Efficient Privacy Preserving Multicast DNS Service Discovery

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

***Abstract*—In today's local networks a significant amount of traffic is caused by Multicast DNS Service Discovery (mDNS-SD), a prevalent technique used for configurationless service distribution and discovery. It allows users to offer and use services like device synchronization, file sharing, and chat, when joining a local network without any manual configuration. While this is very convenient, it requires the public exposure of the offering and requesting identities along with information about the offered and requested services, even when services do not need to be public. Some of the information published by the announcements can be very revealing, including complete lists of family members. Another problem is the huge amount of multicast traffic caused, which is especially relevant for large WiFi networks.**

In this paper we present a privacy extension that does not publish private information and reduces the number of packets sent while still not requiring any network configuration except for an initial pairing per pair of users. A key feature of our solution is the ease of upgrading existing systems, a must for widespread deployment and acceptance. We developed an implementation based on the open-source *Avahi* daemon to show the feasibility of our privacy extension. Our solution grants tunable privacy and reduces multicast traffic without affecting user experience.

***Index Terms*—Privacy, DNS, Multicast, Service Discovery.**

I. Introduction

Multicast DNS Service Discovery (mDNS-SD) is a prevalent technique widely used to distribute services in local networks without configuration. It uses the upper two layers of the Zeroconf stack[1], namely, DNS Service Discovery[2] built on Multicast DNS[3], and provides great user experience. For example it allows a student who enters his campus network to automatically connect his mobile devices (smartphone, tablet, notebook) among themselves, allowing file sharing and synchronization; it further allows to automatically connect to friend's devices on campus allowing to chat or share data and to connect to infrastructure devices like printers. This works seamlessly, without user interaction and regardless of the IP addresses and ports the corresponding services use.

Since Zeroconf is built on multicast, every machine in the same network will automatically receive all the announcement traffic and thus obtain a lot of information about the users in the network without having to send a single packet itself. Using mDNS-SD, devices publish their hostnames, commonly containing the user's name, when entering a network, followed by information about offered and requested services. When a user named Daniel enters the campus network, his Notebook publishes "Daniel's notebook joined the network" to all devices in the network. Many users are completely unaware of how chatty their devices are [4]. Most users do not consent to this information being published whenever they approach a McDonald's or Starbucks [4]. However, there is no user-accessible mechanism to limit or prevent this chattiness.

The device might further publish:

- "I want to sync Daniel's mobile folder with Daniel's smartphone."
- "I share the folders `/home/daniel/share`, `/home/alice/share`, and `/home/bob/share`."
- "I am online using *iChat* and my status is *gaming*."

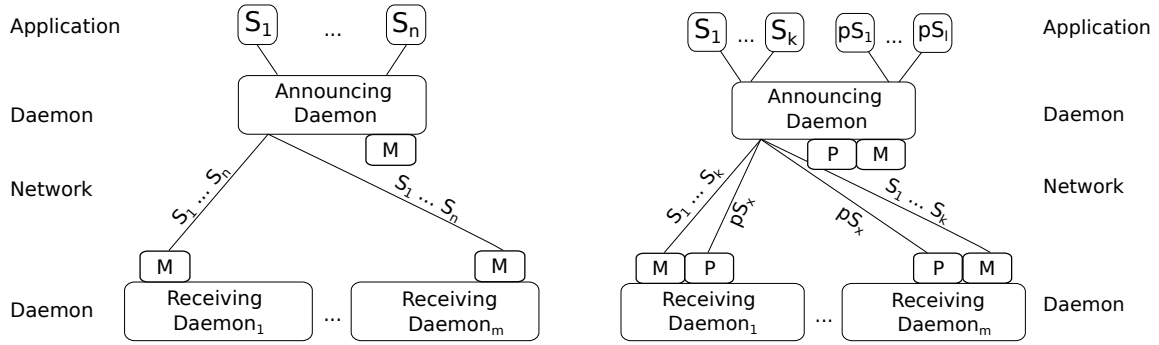
Offering shares might allow inferring names of family members, furthermore opening pathways to social engineering attacks, while a chat application shows the user's activity status to *everyone* in the same network. Most users do not even know how much information is published via mDNS-SD every time they connect their mobile devices to a network or come close to a known WLAN access point.

In this paper, we present a privacy extension for Zeroconf daemons, which

- guarantees privacy by sending private discovery data only to trusted hosts instead of multicasting it to everyone in the same network,
- significantly reduces multicast traffic, moving further towards the Zeroconf group's goal of reducing multicast traffic,
- is transparent to client software using mDNS-SD,
- is transparent to the existing network infrastructure,
- allows automatic service discovery like standard mDNS-SD,
- allows to present only services a user can use, reducing visual clutter,
- is fully backward compatible,[1]
- is very efficient, in all of network traffic, memory consumption, CPU time, and wall clock time.

To grant these features, a relationship between two hosts has to be established. This is done by an initial pairing during which a shared secret is exchanged. After this initial pairing, service discovery requires no further configuration allowing hosts to offer and request arbitrary service instances, which do not have to be known at the time of pairing. The IETF

[1]Public services' operation remains entirely unchanged; private services *seem* entirely unchanged within the trusted group.



(a) Traditional mDNS-SD. The Zeroconf daemon announces every service $S_1, \ldots, S_n$ to every device in the same network, by multicasting it to the *multicast sockets* M.

(b) mDNS-SD using our extension. While our solution still allows to announce services $S_1, \ldots, S_k$ in the traditional way, it is possible to announce services $pS_1, \ldots, pS_n$ in a privacy preserving way. This is done by sending chosen services pS_x to chosen devices' *privacy sockets* P using unicast.

Fig. 1. Standard mMDS-SD transmits all announcement traffic using multicast, while our extension can unicast announcements to *privacy sockets*.

Zeroconf charter[2] states that minimal configuration is tolerated for security's sake; we believe this includes privacy as well. To realize our goals, our solution extends mDNS-SD by

- an additional socket per device called *privacy socket*, which receives private unicast queries from paired hosts (see Figure 1),
- a meta service that distributes *privacy sockets*,
- a meta service that synchronizes pairing data among devices of the same user,
- an enhanced service browsing tool giving users control over mDNS-SD allowing tunable privacy.

All those concepts work seamlessly together and are transparent to both applications and the network.

II. Related Work

In previous work[5] we showed privacy problems arising while using mDNS-SD, and presented a proof of concept solution that shows that privacy in mDNS-SD can be granted with little effort. While this solution provides privacy with imperceptible additional CPU time, the solution we present in this paper

- grants even more privacy, because discovery data is only unicast to paired hosts rather than multicast to everyone although in encrypted form,
- significantly reduces the number of multicast messages,
- is much more convenient, because it needs only one pairing per pair of users, instead of one pairing per pair of service instance and service requester.

The solution presented in this paper outperforms the one we presented in [5] in privacy, network load, and user experience.

We are not aware of any other publication regarding privacy extensions for mDNS-SD. Nevertheless, the problem of mDNS-SD publishing device names has been addressed [6], [4].

[2] http://datatracker.ietf.org/wg/Zeroconf/charter

Aura et al. [6] investigate private information published on different network layers when connecting devices to a network. While they mainly look at other protocols, they also mention the privacy problem of device names published by mDNS-SD. As a solution, they propose network location awareness, that is allowing service discovery only in trusted networks. We consider that too restrictive and want to give the user the possibility to request and offer services in a privacy preserving way even if the network cannot be trusted. Further we want to mitigate the configuration overhead of making assumptions about the trustworthiness of the network in use.

The public announcement of device names by mDNS-SD is discussed in more detail in [4]. The authors conducted studies showing that almost 60% of the published device names contain real user names and that 90% of the users consider this as privacy problem. They propose making users aware of the problem and changing hostnames as a solution and also refer to network location awareness and identifier free networks[7]. The privacy problem of SRV and TXT records being published is not mentioned. We consider it important to hide all possibly private information, because it can be seen by anyone as easily as the hostname and might deserve more privacy protection (e.g. TXT records). Furthermore, users should not have to change hostnames.

Much research has been done in the area of privacy in wireless networks. Especially finding access points in WiFi-networks is related to our paper, because it also has to solve the task of finding entities in a network using broadcast, where everybody in the vicinity can listen to the messages. [8] presents a privacy preserving protocol for discovering WiFi access points. Since it is an extension to the 802.11 MAC protocol, it does not meet our requirement of not changing any deeper protocol layers.

Greenstein et al. [7] present an identifier free wireless OSI layer 2 protocol, which allows to find access points and communicate in the wireless network in privacy. Since it provides privacy on the link layer, it also solves the private service

discovery problem. But to allow private service discovery, the whole network infrastructure has to be updated, while our solution allows private service discovery by just updating the local Zeroconf daemon. Nevertheless, an identifier free network fulfills more privacy requirements than those we stated in [5]. It allows to hide information about a certain device being online in a network, because there are no MAC addresses published; thus allowing users to be untrackable. While we consider it beneficial to be able to meet more privacy requirements, we also consider it important to give users the possibility to easily hide the cleartext information published, while being independent of the network infrastructure.

Generic protocols for privacy preserving service discovery [9] and presence sharing [10] have been proposed. Since they use a central entity [9] and depend on a trusted broker [10], respectively, we did not use them because we want to keep mDNS-SD decentralized and do not want to rely on trusted third parties.

There is much research in the area of service discovery in pervasive computing environments[11] and mobile ad hoc networks[12]. Zhu et al. present a model for privacy preserving service discovery [13]. Since it is a generic model, we consider it unnecessarily complex for our application area. Nevertheless it is a sophisticated model and we want to incorporate ideas like the usage of bloom filters in our privacy extension in future work.

Mechanisms such as IPv6 privacy extensions[14] and locally administered MAC addresses also aim to reduce the ability of others to trace the whereabouts of end systems. These methods work independently of our privacy extension, on OSI layers 3 and 2, respectively, and thus can be used at the same time.

Device fingerprinting [15], [16] can be used to track the devices, even if the protocol in use has no explicit identifiers. It is outside the scope of this paper to address tracing using information leaked by the physical layer [17] and tracing of the users by other 'metadata,' such as the number or rate of connections devices open, or where they connect to [18].

Another research topic that is related to our work is efficiency and bandwidth usage of multicast in general and especially in 802.11 wireless networks. The efficiency problem of multicast in wireless networks is explained in [19]. Chow et al. [20] introduce mechanisms to mitigate the problem.

III. Efficient mDNS-SD Privacy Extension

Our main goal is to provide an extension for mDNS-SD that avoids application-layer distribution of private data. Figure 2 shows resource records published by mDNS-SD when using a chat application that is based on the `_presence` service. Since these records contain private data, we want only paired hosts to be able to receive them.

We also want to reduce the number of multicast packets sent. Traditionally, services that are restricted to a small user group, e.g. file-sharing, are implemented by selectively handing trusted users' access credentials, mostly in the form of a user-specific or shared password. Thus, information about services that can only be used by a few users is published to the whole local network, only to deny almost everyone access later. According to Cheshire[1] mDNS-SD is used to query services of a type one can sensibly use; services one can just connect to without understanding its language should not be offered. We extend this by also not offering services someone is not allowed to access.

```
_presence._tcp.local: type PTR,
daniel@Daniel's Notebook._presence._tcp.local
```

(a) PTR record containing the device's hostname, which by default typically contains the user's name.

```
daniel@Daniel's Notebook._presence._tcp.local:
type SRV, port 5298,
target Daniel's Notebook.local
```

(b) SRV record showing the hostname and port.

```
daniel@Daniel's Notebook._presence._tcp.local:
type TXT,
vc=! ver=2.10.6 node=libpurple
port.p2pj=5298 txtvers=1
status=gaming
last=Kaiser
1st=Daniel
```

(c) TXT record that contains several critical key value pairs like the first and last name of the user, the chat status and the version of the service.

```
Daniel's Notebook.local: type A, addr 134.34.10.36
```

(d) A record presenting a mapping of hostname to IP address.

Fig. 2. Resource records multicast when using a chat application that is based on the `_presence` service. Each of these resource records violates privacy.

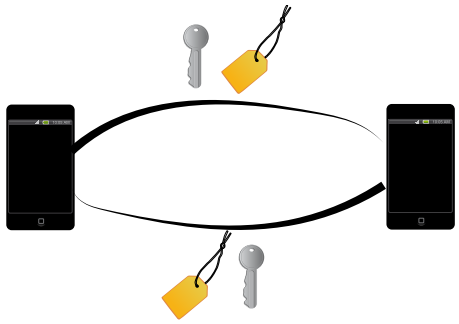


Fig. 3. During pairing, two devices exchange a symmetric key and a label (the name of a meta service instance described in subsection III-C) used to discover privacy aware services.

A. Pairing

Pairing is the process of establishing a relationship between two devices (see Figure 3). We distinguish two kinds of pairings, namely *inter user pairing* and *intra user pairing*; the first allows users to offer services to each other in a privacy preserving way, the latter allows privacy aware service discovery among devices of the same user. Users become "friends" with respect to our privacy extension, if they pair one of their devices with each other. To limit pairing to a single *inter user pairing* and one *intra user pairing* per

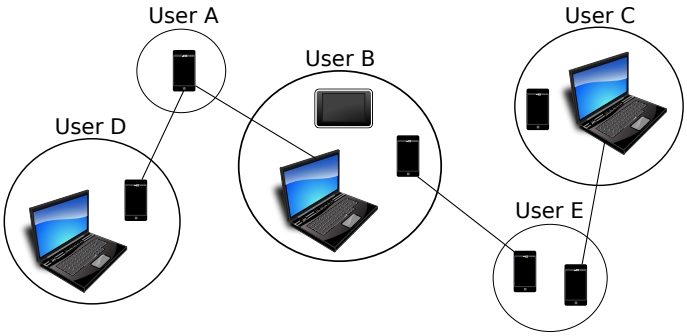


Fig. 4. If users' devices (in circles) are paired to each other, it is sufficient to pair one of their devices to another users device to be able to discover services offered by all devices of this user in a privacy preserving way. This is possible because pairing data can be synchronized among all devices of a user (see subsection III-C). User A paired his smartphone with B's Notebook which makes A and B friends; the pairing data synchronization allows B to discover A's services with his smartphone, even if A does not know this device.

device a user has[3], we introduce a meta service described in subsection III-C which allows a user to synchronize pairing data among his devices. Pairing data synchronization allows to discover services offered by all devices of friends[4], even if just one of them has been paired (see Figure 4).

Both *inter user pairing* and *intra user pairing* are realized by exchanging data via a secure or out-of-band channel, such as Bluetooth, NFC, photographing a QR code, encrypted Email, or SMS. The data consists of a label[5] and a random key only known to the pair of users. After exchanging this information, each user chooses an ID for the new friend, and stores the information received from the friend indexed by this ID in a hash table. The keys are needed to discover privacy aware services that do not have to be known at the time of pairing.

Queries for privacy aware services are only sent to friends that offer those services, and announcements of private services are only sent to friends that are authorized to connect. This might seem like destroying the possibility to connect to previously unknown services, but this can be allowed by just announcing a new service to a friend for the first time.

B. Privacy Socket

To grant privacy not only by encrypting multicast traffic, which would still allow inferring information, and to reduce multicast traffic, we introduce a *privacy socket* each host listens on in addition to the standard mDNS-SD socket. On this socket each privacy extension aware host receives queries from paired hosts. When receiving a query from an unknown host it is dropped silently. Testing whether a received packet is from a paired host can be done efficiently in $\mathcal{O}(1)$ time. Using a single *privacy socket* per host it is possible to offer instances of arbitrary service types to paired hosts via unicast. Figure 5 compares querying for a service using standard mMDN-SD 5(a) to querying using *privacy sockets* 5(b) and 5(c).

[3]A user can pair all of his devices to one of them and then use synchronization to get a fully connected pairing graph among his devices.

[4]If the friend wants services to be offered only by some of his devices, he is able to configure this using the tool we introduce in section V.

[5]The users' `_ppSOS` service instance name described in subsection III-C.

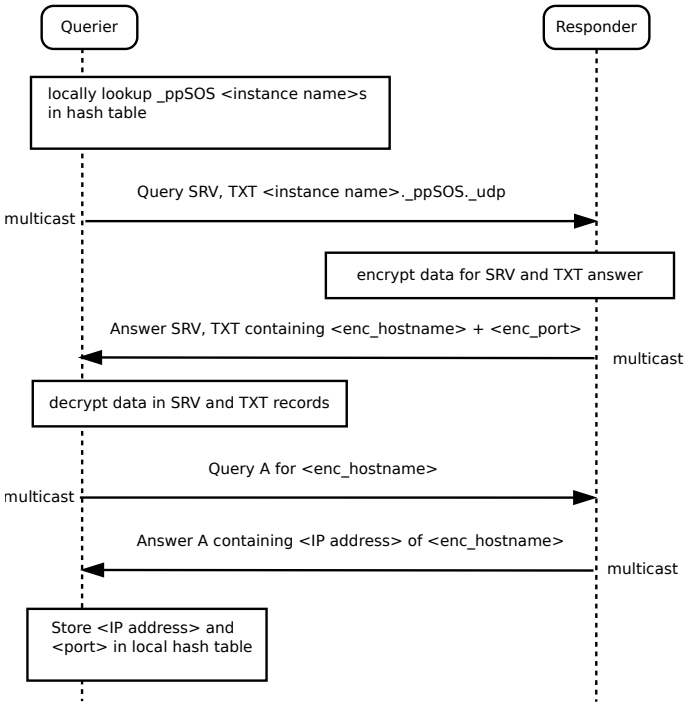
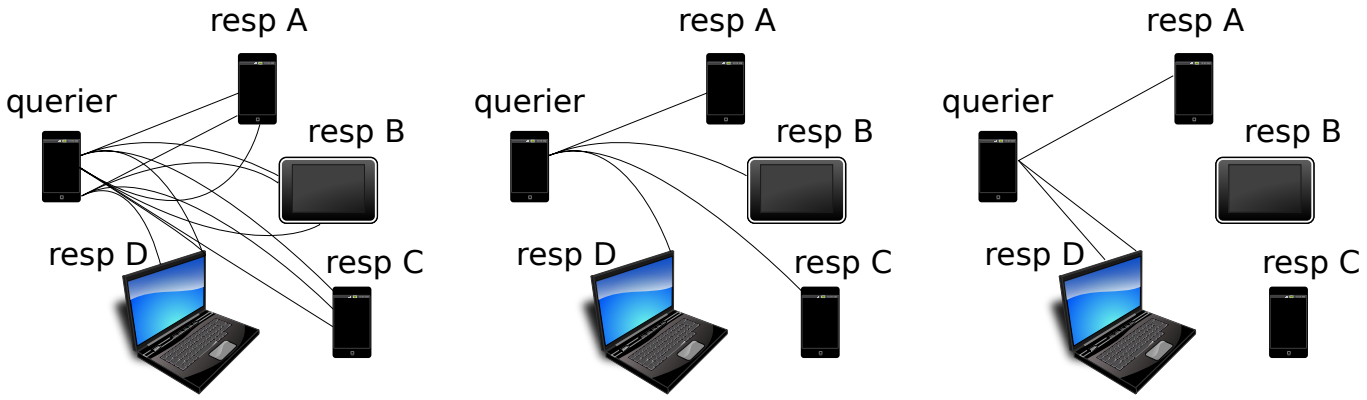


Fig. 6. Queries and responses sent for `_ppSOS` discovery. Discovery data corresponding to `_ppSOS` is multicast to the standard mDNS-SD socket; to grant privacy, possibly private data in the SRV and TXT records are encrypted before transmitting.

C. Meta Services

To fulfill our goals we need privacy preserving, efficient and convenient means to distribute information about the *privacy socket* to paired hosts, and to synchronize the pairing data of users' devices. To achieve this, we introduce the concept of *meta services*, which can be used to exchange information about other services or service groups. Meta services are published directly by our modified daemon. In the eyes of an unmodified daemon, a meta service looks like any other service. The meta service concept allows us to seamlessly integrate service information control in our privacy extension by using mDNS-SD itself. We introduce the following two meta services.

1) Privacy Preserving Service Offering Service: represented by the service type `_ppSOS`, is a meta service we use to publish information about a host's *privacy socket* to paired devices. When joining a local network, our modified daemon offers the hosts `_ppSOS` instance and browses for `_ppSOS` instances in order to find friends that are online in this network. The port and IP address of the *privacy socket* are extracted from the received SRV and A records, respectively, and stored in a local hash table indexed by the users' IDs. Information within the SRV and TXT records is encrypted with the symmetric key exchanged during pairing, granting privacy preserving *privacy socket* distribution. When receiving



(a) Using standard mDNS-SD each device using mDNS-SD in the same network is queried.

(b) Our extension just queries for the _ppSOS meta service using multicast. Since the packet content is encrypted, there are no privacy leaks.

(c) Queries for any other service are directly unicast to the *privacy socket* of the paired host offering the requested service.

Fig. 5. While standard mMDS-SD multicasts all query and response packets revealing a lot of information about users and causing a lot of multicast traffic (a), our solution only multicasts encrypted information about the _ppSOS service (b); all other services are queried directly using unicast (c). Private information is only accessible by chosen paired hosts.

a _ppSOS instance on the *multicast socket* a host looks up the service instance name in his hash table to determine whether it is from a friend without having to decrypt it. Figure 6 demonstrates querying for the _ppSOS service. Distribution using multicast is only necessary for the bootstrapping _ppSOS service. Other meta services can use the privacy preserving means of service distribution granted by the *privacy sockets*.

2) Privacy Data Sync Service: represented by the type _pdss, allows synchronizing pairing data. Since we want to minimize the pairing overhead to once per pair of users, there have to be means for a user to synchronize pairing information of his devices. The _pdss meta service grants very convenient synchronization of pairing data, allowing a user to automatically synchronize his devices by letting them join the same network. The *Privacy Data Sync Service* can also be used to backup pairing data. Using this method, no private data is leaked while synchronizing or backing up pairing data. This is due to the fact that the _pdss meta service only sends encrypted information to privacy sockets. This means secure meta service operation is granted by using the same privacy sockets other private services use[6]. Thus, our solution uses itself to synchronize and backup pairing data in a privacy preserving way.

D. Service Discovery

The queries and responses of all three stages of service discovery, namely *service browsing*, *service discovery*, and *address resolution*, are sent via unicast, encrypted using the symmetric key exchanged during pairing, to the privacy sockets of offering and requesting devices.

1) Service Browsing: is the process of asking for instances of a service type an application is interested in by asking for PTR records corresponding to a requested service type. Hosts that offer an instance of this type will answer by sending PTR records, which are used to list service instances. For example a chat application like iChat will ask for the service type _presence (see 2(a)). Instead of multicasting the PTR query, it is sent directly to the *privacy sockets* of

[6]except for the _ppSOS service which transmits encrypted data using multicast

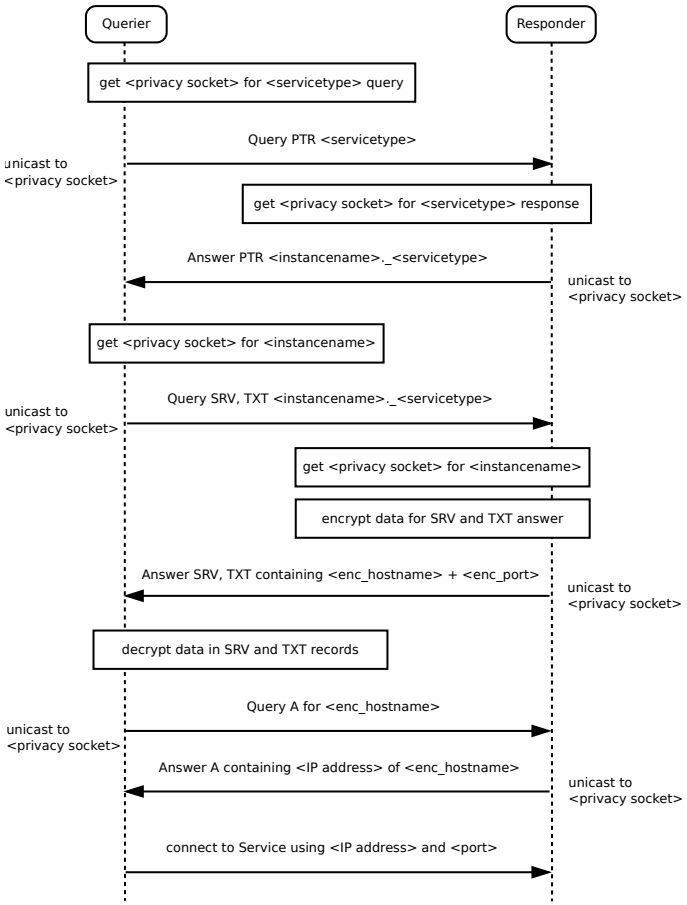


Fig. 7. Service discovery over unicast using a *privacy socket*.

paired hosts offering this service type. A list of IDs of users offering this service type is retrieved from a hash table set up by the enhanced service browser (see section V), a tool that allows users to manage mDNS-SD privacy in a fine grained way. The *privacy sockets* of these users have been distributed by the *Privacy Preserving Service Offering Service* (see subsection III-C) and stored in a hash table, allowing to retrieve all data needed to send a query in $\mathcal{O}(1)$ time. Service browsing is shown by the first pair of query and response in Figure 7. The received PTR answers are stored in another hash table associated with the IDs of the users that sent the answers.

2) Service Resolving: is the process of asking for the port a service instance uses and other information about this service instance, by asking for SRV and TXT resource records. The *privacy sockets* these queries have to be sent to are retrieved from the hash table built during service browsing. Service resolving is shown by the second pair of query and response in Figure 7. Further we offer the possibility to cache service instance names, which allows for direct resolving omitting the step of service browsing. Mimicking the concept of static service instances, we also offer the possibility to exchange static service instance names during pairing.

3) Address Resolution: is done by asking for A and AAAA resource records as standard DNS does. The *privacy sockets* the queries are sent to are the same as for service resolving. For each resolved service there is one corresponding A record. If one host offers two or more service instances, the A record of this host is queried only once. Since the only private data exposed during *address resolution* is the hostname of the user, which can be set to something less revealing, our extension also allows to send queries and responses for A records using multicast.

E. Implementation

We implemented our privacy extension based on the open source Zeroconf daemon Avahi[7]. We changed very little code in the avahi-daemon source files, just adding hooks to our loosely coupled privacy module before queries and answers are sent to and received from the client, respectively. Because of the very limited code changes in the standard avahi source and the loose coupling, it is easy to merge updates from Avahi to our modified daemon. The main part of our implementation is in a separated privacy library.

IV. Performance Evaluation

Users do not want to pay for privacy by complicated configuration, but they also do not want to pay by performance loss. One of the main goals of our privacy extension is to be at least as efficient as standard mDNS-SD in terms of both network and host performance. Our privacy extension does not increase the network load; in fact we reduce it by using mostly unicast instead of multicast. Further the computational overhead on the host devices is imperceptible for the user, both in terms of responsiveness of the system and battery life.

[7]http://avahi.org

A. Network

To be efficient with respect to network load, our privacy extension allows to offer private services only to friends that are allowed to connect and to send queries only to friends that offer the corresponding service, not bothering other hosts in the network with announcements of services they are not authorized to use and queries for private services they cannot possibly answer. Recall that private service queries are only sent to friends that offer those services, and announcements of private services are only sent to friends that are authorized to connect.

If a service instance has to be offered to multiple friends, or if a service type is offered by several friends, our privacy extension sends a unicast to each of those friends instead of sending a single multicast. This might look like a performance problem, but in small networks, the influence of mDNS-SD packets on the network load is negligible, whereas in huge networks sending multiple upstream packets has a lower impact then having to receive multitudes of downlink packets which is the case when using multicast.

Assuming each user wants to transmit one service discovery packet per time unit, using multicast, he has to transmit one packet but receive as many packets as there are users in the network; using unicast, he has to send and receive at most as many packets as there are online friends' devices. Since there are many services, that are only offered to very few friends, e.g. device synchronization, assuming to send each packet to 20 devices is already much for networks with up to 1000 members. If the network has more members the number of online friends devices rises as well, but multicasts get even more inefficient, making the unicast solution superior with respect to network load.

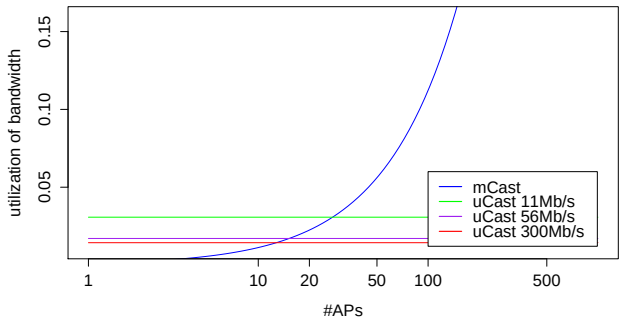
The influence of many multicasts on network load is especially severe in huge WiFi networks, because multicasts are transmitted using a very low transmission rate so that older devices not supporting higher transmission rates can receive the multicasts as well [19]. Hong et al. [21] show that 13% of their campus network bandwidth is used by mDNS-SD. To analyze the impact of our solution on network bandwidth in 802.11 wireless networks[22] compared to standard mDNS-SD we extend on [21].

The utilization of network bandwidth caused by service discovery can be expressed as

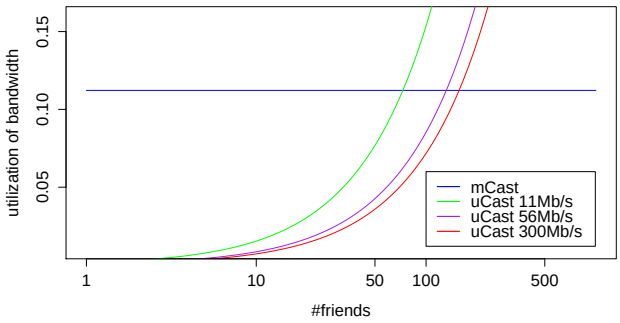
$$utilization_{\text{SD}} = \frac{T_D + T_U}{timeunit},$$

where T_D and T_U are the total transmission times for all service discovery packets on up- and downlink, respectively. The utilization contains all overhead that is needed when using 802.11, including physical layer data, namely PLPC Preamble and Header, and the inter frame spaces DIFS and SIFS. Definitions of T_D and T_U can be found in [21]; specifications of the 802.11 overhead sizes can be found in [22].

To compare our solution to standard mDNS-SD in terms of network bandwidth usage in 802.11 networks, we will examine how the number of access points and the average



(a) Bandwidth utilization caused by multicast and 20 unicasts dependent on the increase of access points in the wireless network. Multicast is less efficient than 20 unicasts in a network with just about 12 access points, when using the 802.11n (300 MB/s). Even when using the outdated 802.11b (11 Mb/s), unicast is more efficient in networks with about 24 and more access points.



(b) Bandwidth utilization caused by multicast and a variable number of unicasts in a wireless network with 100 access points. When using 802.11n (300 Mb/s), over a hundred unicasts can be sent while still being more efficient with respect to bandwidth utilization compared to multicast. Using 802.11b (11 Mb/s), 50 unicasts are still more efficient than multicast.

Fig. 8. Comparison of estimated network bandwidth utilization caused by multicast and unicast in 802.11 wireless networks without taking co-channel influences into consideration. Several unicasts are more efficient than a single multicast with respect to utilization of network bandwidth.

number of friends of a user influence the utilization of network bandwidth. The number of access points is important to consider, because while a single multicast needs one packet upstream, which can be transmitted with a possibly fast transmission rate, it needs as many downstream packets as there are access points, which have to be transmitted using the lowest bandwidth. When there are some access points in close proximity there is also the problem of side channel influence. We did not take that into consideration, but it further increases the bandwidth utilization caused by multicast. Like described above, privacy aware services need to send one packet per device that is online and authorized to receive the corresponding service discovery message. Again assuming 20 unicasts per message, Subfigure 8(a) shows that in a network with about 12 access points sending unicasts to 20 devices on average is more efficient then multicast, when using 802.11n (300 Mb/s) [22]. Even when using the outdated 802.11b (11 Mb/s) [22] 20 unicasts are more efficient in networks with a little more than 20 access points. Subfigure 8(b) shows that in a network with 100 access points sending each message to over 100 devices is still more efficient than multicast, if the devices support 802.11n[22] and to about 50 devices if they use the outdated 802.11b[22].

There are other disadvantages of multicast in 802.11 wireless networks described in [19], like handling host sleep mode, which further increases the bandwidth utilization and battery drain, because devices have to stay awake if multicast traffic is waiting.

B. Host

Our privacy extension has an imperceptible computational time overhead compared to standard mDNS-SD, which even on mobile devices vanishes compared to packet delays of mDNS-SD that are in the range of 20 to 120 milliseconds. These delays mitigate the problem of multicast bursts when multiple devices are reset at the same time[3].

The computational overhead caused by our privacy extension consists mostly of regular expression checking, hash table lookups and sending packets multiple times. For both incoming and outgoing packets we use one regular expression on the resource records' names and one hash table lookup to check if the corresponding service is private. For private outgoing packets we need $\mathcal{O}(n)$ hash table lookups and have to send n packets, where n is the average number of online friends that are authorized to connect to a service. For private incoming packets we only need $\mathcal{O}(1)$ hash table lookups, because the packet comes from exactly one friend, and $\mathcal{O}(1)$ hash table updates, because this packet might contain new information that has to be stored. Assuming $n = 20$, which is a high number, we need less than a hundred hash table lookups and a regular expression lookup, causing an estimated overhead of a few hundred micro seconds. Together with sending about 20 additional packets this is still negligible compared to mDNS-SD delays. On the receiving side, where having to handle bursts of packets is more likely, computational overhead is virtually non-existent.

To check if there are perceptible delays for the user, we tested our implementation in our work group network with a few clients and 500 private services published, and could not observe any delay compared to standard mDNS-SD.

V. Adding User Control to mDNS-SD

The daemon alone is not sufficient without a user-friendly service browser, extended to give users straightforward control which services types and instances they want to be published publicly, which services they want to be published in private, and which services they do not want to be published at all. Together they empower the users with mDNS-SD privacy and transparency. The *Enhanced Service Browser* grants these features giving users means to tune mDNS-SD privacy in a

fine grained way. Further it informs the user when a (newly installed) application wants to offer or request a service which is not configured yet, instead of publishing the information without the user's awareness. To ease configuration the *Enhanced Service Browser* allows to set sensible defaults. The tool also gives the functionality of a traditional service browser, namely listing available service instances; in addition it shows friends and other devices of the user that are currently online.

VI. Conclusion and Future Work

In this paper we introduced a privacy extension allowing Zeroconf service discovery without multicasting private data, while being efficient and transparent. It is very user friendly both in terms of overhead and control, and significantly reduces the number of multicast messages sent, making mDNS-SD more efficient. A major advantage of our extension to existing solutions (see section II) is that none of the existing OSI layer protocols and none of the existing client software has to be altered. Only the Zeroconf daemon running on the users devices has to be modified, while afterwards still being able to exchange service information with unmodified daemons. While our privacy extension provides means to publish services in a privacy preserving way, it still allows to publish nonsensitive services to all users in a network via multicast, and thus is fully backwards compatible.

In the future we plan to introduce a better key management, that will also efficiently support offering private services to groups. The key management will be based on asymmetric cryptography, and should allow to use a mechanism similar to DTLS[23] and a zero knowledge authentication mechanism. We further want to give a security evaluation of the updated system and plan to evaluate our implementation in detail. We also plan to include two user friendly pairing methods: for users meeting each other, allowing them to pair their devices using NFC, and when they cannot easily meet. Furthermore, we hope to improve the privacy protecting abilities such that the possibilities of inferring information is reduced, while still remaining compatible with standard mDNS-SD. This includes substituting the fixed `_ppSOS` instance name with identifiers that still allow efficient discovery; this is important to avoid tracking and to be able to exclude friends from being able to see the `_ppSOS` announcements. Moreover we want to extend our solution to Wide Area Service Discovery.

References

[1] D. H. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*. O'Reilly, 2006. I, III

[2] S. Cheshire and M. Krochmal, *DNS-Based Service Discovery*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6763. I

[3] ——, *Multicast DNS*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6762. I, IV-B

[4] B. Konings, C. Bachmaier, F. Schaub, and M. Weber, "Device names in the wild: Investigating privacy risks of zero configuration networking," in *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, vol. 2. IEEE, 2013, pp. 51–56. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6569062 I, II

[5] D. Kaiser and M. Waldvogel, "Adding privacy to multicast DNS service discovery," University of Konstanz, Tech. Rep., 2014. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:bsz:352-282346 II

[6] T. Aura, J. Lindqvist, M. Roe, and A. Mohammed, "Chattering laptops," in *Privacy Enhancing Technologies*. Springer, 2008, pp. 167–186. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-70630-4_11 II

[7] B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall, "Improving wireless privacy with an identifier-free link layer protocol," in *MobiSys '08: 6th International Conference on Mobile Systems, Applications, and Services*, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378607 II

[8] J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe, "Privacy-preserving 802.11 access-point discovery," in *Proceedings of the second ACM conference on Wireless network security*, ser. WiSec '09. New York, NY, USA: ACM, 2009, pp. 123–130. [Online]. Available: http://doi.acm.org/10.1145/1514274.1514293 II

[9] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 24–35. [Online]. Available: http://dl.acm.org/citation.cfm?id=313462 II

[10] L. Cox, A. Dalton, and V. Marupadi, "Smokescreen: flexible privacy controls for presence-sharing," in *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007, pp. 233–245. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247688 II

[11] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive computing*, vol. 4, no. 4, pp. 81–90, 2005. II

[12] F. Outay, V. Vèque, and R. Bouallègue, "Survey of service discovery protocols and benefits of combining service and route discovery," *IJCSNS*, vol. 7, no. 11, p. 85, 2007. II

[13] F. Zhu, W. Zhu, M. W. Mutka, and L. M. Ni, "Private and secure service discovery via progressive and probabilistic exposure," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 11, pp. 1565–1577, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4339200 II

[14] T. Narten and R. Draves, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, ser. Request for Comments. Internet Engineering Task Force (IETF), January 2001, no. 3041. II

[15] T. Kohno, A. Broido, and k. claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, no. 2, pp. 93–108, May 2005. II

[16] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, "Identifying unique devices through wireless fingerprinting," in *Proceedings of the first ACM conference on Wireless network security*, ser. WiSec '08. New York, NY, USA: ACM, 2008, pp. 46–55. [Online]. Available: http://doi.acm.org/10.1145/1352533.1352542 II

[17] K. Bauer, D. McCoy, B. Greenstein, D. Grunwald, and D. Sicker, "Physical layer attacks on unlinkability in wireless lans," 2009. II

[18] B. Greschbach, G. Kreitz, and S. Buchegger, "The devil is in the metadata - new privacy challenges in decentralised online social networks," in *Proceedings SESOC, PERCOM*, 2012. II

[19] E. Vyncke, *Why Network-Layer Multicast is Not Always Efficient At Datalink Layer*, ser. Internet Draft. Internet Engineering Task Force (IETF), 2014. II, IV-A

[20] C. T. Chou and A. Misra, "Low latency multimedia broadcast in multi-rate wireless meshes," in *IEEE Workshop on Wireless Mesh Networks*, 2005. [Online]. Available: http://www.research.ibm.com/people/a/archan/chou-misra-wimesh.pdf II

[21] 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. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5426121 IV-A

[22] *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. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11-2012.pdf IV-A

[23] N. Modadugu and E. Rescorla, "The design and implementation of datagram tls." in *NDSS*, 2004. VI