglected actions to amount to approximately $10\,\mathtt{kB/s}$. Figure 5(b) shows the estimated network load caused by the different SNS actions.

C. SNS Leader

The communication to the reflector is negligible; even using the minimal TTL it only happens once every 10 seconds.

The Raft message load the SNS leader is exposed to corresponds to k times the load of the follower SNSes, because the leader exchanges messages with every follower: $n_o s_o k * \mathtt{size}_{\mathtt{PST}} = 16.\overline{6} * 5 * 20 * 600 = 1\,\mathtt{MB/s}$. This is a significant load, but our example scope is very large (as well as the average resource records size) and the SNS leader in such a large scope is expected to be very strong and connected to Gigabit Ethernet. Recall that a scope of this size takes time to grow; there will be many leader elections, eventually resulting in a strong leader and also strong followers. To reduce the load on the SNS leader in very large scopes, it can delegate all host queries to other SNSes or even forgo joining the listed SNSes.

D. Reflector

The load on the reflector is really low. For each scope that is part of its authority zone it has to communicate once with a single SNS leader before the corresponding TTL is about to end. Even if a single service discovery domain had so many scopes that it would be hard for a single reflector to handle them, several reflectors using anycast could be used without synchronizing, because the reflector does not hold any state.

E. DNS-Cache

The load increase for the DNS-Cache when using DNS-SD/sDNS is imperceptible as usual web surfing causes a myriad of DNS-Cache requests. There is only one new cache entry per SNS TTL for a scope and hosts only ask for the SNS list when entering the network or when the SNSes stop answering.

F. Unicast vs. Multicast

Compared to DNS-SD/mDNS we reduce network load in most scenarios as we forgo multicast. The influence of many multicasts on the network load is especially severe in huge 802.11 wireless networks [21], because multicasts are transmitted using a very low transmission rate so that older devices not supporting higher transmission rates can receive the multicasts as well [22]. Hong et al. [9] show that 13% of their campus network bandwidth is used by DNS-SD/mDNS. We analyzed the expected network bandwidth savings when using our service discovery privacy extension [19] which also forgoes multicast.

There are other disadvantages of multicast in 802.11 wireless networks described in [22], like handling host sleep mode, which further increases battery drain, because devices have to stay awake if multicast traffic is waiting to be sent by the access point.

VII. Integration and Architecture

This section gives a short introduction to our service discovery framework. It returns control to the user supporting scalable scopes and scalable privacy. Scalable scopes, as described above, allow users choosing the scope in which the services are offered and requested; our privacy extension [19] allows to selectively offer services to chosen friends, chosen groups, or everyone in a scope.

A. Service Discovery Framework

Figure 1 classifies service discovery techniques with respect to the reachability scope they are used in and the privacy they offer. To the end of providing such a service discovery framework, we combined existing techniques and found solutions for the yet missing parts. Solutions for the smallest scope of single-link local networks and the biggest scope of the whole Internet are given by DNS-SD/mDNS and DND-SD/DNS. To provide a solution for the gap in between, we developed DNS-SD/sDNS which has been proposed in this paper. These techniques combined do not cover the whole desired area as they do not provide privacy. DNS-SD/mDNS publishes private information about services in an unsolicited way [18]. To give users control over the offered and requested services, we do not only provide privacy in a binary way - either it is on or off - but give the possibility to scale the privacy, i.e. choose a privacy level. This is important because the less privacy is demanded the easier is the necessary process of device pairing. To allow all users in the current network to discover a certain service, we provide an auto-pairing meta service that exchanges pairing information with all devices in the current network. This allows the paired devices not only exchanging service information in the current network but also in all other networks. Pairing at this level of privacy works without *any* configuration. Our privacy extension [19] allows to offer services to chosen friends. While the basic privacy extension, which we implemented as an extension to the Avahi[17] Zeroconf daemon, is limited to single-link networks like DNS-SD/mDNS, augmenting it using Stateless DNS allows scalable scopes within the privacy extension. We also implemented an enhancement of our privacy extension leveraging Stateless DNS to avoid the need of multicast [20][18].

B. User Control

The user interface to control the scope and privacy is provided by our enhanced service browser. Sensible defaults are preset to still allow configurationless service discovery. The enhanced service browser also manages user groups for the privacy extension. It does not matter if group members were paired using the auto-pairing meta service or a user pairing [19]. Users can manage groups, but configurationless group management is also offered: e.g. all friends paired using the auto-pairing meta service in a certain network allowing e.g. to automatically get a group of all devices used in the home network.

C. Architecture

To ease the integration of alternative ways of service discovery, and to integrate into existing service discovery daemons, we propose a service discovery daemon (SDD) that is responsible to demultiplex client requests to different resolvers. Leveraging the proposed service discovery daemon, client software can use a unified interface for service discovery. Figure 6 illustrates our proposed architecture.

The SDD can be controlled by users via our enhanced service browser that allows to set the scope of discovery and privacy for single service instances, certain service types or all services. The SDD is also connected to a pairing module that handles pairing for privacy preserving service discovery [19].

Based on sensible defaults or decisions made by the user overriding the defaults, the SDD decides – given a client

[17] http://avahi.org

[18] To also support service listing for unpaired devices and arbitrary scopes the techniques proposed in this paper have to be used.

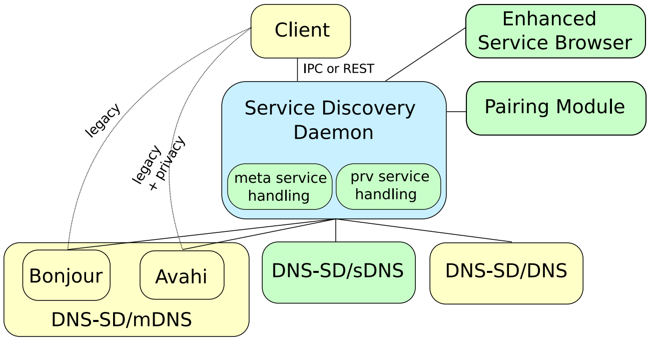


Fig. 6. Service discovery daemon (SDD) architecture. The SDD offers a unified interface to client software and demultiplexes client requests to different means of resource record distribution. Backwards compatibility is granted by the legacy interfaces.

request – which means of resource record distribution has to be used.

Backwards compatibility is provided, because software not supporting the interface to the SDD still works as the interface to existing service discovery daemons has not been changed. Since we provide extensions to Avahi, existing clients can also use the privacy preserving service discovery and offer services in a multi-link scope.

VIII. Conclusion and Future Work

Multicast DNS Service Discovery over Stateless DNS (DNS-SD/sDNS) provides a versatile, convenient and easily deployable means of resource record distribution for scalable DNS Service Discovery. It offers a zero configuration mode of operation and seamlessly integrates in the DNS discovery process, allowing core-independent, user-controllable device interaction in the edge of the Internet. Our proof-of-concept implementation – realized as an extension to the Avahi Zeroconf daemon – already allows to use DNS-SD in our campus WLAN where multicast is disabled. We showed how to integrate DNS-SD/sDNS in our service discovery framework making it part of a user-friendly, efficient service discovery solution supporting both scalable scopes - with the help of the technique proposed in this paper - and scalable privacy. We plan to address further security and privacy problems arising when offering service information across links and in scalable scopes. Further we will evaluate our scope extension with respect to network efficiency using the Omnet++ discrete event simulator[19].

We also plan to integrate DNS-SD hybrid proxy [23] capabilities in the SNSes as soon as the Internet draft becomes an RFC, which is likely to happen soon. This will allow hosts that are not aware of SNSes to use DNS-SD in multi-link networks providing a very elegant way of being backwards compatible.

References

[1] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763 (Proposed Standard), Internet Engineering Task Force, Feb. 2013.

[2] ——, "Multicast DNS," RFC 6762 (Proposed Standard), Internet Engineering Task Force, Feb. 2013.

[3] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[4] K. Lynn, S. Cheshire, M. Blanchet, and D. Migault, "Requirements for Scalable DNS-Based Service Discovery (DNS-SD) / Multicast DNS (mDNS) Extensions," RFC 7558 (Informational), Internet Engineering Task Force, Jul. 2015.

[5] (2013) Petition from educause higher ed wireless networking admin group. [Online]. Available: https://www.change.org/p/from-educause-higher-ed-wireless-networking-admin-group

[6] D. Kaiser, M. Fratz, M. Waldvogel, and V. Dietrich, "Stateless DNS," University of Konstanz, Tech. Rep. KN-2014-DiSy-004, Dec 2014.

[7] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annual Technical Conference*, 2014, pp. 305–320.

[8] C. N. Ververidis and G. C. Polyzos, "Service discovery for mobile ad hoc networks: a survey of issues and techniques," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 3, pp. 30–45, 2008.

[9] S. Hong, S. Srinivasan, and H. Schulzrinne, "Measurements of multicast service discovery in a campus wireless network," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE, 2009, pp. 1–6.

[10] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6282, 6775.

[11] R. Klauck and M. Kirsche, "Bonjour contiki: A case study of a dns-based discovery service for the internet of things," in *Ad-hoc, Mobile, and Wireless Networks*. Springer, 2012, pp. 316–329.

[12] ——, "Enhanced DNS message compression-optimizing mDNS/DNS-SD for the use in 6LoWPANs," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE, 2013, pp. 596–601.

[13] B. Djamaa, M. Richardson, N. Aouf, and B. Walters, "Towards efficient distributed service discovery in low-power and lossy networks," *Wireless Networks*, vol. 20, no. 8, pp. 2437–2453, 2014.

[14] B. Djamaa and M. Richardson, "Towards scalable DNS-based service discovery for the internet of things," in *Ubiquitous Computing and Ambient Intelligence. Personalisation and User Adapted Services*. Springer, 2014, pp. 432–435.

[15] A. Rain, "An analysis of multicast traffic in wireless networks," Master's thesis, University of Konstanz, 2015.

[16] S. Son and V. Shmatikov, "The hitchhiker's guide to DNS cache poisoning," in *Security and Privacy in Communication Networks*. Springer, 2010, pp. 466–483.

[17] T. Pusateri and S. Cheshire, "DNS push notifications," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-dnssd-push-03, November 2015.

[18] D. Kaiser and M. Waldvogel, "Adding privacy to multicast DNS service discovery," in *Proceedings of IEEE TrustCom 2014 (IEEE EFINS 2014 Workshop)*, 2014.

[19] ——, "Efficient privacy preserving multicast DNS service discovery," in *Workshop on Privacy-Preserving Cyberspace Safety and Security (IEEE CSS 2014)*, 2014.

[20] D. Kaiser, A. Rain, M. Waldvogel, and H. Strittmatter, "A multicast-avoiding privacy extension for the avahi zeroconf daemon," *Netsys 2015*, 2015.

[21] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Computer Society LAN MAN Standards Committee IEEE Std 802.11 TM - 2012, 03 2012.

[22] E. Vyncke, P. Thubert, E. Levy-Abegnoli, and A. Yourtchenko, "Why network-layer multicast is not always efficient at datalink layer," Working Draft, IETF Secretariat, Internet-Draft draft-vyncke-6man-mcast-not-efficient-01, February 2014.

[23] S. Cheshire, "Hybrid unicast/multicast DNS-based service discovery," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-dnssd-hybrid-02, November 2015.

[19] https://omnetpp.org/