# Benchmarking Wireless Protocols for Feasibility in Supporting Crowdsourced Mobile Computing

João Rodrigues[1], Joaquim Silva[1], Rolando Martins[1], Luís Lopes[1], Utsav Drolia[2], Priya Narasimhan[2], and Fernando Silva[1]

[1] CRACS/INESC-TEC, Faculty of Science, University of Porto
[2] ECE, Carnegie Mellon University

**Abstract.** Recent advances in mobile device technology have triggered research on using their aggregate computational and/or storage resources to form edge-clouds. Whilst traditionally viewed as simple clients, smartphones and tablets today have hardware resources that allow more sophisticated software to be installed, and can be used as thick clients or even thin servers. Simultaneously, new standards and protocols, such as Wi-Fi Direct and Wi-Fi TDLS (Tunneled Direct Link Setup), have been established that allow mobile devices to talk directly with each other, as opposed to over the Internet or across Wi-Fi access points. This can, potentially, lead to ubiquitous, low-latency, device-to-device (D2D) communication. In this paper, we study whether D2D protocols can support mobile-edge clouds by benchmarking different protocols and configurations for a specific application. The results show that decentralized device-to-device techniques can be used to efficiently disseminate multimedia contents while diminishing contention in the wireless infrastructure, allowing for up to 65% traffic reduction at the access points.

## 1 Introduction and Motivation

Mobile devices are now ubiquitous [1]. The increase in the sheer number of devices has also led to the increase in the density of devices, i.e. more often than not, there will be multiple mobile devices in proximity of each other. Moreover, these devices are now equipped with multi-core processors, multi-GB memory and multiple communication interfaces. This trend has been leveraged by a new class of systems, *mobile edge-clouds* [2] [3] [4] [5] [6] [7] [8]. These systems aggregate computation and storage resources across nearby mobile devices to enable resource-efficient applications.

Simultaneously, new standards and protocols, such as Wi-Fi Direct and Wi-Fi TDLS (Tunneled Direct Link Setup), have been established that allow mobile devices to talk directly with each other, as opposed to over the Internet or across Wi-Fi access points. This can, potentially, lead to ubiquitous, low-latency, device-to-device (D2D) communication.

We believe that these new D2D protocols can boost the efficiency of mobile edge-clouds. Since these systems are meant for scenarios where devices are in

proximity of each other, D2D communication seems like a natural fit. For example, let us consider the following scenario: watching video replays on mobile devices at live events, e.g. soccer match. There is a growing market of apps that provide users within (and outside) the venue with almost real-time statistics and multimedia contents like the number of kilometers a player has run or video replays for goals or interesting events [9] [10]. If a fan chooses to watch a video replay, the content is downloaded from the central servers through stadium installed access points (Wi-Fi or cellular), and then played on the device. If, however, the venue is crowded, the large number of requests can stress the infrastructure [11] [12].

One way to solve this problem is to use mobile edge-clouds. In this way, you and your neighbours may form a local cache for the server contents - users of the service can be encouraged to share in several ways, e.g., sweepstakes of team merchandise or lower rates for the service. For example, before asking the server for a video replay, your app might ask the other phones in the mobile network whether they have a copy of the video. If a copy of the replay is located, your app can retrieve it directly from a neighboring device. Our first hypothesis is that such retrievals can be accelerated through D2D protocols. Moreover, if the retrievals in mobile edge-clouds are done over infrastructural access points, the access points would be congested in-turn leading to high-latency downloads and an overall bad user-experience. Our second hypothesis is that D2D protocols can alleviate such infrastructural stress.

In this paper, we study whether D2D protocols can support mobile-edge clouds by benchmarking different protocols and configurations for a specific application. The results suggest that decentralized, device-to-device techniques can be used to efficiently disseminate multimedia contents while diminishing contention in the wireless infrastructure, e.g., central servers and access points, namely through the use of TDLS and WiFi-Direct.

The remainder of this paper is structured as follows. Section 2 describes the scenarios we are interested in exploring and the experimental setup. Section 3 describes the application we developed to perform the experiments. Sections 2 and 4 describe the experimental setup, the results obtained and discusses their implications. Section 5 overviews related research work and, finally, Section 6 ends the paper with the conclusions and future work.

## 2 Assumptions and Scenarios

We make the following assumptions: (a) all the devices are non-rooted since we are interested in improvements that can be readily implemented in current infrastructures and do not rely on invasive procedures for the devices; (b) we do not control the usage of the radio channels and the choice of the radio band, 2.4GHz or 5GHz, for communication; (c) all devices are within radio range and can make requests and transfer contents using direct connections or using an AP or a hot-spot as an intermediate; (d) there is no application-level routing or discovery mechanisms, we only use what is provided by the underlying wireless

protocols; (e) in the scenarios that involve mobile servers, the latter have all the files that will be requested by the clients, and that would otherwise be present at the central server in a traditional infrastructure, and; (f) clients know the location of the servers from the start.

We define a set of content dissemination scenarios for downloading video replays from a soccer game in a stadium (Figure 1). In all scenarios we assume WiFi is used for communication. In the case of pure WiFi, all scenarios include a traditional, dedicated, access point that the devices use for communication. When using WiFi with TDLS, the access point is used only for the initial contact between the nodes. In the case of WiFi-Direct, this role is performed by a mobile device known as the "group owner".

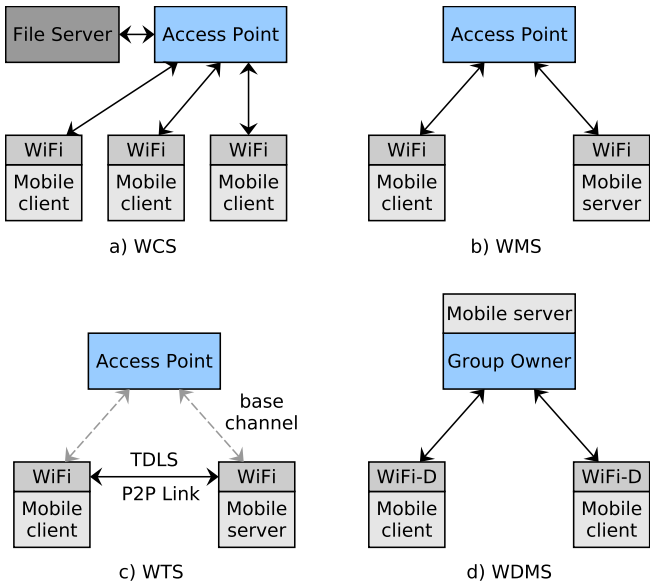


**Fig. 1.** a) WCS - central server, access point, mobile clients, using WiFi. b) WMS - access point, mobile servers, mobile clients, using WiFi. c) WTS - access point, mobile servers, mobile clients, using WiFi/TDLS. d) WDMS - group owner, mobile server, mobile clients, using WiFi-Direct.

In the first scenario (WCS - WiFi Central Server) we have a main server, part of the infrastructure of the stadium, acting as the source of the video replays. The server is connected to access points in the stadium via high-speed Ethernet links. Each access point handles tens of devices in the part of the venue it provides coverage [12]. All mobile devices are clients.

In the second scenario (WMS - WiFi Mobile Server) we remove the main server, and just stick with the access point. Part of the mobile devices in the

venue will act as servers (servers only) for the multimedia contents, thus providing the same content as the central server in the previous scenario.

In the third scenario (WTMS - WiFi TDLS Mobile Server) we use the access point only for the first contact between the devices. TDLS then allows the devices to communicate directly and transfer data. Again, part of the devices in the venue will act as the servers for the video replays. As in the previous scenario, mobile servers are not also clients and contain all the content that might be provided by the central server.

In the fourth scenario ( WDMS - WiFi Direct Mobile Server) the configuration is similar to WMS with the access point replaced by one of the devices acting as a hot-spot, known as "group owner". All communication between mobile servers and clients must go through the group owner.

We identified an additional scenario, similar to WTS, in which the access point is replaced with the group owner. Here, however, we use TDLS to allow servers and clients to communicate directly (mostly) without the intervention of the group owner. We did not experiment with this last configuration. We were not able to form networks of more than 5 or 6 devices using WiFi-Direct and thus present only preliminary results for the WDMS scenario. This difficulty arises not only from the fact that we are using non-rooted devices, and thus cannot fully control operating system configurations, but mostly because most network card drivers/firmware are proprietary software.

## 3 Test Application

We developed Java/Android applications to implement the clients and servers in the aforementioned scenarios. The servers, both fixed, in the first scenario, and mobile, in the others, follow the standard client-server architecture. The server waits for a connection from a client; when one is received, they launch a new thread that examines the request and transfers the requested file. The mobile clients are tailored to be used with the Android Debug Bridge (ADB) tool. We developed a template from which all client behaviours could be implemented by extending a base class. This class has three main methods that are overridable for different scenarios, namely: `startApp`, `transferFiles` and `endApp`. `startApp` executes a sequence of checks to ensure that a client will run correctly, e.g., if the device is connected to the correct WiFi network. It also initializes a logger in order to record information about the run, and fills a work queue with the file requests, a permutation of the file names, to be downloaded by the device. Each file is mapped to a set of predefined servers that can provide it. The clients choose one of these for each download. After that, method `transferFiles` is called to download the files, which are transfered, one by one, after selecting appropriate servers. Once a transfer finishes, the next file on the queue begins to be downloaded. When all files are downloaded, the application calls the `endApp` method to perform all the necessary cleanup.

The execution of the application by the devices starts simultaneously thanks to a barrier implemented (a shared file is used for this) at the end of the `startApp`

method. The devices also synchronize before ending, again by using a barrier at the beginning of the `endApp` method. We used a set of shell scripts to automate the execution of the application and to control the devices through the Android Debug Bridge (ADB). The `adb` command line tool allows to execute ordinary Linux shell commands remotely on Android devices, e.g., `ls`, `cp`. This allows, for example, uploading all log files from the devices to a server for processing and commanding all experiments using simple scripts.

**Algorithm 1** Execution cycle

```
function RUN(n, m, l)
    devices ← GET_DEVICES(n)
    servers ← GET_SERVERS_LIST(m)
    clients ← GET_CLIENTS_LIST(n − m)
    run ← 0
    while run < l do
        REBOOT(devices)
        RUNSERVERS(servers)
        RUNCLIENTS(clients)
        run ← run + 1
    end while
    COPY_LOGS(devices)
end function
```

**Fig. 2.** Algorithm for running the experiments.

For better control, we decided to use ADB through the USB interface (instead of WiFi), with all devices connected to the control computer through an USB hub. Figure 2 shows the basic procedure for running the experiments. The procedure takes three arguments: $n$, the number of devices, $m$, the number of servers and, $l$, the number of times the experiment is to be repeated. We start each run by rebooting the mobile devices to ensure that all devices are in similar circumstances and that there are no extra processes running that could interfere with the results. Afterwards, the RUNSERVERS procedure sends a command to all the server devices to start the local server application. Next, RUNCLIENTS starts the local client application at each client device, that in turn calls the `startApp` entry point. Once all the runs are performed, the logs are copied from the devices to a desktop computer for analysis.

## 4 Experiments and Results

Our experimental setup is composed of the following hardware: 1 Asus RT-AC56U router (AP), 20 non-rooted HTC Nexus 9 tablets, 2 Trust USB hubs (10 ports each) and 1 desktop computer. The layout of the deployment can be seen in Figure 3. The devices were placed on top of a table, side-by-side, in a 4×5 pattern, as in a typical of stadium seats arrangement. They all run Android Lollipop 5.1.1 (API level 22) on top of which our test software was executed. The devices were connected to a control desktop computer via the Android Debug Bridge.

The experiments were setup in such a way that we guarantee that all the content requested by the clients during the runs exists in all servers. In each experiment, clients must download 20 video files, each 3 Mbytes in size from the servers. This size represents a video clip with 10 seconds duration and encoded

with H264 video codec using 480x270 (width x height) as the frame size, a typical format used by mobile apps. Before starting the transfer, each client computes a random permutation of the 20 file names using a uniform distribution, to even out the requests for each individual file during an experiment. Accesses are also performed randomly in time, as each client waits a random time interval, within given bounds, before requesting the next file of the sequence. For each scenario, we run a set of experiments with a varying number of mobile servers/clients. Each experiment was repeated 8 times to smooth out statistic flukes.

The average download time for each video is represented in the graphs in the next sections, together with the bars for the 95% confidence intervals. The values for power dissipation are given as the average per download, as measured using the Android API (`android.os.BatteryManager`) The instantaneous current intensity and the voltage (mostly constant) are read periodically to compute the dissipated power which is then integrated over the complete experiment to get the total energy spent per download.

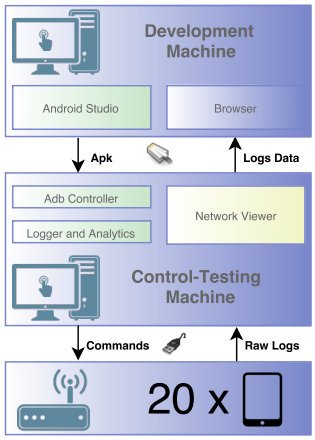


**Fig. 3.** The experimental setup.

To characterize the radio environment, we performed a WiFi analysis on our campus and detected that the 2.4Ghz band was extensively utilized by various services, while the 5Ghz band usage was minimal. All the experiments were performed using the 802.11n for packets exchanged between the AP and the devices; 802.11ac was used whenever packets were exchanged device-to-device[3].

### 4.1 Single Server

We start by restricting the experiments to only one server (infrastructural or mobile), and experiment with four scenarios: WCS, WMS, WTMS, and WDMS. The results for WiFi-Direct (WDMS) are just preliminary. We vary the number of mobile clients downloading videos from 1 to 16. Figure 4 shows the average download time per file. Figure 5 shows the total traffic processed by the AP, in the WCS, WMS and WTMS scenarios. For figure 5, given that the amount of traffic for WCS is exactly the same amount as in the case of WMS (we only accounted for outgoing traffic), we decided not to show it for clarity. Note the small 95% confidence interval error bars.

In Figure 4 WMS is clearly the worst performer, taking almost twice as much time comparing to the other scenarios. This is due to the presence of two wireless hops in the communication path, more specifically, from the mobile client device to the AP and then from the AP to the mobile server device. WCS wins as a

[3] As a note, Android TDLS implementation switches automatically between the 2.4 and 5GHz bands and this is neither controlable and observable from the API.

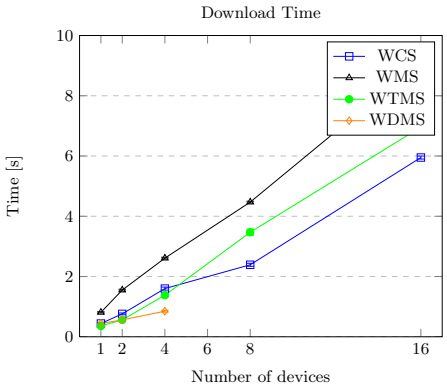


**Fig. 4.** Average download time per file.

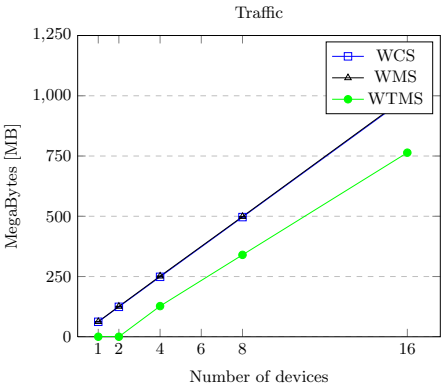


**Fig. 5.** Traffic handled by the AP.

result of the much faster link between the central server and the AP. As we stated, WiFi-Direct could only be used up to 5 devices - 4 clients and 1 server. We present only experiments with 1, 2 and 4 clients. It is hard to extrapolate the behaviour for larger number of clients but for low numbers of clients the results are similar to the best performer (WCS).

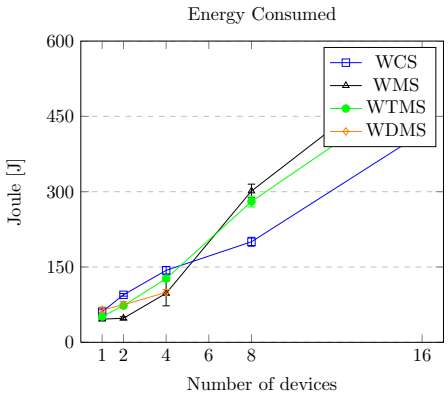


**Fig. 6.** Average energy consumed per experiment.

The behaviour of WTMS is more subtle. While the 802.11z [13] specification extension allows for a peer to have multiple links to different base stations, in practice only 2 channels were available from the server at any given time, leading us to conclude that only one channel is available per radio (since our test devices, Nexus 9, have 2 radios), and thus no multiplexing is performed. This is visible in figure 5, where no traffic is routed through the AP up to 2 clients. The introduction of additional clients only shows limited savings on the traffic being routed through the AP. This is due to the 802.11z's mandatory periodical switch-over to the base channel (the regular AP's WiFi communication channel) to process buffered messages (by the AP).

Given this, if there are only two clients then the two radios can be used to setup TDLS channels. Adding more clients will result in the traffic being routed through the AP, that in turn, will lead to the server device to spend more airtime

on one of the available channels to communicate with the AP. This will effectively only allow for 1 offloading channel to be available.

Figure 6 shows the energy consumed per experiment. The figure shows some similarity to Figure 4 which is expected since our application uses almost exclusively the WiFi hardware to send/receive replays; it does not play the videos on arrival, it just stores them. Hence, download time is an excellent proxy for energy consumption. We observed this correspondence for the other experiments we performed and therefore will not present more graphs on energy consumption in this paper. In absolute value, the energy spent in the downloads is rather small. For example, given the 25.5 Wh specification of the battery from the Nexus 9, this is equivalent to about 91000 Joule. The energy spent per download in, e.g., the WCS scenario, with 1 server and 1 client, is about 50 Joule/20 downloads = 2.5 Joule, corresponding to 0.002% of the total charge; this value grows to 0.02% when using 16 clients.

### 4.2 Multiple Servers

We extended our experiments to examine the impact of introducing multiple mobile servers in the mobile scenarios WMS and WTMS. Figures 7 and 8 show the results obtained. In WTMS we introduced an extra level of refinement. In one case we let the clients choose a random server for each file to be downloaded from a predefined set - this is shown as WTMS$^D$ in the graphs. The alternative process is to statically map each file to a specific server from the start, potentially decreasing competition between clients - this is shown as WTMS$^S$. We increase the number of servers, from 1 to 8, while using a fixed number of clients (12).

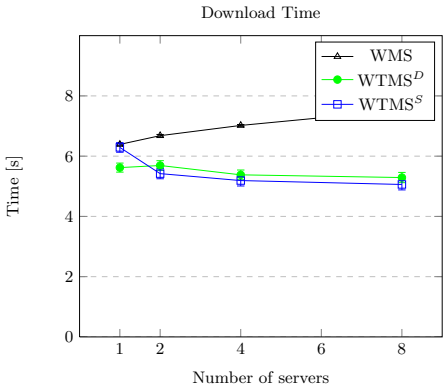


**Fig. 7.** Average download time per file.

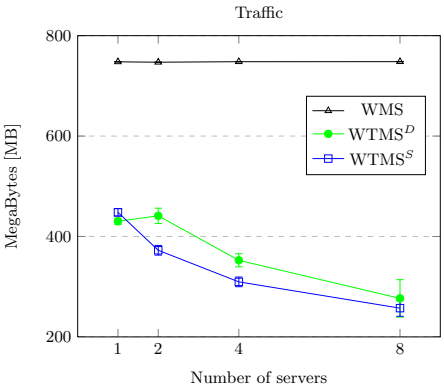


**Fig. 8.** Traffic handled by AP.

Figure 7 shows the average download time with an increasing number of servers. The introduction of TDLS clearly decreases the download time, with the static assignment performing slightly better than the random assignment. This is

due to the better client distribution that the static approach provides, decreasing the possibility of clients overloading a subset of the servers. The results show a slight increase of 20% in the download time with WTMS scenarios from 4 servers relative to the WCS scenario (for 12 clients) (c.f. Figure 4). This, however, is compensated by a 65% decrease in the traffic handled by the AP under WTMS with 8 servers (note that, in WMS as in WCS all traffic goes through the AP). Relative to the WMS scenario we also observe a 33% decrease in the download time.

We resorted to simulation using ns-3 to better understand the behaviour of TDLS, namely, how the number of successful TDLS links varied with the number of available servers in the WTMS (static) scenario. Since there is no native support in ns-3 for TDLS, we had to implement our own code to mimic the behaviour of the radio interfaces. Based on the available documentation and on our empirical experience we deduced that the Nexus 9 allows the use of 2 simultaneous channels that can be any combination of 802.11n and 802.11ac. To simulate this behaviour, we implemented each ns-3 node with 3 network interfaces (2×802.11ac, 1×802.11n), so that at any given time we can have 2 channels as in the Nexus 9. So, although the nodes have 3 software radio interfaces, if we have 1 or 2 clients connected to a server, the server node can only use 2 TDLS interfaces. If more than 2 clients are connected to a server, then we can only use 1 TDLS interface, and all the other connections are done via 802.11n (through the AP). An auxiliary class manages the TDLS connections, parametrized on the percentage of successfully established TDLS links.

We ran several simulations varying the percentage of TDLS connection success. The results show that, to explain the traffic values from Figure 8, the percentage of successful TDLS links must vary from 50% with 1 server to 60%-65% with 8 servers. We attribute this variation to the fact that, with more servers available, there are less clients per server. For example, with 8 servers we have 16 (2×8) 802.11ac channels available and hence, with 12 clients we can more easily establish TDLS connections, despite the extra global radio interference that leads to collisions.

Also, it is apparent from Figure 7 that, in WTMS, the performance does not improve significantly beyond 4 servers. To understand why this happens we performed another experiment where we reduced the number of downloads from 20 to 1 and increased the size of the file to be dowloaded from 3MB to 60MB, so that, overall, the amount of traffic is identical.

Figures 9 and 10 show the results of the experiment. By decreasing the number of downloads per experiment, we observed a decrease in the amount of traffic going through the AP (figure 10). This lead us to conclude that the initial setup for TDLS channels is causing a significant overhead. Since we are unable to inspect the internal behaviour of the network driver, we infer that the creation of a TDLS channel requires multiple attempts, probably due to contention and interference, resulting in a performance degradation. By decreasing the number of downloads (and thus the number of TDLS tunnel negotiations), the expected behaviour was verified with the introduction of additional servers, namely, each

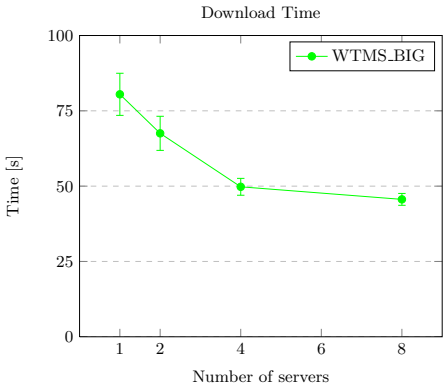


**Fig. 9.** Average download time while using a 60MB file.

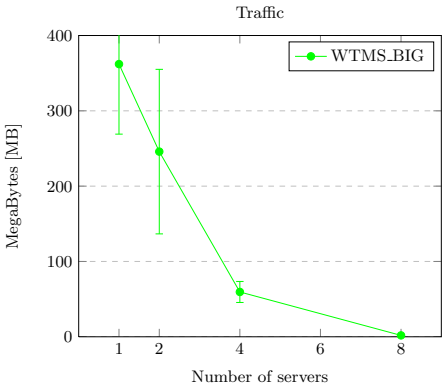


**Fig. 10.** Traffic handled by AP while using a 60MB file.

server is able to support 2 distinct channels, with 1 being allocated to the AP base channel in the presence of 3 or more clients. For example, in the case of 4 servers, there were 3 clients statically allocated with each server (statically assigned, represented by WTMS$^S$), while with 8 servers, we had 4 servers assigned with 2 clients and 4 servers assigned with 1 client.

The download time and amount of traffic tend to lower with the introduction of additional servers, as depicted in figures 9 and 10. However, there is a substantial amount of variation on the results due to the negotiation process of the TDLS links. As stated in 802.11z, if during the setup of a direct link the driver detects a beacon then the negotiation is aborted; or if the amount of traffic on a channel is high enough then the TDLS link could be renegotiated to a different channel, with the traffic between the two peers being redirected through the AP during that time.

### 4.3 Discussion

The fastest download time is achieved by the traditional infrastructure, mostly because it uses a very fast wired 1Gbit/s network between the server and the AP, and also because the AP did not reach saturation in our experiments, even with the highest number (16) of clients. However, the use of multiple servers in an ad-hoc network of devices, all within radio reach, allowed significant removal of traffic from the AP, up to a maximum of 65% less traffic using 8 servers. The corresponding download time for files is 20% higher than the one observed for the traditional server/AP infrastructure, which, in absolute figures, for 3MB files, results in a delay of $\approx 1$ second. Energy-wise, the cost of transfer is small for the Nexus 9, with a nominal battery capacity of about 91000 Joules, amounting to 0.02% with the maximum number of clients competing for the files. Smartphone batteries have typical capacities an order of magnitude smaller and thus we would expect each download to require a few tenths of percent of the battery. Since

each user individually is not expected to make many downloads during a game, battery shortage is, evidently, not a limitation. Thus, the user experience is not significantly diminished by the use of an ad-hoc network of devices to distribute contents while there are considerable gains in terms of removing stress from the AP.

## 5 Related Work

Managing the aggregate computational/storage resources of ad-hoc networks of mobile devices, such as smartphones and tablets, has become a hot topic of research in recent years. This results mostly from ongoing technological advances and the widespread use of the devices. Several projects have explored the technology's many angles, e.g., offloading, crowdsourcing, cost models, protocols, security, and its applications to distinct areas, e.g., commerce, learning, health, entertainment [14] [15].

FireChat [6] is a proprietary mobile app, developed by Open Garden, that allows smartphones to organize into a wireless mesh network and exchange messages using available technologies such as Bluetooth or Wi-Fi. The iOS version uses Apple's Multipeer Connectivity Framework [16] that enables services to be advertised and discovered between nearby iOS devices using different wireless protocols. Services can be provided through personal area type-of networks using infrastructure WiFi, peer-to-peer WiFi and Bluetooth.

Bonjour [17] focus on three main areas: Addressing (allocating IPs to Devices), Naming (creating alias for each network device) and Service Discovery. The discovery works similarly to a publish-subscribe, where nodes advertise their services which then other nodes can use them. In order for devices in a network to be discovered, upon turning on the bonjour service, they send a announce themselves to the network. At the same time, other nodes in the network, already running a Bonjour Service, periodically ask the network what devices are available.

Alljoyn [18] is an, open source, agnostic network framework that allows different types of devices and apps to discover and communicate in an abstract way, hiding the complexity of distinct network protocols and hardware. Generically, it publishes APIs over the network through a general bus, which permits distinctive network technologies to be used, such as Wi-Fi, Wi-Fi Direct, Bluetooth, Ethernet and PowerLine. To the best of our knowledge only infrastructure WiFi is actually implemented. Regarding the network formation, Alljoyn, uses a super peer paradigm, mesh of stars network, where the leaf nodes are connected to router nodes and these act as bridges to the others router nodes.

Efforts to transform mobile networks into actual computational and/or storage resources have been the subject of fundamental work. In [19] [20] Hadoop was successfully ported into mobile devices connected though WiFi to perform map reduce computations. The goal was to identify the problems that might arise from porting a system developed for full-fledged cloud servers to a resource-starved cloud of devices. Doolan et al. [3] try a different approach by adapting

the well established Message Passing Interface (MPI) for mobile systems, with the goal of performing parallel processing over these platforms.

In [21] the authors study the trade-offs between offloading computation to an infrastructure cloud versus retaining the computation within a mobile edge-cloud. They present two diverse workloads for mobile edge-clouds based on the distribution of data and motivate the use of edge-clouds for one of them, specifically when data is inherently distributed in the edge-cloud it is better (in terms of latency and power consumption) to process them in an in-situ manner as opposed to transferring them to the infrastructural-cloud for processing.

## 6 Conclusions

In this paper, we study whether D2D protocols can be used to efficiently disseminate multimedia contents in networks of mobile devices. We do this by benchmarking different protocols and configurations for a specific application. Our first hypothesis, that the download speed would be improved with D2D protocols was not vindicated, although the number of devices we used (and had available) was really not enough to take the AP close to saturation, in which case we would expect such improvements. The observed difference and corresponding extra energy usage is small and does not degrade user experience. On the other hand, the experiments suggest that our second hypothesis, that the use of D2D protocols could significantly remove load from the AP, is valid for we observed a decrease in traffic up to 65% for a multiple server configuration.

Thus we conclude that D2D protocols that take advantage of existing wireless technologies can indeed be used to efficiently disseminate multimedia contents and, especially, to diminish the load in the traditional wireless infrastructure, a critical problem for service providers in large sports venues, while maintaining a good user experience.

We plan to expand this line of work to include network overlays in order to create a fully decentralized and self organizing mesh that leverages the different underlying wireless protocols.

## Acknowledgment

This work has been sponsored by projects HYRAX (CMUP-ERI/FIA/0048/2013), funded by FCT, and SMILES (NORTE-01-0145-FEDER-000020), funded by NORTE 2020, under PORTUGAL 2020, and through the ERDF fund.

## References

1. "Global mobile statistics 2013," http://mobiforge.com/, last visited in 19/02/2016.
2. U. Drolia, N. Mickulicz, R. Gandhi, and P. Narasimhan, "Krowd: A key-value store for crowded venues," in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch'15. ACM, 2015, pp. 20–25.

3. D. C. Doolan, S. Tabirca, and L. T. Yang, "MMPI a Message Passing Interface for the Mobile Environment," in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '08. New York, NY, USA: ACM, 2008, pp. 317–321.
4. E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, Master's Thesis, Carnegie Mellon University, 2009.
5. T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner, "mcrowd: A platform for mobile crowdsourcing," in *7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*. New York, NY, USA: ACM, 2009, pp. 347–348.
6. "OpenGarden's FireChat App," http://opengarden.com/firechat/, last visited in 19/02/2016.
7. "A security, private internet and tactical cloud at the edge." Kurzweil News, August 2013.
8. P. Wait, "Darpa creates cloud using smartphones," Information Week, August 2013.
9. "YinzCam," http://www.yinzcam.com/, last visited in 19/02/2016.
10. "Agile stadiums bring digital content to sports fans," http://goo.gl/4BHxr6, June 2015, last visited in 19/02/2016.
11. P. Kapustka and C. Stoffel, "State of the Stadium Technology Survey," Tech. Rep., 2014.
12. J. Erman and K. Ramakrishnan, "Understanding the super-sized traffic of the super bowl," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. ACM, 2013, pp. 353–360.
13. "802.11z Amendment 7: Extensions to Direct-Link Setup," http://goo.gl/acQa0i.
14. N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computing Systems.*, vol. 29, no. 1, pp. 84–106, Jan. 2013.
15. H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
16. "Apple's Multipeer Framework," https://goo.gl/332lwR, last visited in 19/02/2016.
17. "Apple's Implementation of Bonjour," https://www.apple.com/support/bonjour/, last visited in 19/02/2016.
18. "Alljoyn Framework," https://allseenalliance.org/framework, last visited in 19/02/2016.
19. E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce." Master's thesis, Carnegie Mellon University, 2009.
20. C. L. V. Teo, "Hyrax: Crowdsourcing mobile devices to develop proximity-based mobile clouds." Ph.D. dissertation, Carnegie Mellon University, 2012.
21. U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan, "Motivating mobile edge-clouds," in *10th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC'13)*, 2013.