Collaborative On-demand Wi-Fi Sharing

Hanno Wirtz, Torsten Zimmermann, Martin Serror, Klaus Wehrle
Chair of Communication and Distributed Systems, RWTH Aachen University
{wirtz, zimmermann, serror, wehrle}@comsys.rwth-aachen.de

***Abstract*—While users can ubiquitously access the Internet via their Wi-Fi network or their mobile carrier network, access to foreign private Wi-Fi networks is mostly prohibited. This is because private APs have no means of authenticating foreign mobile users prior to granting access to their network, entailing severe security and liability risks. However, such *Wi-Fi roaming* would make the vast network resources of private users available on a collaborative basis. We propose Collaborative On-demand Wi-Fi Sharing (COWS), offering 802.1x-equivalent authentication of foreign users at private APs without the need for elaborate, hierarchical authentication infrastructures. COWS embeds authentication credentials into 802.11 association requests, enabling APs to establish 802.11 AP networks exclusively *on-demand* and *after* an authentication of the mobile user at her home network provider. Our evaluation using Android smartphones and various authentication provider instances shows the real-life applicability of COWS, enabling lightweight, collaborative Wi-Fi roaming at the APs of private users.**

I. Introduction

While users comprehensively enjoy broadband Internet access at home and via the mobile network carrier in their country, high-speed access to the Internet remains scarce and expensive while roaming, i.e., in foreign countries. The same holds true for locations where neither the home Wi-Fi network nor the mobile network is available. This lack of network access stands in stark contrast to the global proliferation of private Wi-Fi Access Points (APs) that, in principle, offer comprehensive wireless network coverage for mobile users. However, access to foreign Wi-Fi networks, i.e., *Wi-Fi roaming*, is typically prohibited for security and liability reasons [9], [17].

This is because granting network access to a foreign user, without any form of authentication, entails unforeseeable security issues resulting from the user's action in the network. Especially, the legal liability for all Internet traffic originating from the private network lies with the owner, i.e., the user granting access is responsible for all malicious or illegal actions taken by the foreign user, e.g., attacks on other network entities or the exchange of illegal content.

While roaming agreements exist between mobile network providers, at additional costs, no widespread approach exists for Wi-Fi roaming. The *Hotspot 2.0* [4] task group at the Wi-Fi Alliance addresses this fact along with the usability of current commercial Wi-Fi hotspots. Similar, the *eduroam* [7] initiative affords a designated Wi-Fi network at participating educational organizations. Both solutions (cf. Fig. 1, left) build on contractual agreements between large-scale commercial hotspot providers or organizations, respectively, and leverage 802.1x [1] authentication of mobile users that follows the provider or organization hierarchy to alleviate the aforementioned security and liability issues. In the requirement of long-lasting agreements and a hierarchically managed backbone network, both solutions are unsuitable for Wi-Fi roaming at private APs, leaving their comparatively larger potential unused.

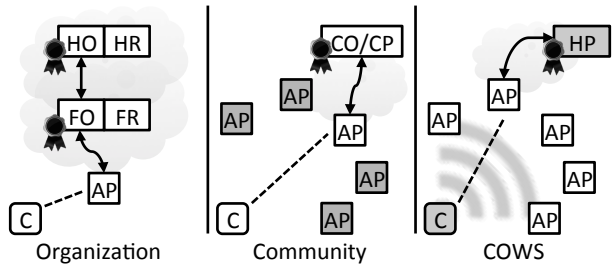


Fig. 1. Organizational Wi-Fi roaming (left) require infrastructures for hierarchical 802.1x authentication at the RADIUS servers (FR, HR) of the foreign and home organization ((FO, HO). Commercial or collaborative Wi-Fi sharing communities (middle) only make members' APs available. COWS comprehensively discovers roaming networks and offers 802.1x-equivalent authentication at private APs by embedding user credentials in 802.11 PREQs. Internet traffic is tunneled through the user's home provider (HP).

Collaborative approaches (cf. Fig. 1, middle) target this potential by exploiting the coverage provided by private APs. As a commercial solution, *FON* [5] establishes a designated Wi-Fi network via proprietary AP, in order to achieve centrally authenticated network access overlaid over the respective private networks. In contrast, community approaches grant access to a dedicated network at the APs of members unconditionally [6] or authenticate users decentralized within the community [9], [12], [13], [17]. Still, collaborative approaches only offer network access to members of a single community at other members' APs, limiting network availability to the (typically limited) size of the community and accidental encounters of suitable APs.

We thus propose Collaborative On-demand Wi-Fi Sharing (COWS) (cf. Fig. 1, right), embedding 802.1x-like authentication of mobile users at private APs, without the need for a managed network hierarchy, *within* the on-demand request and instantiation of a personalized 802.11 network. Specifically, users in COWS opportunistically broadcast, in a standard 802.11 PREQ, a request for a personal, authenticated, and secure Wi-Fi network. The request contains the provider information as well as credentials to authenticate the mobile user at its home network provider[1], enabling overhearing APs to authenticate the user at the provider *prior to instantiating the Wi-Fi network*. Upon authentication, the provider securely communicates the network parameters, i.e., the network SSID and WPA2 PSK, back to the AP, enabling it to instantiate a secure, personal network to which the user can associate. Within this network, all payload traffic is then routed via an encrypted tunnel to the home network provider, removing any legal liability from

[1]Mobile network or residential broadband provider.

the AP operator. Traffic provided for a foreign user can be accounted for at both the AP and the provider, rendering a variety of incentive or micro payment schemes possible.

COWS thereby mitigates the infrastructure and contractual requirements of existing roaming or Wi-Fi sharing approaches (Section II). By embedding network requests in the legacy 802.11 association mechanism and adapting the 802.11 AP functionality to authenticate users at the provider, COWS enables flexible and on-demand network instantiation. Especially, COWS comprehensively protects authentication credentials to mitigate misuse and attacks on the provider or AP (Section III). Our implementation for Android smartphones and the Linux `hostapd`[2] 802.11 AP daemon demonstrates the applicability and performance of our design (Section IV), affording legacy-compliant, collaborative Wi-Fi roaming via personal and secure, on-demand 802.11 network instantiation (Section V).

II. Related Work

COWS departs from commercial or organizational Wi-Fi roaming approaches in favor of private Wi-Fi sharing. We hence embed our design in the state of the art of both directions. Furthermore, we point out an SDN-based solution for managed provider networks instead of private networks.

A. Wi-Fi Roaming

802.1x [1] constitutes the basis for current Wi-Fi roaming approaches, such as the *eduroam* [7] initiative. Organizations thereby contribute their managed wireless network infrastructure and forward the authentication details of roaming users along a hierarchy of RADIUS authentication servers to the host organization of the mobile user. After authentication, the visited organization grants port-based, WPA2-secured network access. Such approaches build on long-lasting agreements between the underlying national research networks as well as the provision of a hierarchical authentication infrastructure. In addition, the requirement of being a member of a participating organization, e.g., a student or a faculty member, and the infrastructure requirement prevent the application to personal Wi-Fi roaming.

eduroam thereby serves all users in a single, dedicated wireless network at participating APs. In case multiple commercial network providers want to offer Wi-Fi roaming for their users, this setting might be too rigid and undifferentiated. Enabling the provision of distinct networks with predefined payment and access settings, the *Hotspot 2.0* initiative [4] extends hierarchical 802.1x [1] authentication with 802.11u [2] network selection. While participating APs, i.e., hotspots, still permanently operate and announce one or multiple networks, mobile clients can discover their appropriate network prior to an association. Again, spanning a hierarchical infrastructure over arbitrary private APs as well as the permanent operation of foreign networks renders such an approach infeasible.

B. Wi-Fi Sharing

Recognizing the potential that private APs offer, *Wi-Fi community networks* and *Wi-Fi sharing* strive to make foreign networks accessible to mobile users. In a commercial Wi-Fi community realization, *FON* [5] offers a dedicated network at community members at the cost of an additional, proprietary Wi-Fi AP that is inserted in the user's network. All *FON* members may access this network and traffic is then handled by the *FON* backend infrastructure. Users are thus bound to a single Wi-Fi sharing provider and need to trust this third-party provider with authentication, traffic security, and accounting.

Approaches may furthermore build on distributed 802.1x authentication of mobile users in a backend, e.g., *SWISH* [12] and *Wireless Roaming via Tunnels* (WRT) [13]. Based on the proposed key establishment protocols, traffic is securely tunneled to the home network of the user and forwarded to the intended Internet address. Both approaches assume the existence of a home organization, equivalent to the host organization in the *eduroam* initiative, and authentication infrastructure.

Alternatively, traffic can be tunneled directly to the actual home AP or router of the mobile user, e.g., in *PISA* [9] and the design presented in [17]. In this, the certifiable membership to a (city-wide) Wi-Fi sharing community serves as the authentication means of users. Instead of tunneling traffic to a dedicated network element at the home network, both approaches target commodity AP devices. This requires a dedicated protocol on both the mobile device and the home network AP, e.g., the Host Identity Protocol (HIP) [14] in *PISA* [9], for authentication, tunnel management, and bidirectional traffic security.

We target similar scenarios and propose a roaming structure that is equivalent to the aforementioned approaches. Our design strives to improve on the following two shortcomings. First, all presented approaches require the operation of a *single, permanent* network that announces and supports the roaming service, wasting resources if no user wants to use the service. Second, users are bound to a single community (provider) and must grant foreign users access to their network prior to [12], [13] or without explicit authentication [6], [9], [17].

Conversely, we strive for flexible Wi-Fi roaming that is instantiated on-demand, exclusively driven by the indication of demand for a roaming network by mobile users. We place all control over the provision of the roaming service, i.e., whether a Wi-Fi network is instantiated and whether a foreign user is granted access, at the AP and, in extension, its owner. Our design can incorporate arbitrary sharing community or provider memberships and flexibly and spontaneously accommodates arbitrary roaming (and thereby incentive) configurations. Furthermore, COWS only requires a lightweight software adaptation of the AP and is fully compatible to the capabilities of unmodified smartphones, ensuring real-life applicability.

C. Orchestration of Wireless Networks

Functionally related to our design, *Odin* [18] proposes a Software-Defined Networking (SDN)-based solution to the instantiation (or orchestration) of wireless AP networks in managed provider scenarios, e.g., at universities or companies. In *Odin*, mobile users access 802.11 APs that are attached to an SDN-enabled backend network in which control over the instantiation and configuration of wireless networks resides at a central controller of *Odin* and OpenFlow. APs then run both an OpenFlow and an *Odin* client, supporting mobility management, load balancing, and channel reconfiguration.

[2] http://w1.fi/hostapd/

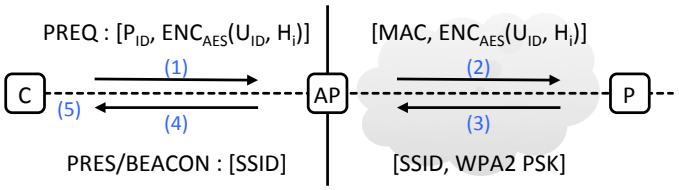


Fig. 2. COWS design overview. Mobile clients indicate their authentication credentials within an 802.11 network request (1). Via a secure HTTPS connection, the AP authenticates the client at her/his home network provider (2) and, based on the retrieved configuration (3), instantiates an 802.11 network (4). All client payload traffic (5) is then routed in a secure tunnel over the home network provider (dashed line).

Analogous to *Odin*, we address Bring-Your-Own-Device (BYOD) settings due to the unmanageable diversity of user devices. Also, COWS instantiates multiple, custom AP networks, similar to the concept of *Light Virtual APs (LVAPs)* belonging to *network slices*, as a means of incorporating diverse roaming and mobility configurations. However, COWS realizes on-demand network instantiation in unmanaged private setups, in contrast to the managed provider setting in *Odin*. This requires the provision of user authentication credentials and provider details *by the user* contrary to the trusted scenario in *Odin*.

III. COWS Design

Fig. 2 illustrates a high-level overview of our design. COWS builds on *instantiating* Wi-Fi sharing networks for authenticated clients, mitigating the liability risks of granting access to a standing network for unauthenticated clients. We briefly explain the concept and detail each step in the respective sections.

A mobile client **(C)** transports both a request for a personal Wi-Fi roaming network and his authentication credentials in the *space-constrained* 32 Byte SSID field of an 802.11 Probe Request (PREQ) (step 1 in Fig. 2) and broadcasts this PREQ within the 802.11 association process. A COWS-capable Access Point **(AP)** that overhears this request forwards the credentials, via a HTTPS connection over the Internet, to the user's indicated home network provider (step 2, Section III-B). The provider **(P)** validates the user credentials and calculates the 802.11 network configuration, i.e., the network SSID and WPA2 PSK key, and indicates the validity of the request to the AP by sending the configuration (step 3, Section III-C). Using this configuration, the AP instantiates an on-demand, secure 802.11 AP network to which the client associates (step 4, Section III-D). The client then securely tunnels all traffic through the home network provider to relieve the AP operator from all liabilities (step 5).

We discuss incentive and accounting models for collaboration as well as argue its applicability to current network scenarios (Section III-E). Last, we detail the security characteristics of COWS regarding possible attacks or misuse (Section III-F).

A. Assumptions & Prerequisites

While we assume the AP operator to share his Internet connection with a mobile user in COWS, we refrain from the assumption of reciprocal trust or authentication between the two parties. As such, the AP operator needs to authenticate the user at an accountable third party, in this case the user's home network provider. In turn, the client does not want to disclose his identity and authentication credentials to the AP operator. Last, the AP operator and client share an interest in not handling Internet traffic of the client directly at the AP because of the operator's legal liability and the client's lack of trust regarding traffic confidentiality and handling. We thus sketch the prerequisites for COWS with regard to client authentication, confidentiality, and traffic security within.

Initially, the client registers for the roaming service at her/his home network provider and generates a public/private key pair specifically for COWS. We envision this to occur "offline", i.e., while the client is attached to his home network. In this step, the client and the provider also establish a shared symmetric key for the encryption and decryption of authentication credentials. We choose a symmetric encryption scheme to meet the space constraints in the SSID field of a standard 802.11 PREQ. The client constructs these credentials in the form of an n-step *hash chain* of which the client shares the anchor element H_n with the provider. By proving the possession of prior authentication tokens of the hash chain (H_{n-1}, H_{n-2}, ...) in subsequent interaction, the client can authenticate itself at the provider, i.e., by proving the input H_{j-1} to generate a result $H(H_{j-1}) = H_j$ of a computationally irreversible hash operation $H()$ [11], [15]. Currently, we use 128 Bit `CityHash`[3] tokens due to the space constraints of the 32 Byte SSID field in 802.11 PREQs. Because COWS never sends tokens unencrypted, $H()$ also does not necessarily have to be a cryptographic hash function.

For identification, we assume a unique user ID U_{ID}, assigned by the provider, for the human owner of each mobile client. In turn, we identify providers by the unique combination P_{ID} of their publicly assigned Mobile Country Code (MCC) and Mobile Network Code (MNC)[4], e.g., 310004 for Verizon in the USA and $26201X$ for T-Mobile in Germany. While other identifiers are possible, we choose this structure to match the aforementioned SSID space constraints as well as to prevent fake or malicious "providers" from entering the system by relying on the public MCC/MNC assignment.

B. Network Request

Clients in COWS *request* a personal Wi-Fi sharing network while simultaneously enabling overhearing APs to *authenticate* the client in 802.1x fashion at the client's home network provider. We realize this functionality in a single step, as illustrated in Fig. 2 (step 1), within the client-side broadcast of an 802.11 PREQ as part of the 802.11 association process. We thus reuse the function of 802.11 PREQ frames, i.e., requesting a Wi-Fi network, but transport within the 32 Byte SSID field a COWS prefix, a provider ID P_{ID}, and the user authentication credentials (cf. Fig. 3). The credentials thereby are the unique user ID U_{ID}, as assigned by the provider, and the next unused element in the hash chain H_i. *Overloading* network requests this way on unmodified devices is possible for applications by triggering a scan for a specific Wi-Fi network, i.e., SSID.

Using the provider ID P_{ID}, the AP is able to forward the request to the home network provider via a lookup that resolves P_{ID} to an IP address. Because of the very limited number of commercial home network providers, this lookup can occur via

[3] https://code.google.com/p/cityhash/

[4] http://en.wikipedia.org/wiki/Mobile_country_code

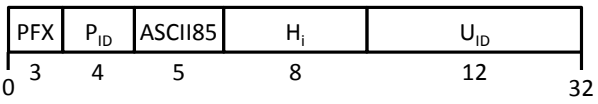


Fig. 3. Current 32 Byte SSID partitioning in PREQs: 3 Byte COWS prefix (PFX), 4 Byte provider ID (P_{ID}), 8 Byte authentication token (H_{ID}), and 12 Byte user ID (U_{ID}) (both AES-encrypted). Ascii85 encoding (ASCII85) for legacy device and OS support inflates 20 Byte user credentials to 25 Byte.

a local table on the AP or acquired via DNS. In forwarding the request, the AP attaches the MAC address of the client device.

The client protects the user ID U_{ID} and the authentication token H_i by encrypting them using the symmetric key established off-line with the provider (cf. Section III-A). We choose symmetric encryption due to the space constraints of the 32 Byte SSID. In our implementation, we use the Advanced Encryption Standard (AES) in Counter mode (CTR), with the *index* i of the current authentication token H_i combined with a zeroed counter serving as the Initialization Vector (IV) for encryption on the client. We use a zeroed counter as the client encrypts only a single block instead of a data stream.

The encryption of authentication credentials serves multiple purposes. First, broadcast 802.11 PREQs are observable by all wireless devices in transmission range. Encrypting credentials thus prevents capture and misuse, e.g., for replay attacks by malicious APs. Second, we do not assume a trust relation between the client and AP operator, as we explicitly want to enable roaming outside of the small possible set of such trust relations. Third, the AP functionality in COWS does not require knowledge of the user's identity. We discuss the security considerations and implications of replaying, attacks, and lost frames in detail in Section III-F.

In COWS, we reuse concepts of our prior work [19], in which we embedded semantic content identifiers in 802.11 broadcast frames to match content requester and provider. Notably, while we strive for 802.1x-equivalent authentication in COWS, our original design in [19] does not address this issue. In this paper, we address a comprehensively different communication scenario and propose a novel combined on-demand authentication and network request scheme.

C. Authentication at the Provider

Upon receiving a roaming request from an AP, the provider uses the MAC address to look up the *index* $i+1$ of the current, i.e., last used/seen, hash chain element H_{i+1}. Expecting a new request, the provider calculates the AES IV using the next index i plus a nulled counter to decrypt the encrypted user ID U_{ID} and authentication token H_i using the shared symmetric key. By validating the freshness of the included authentication token, i.e., calculating if $H(H_i) = H_{i+1}$, the provider ensures the authenticity and timeliness of the request. If the check fails, the provider assumes that the request has been tampered with or is of malicious nature. If multiple identical requests arrive, e.g., from spatially close APs overhearing the request, the provider will only serve the first.

Note that the client MAC address is used only for identification and does thus not need to be the actual hardware address of the 802.11 card. Instead, the client and the provider can establish a method of deterministically calculating MAC address sequences, similar to hash tokens, or cryptographic addresses [8]. This way, a client can not be tracked or de-anonymized based on observed broadcasts.

However, requests and authentication tokens within them might get lost, either on the wireless link between a client and a potential AP or if no COWS AP is in transmission range. In case the request contains a valid MAC address, the provider can iterate through possible hash chain indices, calculate the respective IVs, and try to decrypt the authentication credentials. This fail-over mechanism is limited by the length of the hash chain and offers a natural, adjustable "cut off" point, i.e., the point where a client is required to first contact the home network provider in order to validate her/his integrity. We discuss the (attack) scenarios that make this fail-over mechanism necessary in Section III-F and quantify the performance penalty of increasing numbers of missing requests in Section IV-D.

In case of a positive check, the provider serves the request by calculating the 802.11 SSID and WPA2 PSK, i.e., the Wi-Fi network configuration, and securely, e.g., via HTTPS, sends this configuration to the AP (step 3 in Fig. 2). Because no communication channel between the client and the provider exists yet, we enable both to calculate the configuration autonomously based on the current authentication token H_i. Specifically, the client and the provider calculate $S_i = \texttt{sha256}(H_i)$[5]. We thereby irreversibly compute a 256 Bit string from H_i and derive the SSID from the first 128 Bit substring of S_i and the WPA2 PSK from the second 128 Bit substring of S_i. Again, we encode both substrings in Ascii85 and use them as the network SSID and input to the PBKDF2 function specified in 802.11 for the actual WPA2 PSK generation, respectively.

In this, the client and the provider generate a one-time network configuration that is different for each network instantiation. Furthermore, the authentication token H_i is not revealed to the third-party AP to ensure the integrity of the hash chain against any kind of capture attack by malicious APs.

D. Network Instantiation, Association, and Usage

The reception of a network configuration by the provider signals a positive check of the roaming request and triggers the instantiation of the indicated Wi-Fi network at the AP (step 4 in Fig. 2). We refer to our previous work [19] for a detailed discussion of the actual Wi-Fi network instantiation process. If no or a negative reply occurs, the AP does not instantiate a network. The client observes the availability of the network, i.e., the calculated SSID, either via the beacon frames sent by the AP or an active 802.11 scan. Subsequently, the client associates to the network using the calculated WPA2 PSK network key.

We make use of the widely supported technique of *wireless network virtualization* [3] at the AP to support concurrent network instantiation for multiple clients. Furthermore, this ensures the availability of the Wi-Fi network used by the original owner of the AP. Additionally, while not addressed in this paper, traffic shaping techniques become possible for each virtualized wireless network, affording fine-grained control as well as protection of the owner's network performance.

[5] Any secure cryptographic hash function might be used.

Isolation and protection of client payload traffic then occurs on two levels (step 5 in Fig. 2). First, the client establishes a secure tunnel to the provider by encrypting all traffic with a symmetric key derived via standard public key cryptography. Second, the AP configures a strict routing rule for the instantiated network that forwards all traffic to the home network provider, e.g., using `iptables`. The AP thus forces potentially malicious Internet traffic to originate from the home network provider, thereby avoiding any legal liability [9], [17]. In turn, the client ensures the confidentiality and integrity of her/his traffic by handling it via the trusted network provider.

E. Control and Incentives for Collaboration

COWS enables collaborative Wi-Fi sharing at private APs. We build on the observation that typical residential Internet uplinks are accounted on a flat rate basis and forwarding foreign users' Internet traffic to their home provider does neither incur costs for the AP owner nor for the mobile user. However, it is important to offer incentives for collaboration as well as control over the system in the case of malicious or free-riding users. To this end, COWS inherently supports accounting of client traffic at both the AP and the provider as well as the possibility to compare the respective measurements to prevent entities from cheating.

Accounting "receipts", that are cryptographically signed by the AP and provider, could then be collected and managed by a service that facilitates a variety of micro payments [16]. For example, in a simple "tit-for-tat" scheme, the AP owner would gain roaming credits to be used at other users' APs in the system. Alternatively, network access could be repaid via Bitcoins or credits in crowdsourcing platforms, e.g., Amazon Mechanical Turk. Please note that arbitrary accounting and micro payment schemes are possible *on top* of COWS. However, we do not cover such add-on services in this work since (as of now) we lack the necessary user base to validate them.

F. Security Considerations

The security aspects of COWS revolve around the loss or replay of requests and the authentication credentials within them. Attackers could hence strive to trigger of aggravate such losses. We differentiate between two types of attackers: 1) Passive attackers that overhear and possibly replay requests, and 2) active, malicious APs that mount a Denial of Service (DoS) attack against the client or provider by either dropping requests or flooding the provider with fake requests.

COWS offers protection against passive attackers by encrypting user ID and authentication token using a shared key and ever-changing IV. Furthermore, while capturing and replaying a fresh request, e.g., by jamming the client, triggers the instantiation of a network, the attacker is unable to calculate the WPA2 network key as he is not in possession of the required token H_i and cannot calculate it from the encrypted request. While the AP could make the network accessible to malicious clients, the establishment of an encrypted tunnel would fail as neither the AP nor the client in question is registered at the provider and all unencrypted traffic would be dropped.

A malicious AP may furthermore drop all requests that it observes, thus denying service to benign clients. This is equivalent to the AP not participating in COWS in the first place, the only damage is the loss of the authentication token(s) contained in the request(s). The client can still gain network access at other APs, the overall system remains functional.

An active attacker may, either as a client or AP, inject fake requests into the system. If such an attacker uses a MAC address that is not registered at the respective provider, the request will get dropped. Using an overheard MAC or guessing a registered MAC address results in the provider trying to decrypt the request. As attackers are neither in possession of the symmetric key nor of the correct IV, decrypting the cipher text will not yield a correct request. However, since authentication tokens, i.e., steps in the hash chain, may get lost, the chain index at the client may be lower than at the provider, leading to the provider using the wrong IV. In fact, this is the expected scenario in COWS, as clients opportunistically probe the environment for roaming networks and never use an authentication token twice. Trying to recover the correct chain index, the provider would thus try to decrypt the request by iterating through the hash chain indices.

The number of calculations is thereby determined by the length of the hash chain. This length thus presents a trade-off between the maximum number of requests, and the amount of work the provider is willing to invest. We evaluate the actual computation cost of this trade-off in Section IV-D and show that the use of symmetric cryptography enables hash chains of substantial length without overloading the provider. Note that the provider only adjusts the chain index in the case of a positive decryption, sending a random request does not deplete the hash chain of the respective user.

IV. Evaluation

We prototypically implement the described client, AP, and provider functionality in order to evaluate the feasibility and applicability of COWS. Our evaluation comprises the connection establishment time between the client and the provider (Section IV-A), the impact of concurrent client connection requests (Section IV-B), the scalability of provider-side functionality with regard to local processing and management of concurrent TLS connections (Section IV-C), and the security functionality (Section IV-D). We implement the client functionality on Galaxy Nexus and Nexus S smartphones running Android 4.1.2. The Galaxy Nexus has a 1.2 GHz CPU and a Broadcom BCM4330 802.11n chipset, while the Nexus S contains a 1 GHz CPU and a Broadcom BCM4329 802.11n chipset.

We realize the AP functionality on a Lenovo S10-3 Ideapad with a 1.5 GHz CPU and an Atheros AR9285 802.11n card. Our Linux-based implementation makes use of the `hostapd` user space daemon to realize and manage an 802.11n AP.

In order to realistically evaluate the communication setting in COWS, we use provider machines in our local network and on remote Amazon Web Services (AWS) *t2.micro*[6] instances. The local provider runs on a Ubuntu Desktop 14.04 machine with an Intel i7 2.93 GHz CPU and 4 GB RAM. Provider instances provided by AWS have a 2.5 GHz vCPU and 1 GB RAM and run Ubuntu Server 14.04. A multi-threaded Python application handles (concurrent) client requests at the provider.

[6] http://aws.amazon.com/ec2/instance-types/

A. Timing

As a measure for the real-life applicability, we first evaluate the time overhead of requesting and instantiating a Wi-Fi network in COWS as observed at the client, AP, and provider. We furthermore measure the time requirement of establishing an `OpenVPN`[7] connection as the most heavyweight variant of establishing a secure tunnel from the client to the provider.

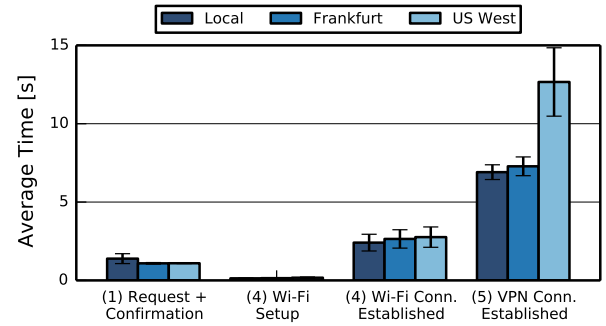
In our evaluation scenario, the client and AP are located at our institute in Aachen, Germany and we vary the geographic placement of the provider to analyze the impact of distance between the AP and provider in the authentication step. We thus first place the provider on a machine in our local network, resulting in an average round-trip time (RTT) of 0.4 ms between AP and provider. Increasing the distance, we realize the provider on an AWS instance located in Frankfurt, Germany, with an average RTT of 5 ms, and on an instance located in Oregon on the US west coast, with an average RTT of 165 ms. The local network scenario thereby serves as a baseline against which we measure the possible realizations of COWS from a provider point of view, namely managing mobile users per-country or globally at a single site.

We construct our evaluation according to the sequence of steps depicted in Figure 2 and conduct 30 complete runs in each setup. Fig. 4(a) – 4(c) then show the average and standard deviation of the time requirement of each step in Figure 2 as observed at the client, the AP, and the provider, respectively. Please note that we only include the VPN connection results (Fig. 4(a), step (5)) for completeness; since we make use of `OpenVPN`, this functionality resides outside of our design.

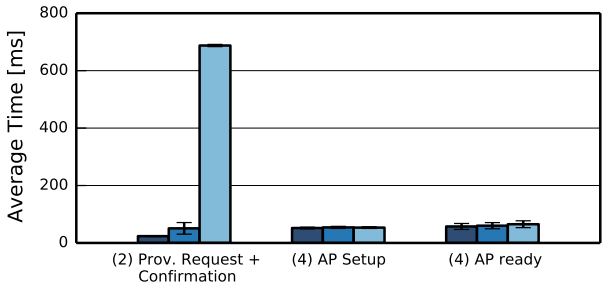
In COWS, a client requests a roaming network and can observe its instantiation, i.e., steps (1) and (4) in Fig. 2, in less than 1.4 s (Fig. 4(a), step (1)) and then connects to the network (Fig. 4(a), step (4)) in less than 2.8 s, regardless of the location of the provider. Indeed, requesting and connecting to an on-demand, secure roaming network in COWS (incl. DHCP) only requires marginally more time than connecting to a permanent Wi-Fi network without authentication at the provider in the same evaluation setup.

The time measured at the client (Fig. 4(a), step (1)) includes the time requirements of all steps at the AP and provider, i.e., steps (2)–(4) in Fig. 2. Fig. 4(b) and 4(c) show the respective time requirements for these steps. As expected, forwarding the client request and waiting for the provider response (Fig. 4(b), step (2)) thereby depends on the placement of the provider instance, i.e., the RTT. Local operations, i.e., authenticating the client and calculating the network configuration (Fig. 4(c), step (3)) as well as instantiating the network at the AP (Fig. 4(b), step (4)) are independent of the provider location. Notably, the client-side time overhead of scanning for and connecting to the network (Fig. 4(a), step (4)) largely masks RTT differences.

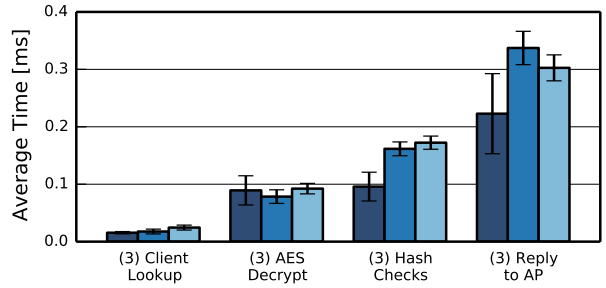
Furthermore, processing the request on the AP (Fig. 4(b)) and at the provider (Fig. 4(c)) does not impose significant computation or time overhead. Note that every request in this evaluation is valid, i.e., the hash chain index used in the request was identical to the one the provider expected and used for



(a) Time overhead of functionality at the client.



(b) Time overhead of functionality at the AP. Legend as in (a).



(c) Time overhead of functionality at the provider. Legend as in (a).

Fig. 4. Time overhead of COWS functionality at the client, AP, and provider for local, in-country, and intercontinental placement of the home provider.

decryption. No time overhead thus incurred due to recovering the correct index. We address this issue in Section IV-D.

Last, establishing a secure tunnel *after* connecting to the network (Fig. 4(a), step (5)) induces a dominating communication and time overhead. We prototypically used `OpenVPN` in certificate mode as a proof-of-concept realization as it is widely accepted and offers a working Android implementation that does not require root access, supporting our goal of a BYOD setup. The results present a worst-case evaluation as we chose the most communication- and computation-heavy key establishment mode as well as a rather bulky VPN implementation. In future work, we will investigate the reduction of this overhead via slimmer VPN implementations in pre-shared key mode, e.g., `tinc`[8], and the applicability of protocol solutions, e.g., *HIP* [14], to BYOD scenarios. Depending on the communication overhead of establishing the secure tunnel, COWS benefits from a lower distance between AP and provider, e.g., via per-country provider instances.

[7] https://code.google.com/p/ics-openvpn/

[8] http://www.tinc-vpn.org/

B. Multiple Concurrent Requests and Network Instantiations at a Single Wi-Fi AP

COWS explicitly supports the concurrent instantiation and operation of multiple Wi-Fi networks in order to preserve the incumbent network of the AP owner as well as to make the AP available to multiple clients in isolated networks. We thus evaluate the feasibility and time characteristics of multiple concurrent requests and network instantiations. In this, the `ath9k` 802.11 driver limits the number of concurrent AP networks on a single physical 802.11 card to four, we hence perform this evaluation with three distinct client devices in addition to an assumed network of the AP owner.

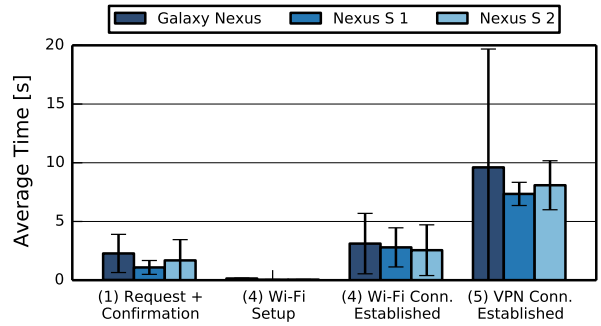
Equivalent to the previous evaluation, we measure the time requirement for distinct functional steps at each client, the AP, and the provider to assess the impact of simultaneous connections. We only consider a local provider as we focus on the interaction between client and AP and we use one Galaxy Nexus and two Nexus S devices as client devices. In order to illustrate the effect per device, we measure the time overhead at each client device and associate the time overhead incurred at the AP and the provider to the respective client device. Fig. 5(a)–5(c) then show the average time overhead and standard deviation of 30 requests and network instantiations at the clients, the AP, and the provider.

We observe that multiple simultaneous requests introduce contention on the wireless link, as the time overhead of requesting and observing the Wi-Fi network instantiation (Fig. 5(a), step (1)) requires between 1.1 s (Nexus S1) and 2.3 s (Galaxy Nexus) compared to 1.4 s for a single client. This is to be expected since the AP multiplexes the available virtualized Wi-Fi networks in the time domain, as can be seen in the successive confirmation of the network instantiation (Fig. 5(a), step (4)). Due to the dominating time overhead of establishing a Wi-Fi association to each instantiated network, the overall time requirement of each client is only slightly higher than in the single-client scenario. In total, concurrently requesting and associating to a network requires on average between 3.9 s (Nexus S1) and 5.5 s (Galaxy Nexus), compared to 3.8 s (Galaxy Nexus) for a single client.

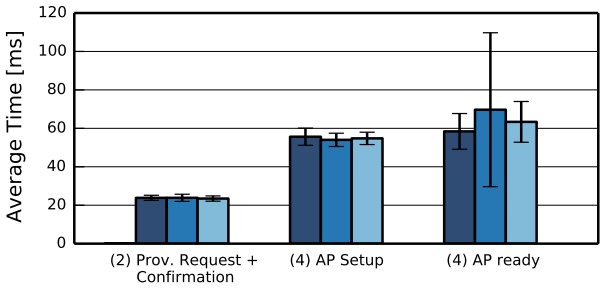
In contrast, we observe only a marginal increase of less than 15 ms in the average overall time overhead of a network request and instantiation at the AP in comparison to the local single-client evaluation (Fig. 5(b)). This is due to the small computational and entirely local AP process once a client request or the provider response has been received.

At the provider, we observe a negligible increase of less than 0.1 ms in processing time (Fig. 5(c)) in the presence of concurrent requests. This is because, even for a consumer-grade desktop machine, COWS only imposes lightweight computational tasks on the provider, such as AES decryption of a single block and computation of a single hash operation. Again, this evaluation assumes a correct request, i.e., the hash chain index used by the client in the encryption is identical to the one used at the provider for decryption. As such, only a single AES and hash operation is performed. We address the impact of asynchronous hash chain indices in the next section.

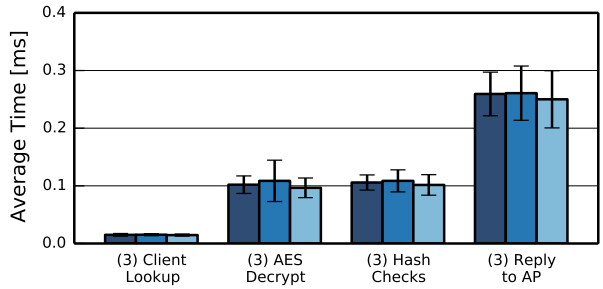
From these results, we derive 1) the feasibility of operating COWS concurrently to the standing Wi-Fi network of the owner,



(a) Time overhead of functionality at each client.



(b) Time overhead for each client at the AP. Legend as in (a).



(c) Time overhead for each client at the provider. Legend as in (a).

Fig. 5. Time overhead of COWS functionality for multiple concurrent COWS instantiations at each client and at the AP and provider for each client.

and 2) the possibility of accommodating multiple roaming clients, with different accounting and incentive configurations, at a single AP. COWS thus mitigates the restrictions of only supporting a single Wi-Fi sharing community as well as permanently operating a community or provider network.

C. Scalability of Provider-side Functionality

In the previous section, we evaluated the feasibility of AP-side support for concurrent roaming clients. Additionally, in a real-world deployment, our design envisions a single provider instance serving numerous APs that each accommodate multiple roaming clients, i.e., their requests for roaming networks. We hence evaluate the provider-side functionality in COWS with regard to the scalability of local processing of authentication requests as well as the time overhead of serving multiple clients at an increasing number of APs.

To this end, we measure the time overhead of both provider functionality aspects, i.e., the purely local processing of requests (cf. Fig. 6(a)) and concurrent handling of multiple TLS connections by APs carrying multiple client requests. We split the evaluation this way in order to clearly distinguish the

impact of either aspect on the actual functionality in COWS, i.e., the processing of authentication requests. For comparison, we evaluate the same steps as in the previous two evaluation sections, i.e., the queuing of the request (including user lookup), the AES-based decryption, and the verification of the hash token (cf. Fig. 4(c) and Fig. 5(c)). We exclude responses to the originating AP as we focus only on the scalability of provider-side processing.

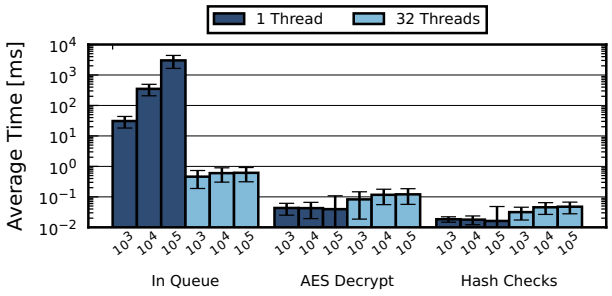
Fig. 6(a) then shows the average and standard deviation of the local processing time for a single authentication request in the presence of an increasing number of queued requests. In this, we approximate a scenario in which a provider instance is asked to process a large number of genuine authentication requests simultaneously. For a naive, single-threaded provider instance (dark blue bars), a request spends the majority of time overhead in the queue and increasing load, i.e., number of requests, induces longer waiting times. In contrast, processing times (AES and hash checks) remain largely constant, independent of the load. In turn, a multi-threaded provider instance reduces the queue times of requests drastically, while an increase in the computation times of AES description and hash checks shows that the comparatively low processing effort of these operations do not amortize the overhead of threading. Still, we conclude that a multi-threaded provider instance running on commodity hardware scales to a large number of concurrent requests.

In addition, we evaluate the communication handling overhead of managing a TLS connection to the actual AP at which the client requests a roaming network by emulating an increasing number of virtual Access Points (vAPs) and clients at these vAPs. Each vAP then instantiates a TLS connection, over which it sends the authentication request(s), to the provider instance, at which the connection request spawns a server thread. We thereby reconstruct the real-life setting of a single provider instance serving a multitude of distinct APs. As in the previous evaluation, Fig. 6(b) shows the average and standard deviation of the time overhead induced by the respective provider-side functionality over an increasing number of vAPs and clients at these vAPs. We find that processing times in either step are largely independent of the number of active APs, i.e., TLS connections, and do not show strong variations over the number of users served per AP.

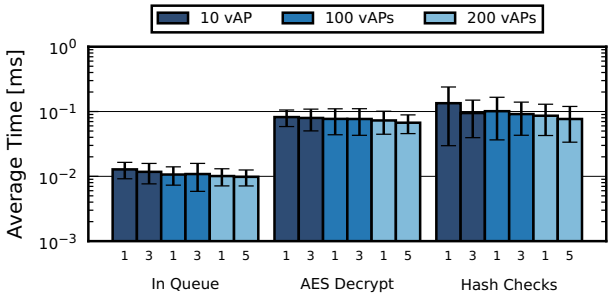
From these complimentary evaluations, we hence derive the scalability of COWS at the provider. Notably, our implementation relies on unoptimized Python modules for both the communication server functionality and the respective cryptography and hash operations. We hence believe that a substantial optimization potential would still be available in case of a dedicated real-world implementation.

D. Security Aspects

As discussed in Section III-F, our design anticipates an asynchronous state of client authentication tokens, i.e., hash chain elements, as roaming requests are purely opportunistic and hash elements are only used once in PREQs. In the resulting case of deviating hash chain indices, the provider tries to recover the correct index by iteratively decrementing the index and attempting to decrypt the request with each resulting IV (cf. Section III-C). Each decryption attempt then requires an AES operation and, if the decrypted user ID U_{ID} matches



(a) Time overhead and impact of threading on provider-side authentication steps for increasing numbers of users (10^3–10^5).



(b) Time overhead of TLS connection handling and authentication over number of users (1–5) simultaneously generating roaming requests at increasing numbers of virtual AP (vAPs).

Fig. 6. Scalability of provider-side functionality. Note the logarithmic scales.

the U_{ID} associated with the MAC address in the request, one hash operation. Authenticating a client request in the face of hash chain indices deviating by m steps thus requires m AES operations and m subsequent `CityHash` operations. For a hash chain of length n, the worst case, i.e., after $n-2$ unsuccessful requests or because of a DoS attack, requires the provider to perform $m = n - 1$ authentication steps on the client request.

In this evaluation, we hence analyze the impact of lost authentication tokens and simultaneously the potential of active attackers (cf. Section III-F) with regard to the induced resource consumption at the provider. We measure the computation time for a hash chain of length of $n = 1000$ and consider the worst case of 999 iterations, a medium case of 500 iterations, and the best case of 1 iteration, i.e., an identical index at both sides. In assuming a moderate hash chain length of $n = 1000$, we aim to provide a tangible notion of the resource consumption. While hash chains may be significantly longer in reality, larger numbers only continue the trends we show in this evaluation. Furthermore, the length of the hash chain corresponds to the number of allowed roaming requests. We deem a number of $n = 1000$ to be a good real-world compromise between the number of attempts and range (or freedom) afforded to the client. Upon exhaustion of n attempts, a sensible protection against misuse would require the client to contact her/his provider via other network means, e.g., from the hotel or coffee shop network. We compare the performances of a local provider to an AWS instance. Fig. 7 depicts the average time overhead induced by increasing disparity of hash chain indices.

Fig. 7 also shows the time requirements of checking the MAC address (“Client Lookup”) and sending a reply (“Reply to AP”) as we measure a complete provider-side operation. On

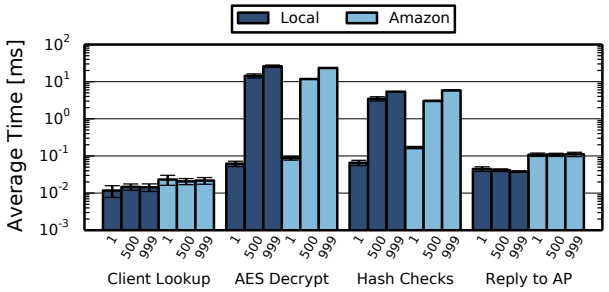


Fig. 7. COWS time overhead at the provider for increasing disparity of hash chain indices between the client and the provider because of unsuccessful client-side network requests or DoS attacks. Note the logarithmic scale.

each provider instance, timings for both operations are constant as these operations do not depend on the hash chain index.

In contrast, a larger gap in the indices induces increasing time requirements to perform the AES decryption and `CityHash` operations. On the local provider instance, for example, the total AES decryption time increases from 0.06 ms for a single, correct decryption to 14.4 ms when 500 iterations are necessary and accordingly to 26 ms for 999 iterations. Correspondingly, the computation time of a single hash check on the local provider instance is 0.065 ms, compared to 3.47 ms for 500 operations and 5.38 ms for 999 operations. The results for the AWS instance are equivalent with only negligible time differences compared to our local provider instance.

We obtain these results with consumer-grade hardware and conclude that attacks on the provider only have a negligible impact per user, i.e., correctly observed or guessed MAC address. The length n of the hash chain thereby provides an adjustable means of controlling the attack impact, as our design requires the actual user to refresh the hash chain at the provider. At the cost of a higher overall time overhead, cryptographic client puzzles [10] in the authentication step between AP and provider would protect providers also from such limited attacks.

V. Conclusion

We propose COWS, a lightweight and flexible approach to providing Wi-Fi roaming networks at private users' APs. By enabling 802.1x-equivalent authentication and relocating the source of outbound Internet traffic to the mobile user's home network provider, COWS removes the security and liability issues from AP providers offering Wi-Fi roaming. In realizing a per-user network configuration and instantiation, we furthermore avoid the overhead and restriction of standing Wi-Fi networks belonging to a single Wi-Fi sharing community. Intuitive and secure accounting techniques at the AP and provider then enable the implementation of incentives for collaboration or micro payment schemes.

Our implementation for commodity Android smartphones and 802.11 APs shows the immediate real-life applicability of COWS. Evaluating our approach in local network settings demonstrates the negligible time overhead introduced by authenticating the user at the provider, while in-country and intercontinental placement of the provider shows the impact of increased RTTs. Last, COWS inherently supports asynchronous states of the authentication credentials at the client and the provider and enables providers to validate incoming requests at negligible computational costs, thereby protecting them against DoS attacks and misuse. Future work targets a real-life deployment of COWS as well as user studies addressing the appropriate means and incentives of collaboration.

Acknowledgments

This work has been funded by the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 "MAKI – Multi-Mechanism-Adaptation for the Future Internet".

References

[1] "IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," *IEEE Std 802.1X-2010*, 2010.

[2] "IEEE 802.11u: Interworking with External Networks," *Amendment to IEEE Std 802.11-2007*, 2011.

[3] R. Chandra, P. Bahl, and P. Bahl, "MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card," in *Joint Conference of the IEEE Computer and Communications Societies*, ser. INFOCOM'04, 2004.

[4] Cisco, "The Future of Hotspots: Making Wi-Fi as Secure and Easy to Use as Cellular," [Online] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/service-provider-wi-fi/white_paper_c11-649337.html.

[5] Fon Wireless, "Global WiFi Network," [Online] https://corp.fon.com/en.

[6] Freifunk Community, "Freifunk Website," [Online] http://freifunk.net.

[7] Global eduroam Governance Committee, "eduroam Initiative," [Online] http://eduroam.org.

[8] B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall, "Improving wireless privacy with an identifier-free link layer protocol," in *International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '08, 2008.

[9] T. Heer, S. Götz, E. Weingartner, and K. Wehrle, "Secure Wi-Fi Sharing at Global Scales," in *International Conference on Telecommunications*, ser. ICT '08, June 2008.

[10] A. Juels and J. G. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks." in *Network and Distributed System Security Symposium*, ser. NDSS '99, 1999.

[11] L. Lamport, "Password Authentication with Insecure Communication," *Commun. ACM*, vol. 24, no. 11, 1981.

[12] D. Leroy, G. Detal, J. Cathalo, M. Manulis, F. Koeune, and O. Bonaventure, "SWISH: Secure Wi-Fi Sharing," *Computer Networks*, vol. 55, no. 7, pp. 1614 – 1630, 2011.

[13] M. Manulis, D. Leroy, F. Koeune, O. Bonaventure, and J.-J. Quisquater, "Authenticated Wireless Roaming via Tunnels: Making Mobile Guests Feel at Home," in *Symposium on Information, Computer, and Communications Security*, ser. ASIACCS '09, 2009.

[14] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host Identity Protocol," RFC 5201 (Experimental), Internet Engineering Task Force, Apr. 2008, updated by RFC 6253.

[15] A. Perrig, "The BiBa One-time Signature and Broadcast Authentication Protocol," in *ACM Conference on Computer and Communications Security*, ser. CCS '01, 2001.

[16] S. Reddy, D. Estrin, M. Hansen, and M. Srivastava, "Examining Micro-payments for Participatory Sensing Data Collections," in *ACM Conference on Ubiquitous Computing*, ser. UbiComp '10, 2010.

[17] N. Sastry, J. Crowcroft, and K. Sollins, "Architecting Citywide Ubiquitous Wi-Fi Access," in *Workshop on Hot Topics in Networks*, ser. Hotnets '07, 2007.

[18] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic Orchestration of WiFi Networks," in *USENIX Annual Technical Conference*, ser. ATC '14, 2014.

[19] H. Wirtz, M. Ceriotti, B. Grap, and K. Wehrle, "Pervasive Content-centric Wireless Networking," in *15th Symposium on A World of Wireless, Mobile and Multimedia Networks*, ser. WoWMoM '14, 2014.