Where Is The Crowd?: Crowdedness Detection Scheme for Mobile Crowdsensing Applications

Desheng Zhang[1], Tian He[1,2], Fan Ye[3], Raghu k Ganti[3] and Hui Lei[3]

[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, PRC

[2]Department of Computer Science and Engineering, University of Minnesota, USA

[3]IBM T. J. Watson Research Center, USA

ds.zhang@siat.ac.cn, tianhe@cs.umn.edu, fanye@us.ibm.com, rganti@us.ibm.com, hlei@us.ibm.com

Introduction

Recently, the mobile devices, such as smartphones, serve as the key computing and communication devices for people's daily life [1]. A unique characteristic in these applications is that the population of sensing devices is much more crowded than mobile wireless sensor networks (MWSN), which gives rise to a novel area of research, termed Mobile Crowdsensing (MCS).

In MCS applications, how to efficiently detect the crowdedness level is an initial yet key step for any further extensions. The crowdedness detection in MCS is similar to the neighbor discovery [2] [3] in MWSN, but it only requires an accurate estimation on the neighbors rather than the specific number of the neighbors. However, this relaxation does not make the crowdedness detection in MCS easier than the neighbor discovery in MWSN, because MCS poses three new constraints. **(1)** The population of mobile devices is highly dynamic. **(2)** In MCS the mobile devices work autonomously, *i.e.*, a device cannot assume that these duty cycles are chosen from a specific set. **(3)** The crowdedness detection is often initiated by the users, so a prompt response to the users is more desired in MCS than in MWSN. These new challenges make existing neighbor discovery schemes [2] [3] unsuitable for the crowdedness detection in MCS.

Based on above observations, in this poster we propose a new crowdedness detection scheme call *Crowd*, which serves as a building block for various MCS applications. Under Crowd, the detecting device adapts its duty cycle according the number of neighbors currently found as so to find the dynamic development of the number of beacons, which is sent by the neighbors for the estimation of crowdedness. This poster hopes to contribute as follows: **(1)** This poster makes an early attempt to detect crowdedness level of a device's neighborhood in mobile crowdsensing applications. **(2)** This paper proposes a crowdedness detection scheme, called Crowd, with practical constraints of mobile crowdsensing in mind. **(3)** This paper presents a mobile crowdsensing application, called *where is the crowd* (WIC), which leverages Crowd to make a suggestion for the users to avoid the crowd.

Related Work

In a duty cycled network, to find all neighbors, a node has to wake up in several active slots in several periods until all neighbors wake up in the same active slots with at least once. However, this straightforward method involves too much latency. To reduce this latency, some advanced schemes are proposed such as Disco [2] and U-connect [3] to set nodes' duty cycles via certain mathematical theorems, which ensures that the time of two nodes encountering is reduced and bounded. The related work is compared in Table.1. However, these schemes fail to detect the crowdedness level of the devices in mobile crowdsensing applications. More importantly, since the duty cycles are generally set according to the battery availabilities in the devices, they cannot be set by some mathematical theorems, *i.e.*, Chinese Remainder Theorem. Nevertheless, the crowdedness detection offers us a leverage that only an accurate estimation of population will suffice, which is a unique opportunity to design the crowdedness detection.

Algorithm 1: **Crowd Scheme**

If (sleeping timer fired){ turn off radio;}
If (sample timer fired){ turn on radio; set up sleeping timer; sample the channel within a slot; compute the new sample DC by Eq.(1); set up sample timer by sample DC ;}
If (active timer fired){ turn on radio; set up sleeping timer; beacon with neighbor table; compute crowdedness level by Eq.(2); set up active timer by original DC ;}

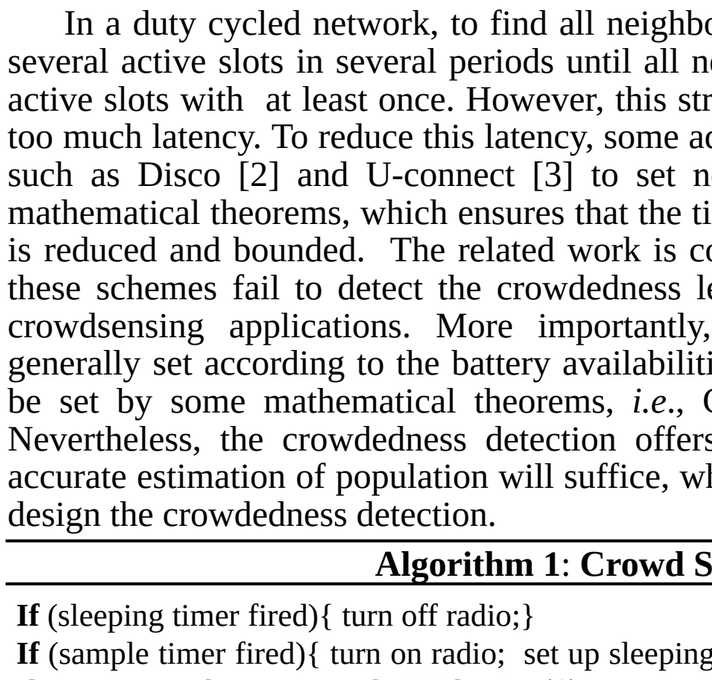

Fig.1 Crowd in the airport

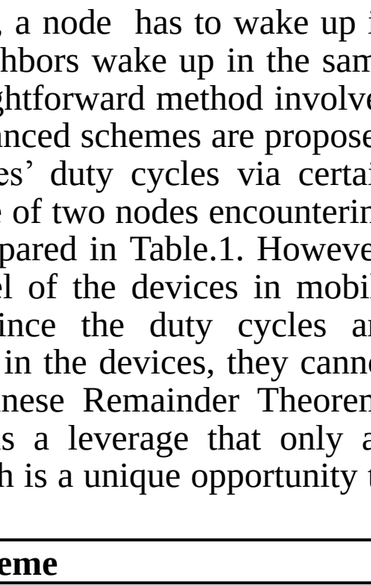

Fig.2 Map on the screen

Main Design

In practical applications, the most of neighbors of a node s will wake up non-uniformly. Crowd is proposed to detect the crowdedness level in this scenario where we can still assume that during some continuous slots (called Slot Set) in one period, the wake-up pattern of s's neighbors of is uniform. Thereby, we can obtain the crowdedness level of s's neighborhood by three steps. (1) Finding every starting slot (denoted as $T_1, T_2,..., T_M$) of all the slot sets (denoted as $S_1, S_2,..., S_M$) in one period. (2) Obtaining the numbers of wake-up neighbors (denoted as $N_1, N_2,..., N_M$) of s in $T_1, T_2,..., T_M$. (3) The crowdedness level N^* can be obtained by $N^* = \sum_{i=1}^{M} N_i \cdot |S_i|$. Under the duty cycle operation, however, s cannot find $T_1, T_2,..., T_M$ due to the sleeping slots. So, Crowd tries to approximately sample in every slot set, *i.e.*, $S_1, S_2,..., S_M$, with a self adaptive sample duty cycle operation.

The high level idea of Crowd is simple. s adaptively increases it duty cycle to sample slots more frequently if the numbers of the neighbors recently discovered in different slots have a large gap (which indicates a inter-set sample) or *vice versa*. Let $2^d/100$ approximately be the original duty cycle (indicated as DC) of s; Let N_i be the number of the neighbors discovered by s in its ith active slots; then the sample duty cycle (indicated as SDC) after $(k-1)$th sample is given by

$$\text{SDC} = \begin{cases} 1, & if\ (2^d + 2^{\sum_{i=1}^{k-1}|N_i - N_{i-1}|})/100 \geq 1 \\ (2^d + 2^{\sum_{i=1}^{k-1}|N_i - N_{i-1}|})/100, & if\ (2^d + 2^{\sum_{i=1}^{k-1}|N_i - N_{i-1}|})/100 < 1 \end{cases}. \quad (1)$$

Given that the number of s's neighbors waking up in the interval $[T_i, T_{i+1}]$ is Poisson distributed with mean λ_{i+1}^i, the expected number (denoted as N_{i+1}^i) of the neighbors waking up during $[T_i, T_{i+1}]$ is given by $N_{i+1}^i = \lambda_{i+1}^i \cdot (T_{i+1} - T_i)/t$, where t is the length of a slot, and $\lambda_{i+1}^i = (N_i + N_{i+1})/2$. Therefore, if let times of s waking up in recent 100 slots be K,

$$N^* = \sum_{i=1}^{K} [N_i + \frac{(N_i + N_{i-1})}{2} \cdot \frac{(T_i - T_{i-1})}{t}]. \quad (2)$$

The detailed scheme of Crowd is given in Algorithm 1.

Application

We propose a simple yet representative MCS application called "where is the crowd" (WIC). In WIC, the user is required to install an app on the mobile device. When the user enters public areas, *e.g.*, airport as in Fig.1, and wants to find a place with a desired crowdedness level, the following steps describe WIC. **(1)** The user turns on WIC on the mobile device such as a smart phone. **(2)** WIC employs Crowd to detect the beacons sent by other mobile devices with WIFI radio. **(3)** WIC shows the crowdedness level of the user's neighborhood on the device's screen by a colored circle, where the color represents the crowdedness level, as shown by Fig.2. **(4)** By collecting the neighbor information of the user's one-hop neighbors (sent by the neighbors in the beacon), WIC also shows the crowdedness level of the user's representative neighbors (*e.g.*, WIFI base stations) on the screen by several colored circles, as shown by Fig.2. **(5)** By the WIFI base stations on the public areas with location information as shown by Fig.1, WIC can turn those colored circles into a crowdedness level map as Fig.2. **(6)** The user can follow this map on the device's screen to the place with a desired crowdedness level.

Conclusions

In this paper, we propose a crowdedness detection scheme called Crowd, serving as a building block for the MCS applications. Different from existing neighbor discovery schemes, our work enables the user to accurately detect the crowdedness of the neighborhood within an acceptable response time. We also present an actual mobile crowdsensing application, called "Where is the crowd", which takes advantages of Crowd. In this application, the user can be informed with the crowdedness level of his or her neighborhood by a virtual map on his mobile device screen, and can make a better decision. In the future work, we plan to further take advantages of the neighbor tables to reduce the response time. In addition, an implement of Crowd and "where is the crowd" on a well known platform with prototype mobile devices is another necessary step to bring mobile crowdsensing applications into reality.

Bibliography

1. Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. 2010. A survey of mobile phone sensing. Comm. Mag. 48, 9 (September 2010), 140-150.
2. P. Dutta and D. Culler, "Practical asynchronous neighbor discovery andrendezvous for mobile sensing applications," in SenSys'08.
3. A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, "U-connect: a lowlatency energy-efficient asynchronous neighbor discovery protocol," in IPSN'10.

Table.1 Related Work

Name	Schedule	Power consumptions	Latency	Power Latency Product (Metric)	Asymmetric duty cycles	Pros	Cons
Optimal Mobihoc'03	Centralized optimal Solution	$P_o = \frac{\sqrt{L - \frac{3}{4}} + \frac{1}{2}}{L}$	L	$\Lambda_o = \sqrt{L - \frac{3}{4}} + \frac{1}{2}$	NO	1. Serve as a Benchmark	1. Not practical, just in theory
Birthday Mobihoc'01	Probabilistical scheme: in every slot, a node in the network transmit, listen, or sleep with different probability.	N/A	N/A	N/A	N/A	1. Better average latency 2. Simple and flexible	1. Aimed for static networks 2. Aperiodic and unpredictable rendezvous
Quorum INFOCOM'02	$\varphi_q(m,t) = \begin{cases} 1, & \text{if } [t]_{\sqrt{L}} = c, \text{or } r \leq \frac{[t]_{\sqrt{L}}}{\sqrt{L}} < (r+1) \\ 0, & \text{otherwise} \end{cases}$	$P_q = \frac{2\sqrt{L} - 1}{L}$	$L_q^S = \sqrt{L}^2$	$\Lambda_{q=}2\sqrt{L} - 1$	$L_q^A = max\{m_i^2, m_j^2\}$	1. Better worst-case latency than B 2. Aimed for both static and mobile networks	1. m is a global parameter 2. only two different quorum systems are allowed.
Disco Sensys'08	$\varphi_d(m,t) = \begin{cases} 1, & \text{if } [t]_{P1} = 0 \text{ or } [t]_{P2} = 0 \\ 0, & \text{otherwise} \end{cases}$	$P_d = \frac{p1 + p2}{p1 * p2}$	$L_d^S = p1 * p2$	$\Lambda_d \geq 2\sqrt{L}$	$L_d^A = p1 * p2$	1.Better Metric than Q 2.Bounded latency	1. Require two primes 2. Either SYM or ASYM, not both.
U-connect IPSN'10	$\varphi_u(m,t) = \begin{cases} 1, & \text{if } [t]_P = 0 \text{ or } 0 \leq [t]_{p^2} < (P+1)/2 \\ 0, & \text{otherwise} \end{cases}$	$P_u = \frac{3p + 1}{2p^2}$	$L_u^S = p^2$	$\Lambda_u = \frac{3\sqrt{L} + 1}{2}$	$L_u^A = p1 * p2$	1. Better Metric than D 2. Only one prime	1. Require primes 2. Coarse Granularity on DC
Searchlight Mobicom'10	$\varphi_s(m,t) = \begin{cases} 1, & \text{if } [t]_P = 0 \text{ and } [t]_p = i, in\ phase\ i \\ 0, & \text{otherwise} \end{cases}$	$P_s = \frac{\frac{p}{2} + \frac{p}{2}}{p * \frac{p}{2}} = \frac{2}{p}$	$L_s^S = \frac{p^2}{2}$	$\Lambda_s = \sqrt{2L}$	$L_s^A = p1 * p2$	1. Better Metric than U 2. No primes in SYM 3. Fine granularity SYM DC	1.Require primes in ASYM 2.Coarse Granularity DC in ASYM