A Web Browser-based Interaction Framework for Mobile Opportunistic Applications

Marcin Nagy
Aalto University
marcin.nagy@aalto.fi

Arseny Kurnikov
Aalto University
arseny.kurnikov@aalto.fi

Teemu Kärkkäinen
Aalto University
teeem.karkkainen@aalto.fi

Jörg Ott
Aalto University
jorg.ott@aalto.fi

Enabling access for legacy devices to opportunistic networking applications has two main benefits to the system and the users:

1. **Support for more platforms.** Implementing a native opportunistic networking platform requires operating system support that many mobile OSes do not provide (e.g., long-running background processes capable of doing networking). However, every mobile OS platform has a modern web browser, which the users are accustomed to using as a gateway to rich online services.

2. **No need to install native software.** Even users of devices that can support a native opportunistic networking platform are required to install and run an application to access the network. While today’s users are accustomed to installing hundreds of apps on their devices, each additional step the user must take will reduce the uptake the service gets. Further, the users are used to installing apps from the OS vendor’s app store, while in an opportunistic scenario the apps are likely to be distributed via other channels. For example, the Liberouter devices distribute the apps directly to the users via a local web portal, which may feel foreign to some users and cause the OS to throw up security warnings.

The main challenge that needs to be solved by the framework is interworking the peer-to-peer style opportunistic networking and client-server style web access approaches. While native opportunistic networking platforms are designed to exploit short, random encounters between nearby nodes to pass around messages, web browsers are designed around the client-server model with always-accessible servers. Further, the software development and distribution models for native mobile apps are well developed and understood, which also holds for many native opportunistic networking applications. However, the development and application distribution mechanisms for web browser based adaptations of mobile opportunistic applications are unsolved problems.

Our solution is based on leveraging two key concepts: 1) a throwbox based deployment, and 2) self-contained and semantically meaningful messages. While not every opportunistic networking system exhibits these characteristics, many practically deployable systems do. In particular, throwboxes are needed for very low density deployments, and self-contained messages for store-carry-forward routing.

The former serves as a natural point to deploy web servers, to which the legacy web browsers can connect. Our framework runs on these throwboxes, serving opportunistic web apps accessible through unmodified web browsers. These apps allow the users to interact with native opportunistic networking applications. For example, we present a web app that can send and receive photos with the native SCAMPI GuerrillaPics opportunistic photo sharing ap-

ABSTRACT

Opportunistic networking is one way to realize pervasive applications while placing little demand on network infrastructure, especially for operating in less well connected environments. In contrast to the ubiquitous network access model inherent to many cloud-based applications, for which the web browser forms the user front end, opportunistic applications require installing software on mobile devices. Even though app stores (when accessible) offer scalable distribution mechanisms for applications, a designer needs to support multiple OS platforms and only some of those are suitable for opportunistic operation to begin with. In this paper, we present a web browser-based interaction framework that 1) allows users to interact with opportunistic application content without installing the respective app and 2) even supports users whose mobile OSes do not support opportunistic networking at all via minimal standalone infrastructure. We describe our system and protocol design, validate its operation using simulations, and report on our implementation including support for six opportunistic applications.

1. **INTRODUCTION**

Research into opportunistic networking over the past decade has spawned a large number of different systems and architectures (e.g., Haggle [56, 61, 43], MobiClique [49], 7DS [57], ShAir [17], SCAMPI [31], IBR-DTN [15], PodNet [34, 22], and NetInf [12]). These platforms come with a variety of applications that work in environments with limited or no Internet connectivity. Instead of detouring information exchange via a centralized “cloud”, these systems pass the information directly between co-located devices over short range links.

The opportunistic networking architectures can be broadly divided into two categories: 1) fully ad-hoc, and 2) infrastructure assisted. The former are based only on direct encounters between nodes, while the latter introduce distributed infrastructure components, “throwboxes” [70], which serve as (possibly temporary) fixed points that offer services to nearby mobile nodes. One such opportunistic infrastructure architecture is the Liberouter system [21], which combines an inexpensive embedded hardware platform (Raspberry Pi or Intel Edison) with an opportunistic networking middleware (SCAMPI, IBR-DTN) to create a do-it-yourself opportunistic router. In this paper we extend our Liberouter platform to support legacy mobile devices running just web browsers. Our extension framework enables interactions with existing opportunistic networking applications from unmodified web browsers within the vicinity of a Liberouter device.
opportunistic networking system. Figure 1 shows the high level architecture at a high level, and provides a justification for our message based approach from a distributed systems viewpoint. We then lay out our throwbox based design for bridging the native opportunistic networks, and legacy web browsers in Section 5. To show the generality of our approach, we present an implementation with two different opportunistic networking platforms, SCAMPI and IBR-DTN, in Section 6. We follow it by presenting the use of the framework to enhance multiple existing applications to support legacy web browsers in Section 7. We end with a simulation based validation of the design in Section 8 and a look at the relevant related work in Section 9 and our conclusions in Section 10.

2. SYSTEM MODEL

As described previously, our approach extends a Liberator based opportunistic networking system. Figure 1 shows the high level model of the system. It is composed of opportunistic infrastructure nodes that act as access points (Liberator AP in the figure) to which nearby clients can connect. Our model considers two types of users: 1) native users who have installed the opportunistic networking software and applications (bottom left), and 2) legacy users who are assumed to only run an unmodified web browser (bottom right). This leads to opportunistic and legacy links respectively. The additional application interactions that we aim to enable through our framework are between the legacy users and native users (the native users can already interact among themselves).

We extend the functionality of the lightweight Liberator devices by adding a Web App Interaction Framework between a local opportunistic router instance and a web portal. The interactions are bi-directional, allowing the legacy users to both consume and produce content for the existing opportunistic networking applications.

We will next explain the underlying messaging model from a distributed systems perspective in Section 2.1 and show how our system can be understood from the applications’ viewpoint in Section 2.2.

2.1 Messaging Model

The messaging model that we use can be derived from the fundamental properties of distributed systems and opportunistic store-carry-forward networks. In general, distributed systems are designed to establish a globally consistent state in multiple independent entities through message exchanges. This is shown in Figure 2a where a state change $S_0 \rightarrow S_1$ in $N_2$ is propagated to the other entities in the system through messaging. This process takes place from time $T_1$ to $T_3$, which is the time it takes for the message to propagate to all other entities. During this time the system is in a globally inconsistent state, and the process of reaching a consistent state is called equilibration. The fundamental problem of distributed systems design is to ensure that the equilibration process results in globally consistent states in the face of concurrent state transitions in the entities.

The CAP conjecture [6] is a useful tool for thinking about the fundamental properties of distributed systems. The major insight to be gained from CAP is that when facing partitions (P), a distributed system design can trade availability (A) against consistency (C). Strict consistency can be enforced in some systems by a global locking mechanisms, but it leads to no availability when the system is partitioned and the lock cannot be acquired. Consensus mechanisms employing quorums alleviate this by requiring only a subset of the entities to agree on state changes. This effectively leads to the larger part of a partitioned system to remain available, at the expense of the other part having no availability and an inconsistent view of the global state. Mechanisms such as using soft state and caches closer to the clients ensure availability in the case of partitions, but can lead to a globally inconsistent state and problems when trying to reconcile inconsistent states after a partition ends. This essentially buys more availability at the cost of going from strict consistency to eventual consistency. In all cases, a key assumption in the design of both centralized and peer-to-peer distributed systems is that partitions are transient phenomena, and the system can eventually reach a consistent and available state.

An important observation regarding the CAP formulation is that
partitioning is tightly coupled with latency, forming a type of latency-partition duality. This is because in the absence of explicit knowledge about the network, a partition is not distinguishable from a long delay, as illustrated in Figure 3a. This forces system designs to use delay thresholds as indicators for partitions, which in turn makes implicit assumptions about the information propagation speed in the network. In particular, it assumes that latencies are in the order of user-acceptable application level transaction times; an assumption that holds in well-connected infrastructure networks where failures are transient conditions.

Opportunistic store-carry-forward networks are composed of pairwise contacts between mobile nodes. This means that the information propagation latency is limited primarily by the inter-contact times between the mobile nodes, and not by the speed of light (and queueing) as in well-connected networks. Combining this with the latency-partition duality, such systems can be seen as being in a constant state of partitioning, causing the breakdown of the underlying assumptions of the mechanisms used to maintain consistency properties in “classical” distributed systems. Another way to state this is that the time for a distributed system built on such a network to equilibrate can be orders of magnitude larger (even unbounded) than the time scales required by meaningful application semantics. I.e., changes are made much faster than the time it takes to reach an equilibrium.

This leads to the need to abandon the idea of global consistency, and instead build distributed systems that are inconsistent by design. A system that is inconsistent by design requires each entity to build their own locally consistent view of the world based on their own set of observations. In concrete terms, it is the client software’s responsibility to create a locally consistent view or state from the (random) set of messages that it has received.

Messaging is used as a way to transition the state of the entities in distributed systems. This can be written as $m: \Delta(S_x, S_{x+1})$, where $S$ is the state of the system. In other words, a message contains the difference between the two states, which the recipient can apply to its state $S_x$ to transition to the state $S_{x+1}$. This has two major implications: 1) the recipient must be in the given state $S_x$ or the message is useless, and 2) as a result of applying the message, the recipient will be in the precise state $S_{x+1}$. This makes sense in distributed systems that are designed to (periodically) reach globally consistent states.

The above messaging model does not make sense in distributed systems that are inconsistent by design, including opportunistic networks. First, the messages should be applicable in any state, otherwise in a system where every entity is potentially in a different state, most messages would be useless. Second, since the system is not designed to ever reach a globally consistent state, the messages do not need to result in all the recipients moving to the same state, just some locally consistent state.

This leads to a different messaging model, where each message will cause a different state transition from one locally consistent state to another in each receiving entity. This can be written as $N(m) : S_N \rightarrow S_{N'}$, where node $N$ applies message $m$ to transition from the locally consistent state $S_N$ to another locally consistent state $S_{N'}$. We can observe that it must be possible to apply the transition from any state, including the empty state, into some locally consistent state; $N(m) : \emptyset \rightarrow S_N$. This is typically expressed in opportunistic and delay-tolerant networking as messages being self-contained and semantically meaningful.

It is this fundamental property of self-contained messages that

---

1There are inconsistent-by-design distributed systems built for well-connected networks too, such as the git distributed source code management system.

---

Figure 3: a) Simplified application structure following the MVC model with selected code embedded in messages; b) running an application-independent interpreter for embedded code to allow web-based access to message contents.

we exploit in building our framework. This messaging model further implies that the network will have a large number of these messages (i.e., content), which can be interpreted independently of any specific application state. However, the applications themselves are still required in order to participate in the system. The key idea of our framework is to ship minimal, generic application logic along with the messages, which allows any node a degree of interaction with the system (e.g., view, respond and send) without requiring the specific native application.

2.2 Application Model

Interactive applications with (graphical) user interfaces, such as running on mobile devices, opportunistic or not, can conceptually be described by the the Model-View-Controller (MVC) paradigm, which may also serve as an implementation model for such systems. We are not tied to this model, but rather use it in the following for illustration purposes. As shown in Figure 3a (left), the MVC model comprises three elements: A data model ($M$) (schema plus actual data) represents the application state as structured data, a view ($V$) of this model is rendered to the user via a GUI, and a controller ($C$) receives input from the user via the GUI and acts upon the input by modifying the model accordingly (after validating the input). The code for such (opportunistic) applications usually resides in the user device (e.g., tablet, smartphone) and thus requires installation before being able to interact with the respective application and its contents.

Our framework overcomes the need for installing apps on the user devices by having an application designer replicate parts of the application view and controller code along with the model for message interpretation and generation inside the message itself in addition to the complete (self-contained) state as shown in Figure 3b (right). To execute the code embedded in the framework-compliant messages, we design an application-independent interpreter that is integrated with an opportunistic networking router and has access to its buffered messages. The interpreter screens messages queuing in the router and processes those containing framework-compliant code. It extracts and formats the contents to make it accessible via standard web technologies (HTTP as transport and HTML5 for presentation/interaction) so that any smart mobile device can interact with the contents, as depicted in Figure 3c. This enables any node to dynamically interpret and render messages of any framework-compliant applications, allowing users to access and interact with the contents, generate replies, and create entirely new messages.

We foresee that the router (e.g., SCAMPi or IBR-DTN) and the interpreter run on dedicated opportunistic (infrastructure) devices, e.g., Liberouters, which also serve as access points and message relays for mobile devices. However, they could also run on other mobile user devices.
3. APP INTERACTION FRAMEWORK

We now turn to a detailed description of the framework. We begin with the description of framework and its functionality at a conceptual level (Section 3.1). The heart of the framework is realized by extending opportunistic application messages with a specific metadata format that is described in Section 3.2. Section 3.3 provides a detailed description of the framework for a DTN router, including functionality and interactions between framework components. Section 3.4 describes the process of bootstrapping applications and nodes. Finally in Section 3.5 we discuss security implication for the framework design.

3.1 Conceptual Model

Recall from the section [2] that self-contained and semantically meaningful messages are the foundation of our framework design. Our goals are to: (1) enable generation and interpretation of the messages by web browsers through the framework and (2) provide interoperability between native applications and their web equivalents. We could easily achieve message interpretation by embedding HTML code into a message that describes its content. However, this would not address the generation issue and would tightly couple the application state with message interpretation. It might also imply replicating (part of) the content, natively and as HTML. Therefore, we choose a more general approach.

The interpretation of messages can be seen as a transformation of opportunistic messages into a view rendered by a web browser, i.e., generation of an HTML page from a set of messages. There are two aspects to this transformation: 1) the individual messages must be transformed into HTML representation, and 2) these individual views must be composed into a coherent application view. For example, a photo sharing application would be composed of a transformation of the individual messages into, e.g., thumbnail views, and a composition of those thumbnails into an application view, e.g., list sorted by the date.

For the first aspect, we define two transformations: message summary and message presentation. The message summary transformation generates a concise view of the message, suitable for inclusion in a listing of a large number of messages. The message presentation transformation generates a detailed view of the message, intended to be displayed on its own to a user who is interested in the message content. Both transformations are pure functions that take as input the message and generate the presentation as output (e.g., HTML code). The transformations are message type and application dependent, and are expected to be included in the message itself by the original generating application.

As messages are self-contained, the message transformations alone already provide a useful view into the messages. However, in many applications there may be further requirements on the display of sets of messages. For example, a message board application should list its messages in time order and possibly threaded by topic. This corresponds to an application level logic which would normally be implemented by the native clients. While our immediate goal is not to enable fully fledged applications to be built on top of the framework, it must be possible to apply some degree of application layer composition logic in order for the generated views to be meaningful. To this end we define application presenter transformations.

Application presenter transformations are similar to message transformations in that they are functions that map a set of inputs into an HTML view to be displayed via a web browser. As input, they take a set of messages (e.g., all messages belonging to a particular application), and optionally some state generated by a previously run presenter. For example, a forum application could have one application presenter transformation that processes all forum messages and generates a list of topics. When the user selects a topic, another application presenter transformation would generate a list of all messages within the topic. In general, each application can have an arbitrary number of linked application presenter transformations that can call each other.

The generation of messages can take two forms: generating new messages, or responding to an existing message. Both take as input a predefined set of values and compose a message, which is then injected into the opportunistic network by the framework. The difference is that the response transformation also gets as a parameter the message to which the user is responding.

3.2 Embedding Code in Messages

Opportunistic applications use the Bundle Protocol [57] as the basic communication protocol. We aim to embed transformations into messages without disrupting opportunistic network functionality for devices that do not understand embedded transformations. To this end, we have developed a simple Web Message Interaction Format (WMIF) that attaches transformations to messages as metadata.

WMIF has key-value format, defines five basic keys, and each key corresponds to the specific transformation. The appPresenter key contains application presenter transformation. Message level transformations are contained in summary and presentation keys. Finally, for generating new content, new and response keys carry appropriate transformations.

Furthermore, WMIF defines also additional keys that help the framework to present the message content in the web browser. The bundleType key indicates type of data contained in the message (e.g., if message carries a photo, bundleType should have “image” value). For simple messages comprising only data of one type in the message body, the framework may take advantage of the bundleType key being present, and provide a simplified interpretation of the message content (e.g., by taking a thumbnail for the “image” bundleType).

WMIF also defines the keys description, service, icon, and system. The description provides a short text summary of the message content (e.g., short comment on the location of photo shot). The service key maps the message to the application name and icon contains an embedded application icon. The system key is used by the framework to find out which opportunistic router (e.g., SCAMPI, IBR-DTN) the message belongs to. The framework uses the system key to provide: (1) a mapping between message and application level transformations and (2) a generic view to the user about applications available at the particular Librouter. Table 1 provides an overview of defined metadata items.

3.3 Framework Design

The framework comprises four main components, namely Message Presenter, Application Presenter, Web Interaction Manager and Message Generator. The first two units are responsible for interpretation of messages, and their arrangement as a part of HTML

| Metadata key | Description |
|--------------|-------------|
| bundleType   | Indicates content type carried inside the message (see table for list of available values) |
| description  | Text describing message content |
| service      | Application name |
| system       | Opportunistic software name |
| icon         | File containing application icon |

Table 1: Web Message Interaction Format summary
completed WMIF data examination, the application Presenter creates a unique message ID as the key (step (2) in Figure 4). Having stored inside the content database (content DB) as a hash map with processing WMIF data contained in the message. All WMIF data are available keys if such simple ones are known for the message type. However, it also implies that legacy users can only generate content for an application if the Liberouter they connect to has already stored message of this app, even if the users wants to create a new message—and this first message would need to be created by a native application. To overcome this limitation, we take two further steps: (1) distributing applications within the framework to transform legacy into opp nodes and, more importantly, (2) allow creating initial content from legacy nodes.

Recall that a native app is essentially structured binary contents (e.g., an executable plus auxiliary data). As such, it can be wrapped up in a message and distributed across the opportunistic network to become accessible via an “app store” [2]. This already makes the network self-sufficient, except for the bootstrap problem that the native opportunistic router (e.g., SCAMPI or IBR-DTN) and the “app store” would need to be installed in the first place.

To address (1), the framework makes all native applications available to web browser users, including the router and “app store” applications: If a received message has bundleType value of “app”, the Message Presenter extracts the binary contained in the message and makes it accessible for download via the web browser (see Figure 5). For (2), we also include the transformation code in messages used to distribute the opportunistic apps. A user visiting the web portal can then not just choose to install an opportunistic app but also to create a message for the respective app by just using the framework.

3.5 Security considerations

Our security considerations cover three issues: (1) secure execution of transformations, (2) message authenticity and (3) impact of communication mode on privacy. More details on privacy and providing access to content of encrypted messages are in our tech report [7].

Threat model. We are concerned with threats that an adversary:
Secure execution of transformations. Execution of transformations of unknown origin poses a serious threat to the secure functioning of the framework, as they may include system calls causing disruption of the framework functionality as well as leakage of data contained in other stored messages. This threat model motivates implementation of the file system level isolation and the system calls filtering (see section 4 for details).

Message authenticity. We address the threat of another user impersonation by providing message authenticity mechanisms. Message authenticity requires providing a binding between an identity of the message sender and his/her personal public key. In this work, we assume availability of such a public key distribution system for legacy nodes (e.g., the SocialKeys [22], or PeerShare [41]). Verifying message authenticity can be realized on the router side or in the web browser. For the former, the framework instance itself verifies message signatures using public keys available in its storage. The browser side verification assumes that public keys are accessible in the browser for the web application (see section 4 for implementation details).

Encrypted message content access. In the framework design we make the assumption that content carried inside messages is unencrypted, thus easily accessible by the framework. However, we argue that in the future legacy users experience can be further enriched by providing them also access to the content encrypted messages. To do this, we assume availability of cryptographic keys in legacy users’ nodes which are either delivered via some key distribution system, or derived by legacy users by means of Password-based Cryptography [28]. Since the message content should be accessible only to users that have an appropriate key, encrypted messages must be decrypted inside the browser (using either the Web Cryptography API, or specialized JavaScript module fetched from the framework). Finally, the transformations of decrypted messages must also be executed inside the browser (thus they must be written in JavaScript), so that they can be directly displayed to the legacy user without the necessity to interact with the framework.

Communication mode and privacy considerations. The current framework design assumes that all messages stored in the router DB should be accessible for the legacy users. This assumption holds well for all applications in which users do not target their messages at a (closed set of) recipient(s), but rather share contents at a (closed set of) recipient(s), but rather share contents openly. Thus, legacy users can read all messages for the group they are part of. To improve user experience, the framework provides real-time content updates by means of the WebSocket Protocol [19] that is implemented using the socket.io library.

The Web Interaction Manager is a set of three PHP scripts. The presentation.php script returns a detailed view about the particular message. The new.php and generate.php scripts are called in case of generating a response to the existing message, or a new message. The new.php returns the web form generated by appropriate transformations to the Application Presenter, so that it can be shown to the user. The generate.php script acts as a form handler that processes data submitted in this web form, and sends them to the Message Generator for the message creation. To enable access to the whole content associated with the particular message in the Redis, all these scripts are executed with the Redis message key as the URL parameter.

The Message Generator is the native opportunistic router application that: (1) reads data submitted by the Web Interaction Manager, (2) using the opportunistic router API, transforms them into an appropriate message and (3) publishes the resulting message in the opportunistic network.

4. PLATFORM INTEGRATION

We now turn to the implementation of the framework and its integration with SCAMP1 and IBR-DTN routers. The current implementation is based on JSON-RPC API, so it can be attached to the message as part of the WMIF format described in the Section 3.2. The only requirement is that if an attached item is a Java serialized class, such a message must also include its URL as a string.

In our implementation there are no constraints on data types that can be attached to the message as part of the WMIF format described in the Section 3.2. The only requirement is that if an attached item is a Java serialized class, such a message must also include its URL as a string.

4.1 Framework Core framework. We implemented the framework as a set of PHP extensions and two native applications. The Message Presenter component is developed in standard Java. It parses newly received messages by the opportunistic network router, and stores their content as a hash map inside the Redis (acting as the content DB). Text items contained in the message are stored directly in the Redis, while binary items are written to the framework’s persistent storage, and only their access paths are stored in the Redis.

The Application Presenter consists of the set of PHP scripts that: (1) communicates with the Redis to learn which application’s content is available and 2) arranges presentation of content generated by transformations in the web browser. To improve user experience, the framework provides real-time content updates by means of the WebSocket Protocol [19] that is implemented using the socket.io library.

The Web Interaction Manager is a set of three PHP scripts. The presentation.php script returns a detailed view about the particular message. The new.php and generate.php scripts are called in case of generating a response to the existing message, or a new message. The new.php returns the web form generated by appropriate transformations to the Application Presenter, so that it can be shown to the user. The generate.php script acts as a form handler that processes data submitted in this web form, and sends them to the Message Generator for the message creation. To enable access to the whole content associated with the particular message in the Redis, all these scripts are executed with the Redis message key as the URL parameter.

The Message Generator is the native opportunistic router application that: (1) reads data submitted by the Web Interaction Manager, (2) using the opportunistic router API, transforms them into an appropriate message and (3) publishes the resulting message in the opportunistic network.

4.1 Framework

Core framework. We implemented the framework as a set of PHP extensions and two native applications. The Message Presenter component is developed in standard Java. It parses newly received messages by the opportunistic network router, and stores their content as a hash map inside the Redis (acting as the content DB). Text items contained in the message are stored directly in the Redis, while binary items are written to the framework’s persistent storage, and only their access paths are stored in the Redis.

The Application Presenter consists of the set of PHP scripts that: (1) communicates with the Redis to learn which application’s content is available and 2) arranges presentation of content generated by transformations in the web browser. To improve user experience, the framework provides real-time content updates by means of the WebSocket Protocol [19] that is implemented using the socket.io library.

The Web Interaction Manager is a set of three PHP scripts. The presentation.php script returns a detailed view about the particular message. The new.php and generate.php scripts are called in case of generating a response to the existing message, or a new message. The new.php returns the web form generated by appropriate transformations to the Application Presenter, so that it can be shown to the user. The generate.php script acts as a form handler that processes data submitted in this web form, and sends them to the Message Generator for the message creation. To enable access to the whole content associated with the particular message in the Redis, all these scripts are executed with the Redis message key as the URL parameter.

The Message Generator is the native opportunistic router application that: (1) reads data submitted by the Web Interaction Manager, (2) using the opportunistic router API, transforms them into an appropriate message and (3) publishes the resulting message in the opportunistic network.

4.1 Framework

Core framework. We implemented the framework as a set of PHP extensions and two native applications. The Message Presenter component is developed in standard Java. It parses newly received messages by the opportunistic network router, and stores their content as a hash map inside the Redis (acting as the content DB). Text items contained in the message are stored directly in the Redis, while binary items are written to the framework’s persistent storage, and only their access paths are stored in the Redis.

The Application Presenter consists of the set of PHP scripts that: (1) communicates with the Redis to learn which application’s content is available and 2) arranges presentation of content generated by transformations in the web browser. To improve user experience, the framework provides real-time content updates by means of the WebSocket Protocol [19] that is implemented using the socket.io library.
carry a "class", or JAR file implementing the class. In such cases, the Message Presenter dynamically loads attached data, transforms it to JSON\(^3\) and stores it in the persistent storage.

**Security components.** File system level isolation constrains transformations to access only data contained inside the message and parts of the content DB that are related to the message. The framework realizes it by: (1) creating a temporal directory and inserting into it all data carried inside the message, (2) binding external resources (i.e., content DB) required by the transformation to the temporal directory (using mount system call), (3) isolating other file system resources from the transformation (by means of chroot call). We realize system calls filtering using the seccomp-bpf facility\(^8\). It allows to whitelist a subset of system calls that a transformation has the permission to execute \([1]\). Calling of a system method that has not been whitelisted causes termination of the transformation process. Since the transformation should be given access to all data carried by the message, our seccomp-bpf profile allows the execution of only system calls that open and operate on files. For system administrators concerned with chroot vulnerabilities, it is trivial to further enhance sandbox design by replacing chroot with pivot_root\(^7\) system call \([44]\). Unlike chroot, pivot_root confines the process into a completely separate root file system, thus even if the process manages to escalate privileges and theoretically break away from the file system isolation, it cannot access file system resources outside its directory, as these resources do not exist for it. Further security improvements can be made by defining mount and network namespaces for the sandbox and running transformations inside Linux containers\(^1\). However, we believe that for our use case it is an overkill.

Browser side message authenticity requires availability of public keys in the browser via the HTML5’s local storage or the device’s persistent storage (e.g., the File API \([51]\)). Both of these solutions are supported by all modern browsers. The authenticity verification process is realized fully inside the JavaScript code via the Web Cryptography API \([53]\). Although it is preferable from the user privacy point of view to use browser-side message authenticity, the framework-side alternative (recall from Section\[3.5\]) is the more realistic option due to poor support of the Web Cryptography API among the modern browsers \([14]\).

**Transformations.** The transformations are implemented in Python, and must follow basic transformation design guidelines described in Section\[3.1\]. Furthermore, to guarantee access to whole message content, the transformation is always executed with its own client libraries to communicate over TCP with middleware services using Wi-Fi, Wi-Fi Direct \([67]\), and Wireless Personal Access Network (IEEE 802.15.4)\(^9\). Like SCAMPI, the IBR-DTN’s design follows the basic DTNRG architecture \([8\ 57\ 13]\). In addition to SCAMPI, it implements also convergence layers for other underlying network technologies (i.e., UDP over IP, IEEE 802.15.4 LoWPAN). Unlike SCAMPI, IBR-DTN does not offer any application level extensions (e.g., content search mechanisms), however, in addition to the Epidemic routing, it provides other opportunistic network routing protocols, namely PRoPHET \([35]\) and Direct Delivery routing.

Applications can be developed use IBR-DTN communication service with Java and C++ client libraries currently available to developers. For other programming languages, developers must write their own client libraries to communicate over TCP with middleware services.

The process of IBR-DTN integration into the framework is identical to the SCAMPI integration. The only difference is that developers must use: 1) ibrdtnlib as a Java library, 2) libapi for C++ development and 3) ibrdtn-api for Android.

**NULL router.** In this mode, the framework cannot take any advantage of libraries provided by opportunistic network platform. Framework access to the router DB is achieved in identically as for SCAMPI and IBR-DTN. However, creation of new messages by the framework requires it to: 1) manually build a message (according to the Bundle Protocol \([57]\)) and 2) copy the created message to the router DB.

### 4.2 Opportunistic router integrations

SCAMPI\(^{1}\), SCAMPI is the opportunistic network platform that provides the network layer implementations capable of delivering messages based on store-carry-forward networking. It is implemented as the middleware that runs on any platform with Java support. It uses Wi-Fi and Bluetooth as the physical layer technology to route messages between nodes. The underlying design follow the DTNRG architecture and protocols \([8\ 57\ 13]\). It also includes a number of extensions to support publish/subscribe messaging, geographically constrained content distribution \([29]\), and content search mechanisms. Among existing opportunistic network routing protocols, only Epidemic \([55]\) and static routing are supported. Namespaced metadata key-value pairs can be attached to the messages to provide application hints to the underlying networking layer.

Applications can be developed to use the communication services provided by SCAMPI. These applications can either be native (Java) applications, or HTML5 web applications. The applications can also be distributed by the middleware without requiring a centralized app store. However, to run the applications, the middleware and the application itself must be running on the device.

Integration of SCAMPI into the framework requires: 1) allowing the framework to access the router DB and 2) implementing new content creation functionality to be in the SCAMPI compliant format. The former one is accomplished by granting the framework access rights to the router DB location, as SCAMPI implements router DB as a set of file system directories. The latter one is achieved by using SCAMPI’s AppLib library that allows to publish new messages to the SCAMPI router instance.

**IBR-DTN.** IBR-DTN is another opportunistic network platform, similarly to SCAMPI working as middleware. It runs on any Linux based operating system as well as on Android and BeagleBone\(^4\). It discovers and establishes communication links with nearby devices using Wi-Fi, Wi-Fi Direct \([67]\) and Wireless Personal Access Network (IEEE 802.15.4)\(^9\) technologies. Like SCAMPI, the IBR-DTN’s design follows the basic DTNRG architecture \([8\ 57\ 13]\). In addition to SCAMPI, it implements also convergence layers for other underlying network technologies (i.e., UDP over IP, IEEE 802.15.4 LoWPAN). Unlike SCAMPI, IBR-DTN does not offer any application level extensions (e.g., content search mechanisms), however, in addition to the Epidemic routing, it provides other opportunistic network routing protocols, namely PRoPHET \([35]\) and Direct Delivery routing.

Applications can be developed use IBR-DTN communication service with Java and C++ client libraries currently available to developers. For other programming languages, developers must write their own client libraries to communicate over TCP with middleware services.

The process of IBR-DTN integration into the framework is identical to the SCAMPI integration. The only difference is that developers must use: 1) ibrdtnlib as a Java library, 2) libapi for C++ development and 3) ibrdtn-api for Android.

**NULL router.** In this mode, the framework cannot take any advantage of libraries provided by opportunistic network platform. Framework access to the router DB is achieved in identically as for SCAMPI and IBR-DTN. However, creation of new messages by the framework requires it to: 1) manually build a message (according to the Bundle Protocol \([57]\)) and 2) copy the created message to the router DB.

### 5. ENHANCED APPLICATIONS

\(^3\)json-io: https://github.com/jdereg/json-io
\(^4\)http://linux.die.net/man/2/pivot_root
\(^5\)https://linuxcontainers.org
\(^6\)Bootstrap: http://getbootstrap.com
\(^7\)ibrdtn-api: https://github.com/jdereg/ibrdtn-api
\(^8\)libapi: http://beagleboard.org/bone
To enhance an opportunistic application for legacy users access, an app developer must implement in Python transformations needed in the application. Among all defined transformations, only the message summary transformation is necessary, as it allows the framework to always interpret the content of the particular message (content generation and application level logic is not available then). Developers embed transformations into messages using opportunistic router libraries (recall from the Section 4.2).

To facilitate the development process, we have also implemented a simple transformation testing environment. It allows the app developer to test correctness of his implementation by: (1) executing the transformation in it and (2) verifying that HTML view generated by the transformation is coherent with the application design.

Now we present description of our enhanced applications. Among the SCAMPI apps we have extended GuerrillaPics, GuerrillaTags, and PeopleFinder, while for the IBR-DTN router we enhanced Whisper, Talkie, and ShareBox.

5.1 SCAMPI apps

GuerrillaPics is an opportunistic network photo sharing application. Since its messages have no dependencies between them (e.g., there are no replies or groupings), this is the simplest use-case for the framework. First, an application presenter transformation is used to generate a grid of photo thumbnails. The transformation calls the message summary for each message to generate a thumbnail and to get the creation timestamp. These summary elements are then listed in a grid based on time ordering and displayed to the users through the framework. If a user selects one of the summaries by clicking the thumbnail, the message presentation transformation is used to generate a full resolution view. Finally, the app has also the ability to share a new photo through the framework by means of the new transformation. Figure 9 illustrates comparison between native GuerrillaPics, and its framework version.

GuerrillaTags is a message board application where messages are not independent as the photos in the previous case, but rather exist in the context of a topic (tag). In this case the summary view is composed from all the unique tags contained in all the messages that belong to the app. This is done by the application presenter transformation, which: 1) applies the message summary transformation to all the app messages and 2) filters out duplicates from the produced list of tags to the list of available topics (each unique topic is a summary item). Each topic has its own state entry, which contains the set of bundles with the common tag. The presentation view is per-topic, and lists all the messages belonging to the given topic. It is generated by another presenter transformation by applying the message presentation transformation on all the relevant messages (as determined from the previously saved topic state), and sorting the resulting list by creation timestamp. This demonstrates how complex application level logic can use the application transformations to generate complex presentation views. The application also allows to post messages via the framework by using the new transformation.

PeopleFinder is an adaptation of Google Person Finder system into opportunistic networks. It allows missing persons records to be generated, and notes to be attached to those records. Each message generated by the application contains the record and a set of all notes related to the record known by the sender. The application summary view is generated from the person records. There may be multiple records for the same person, in which case each is listed separately. The notes attached to a person are displayed in the detailed view, which is generated by the message presentation transformation. Adding a new note for an existing record is done through the response transformation, which gets the original record (and notes) as a parameter. The transformation appends the new note to the existing ones, and generates a new aggregate message containing the record and all known notes. A brand new missing person record is generated through the new transformation.

5.2 IBR-DTN apps

All IBR-DTN apps use unicast communication model. Since the framework currently does not support this communication model (recall Section 4.3), we have modified the apps to use the group communication model.

Whisper is an opportunistic network chat application in which exchange text messages within a group of devices. Thus, the messaging model and transformations logic are identical to the GuerrillaTags application with the group of devices identifier (group ID) playing the role of the GuerrillaTags’s topic. Whisper includes additionally the functionality of responding to the current chat discussion through the framework via the response transformation.

Talkie is an opportunistic network walkie-talkie voice application in which users share voice messages with a group of devices. The messaging model and transformations logic for the message presentation are identical to Whisper’s use case (only chat message is replaced by voice). To generate a new message (or respond to existing conversation), Talkie uses new and response transformations. They use the Media Capture API [7] for recording the audio messages.

ShareBox is an opportunistic network file sharing application in which users exchange pictures or other files within the given group of devices. Since its messages have no dependencies between them, the messaging model and transformations logic is similar to the GuerrillaPics app. The application summary view shows a table of message summary views for all Sharebox messages. The message summary view shows the size of the message together with the device identifier of its sender and the timestamp. Finally, if the user selects a particular message, the message presentation transformation shows the actual files carried inside the message.

6. VALIDATION

Our web-based interaction framework offers, in principle, content access to legacy nodes (in addition to opp nodes). To achieve this, the framework requires code to be shipped along with message state updates, which incurs overhead. In the following, we first evaluate the overhead and its impact. We then turn our attention to how many legacy nodes could be reached by content if those nodes chose to look at the messages.

6.1 Overhead

To evaluate overhead, we measure the actual message sizes of our implementation for three applications with sample contents. The results are shown in table 3: When using a text messaging app with tiny content, the message size grows almost 50-fold. However, this is due only to the small size of the native messages. If the content size of the application messages increases, the overhead becomes more reasonable. For photo sharing, with small photo size of 65–120 KB, including the framework code adds just 5–10% overhead. For the most sophisticated application we looked at, the PeopleFinder, the overhead is roughly factor 15.

Obviously, the code added via the framework is a function of the complexity of the code required to interpret, render, and construct messages: simpler applications will need less code. The overhead is obviously also a function of the content size so that more elabo-

\footnote{Not currently supported by Safari and Internet Explorer.}
rate content will cause, even if more complex code is needed, limited overhead only. One can argue that trends in web site complexity and size\[11\] show that increasing amounts of effort are put into conveying probably roughly the same amount of contents. Thus, adding more sophisticated interaction framework code for a better experience just would mirror what is done on the web.

For opportunistic networks, the most important question is if and how the larger message sizes affect message delivery performance. To this end, we carried out simulations using the ONE simulator \[32\] with two different mobility models: 1) SPMBM: Shortest Path Map-Based Movement between waypoints chosen from the Helsinki downtown map (4.5×3.4km\(^2\)) \[32\] for 50, 100 and 200 pedestrians moving with speeds \(v = U(0.5, 1.5)\) m/s without predefined points of interest. 2) Same as 1 but additionally we introduce 10, 65 and 325 static access points. 3) SMOOTH: a simple way to model human walks \[40\] with the map from KAIST scenario (10×18km\(^2\)) and 50 DTN nodes. SMOOTH is a synthetic model but it is based on real traces and captures most of the properties of human mobility. Our nodes communicate at a net bit rate of 2 Mbit/s with a radio range of 50 m. The nodes use simple epidemic routing \[65\]. We choose a random DTN node to generate a new message every 12 s, 60 s, and 300 s, referred to as high, medium, and low load, respectively. Messages expire after 5400 s. The message sizes correspond to those for native and framework-enhanced messages for the text and photo sharing applications to pick two extremes. We measure the fraction of nodes that obtain a copy of each message, termed coverage, and plot the average of 10 simulation runs, each lasting for 12 hours.

As shown in Figure\[7\] we find that the overhead of the framework does not notably impact the performance results. The coverage remains the same both for using native application messaging and for the framework-enhanced messaging for the text chat application (top) as well as for photo sharing (bottom). For the SMOOTH mobility model we obtain similar results just with much lower coverage rate due to sparser node distribution. This indicates that in the realistic parameter range, the system is not bottlenecked by the

\[11\]www.websiteoptimization.com/speed/tweak/average-web-page/
contact capacity, and therefore the added messaging overhead does not negatively impact the message distribution.

Further, a maximum message rate limit was also observed in past experiments, which have shown that the per-message overhead of protocol implementations appears to be more dominant in communication performance than the per-byte overhead. This was, for example, found in a comparison of three different DTN bundle protocol implementations [35], particularly for growing the payload size from 10 bytes to 10 KB and beyond. Our own (not yet statistically significant) experiments seem to confirm this. While implementation details play one important role here, there are also systematic aspects to consider: nodes that meet need to exchange vectors which messages they have, decide which ones to replicate, and then perform a forwarding process for each message, which causes per-message overhead. Researchers also found that neighbor discovering and pairing with peers is expensive and takes easily tens of seconds [35] while the sending 20 KB of data takes only 160 ms even assuming just 1 Mbit/s data rate. We therefore argue that the framework overhead is not of substantial importance in practice.

6.2 Content Reach

The previous subsection suggests that the overhead introduced by our framework won’t degrade performance for opp nodes. However, how does the framework allow reaching out to legacy nodes? We conduct further simulations to answer this question, using largely the same setup as above. In addition, we introduce two further classes of nodes: access point nodes that, besides running the opportunistic networking protocols, also serve as WLAN access points and run the server side of the interaction framework (cf. Figure 4). And legacy nodes that only interact with these access point nodes, but neither with each other nor with regular DTN nodes. We choose the number of legacy nodes to be equal (L1), five-fold (L5), or ten-fold (L10) the number of DTN nodes.

Obviously, how many legacy nodes we can reach will depend on the movement patterns of those nodes and where the access point nodes are located, and how they move. However, our simulation results shown in Figure 8 hint that we can notably increase the visibility of opportunistic content. For the SPMBM scenario, we find that the content coverage may come close to that of the DTN nodes and reach close to 40% of the legacy nodes. Additionally deploying 10 static access points brings it up to 50%. Comparing this to Figure 7 content availability is roughly equal for DTN and legacy nodes. Note that, the fraction of legacy nodes reached gets bigger as their number increases from L1 to L5 to L10, so the absolute number of additional nodes reached grows even more. We obtain similar findings for high (up to 40%) and low loads (up to 50%) for both text and photo applications. The SMOOTH scenario yields a qualitatively similar picture, again at much lower performance. Results of our simulations with 100 and 200 DTN nodes for all scenarios are also in line with these findings.

7. RELATED WORK

Our framework borrows concepts from different fields of related work to create a unique combination. Most important is the concept of embedding programs into messages and executing them in network nodes, discussed in the past as active networking [62] and mobile code. Lee et al. [33] present a node architecture allowing to deploy in-network services in a next generation Internet. Its main contribution is the concept of making the core network become a distributed service execution environment. It also describes an architecture for extensible router allowing for implementing new router features. Similar concepts of extensible router architecture can be also seen in works of Router Plugins [11], LARA++ [55], PromethOS [50], Pronto [21] and SARA [1], SOFTNET [69], PLAN [23], Bowman and CANEs [56] are examples of active networking systems that assume network packets to contain programs, which are used to manage network nodes. All these systems concentrate on executing code inside the network in order to improve network capabilities, while our solution takes advantage of transformation execution to enable content access to legacy users. Moreover, active networks focus on individual (small) packets, thus limiting the amount of code that can be carried, while our message-based system is not limited by MTU size.

Baldi et al. [4] and Thorn [64] study the applicability of programming languages in mobile code. Ghezzi et al. [20] describe architectures of mobile code applications. Our framework choose one specific programming language and a specific application design tailored to the purpose of read and write access to opportunistic message contents.

Security aspects of mobile code are covered by Arden et al. [2]. He introduces a new architecture for secure mobile code that developers can use, publish and share mobile code securely across trusted domains. Older work presenting security aspects of mobile code are Rubin et al. [54], Zachary [68] and Brooks [53]. Kosta et al. [59] present the concept of using mobile code for improving code execution on mobile devices by offloading part of code execution to the cloud. Similar work by Simanta et al. [60] describe an architecture for offloading mobile code execution in hostile network environments. Unlike these works, our system uses mobile code as a tool for message content presentation and our main concern is offering an isolated execution environment for the application code so that the code does not harm the node running it (which is quite similar to the concerns of protecting routers in Active Networks).

Another set of related work concerns document presentation techniques. MINOS [9] was an early system that allowed for presentation of the multimedia content embedded in a document. Boguraev et al. [5] presents usage of linguistic analysis tools for generation of text document description and its visualization to the user. Our framework is not limited to content presentation, but offers a full-fledged interaction to the legacy user.
Douceur et al. [16] shows an alternative approach for using native applications in the web browser by the legacy device users. This system requires web servers to have an (application-specific) gateway installed that translates a native app into a web app. On the other hand, our framework is more flexible, as it carries all transformation code inside the messages themselves so that no node requires prior knowledge of specific applications.

8. CONCLUSION

In this paper, we have presented the concept of web browser based framework that provides legacy users the ability to take part in an opportunistic network with nothing more than a standard web browser. The users can both see the content published by native applications and publish content themselves. We showed through simulations that this significantly increases the reach of the content both for legacy and native users, with overheads that do not impose a significant penalty on message delivery.

We have described our implementation of our framework for two popular opportunistic networking platforms, and enhanced a total of six applications for those platforms. This has a potential to significantly increase the attractiveness of both networks, as legacy users can access the content without having to install any extra software – a common issue for opportunistic network deployments in the real world.

There are a number of open issues to be addressed in our future work. This includes enhancing security and privacy functionality of the framework by enabling access to encrypted content for legacy users to allow the implementation of closed communication groups. One special case of closed groups will be embracing legacy users also for the point-to-point (user-to-user) unicast communication. Finally, we are exploring how to exploit web browser capabilities for establishing direct browser-to-browser communication for legacy users without a mediating entity (such as an access point).

The framework source code, the simulation code, and the SD card images including the framework are available at http://www.netlab.tkk.fi/tutkimus/dtn/legacy-app-framework.

9. REFERENCES

[1] SECanF in a kernel filter, https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
[2] Arden, Owen and George, Michael D. and Liu, Jed and Vikram, K. and Askarov, Aslan and Myers, Andrew C. Sharing Mobile Code Securely with Information Flow Control. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP ‘12, pages 191–205. Washington, DC, USA, 2012. IEEE Computer Society.
[3] M. Bagnulo, B. Alarcos, M. Calderon, and M. Sedano. Providing Authentication & Authorization Mechanisms for Active Service Charging. In QoJS, volume 2511 of Lecture Notes in Computer Science, pages 337–346. Springer, 2002.
[4] M. Baldi and G. P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In K. Torii, K. Futatsugi, and R. A. Kemmerer, editors, ICSE, pages 146–155. IEEE Computer Society, 1998.
[5] B. Boguraev and R. Bellamy. Dynamic document presentation. SIGCHI Bull., 30(2):72–77, Apr. 1998.
[6] E. Brewer. Cap twelve years later: How the "rules" have changed. Computer, 45(2):23–29, Feb 2012.
[7] D. C. Burnett, A. Bergkvist, C. Jennings, and A. Narayanan. Media Capture and Streams API. http://w3c.github.io/mediacapture-main/getusermedia.html
[8] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838, Apr. 2007.
[9] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria. Multimedia document presentation, information extraction, and document formation in minos: A model and a system. ACM Trans. Inf. Syst., 4(4):345–383, Dec. 1986.
[10] J. Corbet. Seccomp and sandboxing. http://lwn.net/Articles/332974/
[11] Dan Decasper and Zubin Dittia and Guru Purulkar and Bernhard Plattner. Router plugins: A software architecture for next generation routers. In IEEE/ACM transactions on networking, pages 229–240, 1998.
[12] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (netinf) – an information-centric networking architecture. Computer Communications, 36(7):721 – 735, 2013.
[13] M. Demmer, J. Ott, and S. Perreault. Delay Tolerant Networking TCP Convergence Layer Protocol. Experimental RFC 7242, June 2014.
[14] A. Deveria. Can I use Web Cryptography? http://caniuse.com/#feat=cryptography
[15] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf. IBR-DTN: an efficient implementation for embedded systems. In Proc. of ACM CHANTS, 2008.
[16] J. R. Douceur, J. Elson, J. Howell, and J. R. Lorch. Leveraging legacy code to deploy desktop applications on the web. In Proceedings of the 5th USENIX Conference on Operating Systems Design and Implementation, OSDI’08, pages 339–354, Berkeley, CA, USA, 2008. USENIX Association.
[17] D. Dubois, Y. Bando, K. Watanabe, and H. Holtzman. ShAir: Extensible Middleware for Mobile Peer-to-Peer Resource Sharing. In Proc. of ESEC/FSE, 2013.
[18] K. Fall. A Delay-Tolerant Network Architecture for Challenger Internets. In Proc. of ACM SIGCOMM, 2003.
[19] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455, Dec. 2011.
[20] C. Ghezzi and G. Vigna. Mobile code paradigms and technologies: A case study. In K. Rothermel and R. Popescu-Zeletin, editors, Mobile Agents, volume 1219 of Lecture Notes in Computer Science, pages 39–49. Springer, 1997.
[21] GISL Hjalmyrsson, The Pronto Platform : A Flexible Toolkit for Programming Networks using a Commodity Operating System. In Proceedings of International Conference on Open Architectures and Network Programming (OPENARCH), pages 98–107, 2000.
[22] M. Demmer, J. Ott, and S. Perreault. Router plugins: A software architecture for next generation routers. In IEEE/ACM transactions on networking, pages 229–240, 1998.
[23] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, Sept. 2000.
[24] J. Kangasharju, J. Ott, and O. Karkulahti. Floating Content: Information Availability in Urban Environments. In Proc. of IEEE Percom, Work in Progress session, March 2010.
[25] T. Hossmann, F. Legendre, P. Carta, P. Gunnngberg, and C. Rohner. Twitter in Disaster Mode: Opportunistic Communication and Distribution of Sensor Data in Emergencies. In Proc. of ExtremCom, 2011.
[26] IEEE 802.15 Working Group. IEEE 802.15.4-2006 Standard, 2006.
[27] M. T. Islam, A. Turkulainen, T. Kärkkäinen, M. Pitkänen, and J. Ott. Practical Voice Communications in Challenged Networks. In Proc. ExtremeCom, 2009.
[28] B. Kalisky. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, Sept. 2000.
[29] J. Kangasharju, J. Ott, and O. Karkulahti. Floating Content: Information Availability in Urban Environments. In Proc. of IEEE Percom, Work in Progress session, March 2010.
[30] T. Kärkkäinen. Instant Messaging and Presence in Challenged Networks. Master’s thesis, Aalto University, School of Electrical Engineering, Comnet, Oct 2014.
[31] T. Kärkkäinen, M. Pitkänen, P. Houghton, and J. Ott. SCAMPI Application Platform. In Proc. of ACM MobiCom CHANTS workshop, 2012.
[32] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In Proc. of SIMUTOols, 2009.
