Efficient and Transparent Wi-Fi Offloading for HTTP(S) POSTs

Kévin Huguenin, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub

Efficient and Transparent Wi-Fi Offloading for HTTP(S) POSTs

Kévin Huguenin, *Member, IEEE,* Erwan Le Merrer, Nicolas Le Scouarnec, *Member, IEEE,* Gilles Straub, *Member, IEEE*

Abstract—With the emergence of online platforms for (social) sharing, collaboration and backing up, mobile users generate ever-increasing amounts of digital data, such as documents, photos and videos, which they upload while on the go. Cellular Internet connectivity (e.g., 3G/4G) enables mobile users to upload their data but drains the battery of their devices and overloads mobile service providers. Wi-Fi data offloading overcomes the aforementioned issues for delay-tolerant data. However, it comes at the cost of constrained mobility for users, as they are required to stay within a given area while the data is uploaded. The up-link of the broadband connection of the access point often constitutes a bottleneck and incurs waiting times of up to tens of minutes. In this paper, we advocate the exploitation of the storage capabilities of common devices located on the Wi-Fi access point's LAN, typically residential gateways, NAS units or set-top boxes, to decrease the waiting time. We propose HOOP, a system for offloading upload tasks onto such devices. HOOP operates seamlessly on HTTP(S) POST, which makes it highly generic and widely applicable; it also requires limited changes on the gateways and on the web servers and none to existing protocols or browsers. HOOP is secure and, in a typical setting, reduces the waiting time by up to a factor of 46. We analyze the security of HOOP and evaluate its performance by correlating mobility traces of users with the position of the Wi-Fi access points of a leading community network (i.e., FON) that relies on major national ISPs. We show that, in practice, HOOP drastically decreases the delay between the time the photo is taken and the time it is uploaded, compared to regular Wi-Fi data offloading. We also demonstrate the practicality of HOOP by implementing it on a wireless router.

Index Terms—Wi-Fi offloading; Web technologies; Delay-tolerant networking;

1 INTRODUCTION

With the advent of mobile devices, both in terms of their functionalities and of their connectivity, users generate ever-increasing amounts of digital data while on the go. For instance, they take photos and videos with their smartphones and produce or edit possibly large documents on their tablets and laptops. The data is then uploaded (often automatically) to online services, typically through web applications, native apps or system services. They do so for various purposes ranging from social sharing (e.g., sharing photos on Facebook or Flickr, or videos on YouTube) to increasing availability and backup (e.g., uploading all sorts of documents to a cloud storage service such as Dropbox or iCloud). Another example is the case of medical data for the continuous biomedical monitoring of non-hospitalized patients [2] and sportsmen. In many cases, the upload to the online service is performed through HTTP(S) POST operations (e.g., using a browser, or with applications relying on HTTP-based APIs).

- *Manuscript received: June 1, 2015;*
- *This article is a revised and extended version of a paper that appears in the Proceedings of the IEEE International Conference on Web Services (ICWS'14) Huguenin* et al. *[1].*
- *K. Huguenin is with LAAS-CNRS, Toulouse, France. Parts of this work have been done while the author was with EPFL, Lausanne, Switzerland (e-mail: kevin.huguenin@laas.fr).*
- *E. Le Merrer, N. Le Scouarnec and G. Straub are with Technicolor, Rennes, France (e-mail: {erwan.lemerrer, nicolas.lescouarnec, gilles.straub}@technicolor.com)*

To upload the data they produce while on the go, users rely on the connectivity of their mobile devices, namely 3G/4G and Wi-Fi capabilities. To do so, they are offered essentially two options, both with noticeable drawbacks. Cellular Internet connectivity enables mobile users to upload their data from virtually anywhere (and while moving) but drains the battery of their devices [3], [4] and overloads mobile Internet service providers, that, in response, impose data caps (and either, block the traffic, reduce the bandwidth or over-charge the traffic beyond the limit) much to the detriment of the users. Data offloading at Wi-Fi access points [5] (or 3G DropZones as advocated in [6]), be they public (e.g., AT&T WiFi [7]), business (e.g., Starbucks) or community (e.g., FON [8]) hotspots or personal or corporate access points, overcomes the aforementioned issues for delay-tolerant data. This, however, comes at the cost of constrained mobility and/or significant delays for users. Indeed, the users are required to stay in the vicinity of the access point while the data is being uploaded. In the case of personal or corporate access points, the data is uploaded only when the user reaches the corresponding location (i.e., home and work place respectively). A determining factor of the upload time is the up-link speed of the Wi-Fi access point's Internet connection (typically 1 Mbps [9], [10]) which often constitutes a bottleneck compared to the Wi-Fi connection (typically 50 Mbps[1]). The waiting time can reach ten minutes for 20 high-definition photos uploaded on a 1 Mbps

1. A study [3] from 2010 shows that an average Wi-Fi data rate of 1.2 Mbps can be achieved in urban areas, which means that the data rate is often higher than that of the broadband connection; this number should be even higher with the next-generation of Wi-Fi technologies.

Internet link.

In this paper, we propose to leverage on the processing and storage capabilities of common devices located on the Wi-Fi access point's local area network (LAN) to implement a sort of store-and-forward HTTP(S) proxy, thus decreasing the waiting time to the point where the Wi-Fi connection of the access point becomes the bottleneck. First-class candidates for implementing such a scheme include always-on residential gateways [11], [12], routers, network-attached storage (NAS) units, and set-top boxes. One major design challenge, which is paramount for a wide adoption, is to provide a solution that is completely transparent for the users and that requires as-small-as-possible changes to existing software and protocols. We propose HOOP, a system for offloading upload tasks onto devices, such as gateways, in a secure and seamless way. Essentially, when a user reaches an HTML upload form on a HOOP-enabled website, her browser looks for a device running HOOP on the local network (say a gateway) to offload the uploading task. If such a device is found, the user's browser, instead of directly uploading the file to the online service, encrypts and uploads the file to the gateway, together with an authentication token, at a speed determined by the Wi-Fi connection of the access point. At this point, the user can disconnect from the access point and potentially move and switch off her device, while the file is being asynchronously uploaded by the gateway.

HOOP operates seamlessly–from the standpoint of the user–on HTTP(S) POST and relies only on existing web standards (e.g., JavaScript and AJAX) and network protocols (e.g., HTTP(S) and DNS), thus making it highly generic and widely applicable. More specifically, it can be used (through its open API) by any application that relies on HTTP(S) POST to upload data (e.g., HTML forms, Flash/Java uploaders, native apps). HOOP requires only limited changes on the gateways and on the web server and none at the client side (i.e., at the mobile device's operating system and browser). HOOP is secure and it significantly reduces the users' waiting time. Unlike previous works that maintain end-to-end connections between mobile users and online servers despite network disruption (e.g., [13] at the transport layer), HOOP makes the most of connectivity windows by fully exploiting local network connections (at the application layer).

We analyze the security of HOOP and show that HOOP guarantees the confidentiality and the integrity of the uploaded data, with respect to various attackers, including the gateway and eavesdroppers. In addition, we show that HOOP does not create new opportunities for an attacker to disrupt the upload or attack the online service. We evaluate the performance of HOOP in two scenarios. We consider a static user uploading data at a HOOP-enabled Wi-Fi access point and show experimentally that the waiting time is reduced by a factor of 46, compared to regular Wi-Fi data offloading. We consider a mobile user who uploads data while moving, through a network of hotspots, and we show through trace-driven simulations (i.e., by correlating mobility traces with the positions of the Wi-Fi hotspots from a leader ISP), that HOOP increases the upload capacity by a factor of 42. We demonstrate the practicality of HOOP by implementing it on a high-end set-top box and on a wireless router (for the code running on the gateway) and on various websites, including a minimal HTML form-based uploader, the Flash and HTML 5 uploaders of the Gallery [14] web photo organizer and the Java uploader of the ResourceSpace [15] web data management service. Finally, we discuss potential business models for HOOP and show that the involved parties, in particular the users, the online service providers, the Internet service providers (ISP), and the access point operators, all have incentives to adopt HOOP.

The rest of the paper is organized as follows. In Section 2, we survey the related work. In Section 3, we introduce the system model and we give some background about HTTP(S) uploads. In Section 4, we present and describe HOOP. We analyze the security of HOOP and we report on its performance evaluation in Section 5. Finally, we discuss the incentives and the economics behind HOOP in Section 6 and we conclude the paper in Section 7.

2 Related Work

The problem of mobile data upload has received a great deal of attention from the research community over the last few years. More specifically, Balasubramanian *et al.* first proposed [16] to augment the 3G capacity in mobile scenarios by exploiting Wi-Fi access points. They implement a software solution for delaying data exchanges and fast-switching between 3G and Wi-Fi, and they assess the potential of their approach. In [3], Lee *et al.* perform a large-scale experimental performance evaluation of data offloading over Wi-Fi that demonstrates the benefits of this approach, both in terms of the amount of data offloaded from 3G and of battery power. In [6], Trestian *et al.* study the data generation and upload patterns of mobile users and advocate the use of cells with disproportionally upgraded bandwidth, called Drop Zones, for offloading the content generated by mobile users while on the go. In addition, they tackle the problem of the optimal placement and of the dimensioning of the Drop Zones. In [13], Go *et al.* propose a practical implementation of mobile Wi-Fi offloading where they use a transmission protocol that enables mobile devices to maintain an end-to-end connection with a server despite network disruption. As the proposed solution operates at the transport layer, it is transparent to the applications, and thus it is quite generic. However, it requires modifications of the network interface at the client side. Beyond academic contributions, some companies (e.g., GoNet[2]) have successfully designed and deployed Wi-Fi networks for 3G offloading. Finally, some pieces of work, such as [17], complement data offloading with device-to-device communications. In [18], Han *et al.* survey existing offloading techniques. In all these works, it is assumed that the data is offloaded directly over Wi-Fi, at the speed of the access point's connection to the Internet, which constitutes a bottleneck. Although HOOP relies on the same approach, i.e., offloading traffic at Wi-Fi access points, it goes beyond by exploiting the storage capacity at the access points to fully take advantage of the Wi-Fi connectivity for delay-tolerant uploads. In [19], the authors study the trade-off between data downloading delays and user satisfaction in

2. https://www.winncom.com/en/success-stories/73 (Last visited Jan. 2015)

the case of 3G offloading; they show that, by predicting the users' offloading potential and by using appropriate incentives, data downloads can be efficiently delayed without sacrificing the users' satisfaction. Finally, in [20], Kim *et al.* propose an analytical framework to study the performance of mobile data offloading. One of their main findings is that existing Wi-Fi infrastructures deployed in metropolitan areas are sufficient to offload, within reasonable delays, most of the mobile user traffic.

Several works, e.g., [11], [12], advocate the use of the storage capacity of gateways–and other always-on devices with storage capacity–to offload data transfer from user devices. Technical solutions have been proposed and implemented on gateways, set-top boxes and networked area storage units. For instance, many such devices offer HTTP download services and run BitTorrent clients (e.g., Synology NAS). Closer to our work, the Fonera [21] enables users to asynchronously upload files to a number of online services (including YouTube, flickr, and Facebook) by simply copying them over, e.g., ftp, to specific folders. Unlike HOOP, such solutions have major drawbacks that prevent wide adoption in the public domain: The device is trusted with the users' credentials for these online services; the device is given the users' data, in clear, which it can alter; the solution is dependent on the online service (as it relies on their proprietary APIs) and it requires explicit user interactions, as opposed to HOOP that is generic and seamless.

3 System Model and Background

We consider a system composed of the following entities: (1) a local area network (LAN) connected to the Internet by an ISP, (2) a mobile device, controlled by the user, and (3) an online web service, as described in Figure 1. The local network is composed of a router (typically a gateway) that connects the different devices to the Internet, a device with computational and storage capabilities to run HOOP (typically a set-top box or the gateway), and an access point that enables users with wireless-equipped devices to connect to the local network. The user connects to the Internet (through the local network) with her wireless-equipped mobile device and makes use of web services through her installed browser and native apps. We consider an online web service that enables users to upload data through HTTPS[3] POST operations, from an HTML form (potentially with AJAX), a Flash uploader, or a native app. Throughout the paper, we focus on the case of HTML forms; the other cases are in fact simpler, as the service provider controls the application, whereas for HTML forms the service provider does not control the browser.

In a typical HTML scenario (without HOOP), a user connects to the web service and requests the upload page, through HTTP(S), from the browser installed on her mobile device. The web service returns a HTML webpage including a form (e.g., see Figure 2) that contains at least one form element to select the data (typically some files, e.g., photos) to be uploaded, some extra information (e.g., a caption), an authentication token, and the target page (https://www.service.com/post.php) to which the data will be posted. The user then selects the file(s) to upload and submits the form by clicking on the corresponding button, and the data is posted to the target page (typically a php page). The user must stay connected until the data is uploaded. Once the data is uploaded, the target page checks that the user is authenticated (e.g., based on an authentication token stored in a hidden field of the HTML form), and it retrieves and processes the data (e.g, adds the photos to the user's profile in the service's database); then a message confirming the upload is shown to the user. The whole process is depicted in Figure 3.

3. We focus on HTTPS throughout the paper: The case of unsecured HTTP can be solved by implementing a proxy at the IP level; this is not possible for HTTPS, as TLS layer protections rely on session keys that are periodically renewed.

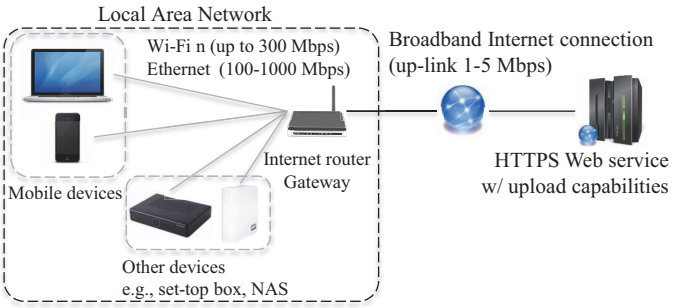


Fig. 1. Setup of Hoop.

```
<form id="upload form" action="post.php" method="post">
  <input type="file" name="data">
  <input type="text" name="caption">
  <input type="hidden" value="..." name="token">
  <input type="button" value="Upload"
    id="upload_button" onclick="upload_form.submit();">
</form>
```

Fig. 2. HTML upload form.

Consider the typical scenario of a native mobile application, written in Java, for the Android platform. The application communicates with the web service through HTTP(S) in order to use the same interface as for the website: The application collects data from the local file system, as well as from various elements of the graphical user interface (GUI); the application embeds the data in an HTTP(S) request that it POSTs to the target URL (e.g., https://www.service.com/post.php) by using a dedicated library (e.g., `org.apache.http.client`).

4 Hoop: Design and Implementation

In this section, we describe HOOP, a system for offloading upload tasks onto devices located on the same LAN as the user's mobile device in a *store-and-forward* fashion. HOOP involves three different entities, as described in the system model: a software component on the device running HOOP (say a gateway), the application running on the user's mobile device, and the web service. We describe the functioning of HOOP by listing and explaining the different operations performed by each of the three aforementioned entities. HOOP operates as follows: The mobile device (be it a script executed by the browser or a native app) searches

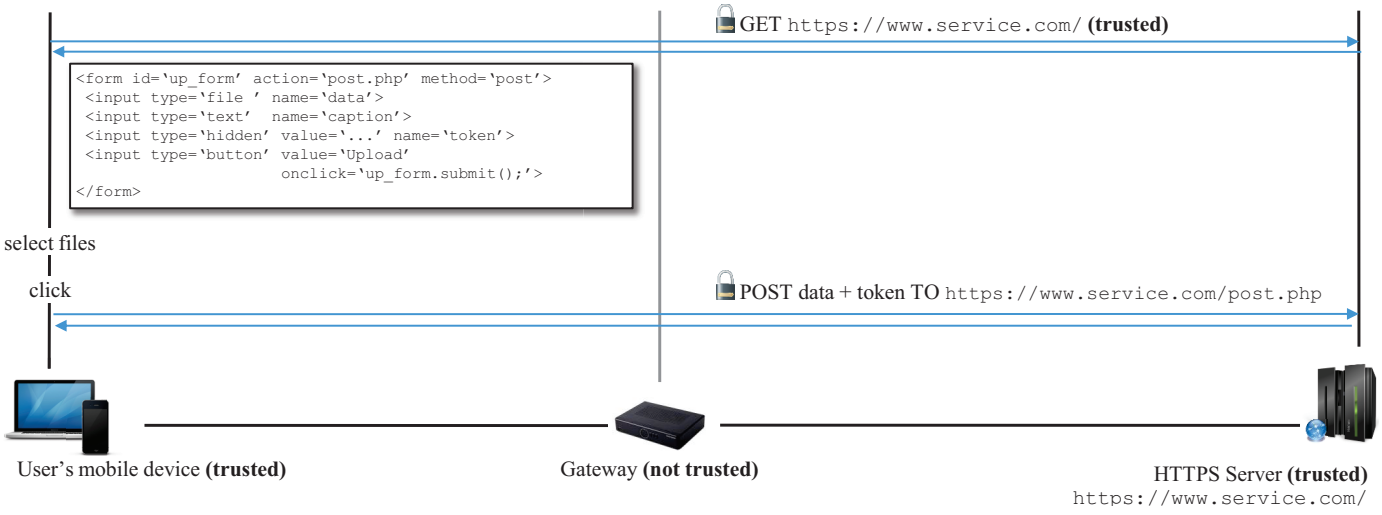


Fig. 3. System overview without Hoop.

for a device running HOOP on the local network and, if any such device is found, it processes (i.e., re-formats and encrypts) the data to be sent and directs the upload to this device (instead of to the web service). The device running HOOP stores the data received from the mobile device and asynchronously uploads it to the web service that handles the data as for a regular upload. We first describe the general functioning of HOOP, depicted in Figure 5; then we describe the specifics of its implementation on the mobile device as a web application running in a browser and as a native app.

4.1 System Description

The HOOP component running at the gateway essentially consists of a daemon acting as both an HTTP server bound to a fixed pre-defined port and an HTTP client. At the startup, the HOOP component registers the hostname `hoop.local` on the local network through the DHCP protocol [22]. Note that as gateways often host a DHCP/DNS server, the hostname registration can be done locally, and the gateway can make sure that no other device on the LAN registers as `hoop.local`. The HTTP server implemented by the HOOP component can be accessed at `http://hoop.local/` and offers two services: `test` (accessible at `http://hoop.local/test`[4]) that allows devices on the LAN to detect its presence and test its availability, and `offload` that implements the store-and-forward operation, as we describe below. In order to allow scripts originating from HOOP-compatible web services to connect to the gateway's HTTP services, the latter implements a cross-origin resource sharing (CORS [23]) policy by adding the rule `Access-Control-Allow-Origin:*` to the HTTP header (or a similar rule in the `crossdomain.xml` file for Flash applications).

The HOOP-compatible web service hosts, in addition to the traditional page `post.php`, a page `post_hoop.php` that handles the uploads that are offloaded to and forwarded by the HOOP component running at the gateway. Although these two pages differ in the way they retrieve and preprocess the uploaded data, they process this data in the same way by relying on the same `php` function. Thus, the modifications required at the web service are limited. The web service has a secret key K_{ws} for symmetric authenticated cryptography. Upon login, the mobile device obtains an authentication token T from the web service. When an upload operation is initiated, the mobile device obtains a fresh random secret key K for symmetric authenticated encryption, together with a version of the key encrypted with the secret key of the web service, i.e., $E_{K_{\text{ws}}}(K)$ where E denotes the encryption operation (typically AES in OCB, CCM or EAX mode), from the web service.

The mobile device (i.e., a web-application running in the browser, a Flash application, or a native app) searches for a device running HOOP on the local network by sending an HTTP request to `http://hoop.local/test`. If the request returns successfully (i.e., the host `hoop.local` is resolved and found, and the request returns the HTTP success code 200–the gateway returns the HTTP service unavailable 503 code if its upload buffer is full), the mobile device sets the target URL to http://hoop.local/offload, so as to offload the upload to the device running HOOP, sets a GET parameter to the target URL of the web service (i.e., http://www.service.com/post_hoop.php), and generates the following post data: $Z = E_{K_{\text{ws}}}(K) \,||\, E_K(T \,||\, D)$, where E denotes the encryption operation (e.g., AES in OCB, CCM or EAX mode), T is the authentication token provided by the web service, and D is the data the user wants to upload (e.g., a photo and a caption). Note that as the content sent by the mobile device to the device running HOOP and by the device running HOOP to the web service is encrypted, there is no need to use TLS encryption (i.e., HTTP suffices); this alleviates the need for certificate management at and for the gateway. Finally, the mobile device posts the data Z to the offload URL http://hoop.local/offload. As the mobile device and the device running HOOP are on the same local network, the speed at which the data is transferred is determined by the technology used on the LAN (typically 100/1000 Mbps Ethernet or Wi-Fi g/n/ac) but is independent from the speed of the Internet connection.

When the gateway receives a request to its `offload` service, it first extracts the target URL of the web ser-

4. Note that we omit the port for the sake of clarity.

vice from the GET parameters (i.e., http://www.service.com/post_hoop.php). Then it extracts the POST data (i.e., $E_{K_{\text{ws}}}(K) \mid\mid E_K(T \mid\mid D)$) and passes this data to its HTTP client that (re-)posts the data to the target URL of the web service. When the device running HOOP has limited processing and memory capabilities (e.g., a wireless router as described in Section 5), the HOOP component is implemented as a stand-alone native executable file that provides basic HTTP server and client features for receiving and (re-)posting offloaded data. When running on a more powerful device (e.g., a set-top box, a NAS, a dedicated server), the HOOP component can also be integrated into an existing HTTP server, e.g., as a module in the Apache HTTP server.

The post_hoop.php page hosted by the web service parses the POST data. It first obtains the symmetric key K by decrypting $E_{K_{\text{ws}}}(K)$ with its secret key K_{ws}. Then, it decrypts the data and the authentication token by using the key K and passes them to the script used to handle regular uploads (i.e., those that do not make use of HOOP). Note that when decrypting the different parts of the POST data, the php script checks the integrity of the data and drops the request if it fails the integrity test. Figure 4 gives a simplified version[5] of a typical post_hoop.php page (note that any language, such as Java or Python could be used for implementing the post_hoop). Note that the web service relaxes its CORS policy for the post_hoop.php page by accepting any origin for this page.

```
function hoopReceive(){
    $fd = fopen("php://input", "r")
    $k = hoopReadAndDecipherSessionKey($kws,$fd)
    $data = hoopDecipher($k,$fd)
    list ($_POST, $_FILE) = hoopMultipartDecode($data)
}
hoopReceive();
include("post.php");
```

Fig. 4. Hoop upload php script.

4.2 Implementation

The implementation of HOOP as a native app on the mobile device is straightforward: Preparing and sending HTTP requests is achieved by using a dedicated library such as org.apache.http.client for Java on Android; the encryption is performed by using a dedicated library as well, e.g., javax.crypto. The authentication token and the encryption key are obtained from web service through HTTPS (e.g., returned in the XML or JSON format).

However, the implementation of HOOP as a web application, is challenging as the web application runs within the browser over which the developer has no control. The code executed by the browser is provided by the web service as a JavaScript. The script contains, in two variables, the symmetric key K and its encrypted version $E_{K_{\text{ws}}}(K)$. When the JavaScript is loaded, it searches for a device running HOOP by making an asynchronous XMLHttpRequest to http://hoop.local/test. If the request returns successfully, the JavaScript modifies the upload form in order to offload the upload to the device running HOOP. This is achieved by setting the target URL of the HTML form (i.e., its action attribute) to the empty string, and by setting instead, through the onsubmit attribute of the submit button, a JavaScript function to be executed when the form is submitted (see Figure 6). This function accesses the data from the files through the HTML 5 File API, performs the encryption by using a dedicated JavaScript library[6] (e.g., crypto-js [25]), and it sends the encrypted POST data to the device running HOOP at http://hoop.local/offload by making an XMLHttpRequest with the GET parameter set to the target URL of the web service (see Figure 7). When the upload terminates, the user is redirected to a dedicated web page by changing the location header.

```
button = document.getElementById("upload_button");

function hoopSetup(){
  // search for a device running Hoop
  req = new XMLHttpRequest();
  req.open("GET", "http://hoop.local/test", true);
  req.onreadystatechange=function(){
    if(req.readyState==4 && req.status==200){
      // switch the upload method to Hoop
      button.onClick = "hoopSend();";
    }
  }
  req.send();
}
```

Fig. 6. Hoop JavaScript function for activating Hoop, if a device running Hoop is found on the LAN.

```
k = "..."  // symmetric key (in clear)
ek = "..." // symmetric key (encrypted)
dest = "http://www.service.com/post_hoop.php"
form = document.getElementById("upload_form");

function hoopSend(){
  data   = hoopMultipartEncode(form) // extract the data
  cipher = hoopCipher(data, k)       // encrypt the data
  req = new XMLHttpRequest();
  req.open("POST", "http://hoop.local/offload?dest=" +
       urlencode(dest), false);
  req.send(ek + cipher);
  window.location = "...";
}
```

Fig. 7. Hoop JavaScript function for preparing and offloading the data to a device running Hoop.

4.3 Additional Features

In addition to its core offload functionality, HOOP offers side features that enable users to monitor their offloaded uploads, at the gateway and at the web service. Upon a successful offload onto the gateway, the user is provided

5. For the sake of clarity, the snippets do not exactly match the actual implementation. In particular, we omit error-handling code as well as diverse optimizations.

6. Note that the W3C is currently working on the specification and the implementation of a JavaScript cryptography API named WebCryptoAPI [24].

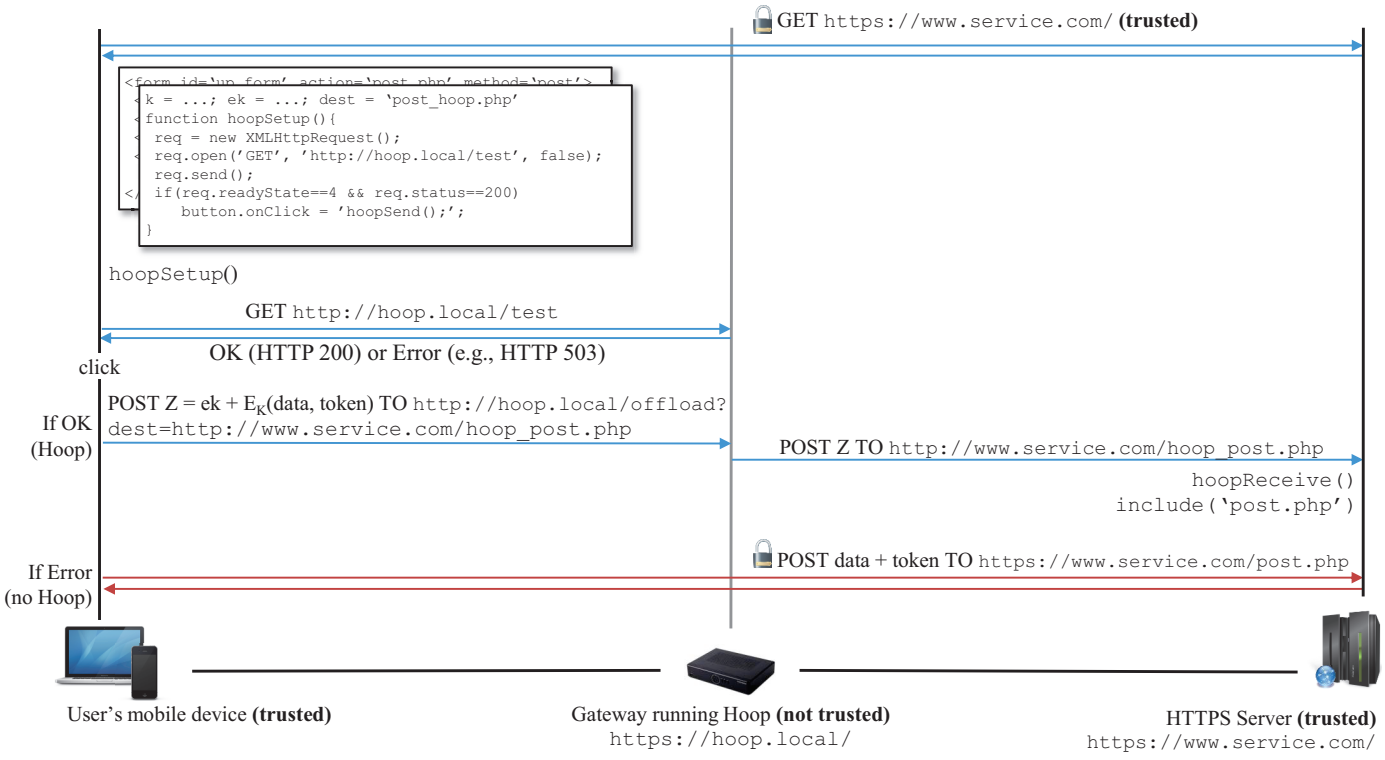


Fig. 5. System overview with Hoop

with a link of the form hoop.local/monitor?ID=..., where ID is a random identifier assigned to the offload, to monitor (i.e., see the current upload status) the (re-)posting of the uploaded data. The operator of the local network can make the monitoring service accessible from outside the LAN; in this case, the local hostname must be replaced by a fully qualified hostname. The monitoring service can be implemented at the web service as well: When an upload is offloaded to a device running HOOP, the web service is notified by the user's mobile device through an HTTPS request that includes the key K and the meta-data (e.g., the caption and the names of the files). The user can subsequently monitor, through her account on the web service, the list of her offloaded uploads and monitor/control (i.e., pause, resume, stop) them.

Finally, the user can specify, in her account settings on the web service, certain policies for deciding whether to use HOOP for offloading her uploads. For instance, the user can decide to never use HOOP, to always use HOOP, or to be asked (through e.g., a check-box) whether to use HOOP when a device running HOOP is found on the local network. More complicated policies can be used so as to, for example, make the decision based on the sizes and types of the files to be uploaded.

4.4 Extension to the *Download* Case

HOOP focuses on offloading *upload* tasks. As download traffic is usually significantly higher than upload traffic, adapting HOOP to handle download tasks is a natural and interesting extension. This, however, is not trivial, especially in a transparent fashion over HTTPS. In particular, it would raise the following difficult question: How do we predict which access points the user is going to connect to (in order to pre-fetch the downloaded content at the access point)? It would also require implementing some sort of "mailbox pick-up" service in which a user has the content provider push the content to an access point in such a way that when she reaches the range of the access point it can download the (pre-fetched) content by using the local link (thus, at a high data rate). For instance, the service provider can push the requested file, or some chunks of it, to the access point with a specific ID (e.g., 4dsf4df2124sdfds2f4) and then redirect the user to the URL http://hoop.local/4dsf4df2124sdfds2f4. This require the users and/or the content provider to have full knowledge of the list (and positions) of available access points. Finally, because for broadband connection (unlike Wi-Fi) the down-link speeds (typically 10 Mbps) are usually much higher than the up-link speeds, such a protocol would be less beneficial for downloads than for uploads. We leave such an extension to future work.

5 EVALUATION

We evaluate HOOP with respect to its security (e.g., the confidentiality and the integrity of the user's data), its efficiency (e.g., technical feasibility of HOOP on various devices), and its efficacy (e.g., in terms of its offload potential). We do not discuss the security of the features mentioned in Section 4.3 as they do not constitute the core of HOOP.

5.1 Security

We look at the security of HOOP by considering different adversarial scenarios. As HOOP is designed for a wide deployment in the public domain, neither the gateway nor the

user is trusted, thus they constitute potential adversaries. In addition, we consider (possibly active) adversaries such as a jammer, an eavesdropper, or another user connected to the LAN. We structure our security analysis with respect to the three entities involved in HOOP.

Confidentiality and integrity of the users' data. The confidentiality and the integrity of the data (users' data, as well as the key K, and the JavaScript or HTML codes) exchanged directly between the user's mobile device and the web service is guaranteed by the TLS encryption of the HTTPS connection: Neither the router nor an eavesdropper (should it snoop on the LAN or on the Internet) can read or stealthily (i.e., without being detected) tamper with this data. The confidentiality and the integrity of the data exchanged between the user's mobile device and the web service, through the HOOP on the gateway (over unsecured HTTP), is guaranteed by the application-layer encryption (i.e., an authenticated encryption (AEAD) such as AES in OCB, CCM or EAX mode), the encryption key K being known only to the user and to the web service as it is exchanged over HTTPS. The authenticated encryption implements integrity checks that prevent an attacker from tampering with the data. Therefore, an adversary, such as a malicious gateway, cannot tamper with the data $(D||T)$ in the post data as it does not know K.

Security of the gateway. An adversary can perform a denial-of-service (DoS) attack against the gateway by issuing a large number of offloading requests. In general, it is difficult to defend against DoS attacks, however they are not specific to the use of HOOP; this means that traditional protection mechanisms can be used and that HOOP does not create new opportunities to attack the gateway.

Security of the web service. Relaxing the CORS policy for the `hoop_post.php` page exposes the web service to cross-site scripting attacks (XSS), e.g., a third-party website stealthily posting data to this page by relying on an existing authentication cookie in the user's browser. However, as the `hoop_post.php` page authenticates users based on the token T encrypted with the secret key K instead of using cookies, such XSS attacks cannot succeed. Finally, an adversary can carry out a DoS attack against the web service, through HOOP, by offloading a large number of requests on a gateway that runs HOOP. Such an attack, however, does not give more power to the adversary as it is similar to making the requests directly from the LAN.

5.2 Efficiency

We evaluate the efficiency of the HOOP components running on the mobile device and on the gateway based on a real implementation on various platforms.

Mobile device: encryption. Encryption (along with file access and communication) constitutes a potential bottleneck when HOOP runs a mobile device. We considered both OpenSSL and CryptoJS (v3.0) libraries for symmetric AES-256 encryption, as they constitute natural candidates for an implementation of HOOP as a native app and as a web application respectively.[7] We conducted our experiments on different devices and settings: a PC laptop (Core i5-2520M) running Firefox 34 on Windows 7, a MacBookAir Mid-2011 (Core i5-2467M) running Firefox 34 on OSX 10.10, an iPhone 5S running Safari for iOS 8.1.2, an iPhone 4 running Safari for iOS 7.1.2, a Nexus 5 running Firefox 34 on Android 4.4.4, and a Galaxy S3 running Chrome 30 for Android 4.2. The results, summarized in Table 1, show that a native app can easily saturate a broadband connection and, in most cases, saturate a LAN connection. The results are not as good for JavaScript. However, they indicate that laptops can saturate a local area network connection and modern smartphones can saturate a Wi-Fi connection. Note that service providers are most likely to provide native apps on mobile devices (e.g., Dropbox, iCloud, YouTube uploader), and the performance of JavaScript is good on laptops, which are likely to use the web version of HOOP within the browsers. Furthermore many factors foresee significant improvements for JavaScript encryption: The processing power of smartphones increases rapidly (the iPhone 4 and the Galaxy S3 have been released in 2010 and 2012 respectively and their successors have significantly improved processing capabilities); developers actively work on improving the JavaScript performance of browsers in general; and the WebCryptoAPI [24] specification of W3C could lead to the use of native code for JavaScript encryption for web applications.

TABLE 1
Benchmark of symmetric cryptographic libraries on various devices (AES-256 encryption).

Device (Browser)	**Throughput (Mbps)**	
	OpenSSL	JavaScript
Laptop i5-2520M	640	138
MacBookAir i5-2467M (Mid-2011)	578	121
iPhone 5S	NC	52.5
Nexus 5	NC	32.9
iPhone 4	96[8]	5.45
Galaxy S3	416[9]	6.50

Gateway component: offload and upload. We implemented the gateway component in charge of receiving and (re-)posting offloaded data on two different devices: a wireless router running OpenWRT and a set-top box (see Table 2 for the detailed configurations). The set-top box has hardware similar to a typical NAS. We implemented the HOOP component in C and compiled it to a stand-alone native executable linked against the `libevhtp` 1.2.6 (static link) and `libevent` 2.0.5 (dynamic link) libraries. The implementation has ∼350 source lines of code (excluding the libraries) that compile to a binary of ∼60 KB (excluding a dynamic library of ∼250 KB) on both platforms. The wireless router embeds a Wi-Fi 802.11n access point and the set-top-box is connected to the router/AP through a 100 Mbps Ethernet network interface.

We conducted our experiments with HOOP running either on the router or on the set-top box, and with our

7. Note that other libraries for JavaScript encryption exist, e.g., [26].

8. Obtained from http://hmijailblog.blogspot.fr/2011/02/openssl-speed-on-iphone-4.html (Last visited Oct. 2013)

9. Obtained from https://jve.linuxwall.info/ressources/taf/aesmeasurements.txt (Last visited Oct. 2013)

TABLE 2
Technical specifications of the devices used for the evaluation. These two devices are representative of ISP-provided equipments: a modem with limited capabilities and a high-end set-top box.

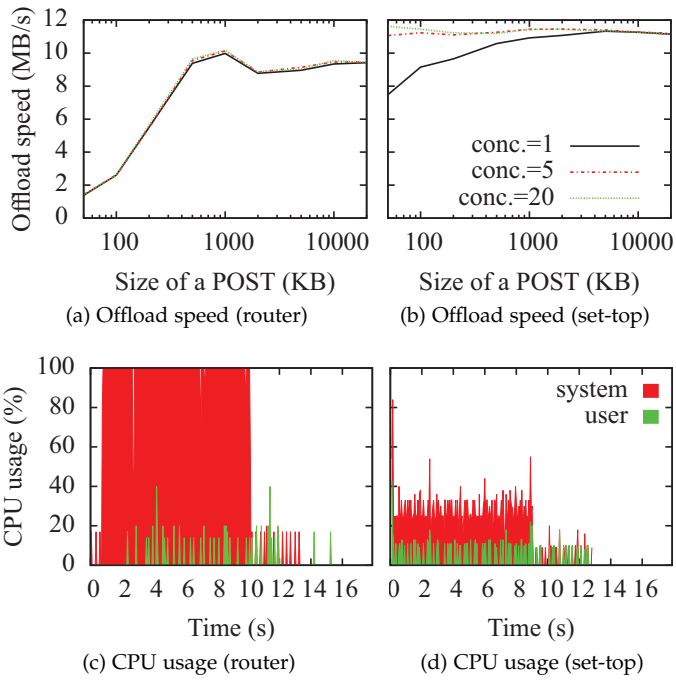
Dev.	Arch.	Proc.	RAM	HDD
Router	MIPS Atheros	AR7241@400 Mhz	32 MB	USB 320 GB
Set-top	x86 Intel	Atom@1.66 Ghz	1 GB	SATA 250 GB

Laptop Core i5-2520M connected to the local network either over 100 Mbps Ethernet or over Wi-Fi 802.11n (the actual negotiated link speed was 78 Mbps). Our experiments with a wired connection between the mobile device and the gateway enable us to assess the performance of the HOOP component running at the gateway (as a wireless connection could have constitute a bottleneck), whereas our experiments with a wireless connection enable us to assess the global performance of HOOP as a whole. We used ApacheBench on our laptop to execute HTTP POST requests and collect statistics.

We evaluate the performance of the HOOP component running on the gateway, in a wired setting, along the following metrics: (1) the offload speed (as a function of the size of the POST, for different concurrency levels[10]), and (2) the CPU usage (and the breakdown between system and user time). The results are presented in Figure 8. It can be observed on Figures 8a and 8b that for small POSTs (e.g., 50-200 KB) offloaded onto the set-top box, sending concurrent requests improves the offload speed as the requests are processed concurrently at the gateway, thus amortizing the connection delays. For large POSTs (i.e., > 1 MB), which constitute the main use-case of HOOP, both the router and the set-top box saturate the LAN connection (i.e., Ethernet at 100 Mbps∼12 MBps) at 10 and 11 MBps respectively. The bandwidth overhead of HOOP (cryptography and headers) is negligible compared to the size of the files. The performance of HOOP is not altered when concurrent POST requests are issued. Figures 8c and 8d show that the system accounts for a large proportion of the CPU usage; this means that the CPU usage is mostly devoted to performing I/Os (i.e., reading from and writing to the network and the disk). It can be observed that, unlike for the set-top box, the CPU of the router saturates; this explains the slight performance gap between the two devices, with respect to the offload speed, observed in Figures 8a and 8b.

To conclude, both the mobile device component and the gateway component of HOOP can run efficiently: the processing capabilities of the devices and the gateways are sufficient to take advantage of the offloading capabilities offered by HOOP: In all configurations, the offloading is performed at a speed significantly higher than the upload speed of an average broadband connection (i.e., 1.15 Mbps). Note that these performances are obtained for slightly outdated devices running somewhat immature JavaScript libraries. In most configurations, HOOP can saturate the local connection (typically at 100 Mbps) hence fully taking advantage of the potential of Wi-Fi offloading.

10. Browsers can issue requests in parallel by opening up to 6-8 concurrent connections.



(a) Offload speed (router) (b) Offload speed (set-top)
(c) CPU usage (router) (d) CPU usage (set-top)

Fig. 8. Performance of the Hoop component running at the gateway. CPU usage is for POSTs of size 1,000 KB with a concurrency level of 1.

5.3 Efficacy

We evaluate the efficacy of HOOP: First experimentally in a static setting where the users do not move and stay connected to the same access point, and then through trace-driven simulations in a mobile setting.

5.3.1 Experimental Results

We experimentally assess the global performance of HOOP in terms of the time needed to complete an offload, based on our implementation on a laptop/router as described in Section 5.2. This metric reflects the immediate gain of a user in a static setting, as it corresponds to the time after which the user can switch off her mobile device and/or start moving out of the range of the Wi-Fi access point. For the web service, we enhance the Gallery [14] web photo organizer with HOOP compatibility[11], and we host it on a server connected to the Internet through a dedicated symmetric connection at 100 Mbps. The local network is connected to the Internet through an ADSL broadband connection synchronized at 12 Mbps (down)/1.15 Mbps (up). Neither the LAN link nor the broadband link has background traffic (i.e., other applications that use the links). Figure 9 shows the results for different POST sizes ranging from 1 to 50 MB in wired and wireless settings (for the connection between the mobile device and the gateway), with and without HOOP. It can be observed that HOOP significantly outperforms regular Wi-Fi offloading (i.e., without HOOP): The offload time is reduced by up to a factor of 85 in a wired setting and by up to a factor of 46 in the wireless settings. These factors roughly correspond to the ratios between the LAN and the broadband link speeds ($100/1.15 \approx 84$ for Ethernet;

11. We implemented HOOP on ResourceSpace [15] as well, to demonstrate HOOP's feasibility for Java-based uploaders.

the observed speed for Wi-Fi 802.11n is consistent with the actual speed of a link at 78 Mbps taking into account the MAC and TCP overheads).

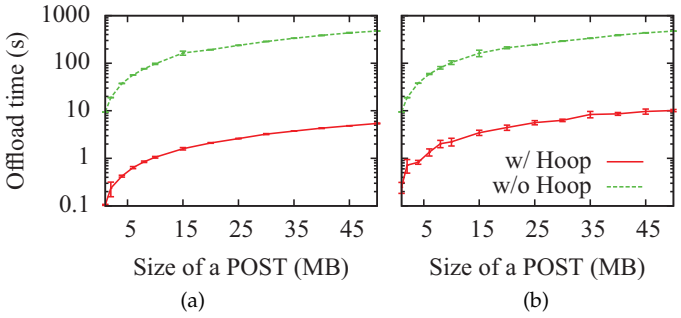


Fig. 9. Global performance of HOOP compared to the baseline in a (a) wired and (b) wireless setting.

5.3.2 Trace-Driven Simulation Results

Through trace-driven simulations, we evaluate the efficacy of HOOP in the scenario of a mobile user, who is equipped with a Wi-Fi/3G-enabled device, moving in a region covered by a community/commercial network of Wi-Fi access points and a 3G network. When the user is in the range of an access point of the network, her device connects automatically to it; this is usually done by the mobile OS (e.g., through the EAP-SIM [27] protocol for AT&T [7] or Swisscom [28] hotspots) or by a dedicated app (e.g., the FON app [8] that uses the user's credentials). The APs are assumed to have an unlimited storage capacity. In addition, the user could have a 3G data plan that enables her to connect to the Internet from anywhere in the region.

Dataset. The evaluation is based on a dataset of Wi-Fi access points from the FON network [8]. FON is a large community network with over 14 million hotspots worldwide[12] (most of them are located in Belgium, France, Japan, Portugal, and the UK–and soon in Germany and in the Netherlands – due to to strategic partnerships with leader national ISPs). The access points composing the FON network are mostly routers and set-top boxes provided and operated by the ISPs that hold total control over them (through automatic firmware updates); as such, they constitute first-class candidates for running HOOP. Users who host a FON access point at their home places or pay a subscription can connect (automatically) to the FON hotspots through a dedicated mobile app. The map of FON hotspots is available at `http://corp.fon.com/maps`. In early 2013, we collected the geographic coordinates of the Wi-Fi access points from SFR, a leading French ISP, which is part of the FON network. In urban and residential areas, the density of the network ranges from one hundred to more than a thousand hotspots per square kilometer. In Paris, France, the average density is 853±346 APs/km^2; 77% of the area is covered by at least one access point and the number of visible access points in covered areas is 6.70 ± 4.62 on average, assuming a communication range of 50 m.

In order to build connectivity traces, we correlate the coordinates of the Wi-Fi access points with synthetic mobility traces of users moving in the Paris area, France. Our dataset comprises two sets of traces: five *tourist* paths and five *commuter* paths. The first type corresponds to pedestrians who explore the city by hopping from one point of interest (POI) to another (including the Eiffel Tower, Notre-Dame, and the Arc de Triomphe). To generate these paths, we obtained the list of the points of interest in Paris from TripAdvisor, and we computed itineraries connecting them by using GoogleMap. We assumed an average walking speed of ∼2.5 km/h with 20-minute stops at POIs. The latter type corresponds to workers who commute between their homes and their work places by using the street public transport system (i.e., bus and tram). To generate these paths, we identified bus itineraries connecting residential areas with business and activity districts. We obtained the path and times from the Paris street transportation system (itineraries and time tables of RATP buses). We assumed that buses make 30-second pauses at stops. The average speed of the buses, determined from the official time tables, is ∼6 km/h.

Methodology. We developed a trace-based discrete-event simulator to compute the various metrics along which we evaluate HOOP. The mobility traces provide discrete samples of the user's position over time. We model the communication range of the access points with a fixed-radius (i.e., R) disc and we use a simple connectivity model[13]. Initially, users are connected to the closest access point in their range: in practice, this would correspond to the visible access point with the strongest received signal strength indication (RSSI), if any. When a user remains in the range of the access point her device is connected to, it does not change access point. As a user moves out of the range of the access point her device is connected to, the connection is interrupted and her device connects to the visible access point with the strongest RSSI, if any. Connecting to a new access point is assumed to take a constant time δt. When connected to an access point, a mobile device can communicate with the access point (and the devices on the same local network) at speed $b_{\mathrm{LAN}}(\cdot)$, which depends on the distance to the access point and the devices on the local network (including the mobile device) can communicate with remote Internet hosts at speed b_{WAN}. We assume that there is no background traffic on the LAN and on the Internet connection, and that the upload buffers of the gateways are empty. We denote by $b_{3\mathrm{G}}$ the speed of the 3G connection. Users from tourist traces generate photos of size S_{pic} at a rate $r_{\mathrm{pic}}^{\mathrm{go}}$ while moving, and at a rate r_{pic}^{POI} when at a point of interest. Users from commuter traces generate documents (or edit documents and upload the modified versions, e.g., to Dropbox or iCloud) of size S_{doc} at a rate r_{doc}. The generated files are stored in a buffer on the mobile device and uploaded to a web service in first-in first-out order. When the connection to an access point is lost, the on-going upload is aborted and it is restarted when the mobile device establishes a new connection. Table 3 summarizes the different simulation parameters together with a brief description and the value used in the evaluation.

12. In fact, 10 million hotspots at the time of the data collection.

13. In practice, wireless communications significantly differ from the unit-disc communication model. To obtain more realistic results, one could perform real-world experiments or use a more realistic network simulator such as NS-3. As we believe that such practical aspects do not alter the core aspects of HOOP and the associated benefit, in this paper, we use a simple communication model.

We consider the following connectivity scenarios and upload strategies:

- **Wi-Fi only (always mobile)**: Users always move according to their mobility trace and upload their data (w/ or w/o HOOP) over Wi-Fi whenever they are connected to an access point.
- **Wi-Fi + 3G (always mobile)**: Users always move according to their mobility trace and upload their data (w/ or w/o HOOP) over Wi-Fi whenever they are connected to an access point, otherwise over 3G.
- **Wi-Fi (mobile + static)**: Users move according to their mobility traces and upload their data (w/ or w/o HOOP) over Wi-Fi whenever they are connected to an access point. When connected to an access point, users move only when their upload buffer is empty (i.e., they wait until their upload buffers are empty before moving).
- **3G Only**: Users always move according to their mobility trace and upload their data over 3G (i.e., they never connect over Wi-Fi, hence they do not make use of HOOP).

TABLE 3
Simulation parameters: description and values.

Param.	Description	Value
R	Wi-Fi communication range	50 m
δt	Wi-Fi connection establishment delay[14]	10 s
b_{WiFi}	Wi-Fi connection speeds (by range)[15]	10–97.5 Mbps
b_{3G}	3G connection speed (upload)	0.8 Mbps
b_{WAN}	Broadband connection speed (upload) [10]	1.15 Mbps
S_{pic}	Picture size	3 MB
S_{doc}	Document size	1 MB
$r_{\text{pic}}^{\text{go}}$	Picture generation rate (on-the-go)	0.2/min
$r_{\text{pic}}^{\text{POI}}$	Picture generation rate (at POIs)	5/min
r_{doc}	Document generation rate (e.g., auto-save)	1/min

Metrics. We assess the performance of HOOP according to the following metrics. Together, they reflect the gains and costs of using HOOP for the mobile users, in different scenarios:

- **Per-session Wi-Fi offload capacity**: The maximum amount of data a mobile user can offload/upload over Wi-Fi during the connection time at an access point.
- **Total Wi-Fi offload capacity**: The maximum amount of data offloaded/uploaded over Wi-Fi (i.e., assuming that the upload buffer on the user device is never empty).
- **Delay**: The delay between the time a file is generated (e.g., the photo is shot) and the time it is uploaded on the web service.
- **Wi-Fi usage**: The amount and the fraction of data uploaded to the web service over Wi-Fi and the time spent uploading it.
- **3G usage**: The amount and the fraction of data uploaded to the web service over 3G and the time spent uploading it (in the Wi-Fi + 3G scenario).
- **Waiting time**: The time spent waiting for the offload or upload to complete (in the Wi-Fi: mobile + static scenario).
- **Energy consumption**: The energy consumed by the network interface to offload/upload the data.

Note that we distinguish between *offloaded* and *uploaded* data. Offloaded data corresponds to the data that has been transferred from the mobile device onto the gateway, whereas the uploaded data corresponds to the data that has been effectively transferred from the gateway to the online service. Without HOOP, these two quantities are the same. With HOOP, however, the amount of offloaded data is always higher than that of uploaded data, as some of the offloaded data could still be in the buffer of the gateway (and thus not effectively uploaded to the online service yet) at the end of the simulation. All of the offloaded data will eventually be uploaded to the online service, without the need for any further action from the mobile device, that could very well be switched off.

Results. Figure 10 shows the Wi-Fi offload capacity per session. This metric is directly proportional to the duration of the Wi-Fi sessions while moving. It can be observed that without HOOP, the 80-th percentile of the amount of data a user can upload is 4 MB for the tourist trace (slightly more than one photo) and 1 MB (the size of a document). This means that, without HOOP, the users from the commuter trace can upload a complete document in only 20% of the sessions. Finally, we note that the capacity is significantly higher (i.e., $\times 2.5$) for the tourist trace than for the commuter trace; this is because users from the tourist trace move more slowly than for the commuter trace (pedestrian vs. public transportation passengers), hence the sessions are longer.

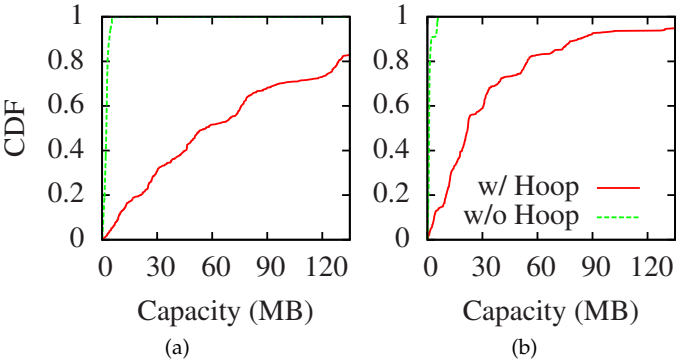


Fig. 10. Wi-Fi offload capacity per session for the (a) tourist and the (b) commuter traces.

We now look at the total Wi-Fi offload capacity, i.e., the total amount of data that is offloaded over Wi-Fi, assuming that mobile users have an infinite number of files to upload,

14. We experimentally estimated the value of this parameter .

15. In our simulations, the Wi-Fi speed is a (decreasing) function of the distance to the access point. We use the link speeds of 802.11n with 3 antennas, 20 MHz channels and a guard interval of 800 ns. The speed at the HTTP layer is assumed to be 50% of the speed at the link layer, as observed in our experiments (see Section 5.3). More specifically, in our simulations, the upload speeds are set as follows: 97.5 Mbps below 1 meter, 88 Mbps below 2.5 m, 76 Mbps below 5 m, 36 Mbps below 13 m, 29 Mbps below 19 m, 19 Mbps below 30 m, and 10 Mbps below 50 m (=R). Because in certain environments and under certain conditions the effective data rates and the communication range could be lower, we also run experiments with lower values. We show the results in Appendix A of the supplemental material.

normalized by the total time of the simulation. Note that this metric takes into account only the amount of *useful* offloaded data: the offloads that are aborted due to Wi-Fi connection loss are *not* taken into account.[16] We also look at the total upload capacity, that takes into account only the amount of data that is actually uploaded to the web service during the time of the simulation. The results are shown in Figure 11. The goal of this simulation is to assess the raw offload/upload *potential*. In practice, however, mobile users might not reach the offload capacity as the upload buffer could be empty if they produce only a small amount of data. For instance, in [20], the authors show that, in a practical scenario, upgrading the Wi-Fi data rate beyond 1 Mbps does not significantly increase the total amount of uploaded data. However, with the increase of the resolution of the cameras embedded on smartphones, the rate at which mobile users generate data is expected to grow significantly. It can be observed that the offload and upload capacities are zero for the commuter trace. This is because the Wi-Fi sessions are too short to upload even a single document (as shown in Figure 10). Finally, we observe that for the tourist trace HOOP increases the upload capacity by a factor of 50 and the offload capacity by a factor of 54. Note that the upload capacity with HOOP is significantly higher than the speed of the Internet up-link (i.e., 1.15 Mbps), which corresponds to the upload capacity in a static scenario where the users stay connected to the same access point during the entire experiment. This is because mobile users offload their data on different access points; therefore the uploads are performed simultaneously, thus increasing the total upload capacity. When users generate small amounts of data, the benefits of HOOP lie in the decrease of delays and energy savings, which we investigate in the next paragraphs.

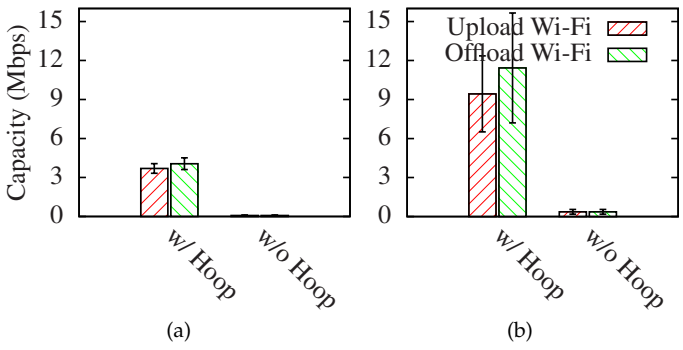


Fig. 11. Total Wi-Fi upload and offload capacities for the (a) tourist and the (b) commuter traces.

Figure 12 shows the cumulative distribution functions (CDF) of the delay in the different connectivity scenarios. In this scenario, we no longer assume that the users generate an infinite amount of data. Instead, we use the data generation model described in the methodology section: Users generate files of a fixed size (S_{pic} for the tourist trace and S_{doc} for the commute trace) at a fixed rate (r_{pic} for the tourist trace and r_{doc} for the commute trace). It can be observed that the delays are drastically reduced with HOOP. Surprisingly, we observe shorter delays in the "w/ HOOP always mobile" scenario than in the "mobile + static" (i.e., Wait) scenarios. This is because, in the mobile scenario, users offload the different files in their buffers to different access points. Hence, the files are uploaded simultaneously, whereas in the wait scenario, the files are uploaded sequentially as they are all offloaded at the same access point. Finally, the results show that 3G connectivity helps reduce the delays. This is because it enables the user to upload some of the files as soon as they are produced while the user stays at a point of interest with no Wi-Fi coverage. Note that with HOOP, the delays are comparable to or shorter than those observed when the users use only 3G, thus offering the users a similar quality of service while reducing her data plan usage. Therefore, unlike previous proposal of Wi-Fi offloading, the use of an active caching equipment prevents the increase of delays.

16. Note that in the case of mobile applications that support *resuming* aborted uploads (e.g., YouTube uploader [29]), all the uploaded data is useful.

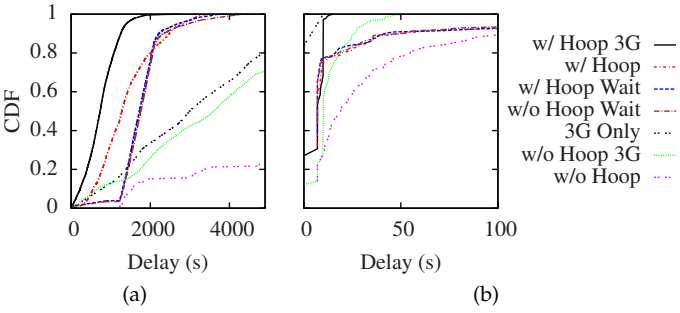


Fig. 12. Delay between the content generation and the upload in different connectivity scenarios for the (a) tourist and the (b) commuter traces.

In order to analyze the effect of the data rates and of the communication range on HOOP, but also to take into account the fact that, in practice, some environments could have communication performances lower than those considered in our first set of experiments, we run these experiments with lower values of the data rates and of the communication range. The results, shown in Appendix A of the supplemental material, show that the conclusions remain unaltered: HOOP still outperforms traditional Wi-Fi offloading and remains beneficial to the users (to a lower extent though).

We now look at the active time (i.e., the time spent uploading/offloading data over Wi-Fi or 3G). Figure 13 summarizes the results for different scenarios: it can be observed that HOOP consistently decreases the active time while increasing the amount of data offloaded. This is consistent with the increase of the amount of data sent over Wi-Fi (and the decrease of the amount of data sent over 3G). This translates into energy savings as the consumption per MB is lower for Wi-Fi than for 3G [4]. Note that the results from the two traces are not directly comparable because the traces have different durations, and that the users generate files of different sizes, at different rates. Note also that for the same trace, the duration of the experiment is longer in the "Wait" scenario. In our simulations, the waiting periods increase the duration of the experiment by 8% with HOOP, and by 10% without HOOP, for the tourist trace.

Finally, we evaluate the energy consumption of the data uploads in different scenarios. We look at the energy

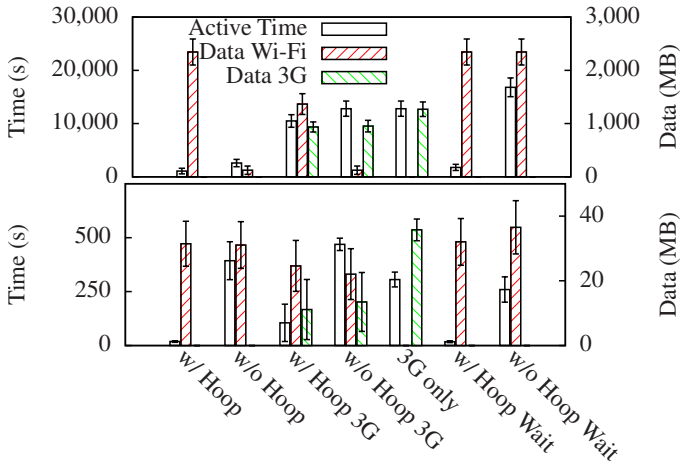


Fig. 13. Time spent sending data and amount of useful data uploaded/offloaded over Wi-Fi and 3G for the tourist (top) and the commuter (bottom) traces.

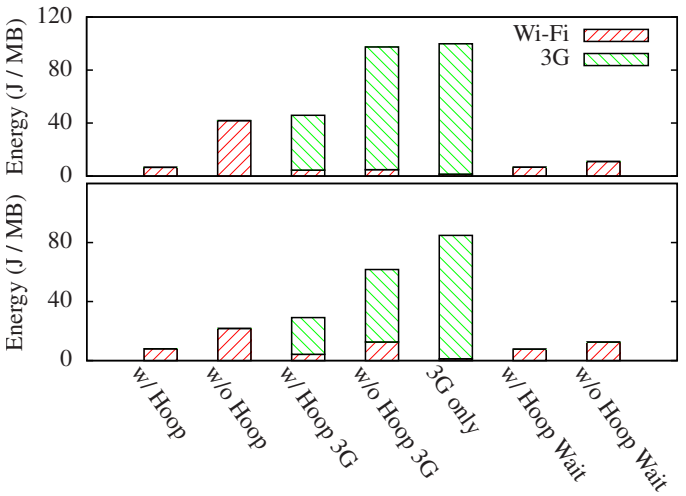


Fig. 14. Energy consumptions of the network interfaces for data upload, per sent KB, over Wi-Fi (incl. active time and scans) and 3G for the tourist (top) and the commuter (bottom) traces.

consumption only of the mobile devices (i.e., we disregard that of the gateway component and that of the web server) and we focus on the data transmission.[17] To do so, we rely on the values of energy consumption of smartphones' 3G and Wi-Fi network interfaces[18] from [4] (including active time and scan energy consumption for Wi-Fi), summarized in Table 4. We further assume that smartphones perform 1-second scans every 8 seconds. We show the results in Figure 14. It can be observed that HOOP consistently reduces energy consumption. The first reason is that the amount of aborted data uploaded (hence the amount of wasted energy) is reduced with HOOP. Secondly, in the case where 3G is used when there is no Wi-Fi connectivity, HOOP also reduces the energy consumption. This is due to the fact that HOOP offloads larger amounts of data over Wi-Fi (because it offloads data at a higher speed); therefore it uploads lower amounts of data over 3G (which is more energy consuming that Wi-Fi). We also observe the price to pay, in terms of energy consumption, for the delay improvement brought by the use of 3G connectivity (shown in Figure 12).

TABLE 4
Energy consumption of smartphones network interfaces [4].

	Transfer (J/MB)	Idle (W)	Scan (W)
Wi-Fi	5	0.77	1.29
3G	100	0	0

5.3.3 *Discussion: Scalability*

In the previous sub-sections, we evaluate the performance of HOOP in a single-user context. In this sub-section, we discuss the scalability of HOOP in a multiple-user context. In particular, we discuss how the performance of HOOP would be affected by the number of users who connect to the access points and transfer data concurrently.

We consider a mobile user and look at the effect of another user connecting to the same access point. In this case, the users have to share the Wi-Fi connection, which result in lower data rates. However, the broadband connection is shared as well, in such a way that, if the second user also upload data, the ratio between the Wi-Fi speed and the broadband up-link speed is not affected. In addition, if the second user does not make use of HOOP, this ratio will in fact increase, as the relative effect on the Wi-Fi speed is lower than on the broadband up-link speed. For instance, if the second user uploads data (without HOOP) at a rate of 0.5 Mbps, the broadband up-link speed is divided by two for the first user, but the Wi-Fi speed is almost not affected (decrease of 10% for a connection at 50 Mbps), which increases the benefits of HOOP. Finally, it should be noted that in the case where the gateway faces scalability issues (e.g., because its buffer is full), it can disable HOOP and switch to traditional offloading. To conclude, scalability can decrease the performance of HOOP in some cases by decreasing the ratio between the Wi-Fi speed and the broadband up-link speed; in general, however, an upload system with HOOP enjoys the same scalability properties as a system without HOOP. Formally analyzing the performance of HOOP in a multiple-user context, as in [31] for Delay-Tolerant Networks (DTN) or in [20] for mobile data offloading, is a very interesting problem, that we leave to future work.

6 DISCUSSION: ADOPTION AND INCENTIVES

In this section, we discuss the incentives for users to use HOOP; and the incentive and the costs, for the commercial parties involved (e.g., the web service), to implement and to deploy HOOP. These factors are key to its wide deployment and adoption. We also discuss additional features that make HOOP more attractive to the different parties.

HOOP is beneficial for the users, as shown in Section 5.3, and it does not require any user intervention as it operates seamlessly. As such, HOOP increases the brand/product value (1) of gateway/access point/NAS/set-top box manufacturers, (2) of the ISPs that provide gateways and/or set-top-boxes to their subscribers, and (3) of Wi-Fi access point

17. With the use of native code for JavaScript encryption (through the WebCryptoAPI), we expect the energy consumption of the encryption to be equivalent to that of HTTPS; in the case where HOOP is used, HTTPS is not needed for the communication between the mobile device and the gateway.

18. Note that 4G LTE is less power efficient than 3G [30].

(network) operators. The cost of deploying HOOP on such devices is minimal: The implementation is simple (∼350 source lines of code) and can be easily deployed via (automatic) firmware updates or via third-party applications available on specific repositories (e.g., optware packages and Synology's third-party packages [32]). The fact that HOOP is generic, and thus can be used by any web service, alleviates the need for the manufacturers and third-party application developers to implement ad-hoc solutions for each service (e.g., YouTube, Flickr, Picasa and Facebook uploaders implemented on the Fonera [21]). HOOP constitutes an interesting marketing argument for service providers as well and offers them an efficient ready-to-use solution that requires only limited changes to the web service. Furthermore, HOOP offers a unique opportunity for ISPs and service providers to control a fraction of their traffic, because they can delay the HOOP uploads. This enables them to smoothen the traffic peaks hence reduce the investments for dimensioning their equipment, as well as their bandwidth costs in the case where burstable billing (e.g., 95-th percentile [33]) is used. Finally, should the use of HOOP be more beneficial to the web service than for the ISP, the web service could pay the ISP for the offloaded uploads (as advocated and studied in [34], [35]), either directly (i.e., in currency) or through advertisement. For instance, we can envision a business model in which the photos uploaded to Facebook through HOOP are displayed with an icon "Uploaded by a Synology NAS" or "Uploaded from AT&T WiFi". The use of HOOP at some businesses could also be included in the framework of existing partnerships between business owners and web-services (e.g., Foursquare discounts [36] and Facebook Wi-Fi [37]).

7 CONCLUSION

In this paper, we have presented HOOP, a system for offloading data uploads on devices with storage capabilities, e.g., gateways, in a store-and-forward fashion. Our system enables mobile users to fully exploit the Wi-Fi link by relaxing the speed constraints due to the link that connects the LAN to the Internet. Unlike existing systems, HOOP operates transparently–from the stand point of the users–and provides a ready-to-use, secure and generic solution to data uploads offloading: The mobile users are not required to trust the gateway with their credentials and the gateway can neither see nor alter their data. We have reported on our performance evaluation of HOOP, demonstrating its efficiency and its efficacy: HOOP can run on devices with very limited capabilities (e.g., MIPS processor at 400 MHz with 32 MB of RAM) and decreases the waiting time of mobile users by up to a factor of 46. We intend to conduct a real-world field experiment to further assess the upload performance of HOOP, as well as the potential energy savings. In addition, we plan to perform a sensitivity analysis to study the effect of the different parameters (including the number of concurrent users) on the performance of HOOP, both experimentally and theoretically (e.g., by building a formal framework as in [20]). Finally, we intend to investigate the feasibility of extending HOOP to handle download tasks.

8 ACKNOWLEDGMENTS

The authors are very grateful to Olivier Heen, Sébastien Henri, Julien Herzen and to the anonymous reviewers for their insightful comments.

REFERENCES

[1] K. Huguenin, E. Le Merrer, N. Le Scouarnec, and G. Straub, "Hoop: HTTP POST Offloading from User Devices onto Residential Gateways," in *ICWS'14: Proc. of the 21st IEEE Int'l Conf. on Web Services*, 2014, pp. 654–661.

[2] F. Guidec, D. Benferhat, and P. Quinton, "Biomedical monitoring of non-hospitalized subjects using disruption-tolerant wireless sensors," in *MobiHealth'12: Proc. of the 3rd Conf. on Wireless Mobile Communication and Healthcare*, vol. 61, 2012, pp. 11–19.

[3] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile Data Offloading: How Much Can WiFi Deliver?" *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 536–550, 2013.

[4] N. Ristanovic, J.-Y. Le Boudec, A. Chaintreau, and V. Erramilli, "Energy Efficient Offloading of 3G Networks," in *MASS'11: Proc. of the 2011 IEEE 8th Int'l Conf. on Mobile Ad-Hoc and Sensor Systems*, 2011, pp. 202–211.

[5] I. IntelliNet Technologies, "Mobile data offload for 3G networks," White paper, 2009.

[6] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, "Taming the mobile data deluge with drop zones," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1010–1023, 2012.

[7] "AT&T WiFi," http://www.att.com/gen/general?pid=5949, Last visited: Jan. 2015.

[8] "FON," http://www.fon.com, Last visited: Jan. 2015.

[9] M. Dischinger, A. Haeberlen, K. P. Gummadi, , and S. Saroiu., "Characterizing Residential Broadband Networks," in *IMC'07: Proc. of the 7th ACM Int'l Conf. on Internet Measurement*, 2007, pp. 43–56.

[10] OECD, "OECD Broadband portal," http://www.oecd.org/sti/broadband/oecdbroadbandportal.htm, Last visited: Jan. 2015.

[11] S. Defrance, A.-M. Kermarrec, E. Le Merrer, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Efficient peer-to-peer backup services through buffering at the edge," in *P2P'11: Proc. of the 11th IEEE Int'l Conf. on Peer-to-Peer Computing*, 2011, pp. 142–151.

[12] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with Nano Data Centers," in *CoNext'09: Proc. of the 5th ACM Int'l Conf. on Emerging networking experiments and technologies*, 2009, pp. 37–48.

[13] Y. Go, Y. Moon, G. Nam, and K. Park, "A disruption-tolerant transmission protocol for practical mobile data offloading," in *MobiOpp'12: Proc. of the 3rd ACM International Workshop on Mobile Opportunistic Networks*, 2012, pp. 61–68.

[14] "Gallery: open source web based photo album organizer," http://galleryproject.org/, Last visited: Jan. 2015.

[15] "ResourceSpace: an Open Source Digital Asset Management," http://www.resourcespace.org/, Last visited: Jan. 2015.

[16] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3G using WiFi," in *MobiSys'10: Proc. of the 8th ACM Int'l Conf. on Mobile systems, applications, and services*, 2010, pp. 209–222.

[17] X. Bao, Y. Lin, U. Lee, I. Rimac, and R. Choudhury, "Exploiting naturally clustered mobile devices to offload cellular traffic," in *INFOCOM'13: Proc. of the 32nd IEEE International Conference on Computer Communication*, 2013.

[18] T. Han, N. Ansari, M. Wu, and H. Yu, "On accelerating content delivery in mobile networks," *IEEE Communications Surveys Tutorial*, vol. 15, no. 3, pp. 1314–1333, 2013.

[19] X. Zhuo, W. Gao, G. Cao, and S. Hua, "An incentive framework for cellular traffic offloading," *IEEE Transactions on Mobile Computing*, vol. 13, no. 3, pp. 541–555, 2014.

[20] Y. Kim, K. Lee, and N. B. Shroff, "An analytical framework to characterize the efficiency and delay in a mobile data offloading system," in *MobiHoc'14: Proc. of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2014, pp. 267–276.

[21] "Fonera 2.0n," http://www.fon.com/en/product/fonera2nFeatures, Last visited: Jun. 2014.

[22] M. Stapp, B. Volz, and Y. Rekther, "The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option," IETF RFC 4702, October 2006.

[23] W3C Working Draft, "Cross Origin Resource Sharing," http://www.w3.org/TR/cors/, April 2012.
[24] ——, "Web Cryptography API," http://www.w3.org/TR/WebCryptoAPI/, June 2013.
[25] "CryptoJS," https://code.google.com/p/crypto-js/, Last visited: Jan. 2015.
[26] "Stanford Javascript Crypto Library," https://crypto.stanford.edu/sjcl/, Last visited: Jan. 2015.
[27] Cisco Systems, "Extensible Authentication Protocol Method for GSM Subscriber Identity Modules (EAP-SIM)," http://tools.ietf.org/html/rfc2616, January 2006.
[28] "Swisscom Public Wireless LAN," http://www.swisscom.ch/en/residential/internet/internet-on-the-move/pwlan.html, Last visited: Jan. 2015.
[29] "YouTube API v2.0 - Resumable Uploads," https://developers.google.com/youtube/2.0/developers_guide_protocol_resumable_uploads, Last visited: Jan. 2015.
[30] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *MobiSys'12: Proc. of the 10th Int'l ACM Conf. on Mobile Systems, Applications, and Services*, 2012, pp. 225–238.
[31] A. Krifa, C. Barakat, and T. Spyropoulos, "Optimal buffer management policies for delay tolerant networks," in *SECON'08: Proc. of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2008, pp. 260–268.
[32] "Synology DSM Packages," https://www.synology.com/en-us/dsm/app_packages, Last visited: Jan. 2015.
[33] "Burstable Billing," http://en.wikipedia.org/wiki/Burstable_billing, Last visited: Jan. 2015.
[34] W. Dong, S. Rallapalli, R. Jana, L. Qiu, K. Ramakrishnan, L. Razoumov, Y. Zhang, and T. W. Cho, "ideal: Incentivized dynamic cellular offloading via auctions," in *INFOCOM'13: Proc. of the 32nd IEEE Int'l Conf. on Computer Communications*, 2013, pp. 755–763.
[35] S. Paris, F. Martignon, I. Filippini, and L. Chen, "A bandwidth trading marketplace for mobile data offloading," in *INFOCOM'13: Proc. of the 32nd IEEE Int'l Conf. on Computer Communications*, 2013, pp. 430–434.
[36] "Foursquare 101: Get a Discount," http://aboutfoursquare.com/foursquare-101/, Last visited: Jan. 2015.
[37] "Facebook Wi-Fi," https://www.facebook.com/help/facebookwifi, Last visited: Jan. 2015.

Kévin Huguenin is a permanent researcher at LAAS-CNRS, which he joined in 2014. Prior to that, he worked as a post-doctoral researcher at EPFL and at McGill University. He also collaborated with Nokia Research and he worked as an intern at Telefonica Research. He earned a M.Sc. degree from École Normale Supérieure de Cachan and the Université de Nice – Sophia Antipolis, France, in 2007 and a Ph.D. in computer science from the Université de Rennes, France, in 2010. His research interests include performance, security and privacy in networks and distributed systems.

Erwan Le Merrer is a researcher at Technicolor in Rennes, which he joined in 2009. Prior to that, he was a post-doctoral researcher at IRISA, France (2007–2009). He earned a Ph.D. in computer science from the Université de Rennes in 2007 (funded with an industrial grant from Orange labs). His research interests include large-scale distributed systems and applications.

Nicolas Le Scouarnec is a researcher at Technicolor in Rennes, which he joined in 2010. He earned a M.Sc. degree from INSA de Rennes in 2007 and a Ph.D. in computer science from INSA de Rennes, France in 2010 (funded with an industrial grant from Technicolor). His research interests include efficiency and reliability of cloud-based storage and computing systems.

Gilles Straub earned his engineer degree from École Nationale Supérieure des Telecom Bretagne in 1991. He started with Thomson-CSF and worked on ATM switching and network adaptation for professional video equipments. He joined Thomson/Technicolor Rennes in 1996 where he now is senior scientist. He actively contributed in the fields of home networking, wireless and broadband standards; he received the Broadband Forum Circle of Excellence Award in 2008 for his involvement in TR-135, which is a TR-069 data model of a set-top box. Since 2008, he has contributed to develop various technologies including large-scale distributed systems, unified content access, and cloud-based dynamic ad insertion.