Liberouter: Towards Autonomous Neighborhood Networking

Teemu Kärkkäinen and Jörg Ott
Aalto University Comnet
Espoo, Finland
firstname.lastname@aalto.fi

***Abstract*—We present the Liberouter framework, a complete communication system design (and its implementation) to enable neighborhood networking without relying on the Internet infrastructure for information exchange. The core is a low-cost router platform that serves as WLAN access points, individual or (dis)connected ones, and offers message storage and relaying in combination with a distributed app store. It allows bootstrapping Android devices to become additional routers (and thus expand the network) and install applications. It also instruments other mobile devices to assist in message forwarding and offers web-based access to content of the neighborhood stored locally.**

I. Introduction

The combination of the Internet's effortless global connectivity and its hugely popular web- and cloud-based services has blurred the distinction between local and remote applications; at least for those well connected. While this has undoubted advantages for providing high quality services to large numbers of users, it has also made people (blindly) dependent on the Internet and centralized third parties. The cloud now stores the users' data and controls the software that executes on that data. We argue that this centralization and globalization should be balanced by the creation of more localized and distributed *neighborhood networks* that are built, operated and controlled by the users themselves.

There may be a number of reasons to desire such localized networking: 1) The centralization of power in the cloud turns the operators into censors for the content published through their platforms. This can manifest through intentional filtering or reviewing, or incidental algorithmic ranking and prioritization of content (and applications in the case of app stores). Even only locally relevant, ephemeral, user created content gets moved to physically remote cloud servers to be stored into third party curated silos. For these classes of content it may often make much more sense to distribute only through localized networks and services. 2) As shown by recent revelations, the centralized networks and services are targets for ubiquitous, ongoing, large-scale monitoring by various government agencies, who indiscriminately collect and analyze end user traffic in bulk both directly from the network and from the cloud service provider systems. This can lead to some users losing confidence in the current infrastructure networks and services. 3) As widely available as Internet access has become, its use is still constrained by cost (e.g., cellular data and WLAN hotspot charges) and by provisioning limitations (e.g., availability, capacity) in many areas. The latter is especially true in remote areas where connectivity is not always available, and in disaster scenarios where the infrastructure has been rendered inoperable.

In this paper, we present the *Liberouter* framework for local networking that i) offers robust communication between reasonably co-located users, ii) facilitates the creation and distribution of new applications within the neighborhood, and iii) provides (limited) infrastructure assistance for running services. As meta goals, the network should be created from inexpensive "infrastructure" components that are self-supplied by the users. The infrastructure may be put and kept in place permanently and may be extended over time, but instantiating networks in an ad-hoc manner should also be supported. While Internet access and gatewaying to Internet services is clearly an option (and the Internet could also be used to interconnect multiple local network islands), those are not the foremost goals. Rather, we emphasize the local character, which also implies that access to content be limited to those in the vicinity.

The two main approaches to local networking that have been proposed in the past are *opportunistic networking* [21], [20] and *wireless mesh networking* [2]. The Liberouter system design arises from an attempt to solve the practical issues discovered in experimental deployments of both of these approaches. In particular, *opportunistic networking* relies on message passing between mobile devices carried by users. This requires the devices to be physically colocated and a radio technology that is capable of discovering nearby peers and forming links between them. Problems arise when the density of peers is so low that physical colocation is rare, and from the energy requirements of continuous peer discovery. Further, in practice many mobile devices are optimized for discovering and using infrastructure networks (such as Wi-Fi access points) and have poor or non-existent support for discovery of direct peer-to-peer communication opportunities. *Wireless mesh networking*, on the other hand, approaches the problem by trying to build end-to-end packet paths through a network of cheap Wi-Fi enabled devices. In practice mesh networks tend to rely on fixed installations of mesh router nodes [1], [16]. Using mobile devices is also possible [25], but the instability of the resulting network topology limits the size and scope of such networks.

The Liberouter approach is positioned between the two previous approaches. Like opportunistic networking, it relies on direct peer-wise contacts to pass messages. However, unlike opportunistic approaches it does not require direct device-to-device contacts, instead introducing autonomous "opportunistic routers" that devices can connect to and swap messages with. This relaxes the colocation requirement as well as the device-to-device discovery requirement. Similar to mesh networking, the Liberouter approach relies on cooperating infrastructure components that are installed by users. However, unlike mesh networking these components are fully autonomous and do not need to be directly connected to each other. This means that already a single Liberouter unit can provide value to users, and the network is not sensitive to topology changes due to components being added or removed from the deployment.

The Liberouter approach is similar to the use of throwboxes [26], [4] to increase throughput and reduce latency in Delay Tolerant Networks (DTN). However, such approaches typically aim to optimize and improve the performance of routing algorithms in one-to-one messaging scenarios, while in our system design focuses on publishing content. Further, we assume application model where there is not a strict receiving set for published messages, but rather users get some value from every received unique message (e.g., in the case of photo sharing each received photo adds value, while being able to receive a specific photo is less important). Designs have been proposed for using throwbox-based systems (also referred to as *infostations*) in urban environments [5], [7], [23]. However, these designs assume infrastructure connected, interoperating throwboxes as content stores that user queries are executed against (in the case of [5]), to which users subscribe with their topics of interest (as for [23]), or that act as Internet gateways (in the case of [7]). In contrast, our design assumes fully autonomous Liberouters and focus on content dissemination instead of fulfilling query-response transactions.

The resulting system properties of our design are also relevant to people in countries with strong censorship mechanisms [3], [18], [6], and a risk of prosecution for those attempting to circumvent such mechanisms. The Liberouter framework presented in this paper should eventually become suitable for such applications as well, but our present focus is on enabling its autonomous operation. Mechanisms essential when targeting censorship circumvention, e.g., for concealing communication, providing real anonymity, and ensuring plausible deniability [12] are subject to our ongoing and future work.

II. Neighborhood Networking

We use a rather lose definition of neighborhood, mainly to contrast our work from obtaining global connectivity: for the purpose of this paper, a neighborhood is a group of at least two users with some common interests. We expect those users to be sufficiently close to each other and/or sufficiently densely distributed that they are often in range of WLAN routers that are either directly connected or can be indirectly connected through message *ferrying*, e.g., because users with mobile devices move frequently back and forth between the different locations. Intuitively, this would yield distances of 100m to maybe a few km between users. Examples for such neighborhoods include a school campuses, parks, groups of houses in a street or around a lake, entire villages, and downtown areas, but also camp sites in the middle of nowhere.

Neighborhood networking may take a number of different shapes, as shown in Figure 1, all of which we seek tos accommodate. At the very core is using 802.11 WLAN to interconnect different nodes, for which our Liberouters serve as the infrastructure, i.e., (groups of) wireless access points: (1) Devices (black dots) may be connected to an individual Liberouter (C) as shown at the bottom, but (2) multiple Liberouters (A, B) may also be interconnected using a wired or wireless link.[1] To extend communication beyond the reach of an individual or connected groups of Liberouters, (3) we utilize delay-tolerant networking concepts [10] and allow mobile nodes (grey dots) to carry messages between Liberouters. Finally, (4) we leverage mobile opportunistic networking ideas for information exchange between mobile nodes when they encounter each other (white dots on the left) [24], [25], possibly with the assistance of WLAN hotspots (white dots on the right) [15].

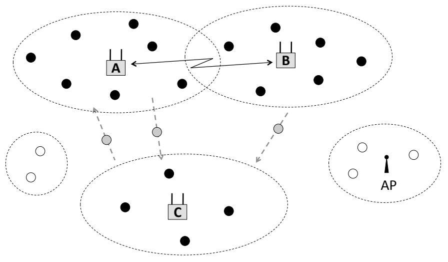


Fig. 1. Sample neighborhood networks: A–C are Liberouters, AP is a WLAN hot-spot, the dots represent nodes. Some nodes (black) are stationary and connected to a Liberouter, some (grey) move between Liberouters and carry messages, some (white) interact directly leveraging other WLANs.

The key constituents are, on the one hand, (stationary) Liberouter boxes providing the minimal connectivity infrastructure and offering persistent storage, and, on the other hand, mobile devices (e.g., feature phones, smart phones, tablets, laptops) serving both as endpoints and as message carriers. Finally, stationary devices connected to Liberouters serve as user equipment or servers.

While delay-tolerant and opportunistic networking for related scenarios has been explored before, our contribution is in the system design: Liberouters bootstrap capable mobile nodes and turn them into Liberouters as well; they serve as distribution mechanism for apps; and they serve as content store and relay for application data.

III. A Low-Cost Disposable Router

The Liberouter network is built around inexpensive do-it-yourself opportunistic routers. They combine cheap, general purpose computing platforms like the Raspberry Pi[2] or the BeagleBoard[3] with the SCAMPI opportunistic communication stack [14]. Each router can bootstrap and provide services to mobile devices.

Like wireless mesh network routers, these opportunistic routers are cheap enough that they can be built and installed by individual users. But unlike mesh routers that require careful configuration and positioning in order to produce a stable topology, Liberouters are fully autonomic, as well as

[1]This could include overlay links across other networks including the Internet, provided using such does not violate the target properties of the neighborhood network. We do not pursue this aspect further in this paper.

[2]http://www.raspberrypi.org/

[3]http://beagleboard.org/

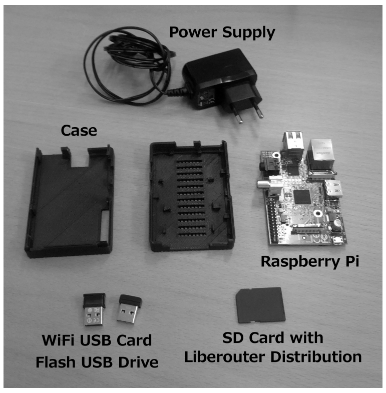


Fig. 2. Hardware configuration for a Liberouter device.

configuration and maintenance free and therefore can be placed anywhere with power and within Wi-Fi range of users. This means that the routers can be deployed, e.g., in homes and businesses, where they then provide services to passing users. They could also be deployed by hiding them in public areas (possibly with a disguised case) or even in moving nodes such as cars. Raspberry Pi based hardware configuration for a Liberouter device is shown in Figure 2.

Liberouters appear as ordinary open Wi-Fi access points to nearby users. Upon connecting to one with a mobile device the user is presented with a captive web portal. The portal allows users to download and install native versions of the SCAMPI router software and other mobile applications. Each Liberouter is also running an instance of the SCAMPI application platform. Mobile devices near a Liberouter Wi-Fi access point and running the SCAMPI router will automatically discover and connect to, and exchange message with, the router's SCAMPI instance without user interaction. Further, the captive portal and the local SCAMPI router could be potentially integrated so that the captive portal site can display content stored in the router to any connected device with a web browser (including those that do not have a native SCAMPI router implementation, namely iOS).

Even a single Liberouter can provide useful communication services to two or more users, while adding more routers will increase the robustness and reach of the network. This is in contrast to mesh networks where a single non-Internet connected mesh node can provide very few meaningful communication services.

A. Basic Operation

Both the Liberouters and mobile devices run instances of the SCAMPI opportunistic router platform. The SCAMPI platform is responsible for discovering nearby nodes, opening communication links, and implementing the store-carry-forward networking that enables the asynchronous multi-hop messaging service between mobile devices and Liberouters. The SCAMPI stack is shown in Figure 3.

The SCAMPI platform is based on the Internet Research Task Force (IRTF) Delay-Tolerant Networking Research Group (DTNRG) architecture and protocols. In particular, the store-carry-forward overlay is implemented using the Bundle Protocol [22]. The messages generated by Liberouter applications

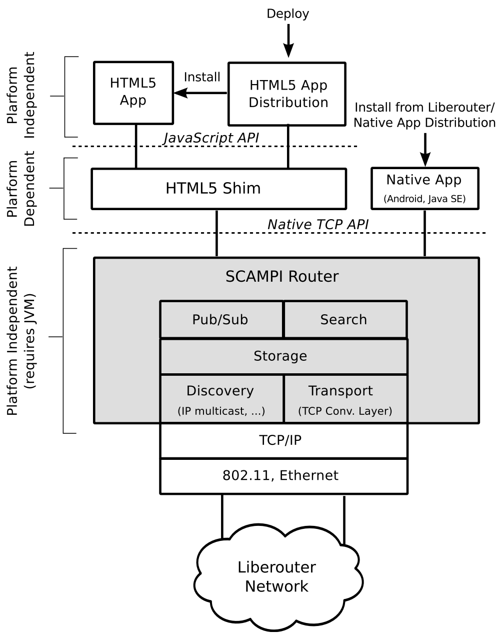


Fig. 3. SCAMPI application platform architecture.

are encapsulated in Bundle Protocol bundles and routed hop-by-hop over direct device-to-Liberouter and device-to-device contacts. The direct device-to-device transport is implemented using the TCP Convergence Layer protocol [8] running on top of the platform's standard TCP/IP stack.

Since every Liberouter access point appears identical to the connected user devices, service discovery can be implemented via known IP addresses and ports. Upon connecting to a Liberouter access point, the SCAMPI instance on the mobile device attempts to open a connection to a well known port on the Liberouter. The Liberouter replies with a service descriptor that contains, among other things, information on how to establish a transport link to the router. The result is a star topology where the Liberouter acts as a hub and the mobile devices as leaves. Thus, the Liberouter is able to store and forward copies of all messages created by the clients that connect to it. This is what allows the Liberouters to act as electronic "drop-boxes" (or: *throwboxes* [26]), passing messages between mobile users that may never be physically co-located, and for the messages to spread to multiple Liberouters by the users' devices.

In addition, the SCAMPI instances on the mobile devices can be configured to discover each other directly when connected to non-Liberouter Wi-Fi networks. This is achieved through IP multicast beaconing based service discovery mechanism, which results in a full connectivity mesh between the co-located nodes. Further, the SCAMPI platform can provide support for other lower layer communication technologies such as Bluetooth and Wi-Fi Direct. However, in current generation devices these technologies require user interaction (e.g., device pairing) before communication can take place, and therefore fully automatic operation is not feasible.

B. Applications

While the Liberouter system forms a general network layer, it is the applications running on this infrastructure that provide value to the users. In general, a network composed of multiple disconnected Liberouters is unsuitable for real-time media applications such as streaming voice or video (although non real-time variants of such are possible, e.g., push-to-talk instead of Voice-over-IP [13]). Further, while unicast messaging applications are possible in such a network, we focus on broadcast style communications where each message has value for multiple users, and therefore leads to efficient use of the community network resources.

The SCAMPI platform provides a TCP-based API for native applications running on the same device as the router. This API can be used by, e.g., Android and Java SE applications to publish and receive messages in the Liberouter network. In addition, the platform exposes an HTML5 API though a lightweight shim, that maps a JavaScript API to the native TCP API and makes it available to HTML5 applications running in a web view component[4]. The different API layers are shown at the top of Figure 3.

These interfaces provide a *publish/subscribe* network abstraction. Unlike TCP/IP APIs that provide datagram or byte stream based abstractions, and Bundle Protocol router APIs that typically provide unstructured binary blob abstractions, the SCAMPI platform API uses a structured message abstraction of *SCAMPImessages*. These are semantically meaningful, self-contained application level messages, which get mapped into Bundle Protocol *bundles* for forwarding in the Liberouter network. The message contents are abstracted as a map of arbitrary key-value pairs. Using a common message content structure allows the lower layer to operate directly on the message contents, for example, to provide versioning and intelligent content merging. The message contents can also be described by attaching metadata. This metadata can be used by the SCAMPI routers to, for example, fulfil search queries.

Native applications can be designed to run in the mobile devices, for example, messaging or photo sharing applications. In addition, native Java SE applications can also run in the opportunistic routers, implementing application layer services that mobile client applications can use. Such services could be, for example, music jukebox applications that publish music files, or applications that pull information such as news from the Internet and publish it to mobile clients (assuming the Liberouter has Internet connectivity in addition to local connectivity).

In addition to the native API, we have created an HTML5 version of the API. Native applications on mobile devices tend to be tightly coupled to their particular platform (e.g., APIs, frameworks and even the languages used by Android, iOS and Windows Phone are completely different.) However, each of these platforms offers rich and mostly uniform support for the HTML5 standards, allowing for a truly cross-platform mobile application development. We provide a light-weight HTML5 shim that offers the messaging functionality through a JavaScript interface to HTML5 applications running inside a web view container. This allows developers to build cross-platform mobile applications using the familiar HTML5 frameworks and tools, while adding the networking functionality through the SCAMPI interface. However, this approach still requires the mobile client to have an implementation of the shim, and therefore these applications will not run on unmodified web browsers. Instead they can be run either through native wrappers around the web view component and shim, or though the HTML5 application distribution client also provided by the platform.

[4]This does not refer to mobile web browsers, but rather to a web view component that is wrapped in a native application; a popular approach to building mobile applications.

C. A Distributed App Store

In order to use the Liberouter applications, whether native or HTML5, the user must first get a copy of the application. Currently the most popular approach to distributing mobile applications are centralized app stores, such as Apple's App Store, Google's Play Store, Amazon Appstore and Windows Phone Store. These stores are curated to different extents, but in all cases the store operator has complete control over what applications are available. Since Liberouter neighborhood networks are fully distributed, it is not feasible to rely on centralized application distribution mechanisms. Therefore a distributed variant of the app store model is required.

As explained before, the Liberouter neighborhood network is based on publishing and spreading of self-contained messages. From the point of view of the network layer it makes no difference whether these messages contain text, images, videos, audio, or opaque binary data objects. This means that the very same mechanisms that applications use to communicate with each other can be used to publish and distribute the applications themselves.

By bundling the applications into messages, anyone can publish applications into the network, and anyone can install and run these applications. On the plus side, this means that no central curator or censor can make arbitrary decisions on what the users can or cannot install on their own devices. On the minus side, the user loses the ability to trust one central source to verify that the application is not malicious (although central curators have not stopped malware from being published in the app stores [11]). Therefore other mechanisms are required to build trust in the distributed applications. While detailed discussion of such methods is out of scope for this paper, we propose that there are a number of possible ways to achieve this: 1) the application developer can sign the application using the platform's native signing tools (e.g., Android APK signing), allowing users to validate the source of the applications, 2) distributed voting mechanisms can be used to flag known malware and known good applications, and 3) (de)centralized curators could still test and sign applications and users can choose which of those curators (if any) to trust.

HTML5-based Applications: Liberouter applications developed in HTML5 can be packaged into self-contained sets of HTML, JavaScript, and CSS files. These files contain the GUI (HTML and CSS) as well as data models and control logic (JavaScript libraries and custom JavaScript). A key benefit of the approach is that the applications contain only code that does not need to be compiled for any specific execution platform, and requires only a modern web browser. These packaged applications can then be sent as SCAMPImessages similarly to all other messages. This allows all SCAMPI mechanisms to be applied to the applications themselves, including metadata-based search and the security features such as message signing to verify the originator of the application. We have developed a *SCAMPI Market* distribution application as a Liberouter HTML5 application, which publishes, receives, validates and executes bundled HTML5 applications.

Native applications: Since the Liberouter network depends on native SCAMPI clients on the mobile devices to spread messages, there must be a way to bootstrap the system by distributing native applications to mobile devices that do not have any special distributed market software installed. We do this by using the captive web portal on the Liberouter devices to distribute native applications. Versions of these applications come pre-installed with the Liberouter distribution, however, they could also be later updated by publishing new versions into the Liberouter network. To support this, the Liberouters must have a SCAMPI update client that receives native application update messages, verifies their origin and then installs them into the captive portal. This mechanism could also be extended to automatically list user published applications on the captive portal site.

In addition to distributing native applications through the captive portal, the SCAMPI Market application can also be extended to support native application installation, using the same mechanisms as HTML5 application distribution described above.

IV. Validation

We have conducted a simulation study using the ONE simulator [17] to validate the feasibility of our design. Our simulation area is 4000 m $\times$ 3000 m section of central Helsinki[5], with simulation duration of one week. On this area, user nodes move according to the ONE map based random movement model with speeds uniformly drawn from $[0.5\text{ ms}^{-1}, 1.5\text{ ms}^{-1}]$, while being constrained to move only on the roads defined by the Helsinki map. Further, we randomly place 20 Liberouter nodes on the map. Each node is capable of Wi-Fi communications with 20 m range and 2 MBs^{-1} transmission speed. User nodes can only communicate with the Liberouter nodes and cannot communicated directly with each other. The Wi-Fi parameters reflect a scenario where the Liberouters are low performance devices placed inside buildings in an urban settings, and therefore realistically will not cover the 100 m range typically assumed for Wi-Fi in the literature. Each user node has 100 MB storage dedicated for a message buffer, while Liberouter nodes have 2 GB, again reflecting a scenario where users are mobile devices and Liberouters low cost dedicated devices. We model a photo sharing type application, where each user publishes 1 MB messages with a period uniformly drawn from $(0\text{ min}, 30\text{ min}]$ with a lifetime of 6 hours. These messages are spread epidemically in the network and received by other users through the Liberouters, and are dropped oldest-first when the storage space is exhausted.

Figure 4 shows the per-user goodput achievable for our system. Goodput is defined as the total volume of unique messages that a user receives over the simulated week. In the case of a photo sharing application, this corresponds to the number of unique photos that the user would receive from the Liberouter network per week. We can see in the figure,

[5]The map distributed with the ONE simulator.

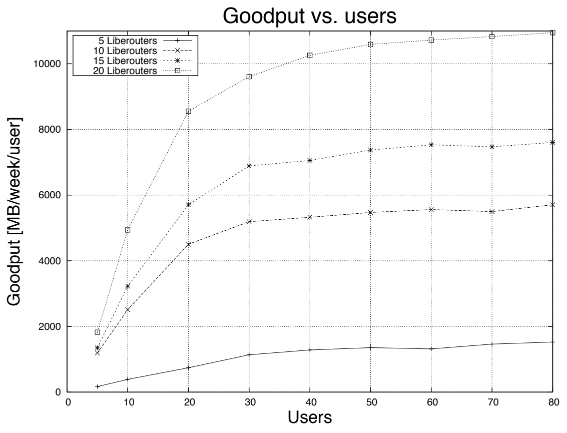


Fig. 4. Goodput of the Liberouter network.

that the goodput first quickly raises as the number of users generating and distributing messages increases. However, the goodput reaches a steady state as the network gets saturated with messages. At this steady state more users can enter the system and they will receive the same goodput as existing users. The amount of data that the system can deliver to a single user is gigabytes per week. To put this into perspective, in the case of 80 users and 20 Liberouters, each user would achieve a monthly goodput of 47 GB, which is an order of magnitude higher than typical data usage caps on cellular networks.

Even though the goodput reaches a steady state as the network gets saturated, the congestion causes published messages to get dropped from the network earlier, which limits the reach of the individual messages. This effect can be seen in Figure 5, which shown the fraction of nodes reached on average by a published message. In an uncongested network the fraction of nodes reached is dependent on the density of Liberouters and the mobility of the users. In the case of 20 Liberouters, the messages will reach well over half the users in the uncongested case. As the number of users increases (and thus the offered load), causing congestion, the reach of the message falls, with only 20% of nodes being reached in the 20 Liberouter case with heavy congestion. However, this effect does not impact the goodput since *any* received unique message provides value to the user. Furthermore, as congestion causes messages to get dropped sooner, the messages that users receive are likely to have been published geographically closer and therefore being more valuable to the user, e.g., in the case of photo sharing where the users would receive photos taken closer to their physical location.

The congestion itself can be seen in Figure 6, which shows the fraction of all message drops in the network that are caused by congestion; the remaining message drops are caused by the messages lifetimes expiring. The congestion is shown for the user devices, the buffer sizes of the Liberouters are sufficiently large that no messages are dropped by them due to congestion (this assumption is likely to hold in real deployments since Liberouters are dedicated devices, while the users' devices are mobile phones where only a limited amount of the resources will be dedicated to the Liberouter system's use). We can see that after reaching the steady state,

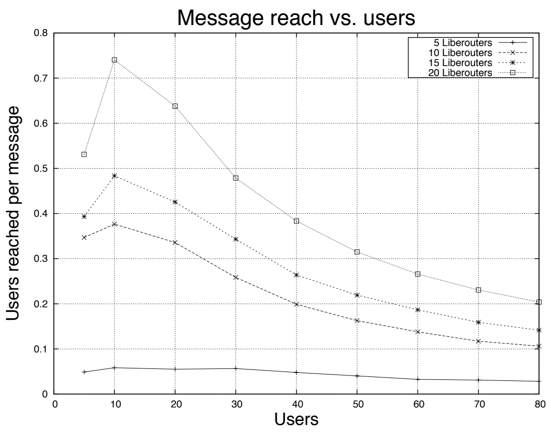


Fig. 5. Fraction of clients reached by each message.

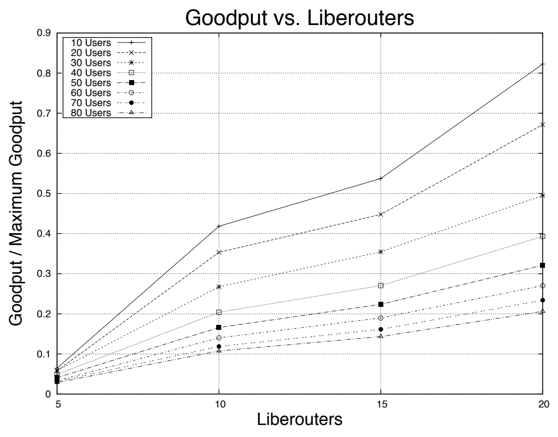


Fig. 7. Fraction of maximum goodput achieved.

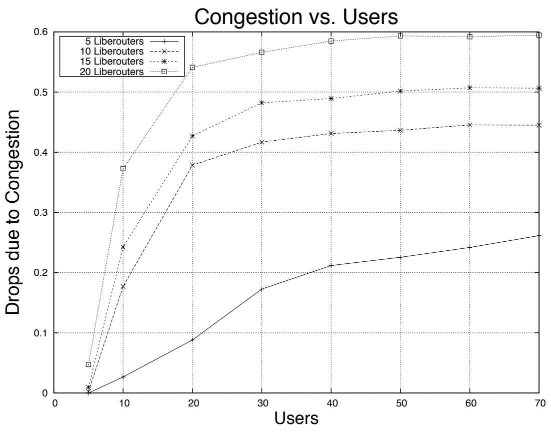


Fig. 6. Fraction of message drops in the user devices due to congestion.

the fraction of messages dropped due to congestion does not increase. However, adding Liberouters to the network has the effect of increasing congestion at the users' devices. This is caused by the increased encounters between the users and the Liberouters causing more messages to be transmitted to the users' devices.

Two main approaches can be taken to scale the network: 1) the number of Liberouters in the network can be increased, or 2) the resources available in the network can be increased. As can already be seen from Figure 4, adding more Liberouters to the network increases the goodput to all users. Figure 7 demonstrates this explicitly by showing the achieved goodput as a fraction of the maximum theoretical goodput[6] when the number of Liberouters is increased. The observed impact on goodput is higher for cases with smaller numbers of nodes. It is due to those cases being closer to the transition range between the congested and uncongested cases, where adding more capacity will directly increase goodput. In contrast, when the network is significantly congested, as is the case with 80 users, adding more capacity will not reduce congestion noticeably and therefore has much less impact on the goodput.

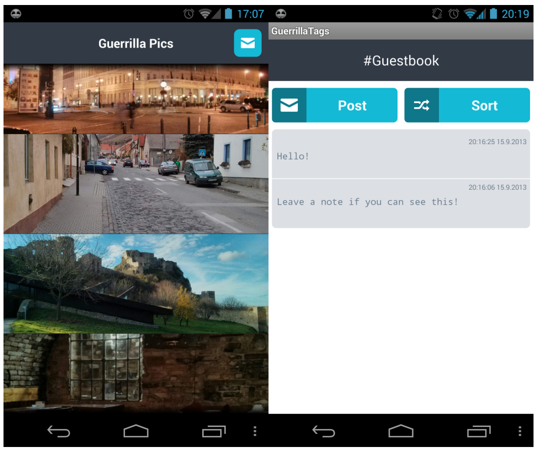
The second approach to scaling is to increase the resources in the network. There are two main resources in the devices: buffer space and transmission range and capacity. As we already showed, in our scenario the Liberouters have enough buffer space to store all received messages. On the other hand, the users have more limited buffers that get congested as the number of users in the network increases. However, our simulations show that there is no impact on goodput nor message reach when the users' buffer is varied from 100 MB to 500 MB, even though at 500 MB the buffers are no longer congested. This indicates that the system is not limited by the buffering resources when it comes to scaling to a larger number of users, although they are likely to be important when scaling to higher message generation rate. This makes intuitive sense, since in the steady state the Liberouters are likely to contain more new messages than can be transmitted to a user during the brief contact, which means that regardless of buffer sizes the user will always get new content when passing by a Liberouter.

This leaves us with Wi-Fi transmission range and capacity as the other option to scale. Since it is typically easier to increase the Wi-Fi range (increased transmitter power and higher gain antenna), we study the impact of increasing the Wi-Fi range. We simulated increasing Wi-Fi range between 20 m and 100 m and found that the goodput and message reach increased significantly, almost reaching maximum throughput for all the studied cases at 100 m range. At the same time congestion in the user devices grew significantly. This makes sense since a longer radio range means that the users are within Liberouter range longer and therefore will receive a larger number of new messages.

The key observation that can be made about the behavior of the Liberouter system is that adding more users to the system does not lead to a congestion collapse[7]. Instead of the per user goodput collapsing under congestion, the reach of the messages drops while goodput remains stable. In the case of photo sharing, this means that each user will get the same amount of new photos, but fewer users will see the same photo. In other words, the system design leads to

[6]Goodput in the case every published message is received by every node.

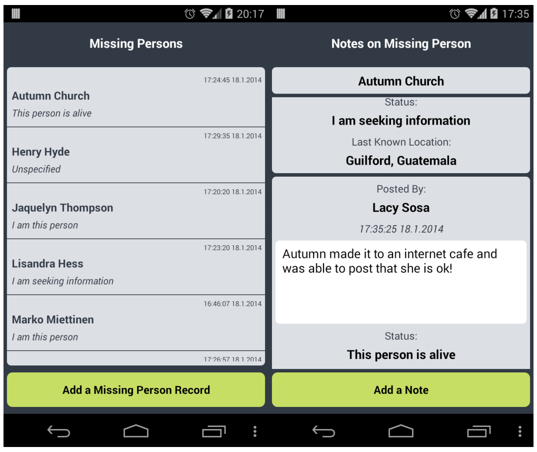
[7]At the system level. A single Liberouter node will collapse given a large enough number of concurrent clients, which is likely to occur in mass events, such as stadiums or concerts. There are ways to mitigate this, but they are outside the scope of this paper.



20:16:25 15.9.2013
Hello!

20:16:06 15.9.2013
Leave a note if you can see this!

Fig. 8. Guerrilla Pics (left) and Tags (right) applications.



17:24:45 18.1.2014
Autumn Church
This person is alive

17:29:35 18.1.2014
Henry Hyde
Unspecified

17:20:20 18.1.2014
Jaquelyn Thompson
I am this person

17:23:20 18.1.2014
Lisandra Hess
I am seeking information

16:46:07 18.1.2014
Marko Miettinen
I am this person

17:26:57 18.1.2014

Autumn Church
Status:
I am seeking information
Last Known Location:
Guilford, Guatemala

Posted By:
Lacy Sosa
17:35:25 18.1.2014
Autumn made it to an internet cafe and was able to post that she is ok!
Status:
This person is alive

Fig. 9. User interface of the People Finder application.

highly diverse and localized set of content under load. This means that the Liberouter network is well suited for broadcast type applications where the aim is to let users publish and receive a diverse set of highly localized content. Conversely, the network is not well suited for applications that require a wide distribution of the same piece of content to a large number of users, or to a specific receiver, even under load.

V. Implementation

We have implemented a Liberouter device using a Raspberry Pi with a USB Wi-Fi for connectivity and USB flash drive for storage. The distribution is built on the standard Raspbian Linux distribution, configured to act as a Wi-Fi access point with a captive portal and the SCAMPI opportunistic router. We use Android devices as the user devices, for which we provide a SCAMPI router application, *Guerrilla Tags* messaging application, *Guerrilla Pics* photo sharing application, and *People Finder* disaster recovery application.

We have been running small scale Liberouter deployments at a number different locations in Helsinki, Stockholm, and Cambridge, with some continuously operating for half a year and using Android phones to carry messages back and forth. Larger scale deployments of some tens of nodes are currently under development. However, anyone can build and deploy Liberouters by using the publicly available distribution[8].

Liberouters appear to users as open Wi-Fi access points with the SSID ”LIBEROUTER”. Users connecting to the network will see a captive portal web site that lets the users download the native Android (2.3.3 or later) applications needed for communicating in the Liberouter network. This includes the Android SCAMPI router (LibeRouter), which provides opportunistic communication services to the other applications. After the user installs and launches the LibeRouter application, it will connect to the SCAMPI router running in the Liberouter access point. It remains running in the background (until stopped by the user), which, when combined with Android automatically connecting to any LIBEROUTER Wi-Fi networks that it finds, ensures that the device will swap messages with any Liberouter within communication range.

[8]Available at: http://www.ict-scampi.eu/results/scampi-liberouter/

The local applications, Guerrilla Pics, Guerrilla Tags and People Finder can be used at any time, regardless of whether the device is connected to a Liberouter or whether the local LibeRouter instance is running. Guerrilla Pics and Guerrilla Tags (show in Figure 8) let the users publish photos and short messages respectively. The applications have internal databases with the contents that the user has published and received. When the local LibeRouter instance is running, the applications fill their internal databases with messages received from the Liberouter network and publish any messages created by the local user.

The third application, People Finder (Figure 9), is an implementation of the People Finder Interchange Format (PFIF) [9] standard for exchanging missing persons records following disasters. PFIF is designed for enabling large, centralized missing persons database operators to synchronize their records, particularly during major disasters when significant number of people are seeking and providing information about their friends and loved ones. People Finder application allows people to publish and collect missing persons records using the Liberouter network, without needing Internet connectivity in order to connect to the large centralized databases. This is especially important, since major disasters often disable the very infrastructure that is required for the Internet based centralized solutions.

We are currently developing various aspects of the Liberouter system. First, we are continuing to improve the underlying SCAMPI platform with more features, especially for more efficient routing and content distribution. Second, we are working on providing content to non-Android users by integrating content from the local SCAMPI router caches into the captive portal. This would allow users to view the content in the network without installing the native applications into their devices. Third, we are preparing a release of the developer frameworks and APIs through the Liberouters, so that everything needed for building new applications for the network can be downloaded from any Liberouter. Finally, we are building a native application distribution system so that native applications can be distributed through other means than the captive portal web site.

VI. Discussion and Conclusion

We have presented our Liberouter framework and a system design to create and dynamically expand autonomous neighborhood networks using stationary and mobile devices. The communication, storage, and application distribution features *complement* the Internet infrastructure and enable localized information sharing that avoids central control or restrictions.

Sharing content and particularly applications in autonomous environments raises security issues: as noted above, while app store control may be seen negative, it serves as a filter for malicious software; and while cloud and server infrastructure may restrict content exchange and make content observable to third parties, it facilitates filtering spam and malware. Any solution eliminating such central check points should offer alternative answers to the above issues.

Since we do not target operation in an empty space, we can leverage existing mechanisms and make use of the Internet if needed. For example, the bundle protocol supports signing and encrypting messages and we can implement similar security mechanisms inside our messages (e.g., using S/MIME) to provide authentication and access control. As we are targeting neighborhood networking, we may assume that (groups of) the involved users will be aware of each other so that individual signatures actually bear a meaning and users can decide when to trust a piece of content or an application before viewing or installing. To simplify introduction and sharing of personal attributes (e.g., certificates), we may also leverage trust between friends to identify friends of friends and link such relationships from online social networks such as Facebook to such personal attributes via servers controlled in the neighborhood, as done in PeerShare for MAC addresses of mobile devices [19].

Protecting neighborhood networks against DoS attacks is no different from other mesh or opportunistic networks. The isolation of the local data path from the Internet has the positive side effect that injecting traffic into the network would require physical presence, thus making attacks potentially more costly and *distributed* DoS attacks even costlier to carry out. Moreover, message authentication may help allocating storage and communication resources.

We note, however, that addressing these aspects becomes substantially harder for systems targeting censorship-resistant communications: access to (sufficiently trustworthy) infrastructure services may no longer exist, message and authentication may need to follow quite different paths, and the cost of being subverted by adversary applications may become immense so that false negatives are no longer tolerable. This is an area for future work.

Acknowledgements

This project has received funding from the European Community's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 611366 (PRECIOUS) and from EIT ICT Labs.

Special thanks to the Liberouter team in Aalto Design Factory Product Development Project 2012: Bryn Haffey, Mikko Heiskanen, Xueyun Li, Zijia Liu, Jessie Mislavsky, Chirapriya Mondal, Lasse Rantanen, Markus Selensky, Shengjun Shi, Yue Shen, and Qiuyang Tang.

References

[1] D. Aguayo, J. Bicket, Biswas S, G. Judd, and R. Morris. Link-level Measurements from an 802.11b Mesh Network. In *Proceeding of ACM SIGCOMM*, August 2004.

[2] I.F. Akyildiz and Xudong Wang. A survey on wireless mesh networks. *IEEE Comm. Magazine*, 2005.

[3] S. Aryan, H. Aryan, and J. A. Halderman. Internet Censorship in Iran: A First Look. In *USENIX FOCI Workshop*, 2013.

[4] N. Banerjee, M.D. Corner, and B.N. Levine. An energy-efficient architecture for dtn throwboxes. In *INFOCOM 2007. IEEE*, May 2007.

[5] A. Bujari, C.E. Palazzi, D. Maggiorini, C. Quadri, and G.P. Rossi. A solution for mobile dtn in a real urban scenario. In *Wireless Communications and Networking Conference Workshops (WCNCW), IEEE*, April 2012.

[6] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *Workshop on Privacy Enhancing Technologies (PET)*, 2006.

[7] F. De Pellegrini, I. Carreras, D. Miorandi, I. Chlamtac, and C. Moiso. R-p2p: A data centric dtn middleware with interconnected throwboxes. In *Proc. of Autonomics '08*. ICST, 2008.

[8] Mike Demmer, Jörg Ott, and Simon Perreault. Delay Tolerant Networking TCP Convergence Layer Protocol. Internet Draft draft-irtf-dtnrg-tcp-clayer-06.txt, Work in Progress, May 2013.

[9] Ka-Ping Yee (ed.). PFIF 1.4 Specification. http://zesty.ca/pfif/1.4/, May 2012.

[10] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proc. of ACM SIGCOMM*, 2003.

[11] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proc. of ACM SPSM workshop*, Chicago, 2011.

[12] S. Hasan, Y. Ben-David, G. Fanti, E. Brewer, and S. Shenker. Building Dissent Networks: Towards Effective Countermeasures against Large-Scale Communications Blackouts. In *USENIX FOCI workshop*, 2013.

[13] Md. Tarikul Islam. DT-Talkie: Push-to-talk in Challenged Networks. In *Demo at ACM MobiCom*, 2008.

[14] Teemu Kärkkäinen, Mikko Pitkänen, Paul Houghton, and Jörg Ott. Scampi application platform. In *Proc. ACM CHANTS workshop*, 2012.

[15] Teemu Kärkkäinen, Mikko Pitkänen, and Jörg Ott. Enabling Ad-hoc-Style Communication in Public WLAN Hot-Spots. In *Proc. ACM CHANTS workshop*, 2012.

[16] R. Karrer, A. Sabharwal, and E. Knightly. Enabling large-scale wireless broadband: the case for taps. *SIGCOMM Comput. Commun. Rev.*, 34(1):27–32, January 2004.

[17] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *Proc. of SIMUTools '09*, 2009.

[18] Zubair Nabi. The Anatomy of Web Censorship in Pakistan. In *USENIX FOCI workshop*, 2013.

[19] N. Asokan Marcin Nagy and Jörg Ott. PeerShare: PeerShare: A System Secure Distribution of Sensitive Data Among Social Contacts. In *Proc. of NordSec*, October 2013.

[20] L. Pelusi, Andrea Passarella, and Marco Conti. Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks. *IEEE Comm. Magazine*, November 2006.

[21] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A Networking Architecture Designed Around Mobile Users. In *Proc. of WONS*, 2006.

[22] Keith Scott and Scott Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[23] G. Sollazzo, M. Musolesi, and C. Mascolo. Taco-dtn: a time-aware content-based dissemination system for delay tolerant networks. In *Proc. of MobiOpp '07*, 2007.

[24] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. WiFi-Opp: Ad-Hoc-less Opportunistic Networking. In *Proc. ACM CHANTS workshop*, 2011.

[25] Hanno Wirtz, Tobias Heer, Robert Backhaus, and Klaus Wehrle. Establishing Mobile Ad-Hoc Networks in 802.11 Infrastructure Mode. In *Proc. ACM CHANTS*, 2011.

[26] W. Zhao, Y. Chen, M. Ammar, M. Corner, B. Levine, and E. Zegura. Capacity enhancement using throwboxes in dtns. In *IEEE MASS*, 2006.