DroidPark: a context-aware amusement park application based on opportunistic data dissemination

E. Carniani, G. Peterle, (IIT-CNR Pisa)
I. Belluco, F. Ricci (University of Pisa)

Abstract - **In this document we are introducing DroidPark, a Mobile Social Network application for the Android Operating System designed to share information about queues, opinions and user ratings of a park's attractions in an opportunistic environment. Droid Park is a case study of data dissemination in a real application context, by means of opportunistic communications based on the CAMEO middleware. We are going to show how the usage of CAMEO may help in providing context and social-aware functionalities, in order to ease application development, and improve overall performance and user experience.**

I. Introduction

The more the handheld smart devices get diffused, the more they become a key technology to allow for new kinds of mobile applications that can be integrated into users' everyday life. As the user gets equipped with mobile and smart devices, new approaches to application interactions can also be explored. Moreover, as mobile devices get more effective to collect, store, analyse and carry information, new applications can be developed, at least in the way they can be designed and concretely work. From the user point of view there are many possible contexts that get quite interesting, and eventually lead to a real and useful application counterpart. We simply need to exploit the paradigms of data exchange that may arise from these scenarios and thus design the best implementation.

As an example of this, we introduce DroidPark, a MSN application that involves people visiting an amusement park and their related activities (queuing to an attraction, telling others their opinions about it, rating it with stars, or simply walking around the park). The application is enriched with opportunistic data dissemination implemented on top of the CAMEO platform [ARN13]. The CAMEO middleware implementation is a stable good starting point that improves the application's ease of development and performance [ARN11].

As well as in [ARN14], a paper explaining a novel approach for a fitness application able to find the best walkway to a destination through opportunistic sensing, we are also going to show how CAMEO may provide context-aware functionalities for DroidPark.

The case study highlights different issues that may arise when developing opportunistic applications such as: i) data dissemination of global information in a small quasi-uniform environment (queues duration and subjective evaluations of attractions), ii) punctual data exchange based on shared interests (text opinions on specific games), and iii) usage of different dissemination policies depending on the context (the user is in queue, or walking around the park).

II. Background and motivation

Smart nodes spread user generated contents and context information by means of opportunistic connections: by providing a direct exchange among peers (i.e. without passing through an infrastructured network topology) the user is enabled to exchange information by simply using the available devices, and nothing more.

Distributed issues should be locally solved as much as possible, i.e. in the local node and possibly with no interactions to avoid additional (time-consuming and unreliable) network overhead.

This is quite a simplification in terms of complexity, setup expenditure, system maintenance, and has many other advantages. By contrast, all communications must be delay-tolerant, more robust, and allow for unreliability; this means that special expedients and additional requirements must be taken into account.

As a matter of fact, since communication must be delay-tolerant, connection reliability should not be a key feature. This, in short, leads to data exchange minimisation in order to make the whole distributed system work better, since it implicitly gets more reliable and responsive. In other words, applications should be designed in the most decentralised and autonomous way: data analysis should be done locally, and decisions (such as data forwarding) should be performed as much autonomously as possible.

The aim of this document is to show another application field that might benefit from opportunistic environments, while further investigating some approaches to data dissemination in well defined contexts arising from a specific scenario: an amusement park.

Our case study runs in a closed environment, with quasi-uniform and sparse distribution of people freely moving around. Inside an amusement park we can fairly suppose that any user will visit all the park areas sooner or later, i.e. simple but still realistic dissemination protocols should be fine here.

Despite this, attractions themselves are characterised by quasi-uniform and dense distribution of people standing in queues.

We eventually tried to figure out all the issues arising from the mixture of both scenarios, what dissemination protocol is best suited for a context or the other, and what happens when a user gets in or out a queue of a game.

III. Dissemination policies

Users walking around the park may move in the whole area, their density be possibly sparse, their location may not be feasibly detected with low effort or be easily predictable: this leads to a specialised dissemination policy, that should benefit from other context-aware information, such as users willingness or ability to move around.

In other words we need to find good nodes that can be fastly elected as the local best carriers: in non-queuing contexts we can suppose for instance that younger people move faster and broader than elderly ones; and this is the simple choice done in our study.

We introduce an interesting data-dissemination approach, which spreads messages by a controlled delayed-broadcast algorithm and bases its dissemination policy on an utility function: as a variant of the original Spray and Wait algorithm [SPY05], it not only delivers a message to a single receiver, but extends the delivery to every node in a controlled delayed-broadcast fashion with a distributed (but still locally-decidable) method for selecting the preferred carrier(s). We will refer to this algorithm as “Spread And Wait”.

By contrast, the amusement park is of course a place that allows for at least another kind of dissemination policy: people in a queue are quite static, squeezed, and well localised. This is almost opposite to the previous scenario, since in a queue user movements can be easily predicted and held in a small range. Because of the devices density in such range, data must be carefully spread to avoid disastrous collisions: nevertheless, if properly exploited, the crowdedness may be an advantage since all data transportation may be done through a simple flooding (but enhanced) mechanism.

We could implement a variant of a multi-hop protocol with a real infrastructure, good in dense networks, such as MinT as seen in [WOO03], or LEACH described in [HEI00]: the latter works better if the cluster heads have higher transmission range than others. This is not the case in CAMEO, since it currently talks by the Wi-Fi Direct protocol, thus all the devices behave almost the same.

Whatever the (flat) multi-hop protocol we could use, a real routing protocol may be a valid choice, because we could elect the routing nodes as the carriers for data dissemination, since they know the routes to any node they manage, and use them to directly spread the copies.

We instead propose a simpler implementation, based on a probabilistic transmission approach, whose density function depends on the cardinality of the neighbourhood. As an advantage, this approach does not need any infrastructured network, and it’s easy to implement. We called it the “Wave Multi-Hop”. Details will come shortly in the next sections.

At last there is another feature we wanted to show: the exchange of opinions (that can be seen as user generated text content) to whom has the same interests. Interests are declared in the application context, and CAMEO helps in spreading these information once the devices get in contact; this resembles a simple one-hop opportunistic data exchange, focused on and filtered by interests; we will refer to this as “One-hop Content Delivery” algorithm from now on.

For the sake of simplicity, we assumed that interests are simply represented by the attractions of the park; so, for example, people interested in the Roller Coaster will get opinions from other users about it during their intercontact time. Of course a more sophisticated algorithm may be developed, for example by setting an interest on games suitable for people at least 14 y.o., shorter than 1.90m. But from a theoretical point of view, this is really analogous to the simpler requirements shown above and in no way constitutes a limitation to the results obtained, for it’s basically a matter of user interface.

To recap, we have two data dissemination algorithms for each device, running alternatively: the “Spread And Wait” runs when the user is walking around the park, while the “Wave Multi-Hop” runs when the user decides to stand in a queue in order to join an attraction. At the same time the “One-Hop Content Delivery” is run all the time to share opinions about common interests among close users.

IV. DroidPark

In order to do what exposed above, we are proposing DroidPark, an application built for such well defined environmental characteristics. The CAMEO middleware helps the whole data dissemination process by managing the opportunistic communications, thus effectively allowing for a novel way of exchanging data among nodes.

We chose to setup a simple emulation with only four attractions: the Roller Coaster, the Big Wheel, the Canoe Game, and the Fast Food (note that a restaurant, as it is included here, is characterised by issues similar to a regular attraction). There is also a sparse area that connects all the attractions - and where people can freely walk around: we call it the Walking Area.

When the user is not queued to any attraction, she stands in the Walking Area; this is where the Spread And Wait algorithm takes place. On the other hand, when she decides to join one of the four queues, the Wave Multi-Hop algorithm applies.

The way the user is detected as standing in a queue is out of scope here. There can be some kind of heuristics, such as the user waiting close to an attraction for more than a given time - even made more robust by some hysteresis mechanism to avoid false positives - that switches the user state automatically.

For the sake of this case study, we simply assume that a manual toggle (represented by the iconic button of an attraction in the main screen) fulfills our needs. Hence, the user presses a given button when she gets in the corresponding queue; this triggers a start signal to a local device queue timer, which holds the queue duration for the related game. Then she presses the button again as she gets into the attraction, thus leaving the queue: when there is such a change in the local context, the timer is stopped and the dissemination algorithm is switched accordingly.

The time elapsed between the start and stop signals is the most updated queue duration for each attraction, and it must be sent over the opportunistic network in place of the (hereon outdated) information sent previously. We call this message “the Queue Message”.

The device immediately injects the Queue Message in the network by using the Spread And Wait algorithm. It also chooses two or more youngest neighbours as carriers that will apply, in turn, their own dissemination algorithm. Further details about this will come shortly.

The user may, at anytime, express a subjective rating of the attractions in the park, from one to five stars; the message carrying the user evaluations for the games is called “the Rating Message”. Users will - sooner or later - receive all rating messages from anyone, and for each attraction. Devices must hold all the values in order to locally calculate and show the average rating for each game; edits are of course possible, and a subsequent Rating Message for a user will substitute the outdated one.

Because Rating and Queue Messages have similar properties, they may be handled in a similar way: both must be delivered to as many devices as possible; only the most recent message is kept, and both algorithms should converge as soon as possible. In the case of Queue Messages, each device must hold a message for each attraction (in our case, a maximum of four most recent messages), regardless the user that has originated it; by contrast, the Rating Messages must take into account the originating user to correctly maintain the most recent rating for each user and attraction, then compute the average value.

Last but not least, the One-Hop Content Delivery algorithm - responsible for disseminating the user textual opinions about games - is applied anytime, for it’s not affected by the local context of the user.

V. The Spread and Wait Algorithm

This algorithm is most suitable for uniform, dynamic and sparse environments, such as the Walking Area, where people is free to move around and join a queue. Copies of the message are sent to all the neighbours of a device, but only the youngest ones are chosen as the carriers in order to spread a given amount of copies. We expect that younger people move faster and broader, and this is why we chose the youngest of the neighbours as the dissemination utility function.

The basic control idea for the number of spread copies is taken from Spray And Wait but, instead of carrying a message to single destination, we carry Rating and Queue Messages that must be delivered to any device. Basically, if there are *m* users in the park, *m* copies should suffice, under the hypothesis that - sooner or later - everybody will meet any other in the park.

This means that every device may be elected as a carrier since it holds a copy of the message in its cache forever, but only the locally-youngest people are entitled to perform the effective dissemination. Actually, to allow for path redundancy, two people (the two youngest ones in the local neighbourhood of a node) are entitled to become carriers.

This is how the algorithm works: we first suppose that each message is uniquely identified. The Queue message is identified by the subject game and the time of issuing (timestamp).

A queue message Q_a is more recent than Q_a’ iff Q_a and Q_a’ regard the same attraction *a*, but Q_a has a greater timestamp than Q_a’.

The same holds for the Rating message, but this time we also need to take into account the user id besides the subject game in order to identify it. The rating message is denoted as $R_{a,u.}$

Generally speaking, M and M’ represent two Queue (or Rating) messages for the same game (and user), with possibly different timestamps. Given *f* carriers, and *n* number of copies to be spread, each node *D* must then implement the following general algorithm:

Phase 1: receiving a message

- a device *D* receives (or generates) a message *M*, along with a number of copies *n* to be spread;
- if D holds a more recent M’, discard M (throw away its copies), then exit (don't run phase 2);
- otherwise if M is more recent than M’ (or no M’ exists), store M in the cache (throw away the old message if any); notify M to the application and set the number of copies to be spread to:

  $n' = n - 1$

  (decrement by 1 since the onboard application has just received its own copy);
- otherwise the timestamps are the same. Set the number of copies to be spread to:

  $n'' = n' + n$

  (add received copies to the amount of already present copies). Then exit (no phase 2).

Phase 2: spreading copies

- Be *S* the neighbourhood of D, and |*S*| its cardinality. Choose *F* subset of *S* by an utility function such that *F* holds the youngest *f* users of *S*; be their devices respectively D^1 to D^f;
- Send a message to each node belonging to *S\F*, each with its own *n* set to 1 (these are the non-carrier nodes);
- Assign D^1 to D^f a congruent number of copies, by splitting the message budget held in *D* into |*F*| integer parts. D^1 to D^f are elected as the carrier nodes and each of them will receive its own number of copies to be spread as:
- $n' = \lfloor \frac{n - |S|}{f + 1} \rfloor$

  while D will keep:

  $n'' = n - n' \cdot f$

  copies of the message;
- Send M to D^1 to D^f along with their own number of copies *n’*.

Whenever a neighbour gets in contact, do the following:

Phase N: contacting a new neighbour

- If the new neighbour is older than the two youngest users, just send her one copy of each message M in the cache if its *n* > 0, and decrement *n* by 1;
- Otherwise send her *n'* number of copies to be spread, calculated as:

  $n' = \lfloor \frac{n+1}{2} \rfloor$

  along with the message M, and set the remaining local number of copies to be spread to:

  $n'' = n - n'$

The higher is the number *f* of carriers, the higher is the path redundancy; the number *n* of copies should be proportional and not inferior to the estimated number of users. For our study we used *f=2* and *n=1000*.

VI. The Wave Multi-Hop Algorithm

This algorithm is most suitable for uniform, compact, and dense environments, such as the Queue Area for each attraction. The idea behind it is the following: an outgoing message is sent in a limited broadcast to any device in the neighbourhood; zero or more message receivers are probabilistically chosen as carriers that will, in turn, flood it again in their own neighbourhood.

Of course the more the users are densely collapsed, the higher is the probability of spreading the information to both ends of the queue. Conversely, the lower is the number of devices that can be found in the neighbourhood, the less is the probability of reaching the end of the queue.

This means that, in a sparse neighbourhood, each node may have a more crucial role as a carrier than in a denser one. If the probability of flooding the information is too high, we may end up in overcrowding the radio channel, discharging all batteries uselessly; in the worst case, we could even not succeed in sending a single information. Conversely, if the probability is too low, the message wave could get interrupted, thus isolating the users in two (or more) partitions without reaching the queue extremities: the 2-partition case may happen when the queue is actually a plain line, while more partitions may be obtained if the queue is distributed over a more complex geometrical shape.

It gets clear that if the probability is constant or simply unrelated to the neighbourhood density, the dissemination could irreversibly fail.

Hence, a possible solution to this problem is basically to vary the probability of re-sending the messages to the neighbours, in such a way that the probability of flooding raises as the neighbourhood get sparser. In our implementation the chosen density function is as simple as:

$$p(s) = \frac{1}{s}$$

being *s* the number of neighbours. This generates a sort of wave in the queue, and that’s why we have called this algorithm the “Wave Multi-Hop”.

To keep the number of copies in the whole system consistent, we need to track - as efficiently as possible - the allowed number of copies of each message. This can be partially achieved by applying the same receiving phase as in the Spread And Wait algorithm (Phase 1).

Given the quasi-static nature of the queue, a simple algorithm that performs data dissemination could be also similar to the Spread And Wait, but with the following variant:

Phase 2: spreading copies

- Be *S* the neighbourhood of *D*, and |*S*| its cardinality. Choose a subset of S as the carriers *F*, such that each element is chosen with a probabilistic utility function *p* such that *p(*|*S*|*)* = *1/*|*S*|. Be |F| the cardinality of F and be its device elements D^1 to $D^{|F|}$;
- Broadcast M to each node belonging to *S*\F, each with its own number of copies *n* set to 0;
- Assign D^1 ... $D^{|F|}$ a congruent number of copies, by splitting them in |F| equal integer parts. Be this number n' such that:

$$n'=\lfloor \frac{n}{|F|} \rfloor$$

- Send M to D^1 ... $D^{|F|}$, each with its own *number of copies* set to *n'*;
- hold in D the possible remainder *n''* calculated as follows:

$$n''=n-|F|\cdot n'$$

Whenever a new neighbour gets in contact, do nothing. This is a strict countermeasure to avoid loops and snowball effects that reverberate over nodes because of a possible (and almost certain) cyclic topology.

Please note that this algorithm differs from the Spray and Wait in several ways: i) the utility function (the youngest users vs. a density function), ii) the number of copies sent to the carriers (the latter "pushes" the copies forward in a stronger way), and iii) the phase-N (when a neightbour is coming in, the first tries to push all cached messages, whilst the second one does nothing). These changes are mainly due to the different nature of the respective contexts: a sparse and dynamic context, against a dense and quasi-static one.

Another noticeable change with respect to the previous algorithm is that we don't count the copies sent in broadcast to non-carriers: this emulated broadcast is necessary to disseminate the messages over normal nodes, that would not get them otherwise.

This is an additional cost due to the fact that in our implementation a message broadcast is emulated by unicast connections through CAMEO, instead of being listened in promiscuous mode. The unicast technique produces an overhead due to the multiplication of message copies, thus relaxing the consistency of *n* to a "not less than" over a "strictly equal to" condition.

Implementing the local broadcast as a custom UDP in place of the multicast emulation done with unicast connection would, for example, allow for a less expensive message dispatching, since no emulation overhead would occur.

Beside the fact that this algorithm is far from being optimal in terms of battery consumption and data transmissions, it does not need to set up and maintain any network topology, nor it has any distributed state to be held (such as the link state based algorithms), allowing for a simple and robust implementation.

VII. Conclusions

We showed how an apparently simple environment, an amusement park, holds many issues to be solved and, at the same time, how a novel approach such as an opportunistic communication middleware may come in handy. Moreover, the realistic but still closed context allows ti apply simple dissemination protocols in a real context.

An application like the one we have shown does not make use of infrastructured networking at all and keeps running autonomously, with no dedicated intervention or external hardware, except for the devices participating in the service, and could be adopted by some enterprises once fully customised and tested. Of course, as this is only a simplified proof-of-concept, a big effort must still be done to adapt the application to the specific park needs.

VIII. Future Work

The best redundancy value has been set to two, to try and avoid high delays (e.g. the user takes a nap in the park). Moreover, the number of copies has been supposed to be equal to the estimated users in the park. But what if the user quits the Park? A higher value should be recommended, but which value is to be considered "good"? The chosen values may be suboptimal, and it may be an interesting field for further investigations.

Wave multi-hop should be tested against the performance of other quoted algorithms, such as minT or LEACH, in order to determine the most suited approach that better fits the chosen context and possibly show the goodness of the proposed approach.

The One-hop Content delivery algorithm, used here to spread user opinions about the games to the direct neighbours, may be enhanced to a multi-hop delivery protocol by some publish-subscribe mechanism as seen for example in SPIN, introduced and described in [HEI99].

Spread and Wait could be compared to other algorithms, or even modified by taking from them some ideas: for instance, it could be compared to proper variants of SimBet [DAL09], BubbleRap [HUI11] or ContentPlace [BOL09].

Implementing some form of neighbourhood limited broadcast (also known as all-neighbours multicast) in CAMEO would allow applications to benefit of new optimised ways to exchange data, and make applications more effective with ease.

References

[ARN11] V. Arnaboldi, M. Conti, and F. Delmastro, “Implementation of CAMEO: a Context-Aware Middleware for Opportunistic Mobile Social Networks”, in IEEE WOWMOM 2011.

[ARN13] V. Arnaboldi, M. Conti, F. Delmastro, “A novel context-aware middleware for opportunistic mobile social networks”, 2013, http://dx.doi.org/10.1016/j.pmcj.2013.09.010

[ARN14] V. Arnaboldi, M. Conti, F. Delmastro, G. Minutiello, L. Ricci, IIT-CNR and Computer Science dept. at the University of Pisa, “DroidOppPathFinder: a context and social-aware Path Recommender System Based on Opportunistic Sensing”, 2014.

[BOL09] C. Boldrini, M. Conti, A. Passarella, “Design and performance evaluation of ContentPlace, a social-aware data dissemination system for opportunistic networks”, IIT-CNR, 2009.

[DAL09] Elizabeth M. Daly, Mads Haahr: Social Network Analysis for Information Flow in Disconnected Delay--Tolerant MANETs. IEEE Trans. Mob. Comput. 8(5): 606--621 (2009)

[HEI99] W.R. Heinzelman, J. Kulik, and H. Balakrishan, “Adaptive Protocols for Information Dissemination in Wireless Sensor Networks”, Proc. ACM MobiCom’99, pp. 174-185, 1999.

[HEI00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-Efficient Communication Protocol for Wireless Microsensor Networks”, Proc. Hawaii International Conference on System Sciences (HICSS 2000), January, 2000.

[HUI11] Pan Hui, “Bubble Rap: Social-Based Forwarding in Delay-Tolerant Networks”, IEEE Transactions on Mobile Computing, 2011.

[SPY05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: Efficient routing in intermittently connected mobile networks. In Proceedings of ACM WDTN, 2005.

[WOO03] A. Woo, T. Tong, and D. Culler, “Taming the underlying challenges of reliable multihop routing in sensor networks”, Proc. ACM SenSys 03, pp: 14-27, Los Angeles, CA, November 2003.