Enabling Ad-hoc-Style Communication in Public WLAN Hot-Spots

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

ABSTRACT

Opportunistic networking between mobile devices relies on the capabilities of those devices to establish ad-hoc communication among each other. While the two dominant wireless interface technologies, IEEE 802.11 wireless LAN and Bluetooth, offer such capabilities in theory, limitations of the protocol specification, chipsets, and operating systems in mobile devices render those features largely unusable in practice. Researchers have recognized these shortcomings and devised mechanisms in which mobile devices act as WLAN access points to simulate WLAN infrastructure-based operation. In this paper, we complement these approaches by instrumenting commercial WLAN APs that do not employ L2 security to serve as link layer packet relays without requiring the mobile nodes to authenticate with the WLAN hot-spot. We present different mechanisms for peer discovery, evaluate their feasibility for a set of commercial hot-spots, and discuss operational considerations for fair use of commercial access points.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols

General Terms

Algorithms, Design, Experimentation

Keywords

Delay-tolerant Networking; DTN; WLAN; Hot-spot; Ad-hoc Networking

1. INTRODUCTION

Delay-tolerant networking between mobile nodes exploiting opportunistic contacts between the devices as their users move around has been a research topic for many years, yielding diverse communication and software architectures [23, 4, 21, 14], a myriad of routing protocols, and a diverse set of applications [1, 20, 7, 6, 13, 8, 9], to name just a few.

CHANTS'12, August 22, 2012, Istanbul, Turkey.

Yet, none of this has seen true deployment in practice. While there may be many "soft" reasons such as lacking incentives for altruistic cooperation between users (fearing for their device battery life) and one could argue about legal implications [12], we are still missing some of the basic technical enablers. Mobile devices are equipped with diverse wireless interfaces that would *in principle* be capable of supporting opportunistic communications. But, *in practice*, mobile device vendors don't really seem to appreciate those technologies and leave their capabilities often crippled by their hardware, operating system, and/or API designs, even though we are seeing some modest improvements (see section 2).

WLAN appears as the most promising technology due to its data rate and reach but it is de-facto bound to operation in infrastructure mode, in which access points support efficient network discovery and node autoconfiguration. Exploiting WLAN nevertheless for mobile opportunistic communication leaves two complementary routes: 1) turning mobile nodes into (ephemeral) access points and 2) utilizing existing nearby access points. Some rather static variant of (1) is inherently supported by many mobile phones and tablets that can act as a manually configured WLAN AP to share cellular Internet connectivity with other devices. Recent research has gone further and suggested that mobile devices should dynamically become APs to establish ad-hoc connectivity (see below). In addition, (2) has been discussed mostly for open access points but not yet addressed commercial hot-spots.

In this paper, we take a stab at exploiting the vast number of available commercial hot-spots that use web-based authentication portals (see section 3). In particular, we believe that two key issues need to be tackled:

1. We devise a number of complementary mechanisms for peer discovery that do *not* require authentication with the respective hot-spots in section 4 and evaluate their effectiveness in the real world in section 5.

2. We discuss fairness issues for using commercial WLAN APs for local communication and suggest using a less-than-best-effort transport mechanism to minimize the impact of ad-hoc communication on the commercial operation in section 6; this is important so that vendors and hot-spot providers do not see a need to close down the present communication opportunities.

2. BACKGROUND AND RELATED WORK

Various studies have explored (the limitations of) ad-hoc communication for one-hop DTN forwarding in practice. Researchers found limitations for Bluetooth pairing in that pairing takes time and quite a few of the pairing attempts do not succeed [19]. Moreover, basic Bluetooth is not very energy-efficient in terms of en-

ergy per bit transmitted [18] and mobile operating systems such as Android did not provide support for establishing ad-hoc networks. These two limitations are about to change with Bluetooth Low Energy[1] and recent Android extensions[2]. At least three issues remain, however: the short communication range that limits contacts to less than a minute at walking speeds; the low bitrate (compared to WLAN); and the perceived insecurity of Bluetooth that makes people turn off their mobile's Bluetooth interfaces to begin with. Therefore, we put our emphasis on WLAN.

The IEEE 802.11 WLAN specifications define an *ad-hoc mode* that allows hosts (precisely: *station adapters*) to communicate directly with one another, i.e., without the need for an *access point (AP)* as is the case in the commonly used *infrastructure mode*. However, ad-hoc mode has suffered from being specified only for the link layer, lacking conventions on how to run higher layer protocols such as IP on top. In effect, ad-hoc mode is not really supported by most operating systems. While a recent specification by the WiFi Alliance, *WiFi Direct* [5], aims at addressing the shortcomings and provides a complete solution for ad-hoc interoperability between devices, mobile device operating systems and APIs have yet to support this mode of operation.

Recent research [26, 27] suggests turning mobile nodes opportunistically into access points to address the lack of ad-hoc networking support. The authors of *MA-Fi* [27] focus on mobility considerations for the APs and maintaining multihop connectivity. In contrast, *WiFi-Opp* [26] aims at opportunistic communications for which the authors suggest several modes of operation: Mobile nodes scan the environment for usable access points and associate with those for some time, optionally taking turns between APs. If no AP can be found, a mobile becomes an AP itself for some time to facilitate communication for other nodes. This also works for *fixed APs*, i.e., access points not created by other mobiles but run as hot-spots. In their paper, the authors investigate the impact of various parameters (e.g., beacon time, scanning interval, connection time) by means of simulations, but they do not discuss how to exploit the fixed APs in practice.

We extend WiFi-Opp in exactly this point: we investigate suitable mechanisms to instrument potentially restrictive commercial hot-spots for ad-hoc communication. We use the APs only to assist local communication and therefore do not need to perform authentication, e.g., in an automated fashion as in [16] or as implemented lately in various WLAN connectivity assistants. Not being authenticated may impose diverse constraints on communication between two mobile nodes, depending on the hot-spot configuration, for which we develop mechanisms to systematically probe each AP. We validate such mechanisms in real-world hot-spots.

3. COMMERCIAL WLAN HOT-SPOTS

We define the essence of a commercial WLAN hotspot as follows: A combination of a WLAN access point and a gateway (GW) as shown in Figure 1—both of which may or may not be integrated in the same physical box—that serve the mobile devices of typically stationary users to obtain Internet access.[3] The AP function provides wireless connectivity, including address autoconfiguration (using DHCP); the WLAN operates without any cryptographic protection so that any host can associate with the AP.

[1]Bluetooth Core Version 4
https://www.bluetooth.org/Technical/Specifications/adopted.htm
[2]http://developer.android.com/sdk/android-2.3.3.html#bluetooth
[3]Multiple APs might be connected via Ethernet or other L2 technologies to form an extended service set and support handover of mobile nodes using the same SSID. APs may also support multiple SSIDs and/or serve multiple service providers.

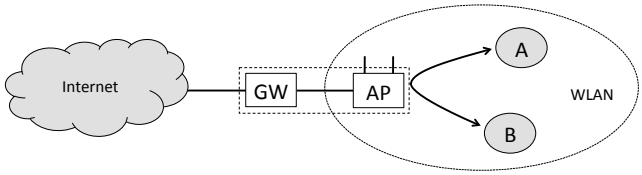

Figure 1: Commercial Hotspot Setup

The GW function is responsible for filtering traffic to the Internet so that only authenticated clients gain access. The latter is achieved using a *captive portal* that answers DNS requests on behalf of the mobile nodes and/or terminates and redirects HTTP connections targeted to a node on the Internet to a local HTTPS authentication server, using 30x or 511 [11] HTTP response codes. The authentication server then prompts the user to enter her credentials. All other traffic is simply dropped until the user has been authenticated (using a local database or some backend service).

While the GW filters most traffic to the Internet (in some cases, DNS may be the only exception), communication between different hosts attached to the AP prior to authentication is possible to different extents depending on the architecture.[4] At the very least, broadcasts of the address resolution protocol (ARP) must be possible because otherwise the duplicate address detection required by DHCP would not work. In the following section, we discuss the different degrees of connectivity and explore how to utilize those for peer discovery.

4. PEER AND SERVICE DISCOVERY

Assuming that two nodes are associated to the same WLAN AP as nodes A and B in Figure 1, they need to become aware of each other and the respective capabilities to support opportunistic communication and they need to determine if and how they can communicate with one another.

Ideally, peer and service discovery occur in a single step: A straightforward approach uses **multicast discovery** as is defined for mDNS [2] or UPnP [3] for generic service discovery. For example, a DTN node would simply respond to a query for a certain DNS service record. But also per-protocol discovery mechanisms could be applied, using protocol-specific multicast addresses, port numbers, and message formats for discovery. Nodes can send beacons indicating their available services or send/react to inquiries for services.

Preliminary experiments carried out by the authors have shown, however, that multicasting between mobile nodes is often blocked, at least prior to authentication.[5] A simple recipe to circumvent this restriction would be reverting to **IP broadcast discovery** but IP layer broadcasting may be disabled as well. For link layer broadcast discovery, port numbers for demultiplexing are no longer available so that different conventions for service discovery using L2 frames would be needed (and not be blocked).

As a more promising fallback, we suggest to decouple the steps of peer and service discovery and utilize, in a first step, ARP requests and replies to find out about other *potential* peers. ARP messages are broadcast as L2 frames and, as noted above, they cannot be blocked since otherwise DHCP would stop working properly.

[4]Note that, even after authentication, some hotspot architectures limit direct peer-to-peer communication between hosts.
[5]And it may be so even after authentication since multicasting takes up an undue share of channel capacity.

We define two parts of **ARP-based discovery**: 1) Each node passively listens to ARP requests and replies (it does so anyway) and populates a table of known peers with this information. 2) Each node regularly broadcasts an ARP reply packet (using some "beacon interval") to make others aware of its existence. This will ensure that new nodes associating with the AP will be made known and learn about nodes associated with the AP.

ARP does not provide information beyond a peer's IP address so that a secondary step for service discovery is needed—which then blends in seamlessly with determining if direct communication between the two nodes is possible.[6]

To check for ad-hoc connectivity, a node A simply attempts to reach its peer B at a known port number using TCP or UDP. This could be done using an explicit service discovery step by means of point-to-point communication for one of the service discovery protocols mentioned above to exchange capabilities. Or, if a port number is reserved for the opportunistic communication protocol (such as port 4556 for the TCP and UDP convergence layers of the DTN bundle protocol [24]), node A attempts establishing a connection to the sought service at B right away. If the connection succeeds, the two nodes can use it to exchange service-specific information, e.g., buffer space, routing tables or contact history, and to (optionally) authenticate each other, before they begin exchanging messages. If it fails, A knows that B either does not support the service or that the AP does not let the traffic pass—which are equivalent in their outcome.

Finally, if (mobile) operating system constraints do not allow sending or receiving ARP packets (an operation that requires raw sockets so that access is usually restricted), a node might perform **unicast discovery**. From its DHCP lease, it learns also the netmask for the subnetwork and thus the address space assigned to this subnet, typically in the order of /24. With such a limited address space, a node can then simply scan all addresses at a reasonable rate to find available peers. However, this should be used as last resort as it is most expensive in terms of energy consumption (and may even conflict with the terms of usage of the hotspot).

If connectivity checks or peer discovery fail on a given port number, one option could be trying other ports and possibly tunnelling a protocol inside another (such as HTTP). However, such an approach is unlikely to succeed in the long run as it would just cause a similar arms race as observed between firewall vendors implementing deep packet inspection and application protocol designers attempting to make their traffic and messages look like HTTP. Nevertheless, in our subsequent evaluation we also investigate if hotspot APs allow connection on certain port numbers between hosts to pass while blocking others.

Table 1 summarizes the above collection of peer discovery mechanisms. Connectivity verification is always done by attempting to establish a TCP connection or exchanging UDP datagrams between the peers.

Note that nodes connected to different WLAN APs forming an extended service set may experience different connectivity, but the mechanisms described above will discover peer-to-peer connectivity when it exists.

Mechanism	Peer discovery	Service discovery
Multicast	request/response beaconing	included included
Broadcast	request/response beaconing	separate separate
ARP-based	passive listening unsolicited response	separate separate
Unicast	request/response one-by-one scanning	partly implied separate

Table 1: Overview of available discovery mechanisms

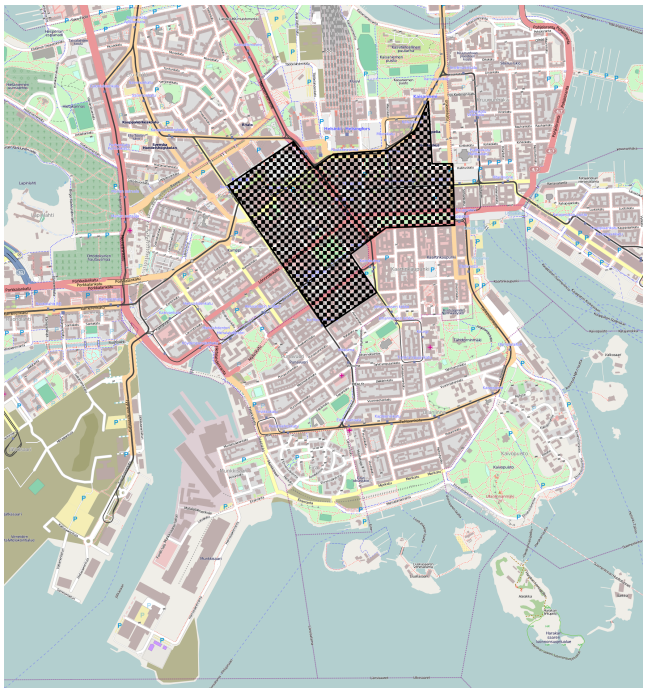

Figure 2: Approximate area of AP scans in Helsinki. © OpenStreetMap contributors, CC BY-SA

5. EVALUATION

To evaluate peer-to-peer connectivity between nodes that are associated to the same open AP, we built a testing framework to be run on two mobile devices. One of the devices acts as a test master and the other as a slave. The master scans available open APs, associates to one and then instructs the slave to connect to the same AP. After both devices are associated and have acquired IP addresses through DHCP, the master initiates a suite of connectivity tests. Control traffic, e.g., commands and address information, between the master and the slave is transmitted over Bluetooth.

The connectivity checks include *IPv4/IPv6 multicast* with various scopes, *IPv4/IPv6 broadcast* to both network and subnet broadcast addresses, *mDNS/DNS-SD* (Bonjour), *ARP* request with TCP echo, *direct TCP* to various ports (with traffic appearing to be HTTP).

We ran the test suite for a random sample of 50 open access points. The access points were picked from an approximately 1 km^2 area in central Helsinki (as shown in Figure 2, which has a high amount of pedestrian traffic and a high density of hotels, restaurants and businesses that provide open access points. Most of the measurements were performed from the street with a few measurements inside shopping centres, all publicly accessible to pedestrians. Only traffic directly between the two devices was tested and no attempt

[6]One could apply proprietary extensions to ARP and, e.g., pad the ARP message with additional information, but such an approach could easily clash with similar extensions devised by others so that we refrain from pursuing this idea further.

Mechanism	Success
ARP + TCP echo	30%
mDNS + DNS-SD	30%
IPv4 local broadcast	30%
IPv4 subnet broadcast	32%
IPv4 multicast - local subnet	30%
IPv4 multicast - Internetwork Control Block	28%
IPv4 multicast - AD-HOC block	30%
IPv4 multicast - Administrative scope	28%
IPv6 multicast - link-local	40%
IPv6 multicast - IPv4 local scope	40%
IPv6 multicast - administrative-local	40%
IPv6 multicast - site-local	40%
IPv6 multicast - organization-local	40%
IPv6 multicast - global	38%
Captive portal - port 80	28%
Captive portal - port 443	4%
Captive portal - port 4556	2%
Captive portal - port 8080	12%
TCP - port 8080	30%
TCP - port 4556	26%
Any communication	48%
IPND + TCPCL	20%
Completely open	16%

Table 2: Test results for 50 open access points.

was made to connect to the public Internet or interact in any way with captive portals. The results are shown in Table 2.

The *IPv4/IPv6 multicast* tests worked by having both test devices subscribe to the same multicast address, after which the slave sent a beacon to the multicast address every second. The master listened for the beacons for a few minutes (to account for possible delays to multicast delivery due to 802.11 power-saving mode) and recorded which beacons it received. The *IPv4/IPv6 broadcast* tests worked similarly (except for the multicast group membership).

The tests show that IPv4 multicast and broadcast works in roughly 30% of the APs. The success rate of mDNS, which is based on IPv4 multicast delivery, is also 30%. However, in many APs that block IPv4 multicast, IPv6 multicast packets will still go through. IPv6 multicast works in 40% of the APs (for all scopes except global), giving it the highest success rate of any single test. This is likely to be due to technical issues (e.g., filtering software not supporting IPv6 or the administrators failing to configure their systems properly) rather than a policy decision since there is no qualitative difference between IPv4 and IPv6 multicast traffic. Furthermore, among the tested APs we found almost every possible permutation for multicast and broadcast filtering rules, including APs that filtered mDNS but allowed all other tested IPv4 multicast traffic (and vice versa), APs that filtered all IPv4 multicast traffic and some but not all IPv6 multicast scopes and APs that filtered local network broadcast (255.255.255.255) but not subnet broadcast. This indicates that any discovery mechanisms cannot rely on a single multicast or broadcast address, but must use multiple in order to maximize the success probability.

While IP multicast is typically used for peer discovery, *direct TCP* is used for actual data transfer. We tested direct TCP communication between the nodes for ports 8080 and 4556 (TCPCL). In order to detect captive portals we further tested opening a TCP connection to ports 80 and 443 even though the test devices were not listening those ports. All TCP traffic was made to look like HTTP requests with the test master sending out an HTTP GET request and the slave sending an HTTP 200 OK response.

The tests showed that 30% of the APs allowed direct TCP connections between peers on port 8080 and 26% on port 4556. The APs employed various strategies for blocking the TCP connections. Some simply dropped the packets leading to the handshake timing out, some replied with ICMP Destination Unreachable, while yet others allowed the initial TCP handshake to succeed but did not allow any user data. In a few occasions TCP connections failed because the AP assigned IP addresses from different subnets to the two test devices.

The TCP test suite was capable of detecting *captive portals* by examining the headers in the HTTP responses. Even though the tests did not include any attempt to connect to the public Internet, only to the other local device, we still found that 28% of the APs had a captive portal intercepting HTTP requests to the port 80. Ports other than 80 had significantly lower interception rates with 12% of traffic to port 8080 and 4% to port 443 being intercepted. Only one AP had a captive portal intercepting traffic to a non-HTTP port (4556), likely through deep packet inspection to identify HTTP traffic. We observed captive portals using the HTTP status codes 302, 307 and 404, but not the 511 code that is supposed to be used to indicate the presence of a captive portal [11].

Out of the tested APs, 16% appeared to allow completely open communication between local devices without any blocking or filtering, enabling any kind of peer-to-peer communication to take place. Roughly half of these were APs operated by businesses for their customers, while the other half seemed based on their names to be privately owned APs that had been accidentally left open.

Typical peer-to-peer communication requires two elements, discovery and transport. We found that such combined interaction will succeed for 20% of the APs for the DTN protocols (IP Neighbor Discovery + TCP CL). Of the failed attempts, 80% failed because both discovery and transport failed, 12.5% because transport failed, and 7.5% because discovery failed.

Towards Practical Application

While the previous results seemingly mean that peer-to-peer communication will fail in 80% of the open APs, we found out that when combining all the methods of data delivery between the devices we could find at least one mechanism to transfer packets between the two devices 48% for of the time. From the results in Table 2 we can see that just by directly scanning for open port 4556 instead of relying on multicast based discovery, we can increase the success rates from 20% to 26%. Using a known HTTP port 8080 we can further increase this to 30%. In order to reach the full 48% success rate, multicast/broadcast based transport is needed.

These results show that no single mechanism is clearly superior, nor reliable enough to depend on solely for peer discovery and contact establishment. Furthermore, different mechanisms place different strain on the network. For example, broadcast/multicast packets are processed by multiple nodes (and may be rate limited by the AP) and therefore can be used for peer discovery, but should be avoided for message transmission. Therefore a protocol is needed for systematically applying the various mechanisms for peer discovery and contact establishment in a way that guarantees the best possible chance of success while avoiding unnecessary strain on the network. While the detailed design of such protocol is outside of scope for this paper, we can postulate some general rules for efficient peer discovery and contact establishment: 1) the nodes should listen for transport connection on multiple ports, including well known HTTP ports (such as 8080) if possible, 2) standard mDNS/DNS-SD discovery should be attempted first, 3) cus-

tom multicast/broadcast based peer discovery should be tried with different scopes, including IPv6 multicast, 4) direct port scanning can be attempted if all other attempts failed.

Another practical consideration is ensuring that nodes connect to the same AP when multiple open APs are detected at the same time. There are at least two approaches to solve this: 1) passive, distributed choice, or 2) active, coordinated choice.

The first approach relies on all clients making the AP choice locally based on some set of rules which result in all the clients choosing the same AP. The simplest of such algorithms is to pick based on the numerical order of the BSSID (every client connects to the AP with the lowest value for BSSID). The downside of such static rule is that the traffic load is likely to concentrate on the same APs. This can be rectified by including some time-varying, but shared information in the decision making. For example, the clients may append the current time with a resolution of an hour to the BSSID, calculate a hash over the concatenation and use that value for selection. This would then shift the load around APs over time.

The second approach would require active probing to discover nodes connected to an AP and then making a coordinated decision to join a specific AP or to change APs in order to spread the load. Such approaches are likely to be more complex but can potentially be more robust.

6. DISCUSSION

6.1 Scaling

For opportunistic ad-hoc operation, we would expect a mobile device to have at most a few neighbors at any point in time; in fact, quite often in DTN research, wireless interference and other aspects of multiparty interaction are discounted because sparse networks are assumed.[7] Leaving physical and link layer aspects aside, at the very least, using hot-spots will enhance the reach of mobile nodes: because using infrastructure mode is likely to increase the number of devices that can communicate to begin with, because APs act as a relay and increase the coverage area, and because multiple APs may form an extended service set.

This means that DTN nodes must exercise care when executing neighbor discovery mechanisms. Of the ones described earlier, passive discovery by observing other traffic (such as ARP or DHCP) would cause only minimal additional traffic whereas active unicast scanning would be most expensive. Multicast-based discovery is somewhere in-between, but multicasting uses lower transmission rates for transmission and therefore consumes more channel capacity. To maintain the channel share used by multicast-based discovery mechanisms, adaptive probing should be used that limits the total amount of discovery traffic within a hot-spot, similar to the restrictions defined for RTCP [22] or Mbus [17].

In an infrastructure WLAN with a large number of nodes associated to it, a mobile node faces a larger choice of peers to communicate with. For exchanging data, a node has to attempt connecting to its peer, exchange DTN metadata (e.g., contact and routing information), determine the messages to relay, and exchange them. A node should limit the number of peers it attempts to communicate with in parallel in order to limit its own resource usage within the hot-spot (see also next subsection). Once a node has scheduled set of messages for transmission to a peer, it should finish this set before choosing a new peer to connect to, so that also messages in the end of the (priority) queue will be replicated; otherwise, the contact durations would be artificially cut short. Investigating this in detail is a topic for further study.

[7]With Bluetooth, some such simplifications may indeed hold because connecting to more than one device may not be possible.

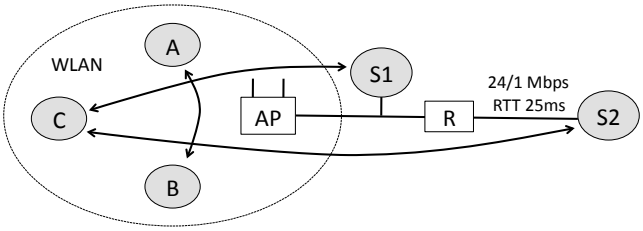

Figure 3: Simple Measurement Setup

6.2 Transport Protocol Considerations for Friendly Hot-Spot Usage

Two mobile nodes leveraging a hot-spot AP for their interaction obviously utilize resources intended for a different—usually commercial—purpose and thus may affect the regular (paying) users connected to the hot-spot for Internet access. Since it is easily possible for an AP to block node-to-node communication (as our experiments confirmed), mobile nodes seeking to "borrow" hot-spot capacity should ideally not disturb the hot-spot operation for regular users—otherwise future hot-spot systems might simply eliminate this opportunity to protect their customers' interests.[8]

Not disturbing traffic essentially means for a node to restrict itself to only using spare capacity so that the additional load does not lead to packet losses and the increased network load yield at most minimal increases in transmission latency.[9] To this end, the opportunistic traffic appears similar in nature to *scavenging* residual path capacity in the Internet as suggested for *background* applications such as software updates or file sharing, except that it is link-local. This suggests mechanisms similar to the *Low Extra Delay Background Transport (LEDBAT)* congestion control schemes devised for the Internet and implemented in, e.g., BitTorrent [10], might be applicable.

LEDBAT is designed to utilize the available capacity while maintaining low queuing delay when no other traffic is present; add only minimal queuing delay to other flows, and yield to flows employing TCP congestion control regimes [25]. To achieve this, LEDBAT closely monitors one-way delay and cuts back its sending rate upon slight increases, thus giving way before TCP congestion control would act.

Initial Experiments

While exploring such alternative transport protocols for peer-to-peer operation in depth is well beyond the scope of this paper, we carry out an initial set of experiments to obtain some first *qualitative* indication about the basic feasibility of this idea. We build the setup depicted in figure 3 comprising five nodes: a WPA-protected 802.11g WLAN is provided by a D-Link DWL2100AP access point to which three nodes are associated: (A) an IBM ThinkPad laptop computer running Ubuntu Linux 11.04 (kernel 2.6.38-15) and (B) a Nokia N900 Maemo phone serve as the two peer nodes generating the potentially interfering traffic for the regular client. (C) a MacBook running MacOS 10.7.4 serves as the stationary client in the commercial hotspots and also orchestrates the execution of the

[8]This is particularly important because the traffic between two mobile nodes hits the WLAN twice, yielding a stronger effect.

[9]Additional traffic also requires more energy to forward packets, but we consider this aspect to be minor.

measurements. For simplistic stationary traffic C interacts with a local Linux server (S1, local) and a remote one (S2, remote) in the Aalto University network. The local network is connected to the Internet via a commercial ISP using DSL access (supposedly 24/1 Mbit/s, but effectively rather some 16/1 Mbit/s); the mean RTT to S1 is 2–3 ms and to S2 some 25 ms.

We choose four different setups: as primary traffic, the client C uploads or downloads data from a local or a remote server for 90 s and logs each transferred segment. We combine these with three different types of peer-to-peer cross traffic between A and B: *none* (to obtain a baseline), *TCP*, and *LEDBAT*. We use *tcpx* [15] to generate TCP upload and download traffic between C and S1/S2 as well as TCP-based cross-traffic between A and B. For LEDBAT-based cross traffic between A and B, we employ the BitTorrent *libutp* library[10], specifically we instrument the tools *utp_send* and *utp_recv* to transfer 8 MB files. For both TCP and LEDBAT cross-traffic, A is the sender and B the receiver. The cross-traffic is started 10 s after the primary TCP traffic.

We analyze the goodput of the client-server and the peer-to-peer connections only from a *qualitative* perspective because, obviously, our measurements are biased by the implementations of the stationary and mobile nodes and the specific access point design and the setup is subject to external influences. For the local server, we find that the mean client (C) bit rate achievable with TCP cross traffic is some 5–20% below the rate without cross traffic, whereas it remains largely unaffected when using LEDBAT cross-traffic. For the remote server, the impact is less noticeable. Here, the capacity limitations of the DSL link clearly constrains the upload rate: none, TCP, and LEDBAT cross traffic yield similar results. For the download, TCP and LEDBAT yield roughly similar performance, some 5–10% below the rate achieved without cross traffic. However, with the unknown impact of the involved Internet, these numbers should be treated with care.

Looking at the bit rate for the cross traffic, we find that LEDBAT indeed makes room for the primary TCP traffic and picks up after the primary TCP connection has finished. In contrast, TCP attains a higher share of the WLAN capacity. The effect is more pronounced the stronger the primary TCP traffic is: for communication with the local server, the primary TCP rate is higher so that latency grows, which is interpreted by LEDBAT as a signal to cut back. Communication with the remote server, in contrast, does not fill up the WLAN so that the cues for LEDBAT are not as strong, yielding less cut-back by LEDBAT, whose performance is closer to TCP.[11]

Overall, this simple experiment provides some initial hints that (for WLANs under load) it could be desirable for mobile peer-to-peer traffic to utilize a transport protocol that yields to traffic of paying customers. LEDBAT appears as a reasonable starting point for such a scavenging adaptive transport mechanism, but will likely require tuning to become more sensitive to early delay cues. We leave exploring those ideas further as a topic for further study.

6.3 Private WLANs

The mechanisms described above to support direct peer-to-peer communication between mobile nodes in commercial hot-spots could equally be applied to access points in private households or businesses. Similar to commercial hot-spots, a key point is that local users are not disturbed by the traffic load caused by the mobile ad-hoc users. Using scavenging transport protocols along the lines discussed above would be applicable as well.

While commercial hot-spots do provide authentication mechanisms and prevent unauthorized access to the Internet by means of captive portals on top of unsecured WLANs, private and business access points usually employ L2 security (WPA2), possibly in conjunction with 802.1x to protect their network. In such cases, an access point would need to host a second SSID for mobile users that is open at the link layer and isolated from the SSID used for internal communications. This could, for example, be achieved using *virtual SSIDs* (as is done for some WLAN sharing services such as FON[12]) in conjunction with VLANs[13].

A suitably designed access point could also easily limit the throughput on the VLAN assisting mobile peer-to-peer communication whenever local traffic is present and thereby prevent (malicious) mobiles from undue resource utilization. In a simple case, all mobile-to-mobile traffic would be treated jointly as a separate traffic class with traffic shaping applied as a whole. Since mobile users will usually be only associated with the AP for a short period of time, their individual impact on stationary users may be limited so that such a data rate cap could be imposed only after a grace period (e.g., 60 s). More complex algorithms, determining the cap and grace period dynamically as a function of the total load incurred my the mobile users (over time) and the demand of stationary users, are also conceivable.

7. CONCLUSION

Our experimentation supports that WLAN access points can be used in practice to learn about other co-located mobile devices in urban areas. However, observations from running complex connectivity scanner show that any single scanning method and address combination does not outperform all others. This calls for using combination of scanning methods and providing appropriate addresses (IPv4/IPv6, unicast/multicast/broadcast) for each method. While the present evaluation shows a need for combination of scanning schemes, efficient semantics for choosing such combinations will be a topic for future work.

A further observation is that discovering device co-location is more widely supported than data transfer between devices via an access point by using a known transport such as the TCP convergence layer for DTNs. This hints that using WLAN based discovery can serve as a signal to start using peer-to-peer communications, such as WiFi-Opp or Wi-Fi Direct, that enable using chosen convergence layers upon discovering device co-location. In contrast, while Wi-Fi Direct provides its own discovery methods, using it all the time would render traditional WLAN access unusable as current devices do not support its use simultaneously to infrastructure mode.

Measurements show that choosing a specific protocol (e.g., the TCP convergence layer) may lead to poor success in data transfer; this calls for future work on additional transport mechanisms, for example, UDP (multicast) based convergence layers. Such future convergence layers also need to avoid interfering with paying customer traffic in hot-spots so that service providers do not block ad-hoc traffic. Exploring a bandwidth scavenging protocol such as LEDBAT appears as a promising first step into this direction, but refinements to capture the delay properties of WLANs may be required.

As a general conclusion we see that using WLAN access points for enabling opportunistic peer discovery and communication can

[10] https://github.com/bittorrent/libutp/

[11] Interestingly, we also find that the TCP connection for the cross traffic occasionally stalls for several seconds whereas LEDBAT traffic continues at a low rate.

[12] http://www.fon.com/

[13] See, e.g., http://www.dd-wrt.com/wiki/index.php/Multiple_WLANs

work in practice for urban pedestrians, but requires careful design to achieve high discovery and communication success rates. Our future work includes exploring the capabilities of commercial WLAN hot-spots at larger scale, devising an algorithmic approach to efficient connectivity establishment between mobiles, and exploring the design of a scavenging transport protocol in more detail.

Acknowledgements

This work received funding from the Academy of Finland in the RESMAN project (grant no. 134363) and from the European Community's Seventh Framework Programme under grant agreement no. 258414 (SCAMPI).

The authors would also like to thank Matthias Wählisch for his support with the early WLAN AP measurements.

8. REFERENCES

[1] Aruna Balasubramanian, Yun Zhou, W. Bruce Croft, Brian Levine, and Arun Venkataramani. Web search from a bus. In *Proc. of the Second MobiCom Workshop on Challenged Networks (CHANTS)*, September 2007.

[2] S. Cheshire and M. Krochmal. Multicast DNS. Internet Draft draft-cheshire-dnsext-multicastdns-15, December 2011.

[3] Microsoft Corporation. Universal Plug and Play Device Architecture. available at http://www.upnp.org/, June 2000.

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

[5] Wi-Fi Alliance Technical Committee P2P Task Group. Wi-Fi Peer-to-Peer (P2P) Technical Specification, 2010.

[6] Md. Tarikul Islam, Anssi Turkulainen, Teemu Kärkkäinen, Mikko Pitkänen, and Jörg Ott. Practical Voice Communications in Challenged Networks. In *Proceedings of the ExtremeCom workshop*, 2009.

[7] V. Lenders, M. May, G. Karlsson, and C. Wacha. Wireless ad hoc podcasting. *ACM/SIGMOBILE Mobile Comp. and Comm. Rev.*, 12(1), 2008.

[8] Anders Lindgren. Social networking in a disconnected network: fbDTN: facebook over DTN. In *Proc. of ACM MobiCom CHANTS workshop (demo)*, Sep 2011.

[9] Martin Lukac, Lewis Girod, and Deborah Estrin. Disruption Tolerant Shell. In *ACM SIGCOMM Workshop on Challenged Networks (CHANTS)*, September 2006.

[10] Arvid Norberg. uTorrent transport protocol. BitTorrent Enchancement Proposal 29, January 2010.

[11] Mark Nottingham and Roy T. Fielding. Additional HTTP Status Codes. RFC 6585, April 2012.

[12] Jörg Ott. Me – Your ISP. *IEEE Internet Computing*, pages 84–87, July/August 2011.

[13] Jörg Ott, Esa Hyytiä, Pasi Lassila, Tobias Vaegs, and Jussi Kangasharju. Floating Content: Information Sharing in Urban Areas. In *IEEE Percom*, 2011.

[14] Jörg Ott and Jussi Kangasharju. Opportunistic Content Sharing Applications. In *Proc. of the ACM MobiHoc NOM workshop*, June 2012.

[15] Jörg Ott and Dirk Kutscher. Drive-thru Internet: IEEE 802.11b for "Automobile" Users. In *Proceedings of IEEE Infocom, Hong Kong*, March 2004.

[16] Jörg Ott, Dirk Kutscher, and Mark Koch. Towards Automated Authentication for Mobile Users in WLAN Hot-Spots. In *Proc. IEEE VTC Fall*, September 2005.

[17] Jörg Ott, Colin Perkins, and Dirk Kutscher. A Message Bus for Local Coordination. RFC 3259, April 2002.

[18] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proc. of ACM MobiSys*, 2006.

[19] Anna Kaisa Pietiläinen and C. Diot. Experimenting with Opportunistic Networking. In *Proc. of the ACM MobiArch Workshop*, 2009.

[20] Mikko Pitkänen, Teemu Kärkkäinen, Janico Greifenberg, and Jörg Ott. Searching for Content in Mobile DTNs. In *Proc. of PerCom*, 3 2009.

[21] Mikko Pitkänen, Teemu Kärkkäinen, Jörg Ott, and et al. SCAMPI: Service platform for soCial Aware Mobile and Pervasive computIng. In *Proc. of the ACM SIGCOMM workshop on Mobile Cloud Computing (MCC)*, August 2012.

[22] Henning Schulzrine, Stephen Casner, Ron Frederick, and Van Jacobsen. *RTP: A Transport Protocol for Real-Time Applications*, July 2003. RFC 3550.

[23] James Scott, Pan Hui, Jon Crowcroft, and Christophe Diot. Haggle: A Networking Architecture Designed Around Mobile Users. In *Proceedings of IFIP WONS*, Les Ménuires, France, January 2006.

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

[25] Stanislav Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind and. Low Extra Delay Background Transport (LEDBAT). Internet Draft draft-ietf-ledbat-congestion-09.txt, Work in progress, October 2011.

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

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