DiVote: A Distributed Voting Protocol for Mobile Device-to-Device Communication

Peter Danielis, Sylvia T. Kouyoumdjieva, and Gunnar Karlsson
ACCESS Linnaeus Center, School of Electrical Engineering
KTH Royal Institute of Technology, Stockholm, Sweden
Email: {pdanieli, stkou, gk}@kth.se

***Abstract*—Distributed aggregation algorithms have traditionally been applied to environments with no or rather low rates of node churn. The proliferation of mobile devices in recent years introduces high mobility and node churn to these environments, thus imposing a new dimension on the problem of distributed aggregation in terms of scalability and convergence speed. To address this, we present DiVote, a distributed voting protocol for mobile device-to-device communication. We investigate a particular use case, in which pedestrians equipped with mobile phones roam around in an urban area and participate in a distributed yes/no poll, which has both spatial and temporal relevance to the community. Each node casts a vote and collects votes from other participants in the system whenever in communication range; votes are immediately integrated into a local estimate. The objective of DiVote is to produce a precise mapping of the local estimate to the anticipated global voting result while preserving node privacy. Since mobile devices may have limited resources allocated for mobile sensing activities, DiVote utilizes D-GAP compression. We evaluate the proposed protocol via extensive trace-driven simulations of realistic pedestrian behavior, and demonstrate that it scales well with the number of nodes in the system. Furthermore, in densely populated areas the local estimate of participants does not deviate by more than 3 % from the global result. Finally, in certain scenarios the achievable compression rate of DiVote is at least 19 % for realistic vote distributions.**

I. Introduction

Distributed tasks and computations, e.g., to estimate the average or sum of a set of values, are often conducted based on inputs supplied by collaborative users. Such aggregation functions are of high importance in large-scale distributed systems where there is a need to compute global system properties [1].

In this paper, we focus on a specific class of distributed tasks, namely distributed voting in the context of *urban polling*. Potential urban polling applications collect and process information on locally-relevant questions and provide users in a community with answers to them [2]. Such questions can relate to urban planning optimization ("Is the switching behavior of this traffic light fast enough?") or to safety in a given region ("Do you feel safe in this area?").

In general, the information obtained during a poll can be processed either in a centralized or in a decentralized manner. Centralized processing requires nodes to submit their votes to a central entity. However, this approach lacks scalability and poses privacy concerns as users might in general not want their votes to be seen by a central entity [3]. In particular, this may be of high importance in countries where people do not trust authorities. Contrary, decentralized (or distributed) processing requires nodes to compute local estimates of the result based on partial system knowledge. As opposed to conventional distributed processing scenarios where nodes are considered to be static or semi-static [4], in this work we examine scenarios, in which nodes exhibit high mobility. We consider a node to be a pedestrian carrying some device equipped with a wireless communication interface such as a mobile phone. We rely on device-to-device communication for disseminating votes among participants in the poll. Mobile nodes opportunistically exchange data whenever they come in direct communication range [5]. For the purpose of urban polling, this data comprises voting information conveyed by means of broadcast messages and nodes immediately update their local estimate upon reception of a message.

In the context of distributed voting in urban environments, a distributed voting protocol needs to comply with the following requirements: (1) be scalable, (2) have fast convergence and high accuracy, and (3) preserve node privacy. Thus, in this work we present DiVote, a distributed voting protocol for mobile device-to-device communication, which provides all of the above characteristics. The main contributions are:

- We show that DiVote is suitable for operation in dynamic environments with high node mobility. DiVote makes use of the benefits of D-GAP compression for *scalability* of the protocol [6]. Furthermore, DiVote *preserves node privacy* by applying a cryptographic hash function to user identities.
- We perform extensive trace-driven simulations using realistic pedestrian mobility. We show that DiVote scales well with the number of nodes in the system. Furthermore, DiVote demonstrates both *fast convergence* and *high accuracy*, with local estimates deviating at most by 3 % from the global value in dense scenarios.
- DiVote is able to achieve at least *19 % compression rate* for realistic vote distributions, which makes it appropriate for execution on mobile devices with limited storage capabilities or with restrictions on the memory to be used.
- DiVote exhibits *low processing load* at the application layer as it requires only a fraction of the received broadcast messages (34 % in dense scenarios and even less in sparser scenarios) to be processed to achieve accurate

978-0-9883045-1-2/16 $31.00 © 2016 ITC

local estimates.

The remainder of this paper is organized as follows: In Section II, we provide an overview of previous work in the field of distributed aggregation, and reason why current solutions are not suitable for operation in mobile environments. Section III introduces DiVote, a distributed voting protocol for mobile device-to-device communication. Section IV outlines the evaluation scenarios, and Section V presents results from realistic pedestrian mobility scenarios. Finally, we conclude the study in Section VI, and present directions for future work.

II. Related Work

Distributed voting belongs to the class of distributed aggregation problems. Distributed aggregation in general comprises computations such as sum, average, minimum, or maximum over unreliable networks, in which no central entity is accessible or required. There are two main paradigms to address this problem, namely *gossip-based* and *tree-based* aggregation. Tree-based aggregation protocols have been shown to perform poorly in dynamic environments with high levels of churn [4], therefore for the rest of this section we focus on discussing the applicability of state-of-the-art gossip-based protocols to our scenario.

Gossip-based aggregation protocols that react to environment changes can be broadly classified in *restarted* and *bookkeeping* protocols.

We first discuss *restarted protocols*, many of which are based on the the push-sum algorithm presented in [7]. The basic idea of the algorithm is that nodes periodically exchange stored values with their neighbors and are thus able to compute the sum or average of all values. However, the main assumption in [7] is that values stay unchanged over time. In [8], the authors propose executing the push-sum algorithm in epochs to reflect changes in the network; the protocol is restarted after each epoch. The distributed random grouping algorithm (DRG) is proposed in [9]. In this algorithm, some nodes can periodically become group leaders and then determine group members by exchanging messages in a handshake manner. The establishment of several roles as well as the information exchange by means of a handshake makes this algorithm too slow to react on changes induced by moving nodes in our envisaged scenario. Another gossip-based distribution estimation approach is suggested in [10]. This algorithm exchanges and merges lists consisting of pairs with value and respective counter between nodes. However, duplicates may occur when applying this approach, which distorts the computed estimate. Most of the restarted gossip algorithms show this shortcoming and do therefore not achieve a high accuracy. In [11], the authors tackle the data duplication problem by simultaneously executing multiple instances of the proposed protocol however the solution exhibits low accuracy [12].

Bookkeeping gossip-based protocols are able to revert changes in the nodes' states. In [13], a node saves the states on its neighbors and recovery is triggered when a node crashes or disappears. Here, a tradeoff between accuracy and protocol overhead has to be chosen so either scalability in terms of memory consumption or accuracy are decreased. In [14], the authors propose LiMoSense, an algorithm for live monitoring in dynamic sensor networks, which takes into account node churn and link failures at runtime. LiMoSense is not appropriate for dynamically changing environments, since it assumes a known set of neighbors, from which it randomly chooses a single neighbor at a time for message exchange. In [15], the authors present Flow Updating, a bookkeeping algorithm, which iteratively averages values towards the global network mean. Every node computes a flow value for each of its neighbors and stores the value in a matrix. The algorithm tries to enforce skew symmetry of this matrix. This however significantly decreases the convergence speed. To summarize, bookkeeping protocols usually require up to thousands of rounds to converge, thus they are not applicable in scenarios with high dynamics.

TABLE I
Comparison of DiVote with state-of-the-art approaches for dynamic environments.

Feature	Tree-based	Restarted gossip	Bookkeeping gossip	DiVote
Convergence speed	Low	Low-medium	Low	High
Accuracy	High	Low-medium	High	High
Scalability	Low	High	Low-medium	High
Privacy	Depends on extra measures			Yes

In Section I, we outlined the main characteristics of a protocol suitable for distributed aggregation in dynamic environments. In Table I, we compare tree-based, restarted, and bookkeeping gossip-based protocols with respect to these characteristics. We also show how our protocol DiVote compares to others in the literature. In the next section, we present DiVote in detail.

III. DiVote: A Distributed Voting Protocol

In this section, we present DiVote, a distributed voting protocol for mobile device-to-device communication. We begin by introducing some of the building blocks of the protocol, i.e., the vector compression scheme and the fundamental vector operations that are introduced by the protocol. We then present the details of the algorithm behind DiVote.

A. The need for D-GAP compression

In the context of distributed voting, each node can cast a binary vote (0 or 1) to a poll. (We note that the assumption of binary votes does not limit the reasoning to follow, and can easily be extended to any number of votes. In this paper, we only consider binary votes for the sake of brevity.) For nodes to be able to calculate the anticipated global vote, they need to keep track of the votes of other peers in their vicinity throughout their lifetime. However, keeping track of a simple moving average may result in votes being counted multiple times if a node and a peer come in communication range more

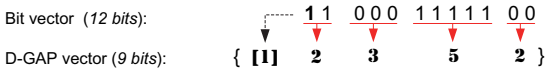


Fig. 1. An example of D-GAP compression. A bit vector of 12 bits is converted into an integer D-GAP vector of 9 bits. The leading bit in the D-GAP vector indicates if the vector starts with 0s or 1s.

than once. Furthermore, keeping track of votes in the form of a binary vector may be consuming a lot of resources, especially if the vector contains long sequences of 0s or 1s. To address these problems, in DiVote we leverage the concept of *D-GAP compression* [6]. D-GAP compression provides a compressed representation of bit vectors in the form of integer vectors (later referred to as D-GAP vectors) and can be treated as a specialized variant of run length encoding. Each integer in a D-GAP vector represents the number of consecutive 0s or 1s that are present in the bit vector at a given position. Whether the integer corresponds to a sequence of 0s or 1s is determined by the leading bit of the D-GAP vector. A leading bit of 0 shows that the first integer corresponds to a number of consecutive 0s, followed by a number of consecutive 1s and so on; a leading bit of 1 indicates the opposite behavior.

An example of converting a 12-bit vector into a 9-bit D-GAP vector is illustrated in Figure 1. We calculate the total number of bits required for representing the D-GAP vector, $\mathrm{N(DGAP)}$, as follows:

$$\mathrm{N(DGAP)} = \sum_{i=1}^{n} (\lfloor \log_2 d_i \rfloor + 1) \tag{1}$$

where d_i is the integer representation at position i of the D-GAP vector, and n is the size of the vector.

For decoding purposes, each D-GAP vector needs to have a corresponding D-GAP *mask* vector. In essence, a D-GAP mask vector is a bit vector of consecutive sequences of 0s and 1s, and each sequence indicates the boundaries of an integer in the D-GAP vector. Let us assume the following D-GAP vector: $\{[0]\ 2\ 1\}$ as an illustration of the problem. For the vector to be stored in memory it will be converted to $\{[0]\ 10\ 1\}$. However this representation alone is not enough for decoding, i.e., it is impossible to tell whether the original D-GAP vector was $\{[0]\ 2\ 1\}$ or $\{[0]\ 5\}$. To decode the D-GAP vector, we need to apply a mask vector $\{[0]\ 001\}$. The mask vector shows that the initial two bits correspond to the first integer while the third bit corresponds to the second integer. For longer D-GAP vectors, the D-GAP vector mask will iterate between sequences of consecutive 0s and sequences of consecutive 1s.

B. Operations on D-GAP vectors

A D-GAP vector is solely a data structure. Hence, we define the following three operations for DiVote that can be performed on two D-GAP vectors of arbitrary lengths: `merge`, `consolidate`, and `append`. For brevity, let us consider vectors of different lengths, DGAP_{min} and DGAP_{max}, denoting the shorter and the longer vector, respectively. Note

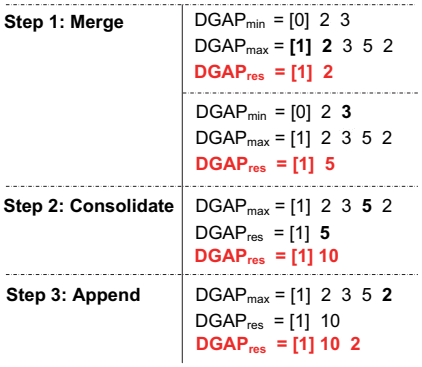


Fig. 2. Merging, consolidating, and appending D-GAP vectors. Bold integers denote positions under consideration, crossed integers are not considered, and the resulting vector is highlighted in red.

that here vector length corresponds to the *expanded bit vector length* and is defined as $\mathrm{N(BVEC)} = \mathrm{L(DGAP)} = \sum_{i=1}^{n} d_i$.

- The `merge` operation combines DGAP_{min} and DGAP_{max}[1:L(DGAP_{min})] vectors into a resulting vector DGAP_{res}. The `merge` operation is performed in an iterative manner until it reaches the end of DGAP_{min}.
- The `consolidate` operation combines the last position of DGAP_{res} with the next position of DGAP_{max} after the `merge` operation is performed. The `consolidate` operation is only performed if the integers at these two positions correspond to the same bit value.
- The `append` operation simply adds the remainder of DGAP_{max} to DGAP_{res}.

Observe that if the two input vectors are of equal lengths, the only operation that will be performed is `merge`.

Figure 2 illustrates an example of all three operations that can be performed with D-GAP vectors.

C. Functional principles of DiVote

With DiVote, each node locally keeps track of nodes it has obtained knowledge of, either directly or via other peers, as well as of their votes. This information is presented in the form of two correlated D-GAP vectors. Whenever a node first casts a vote, it adds itself to the *shared-nodes vector*, and it adds its vote to the *votes vector*. Observe that due to privacy preservation reasons, the position, in which information is stored in each of the D-GAP vectors, is determined by a cryptographic hash function such as MD5, which is calculated over a unique identifier, e.g., the node's MAC address. For instance, if for a node, which casts a vote for 1, the cryptographic hash function returns a position of 25, both its shared-nodes vector and its votes vector would be initialized with $\{[0]\ 24\ 1\}$. If the same node were to cast a vote for 0, the votes vector would instead be initialized with $\{[0]\ 25\}$. If a node is compromised, it could potentially associate the vote to the respective node during such initialization. However, in this work we assume all nodes to be trustworthy.

Note that hash functions can compute the same hash value for different identifiers, i.e., collisions can occur. This would

```
{
  "message" : {
    "node-id" : "0x00:0a:95:9d:68:16",
    "divote" : [ {
      "poll" : {
        "poll-id" : "5",
        "topic" : "sample poll",
        "shared" : [0 24 1],
        "votes" : [0 25],
        "updated" : "2016-02-17T18:30:02Z"
      }
    } ]
  }
}
```

Listing 1. An example of a DiVote message in the JSON format.

lead to positions that some nodes would share, i.e., some nodes would replace their information when shared alternately, which could falsify their local estimate. However, the collision probability of MD5, which computes 128-bits hash values, is as low as $2.7 \cdot 10^{-20}$ when, e.g., calculating hash values from 2^{32} values (assuming the birthday paradox [16]) and is therefore neglected.

Each node periodically broadcasts a beacon containing its shared-nodes vector and its votes vector to peers in its vicinity. An example of a DiVote beacon message is presented in Listing 1. Whenever a node receives information from another peer, it immediately updates both its shared-nodes vector and its votes vector following the DiVote protocol outlined in Algorithm 1. We note that the procedure in Algorithm 1 is performed simultaneously with respect to both vectors. However, here we only show how new votes are incorporated into the votes vector for the sake of brevity. Whenever a node receives a beacon from another peer in proximity, it first extracts the received information and checks in a local database whether the received vector from peer i, $\text{DGAP}_{rec}^{(i)}$, has changed (line 4). This check is currently done by looking up the `node-id` in the local database first. If the `node-id` is found, there has been prior communication with this peer and the advertised `updated` field value is compared to the previously registered `updated` field value. If they do not differ, the local vector DGAP_{loc} stays unchanged. Otherwise, DiVote consecutively executes the operations `merge` (lines 7-11), `consolidate` (lines 12-16) and `append` (line 17) in order to update the local estimate (line 18). Thus, the shared-nodes vector and the votes vector contain cumulative information of all nodes that have been shared over time, and their corresponding votes, even if these nodes have left the system. Furthermore, DiVote allows nodes to disclose peers that they have not encountered physically by propagating the knowledge accumulated by other participants in the system. This allows DiVote to achieve fast convergence and high accuracy in a distributed manner in scenarios with high levels of mobility, such as in urban environments. Finally, correlating votes and nodes is impossible by third party entities such as central collection points or compromised participants in the poll.

Algorithm 1 The DiVote Protocol

1: $\text{DGAP}_{rcv}^{(i)} \leftarrow$ received DGAP vector from node i
2: $\text{DGAP}_{loc} \leftarrow$ local DGAP vector
3: $\text{DGAP}_{res} \leftarrow$ resulting DGAP vector
4: **if** $\text{DGAP}_{rcv}^{(i)}$ changed since last beacon from node i **then**
5: $\quad \text{DGAP}_{min} = \min(\text{DGAP}_{loc}, \text{DGAP}_{rcv}^{(i)})$
6: $\quad \text{DGAP}_{max} = \max(\text{DGAP}_{loc}, \text{DGAP}_{rcv}^{(i)})$
7: $\quad$ **while** ! $\text{DGAP}_{min}.end()$ **do**
8: $\quad\quad \text{DGAP}_{res} = \text{MERGE}(\text{DGAP}_{min}, \text{DGAP}_{max})$
9: $\quad\quad rpos \leftarrow$ end position in DGAP_{res}
10: $\quad\quad mpos \leftarrow$ current position in DGAP_{max}
11: $\quad$ **end while**
12: $\quad \text{state}_{res} = (rpos \mod 2)\ xor\ \text{DGAP}_{res}[0]$
13: $\quad \text{state}_{max} = (mpos \mod 2)\ xor\ \text{DGAP}_{max}[0]$
14: $\quad$ **if** $\text{state}_{res} == \text{state}_{max}$ **then**
15: $\quad\quad \text{DGAP}_{res} = \text{CONSOLIDATE}(\text{DGAP}_{res}[rpos], \text{DGAP}_{max}[mpos])$
16: $\quad$ **end if**
17: $\quad \text{DGAP}_{res} = \text{APPEND}(\text{DGAP}_{res}, \text{DGAP}_{max}[mpos : \text{end}])$
18: $\quad \text{DGAP}_{loc} \leftarrow \text{DGAP}_{res}$
19: **end if**

IV. Evaluation Scenario

In this section, we introduce the mobility scenario as well as the simulation setup and investigated performance metrics.

A. Mobility scenario

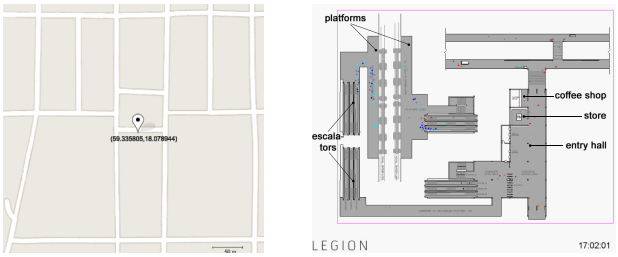
In order to realistically recreate pedestrian mobility, we use the Walkers traces [17] captured in Legion Studio [18], a commercial simulator initially developed for designing and dimensioning large-scale spaces via simulation of pedestrian behaviors. Its multi-agent pedestrian model is based on advanced analytical and empirical models which have been calibrated by measurement studies. Each simulation run results in a trace file, containing a snapshot of the positions of all nodes in the system every 0.6 s.

Fig. 3(a) and 3(b) present the scenarios considered in our evaluation: an outdoor urban scenario, modeling the Östermalm area of central Stockholm, and an indoor scenario, recreating a two-level subway station. We note that it is not possible to capture all states of human mobility with a single setup, however the scenarios are representative of typical daytime pedestrian mobility.

The Östermalm scenario consists of a grid of interconnected streets. Fourteen passages connect the observed area to the outside world. The active area, i.e., the total surface of the streets, is 5872 m^2. The nodes are constantly moving, hence the scenario can be characterized as a high mobility scenario.

The Subway station has train platforms connected via escalators to the entry-level. Nodes arrive on foot from any of five entries, or when a train arrives at the platform. The train arrivals create burstiness in the node arrivals and departures. Nodes congregate while waiting for a train at one of the platforms, or while taking a break in the store or the coffee shop at the entry level. The active area is 1921 m^2.

If not stated otherwise, the input parameters of the Östermalm and the Subway scenario result in approximately



(a) (b)

Fig. 3. Urban scenarios: (a) a grid of streets representing a part of downtown Stockholm, Östermalm, and (b) a two-level subway station.

the same mean node density of 0.1 nodes/m^2. (More information can be found in [19].)

B. Simulation setup

In our evaluation scenarios, we assume that all nodes carry devices and all are participating in the distributed poll in the area. Each node casts a binary vote $v = \{0, 1\}$ upon entry in the simulation, and votes are distributed according to a distribution $f(x)$ with a mean $\mathbb{E}(x)$.

For the evaluation, we use an implementation of an opportunistic content distribution system in the OMNeT++ simulator [20]. Each simulation run is executed in synchronous rounds of 0.6 s which corresponds to the granularity of the mobility traces we use. Nodes broadcast their shared-nodes vector and their votes vector at the beginning of each round. To avoid collisions on the wireless medium, the broadcast transmission of each node in each round is distributed uniformly at random $\mathcal{U}(0, 0.5)$ s. The transmission range is set to 10 m.

C. Performance metrics

We focus on evaluating the following performance metrics.

- *Deviation* Δ: The deviation is a measure of the accuracy of the DiVote protocol, i.e., it shows how close the local estimate of a node is to the anticipated global result. The deviation is calculated as:

$$\Delta = \left| \frac{\bar{x} - x}{x} \right| \tag{2}$$

  where $\bar{x} = \mathbb{E}(\mathrm{DGAP}_{loc})$ is the local estimate, and x is the anticipated global result depending on the nodes currently in the system.
- *Compression ratio* (CR): The compression ratio is a measure of the efficiency and scalability of the DiVote protocol in terms of resource management, i.e., how much storage space does the protocol require for performing distributed voting computations in a mobile environment. The compression ratio is calculated as:

$$\mathrm{CR} = 1 - \frac{\mathrm{N(DGAP)}}{\mathrm{N(BVEC)}} \tag{3}$$

  where $\mathrm{N(DGAP)}$ is calculated as per Eq. 1, and $\mathrm{N(BVEC)}$ is the number of bits required if the data were represented in the form of a bit vector.
- *Information overhead* (IO): The information overhead is a measure of the processing load reduction for a node in the system and therefore indicates scalability as well. It shows how many of the received broadcasts do not need to be processed. The information overhead is calculated at the application layer as:

$$\mathrm{IO} = 1 - \frac{\mathrm{n(BRC)}}{\mathrm{N(BRC)}} \tag{4}$$

  where $\mathrm{N(BRC)}$ is the total number of broadcast messages received by a node throughout its lifetime in the system, and $\mathrm{n(BRC)} \subset \mathrm{N(BRC)}$ is the number of broadcast messages that were used for updating the local estimate of the node.

V. Simulation Results

In this section, we investigate simulation results for:

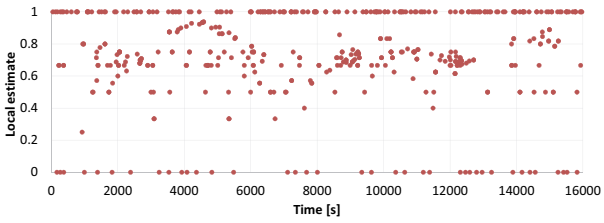
- different arrival rates λ while fixing the scenario and the distribution of nodes voting for one;
- two different scenarios while fixing the arrival rate λ and the distribution of nodes voting for one;
- two different distributions of nodes voting for one while fixing the arrival rate λ.

A. Effect of arrival rate

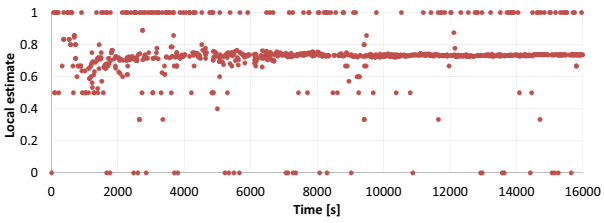
First, we present results for the Östermalm scenario for the arrival rates λ = {0.0025, 0.005 0.01, 0.07, 0.15, 0.30} nodes/s. We assume that votes are deterministically distributed, with a mean $\mathbb{E}(x)$ = 0.75, i.e., 75 % of all nodes vote for one and 25 % vote for zero. (We release this assumption in Section V-C.) In this case, the first node entering the system votes for zero whereas the following three nodes vote for one. After that, this distribution continues for all further nodes. As we will see in Section V-B, this represents the worst case of achievable compression rate.

Figures 4(a)-(c) show the local estimates of all nodes over time for sparsely populated scenarios, i.e., λ = {0.0025, 0.005, 0.01} nodes/s. We see that the convergence of the local estimates towards the global result is strongly dependent on the population density. For low values of λ, Figure 4(a), nodes are not able to estimate correctly the expected global result. As the arrival rate increases, Figure 4(b), a clearer trend can be seen towards convergence, and at λ = 0.01 nodes/s nodes are able to locally estimate the global result. Still, for any of the low arrival rates, some nodes do not gain sufficient knowledge about other nodes or even do not disclose anyone so that their local estimates remain 0 or 1 (see Figures 4(a)-(c)). As the arrival rate further increases, nodes converge earlier to the global result, and outliers disappear. Thus, we omit results for λ = {0.07, 0.15, 0.3} nodes/s for the sake of brevity.

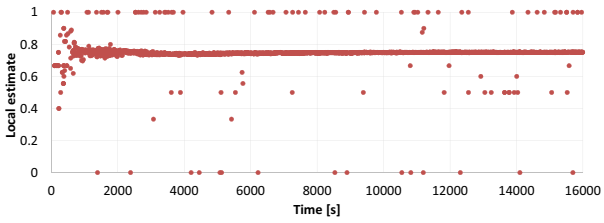
Figures 5(a)-(c) illustrate the proportion of shared nodes over time. In sparser scenarios, Figure 5(a), the proportion of shared nodes is kept below 20 %. Furthermore, information on shared nodes is continuously lost when nodes leave the system. Therefore, we are not able to observe clear convergence of the local estimates towards the global results, Figure 4(a).



(a)

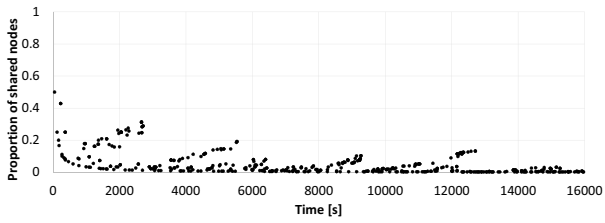


(b)

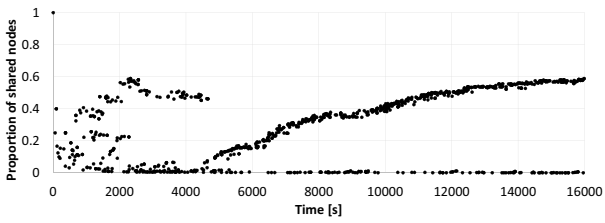


(c)

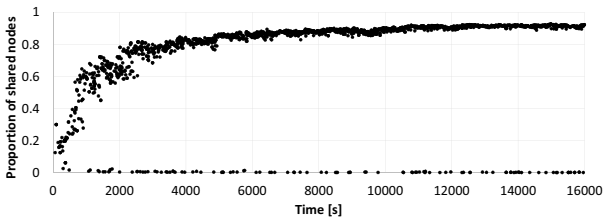
Fig. 4. Local estimates of all nodes for the Östermalm scenario: (a) λ = 0.0025 nodes/s, (b) λ = 0.005 nodes/s, and (c) λ = 0.01 nodes/s.



(a)



(b)



(c)

Fig. 5. Proportion of shared nodes for the Östermalm scenario: (a) λ = 0.0025 nodes/s, (b) λ = 0.005 nodes/s, and (c) λ = 0.01 nodes/s.

With increase of the arrival rate, λ = 0.005 nodes/s, up to 60 % of all nodes in the system are shared leading to a clearer convergence towards the global result, Figure 5(b). At around t = 7000 s when approximately 30 % of all nodes have already been shared, the trend towards the mean $\mathbb{E}(x)$ = 0.75 becomes apparent. We note that approximately 60 % of nodes are already shared around t = 4000 s by some of the nodes in the system. A further investigation shows that indeed the knowledge is kept only in two nodes which have shared around 130 nodes each. However, both of these nodes leave the system in t $\in$ (4600, 4700) s so their accumulated knowledge is lost and the process is restarted. This is reflected in Figure 4(b) as well as the trend towards the global result does not become clear before t = 7000 s when the vast majority of nodes has shared at least 30 % of all nodes. Finally, Figure 5(c) shows an even clearer convergence trend; already at t = 1000 s when approximately 30 % of all nodes have been shared most nodes approach the mean $\mathbb{E}(x)$ = 0.75. Thus, we conclude that even in dynamically changing environments there is a correlation between the percentage of shared nodes and the convergence to the global result.

The observation that disclosing approximately 30 % of nodes is sufficient for achieving precise estimate of the global result is further confirmed in Figures 6(a)-(c), which show the change in deviation Δ with respect to the proportion of shared nodes for the denser Östermalm scenarios with λ = {0.07, 0.15, 0.3} nodes/s. As the arrival rate increases, the deviation Δ significantly decreases once 30 % of the nodes have been shared, from 15 % for λ = 0.07 nodes/s, Figure 6(a), to below 6 % for λ = 0.3 nodes/s, Figure 6(c).

We further evaluate the performance of the system in steady state, i.e., once the average number of nodes in the area stays unchanged despite the arrivals and departures in the system. We then aggregate results from a 1000 nodes. We subsequently exclude the sparse scenario with λ = 0.0025 nodes/s from consideration as the trace does not comprise 1000 nodes after the steady state has been reached, and local estimates do not converge to the global result.

Table II shows the average and maximum deviation Δ including 95 % confidence intervals as well as the information overhead in the steady state depending on the arrival rate λ. Note that minimum deviation values are omitted as they are zero in all cases. These results clearly show that the denser the scenario, the smaller the deviation Δ, i.e., the accuracy increases. Note that for λ = 0.005 nodes/s and λ = 0.01 nodes/s some nodes are still completely wrong regarding their local estimate when they leave the system. As the arrival rate increases, both the average and the maximum deviation are

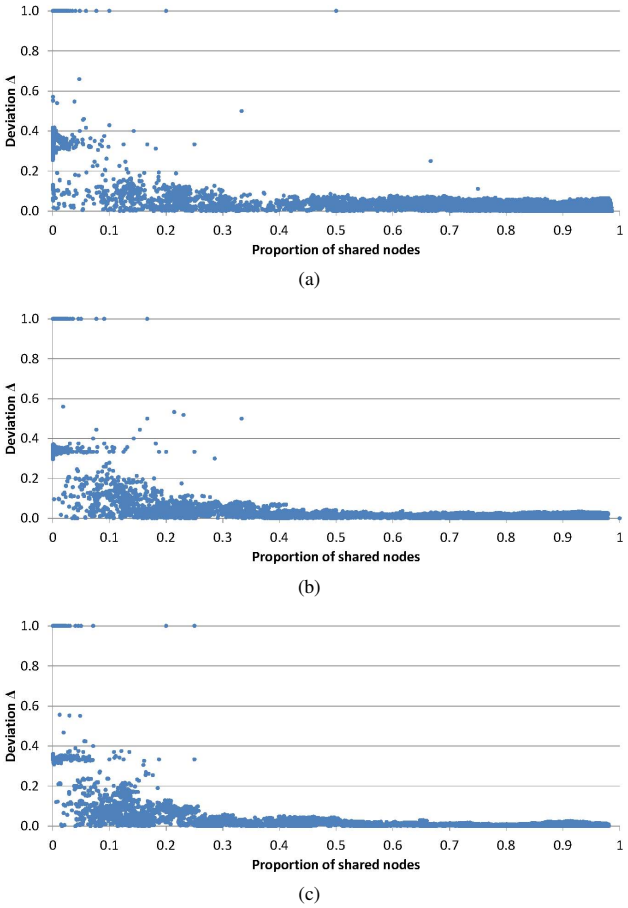


Fig. 6. Deviation Δ depending on the proportion of shared nodes for the Östermalm scenario: (a) λ = 0.07 nodes/s, (b) λ = 0.15 nodes/s, and (c) λ = 0.3 nodes/s.

steadily decreasing. Finally, for the most densely populated scenario, λ = 0.3 nodes/s, the maximum deviation never exceeds 3 %. Moreover, the information overhead is between 93 % and 94 % for $\lambda = \{0.005, 0.01, 0.07\}$ nodes/s, which results from the fact that often the same nodes meet again and do not exchange new information. On the one hand, this underlines the low processing load on application layer and thus DiVote's scalability in sparse scenarios. On the other hand, as shown above, the accuracy is not very high for the sparse scenarios. In denser scenarios, the information overhead amounts to lower values of 90 % and 66 % for λ = 0.15 nodes/s and λ = 0.3 nodes/s, respectively. Due to the higher population density, it is more probable that new nodes meet, which then exchange new information and thus the information overhead decreases.

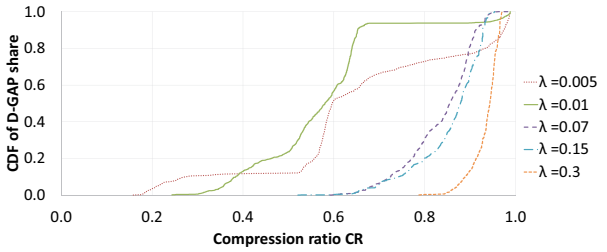
Figure 7 shows the cumulative distribution functions (CDFs) of the compression ratio when storing the D-GAP vectors with shared nodes and with votes across different arrival rates λ. The achievable compression ratio is higher in case of using D-GAP for storing shared nodes, Figure 7(a), than that for storing votes, Figure 7(b). For each shared node, a 1 is set in the D-GAP, which results in long sequences of consecutive 1s and increases compression. On the other hand, depending on the voting distribution and its mean value, sequences of consecutive 1s may be shorter in the D-GAP vector for storing votes resulting in a lower compression ratio. In Figure 7(a), the curves for λ = 0.005 nodes/s and λ = 0.01 nodes/s show an erratic trend as the scenarios are too sparsely populated to show clear convergence. With the increase of the arrival rate, however, the compression ratio also increases, and for λ = 0.3 nodes/s, the average compression ratio amounts to 92 %. This results from the fact that most nodes in the system have been shared, thus almost all bits in the D-GAP are set to 1. The compression ratio of the D-GAP for storing votes approaches 25 % as the arrival rate increases, Figure 7(b). This behavior is strongly dependent on the chosen voting distribution as well as on its mean value $\mathbb{E}(x)$ = 0.75. As mentioned earlier, the first node entering the system always votes for zero, while the following three nodes vote for one, and subsequently the distribution applies to all further nodes. Hence, the D-GAP for storing votes convergences to sequences of three consecutive 1s, which are interrupted by a 0. Applying Equation 1, three consecutive 1s and one 0 requires 3 bits for storage whereas a plain bit vector would require 4 bits. Therefore, the achievable compression ratio convergences to a value of 0.25 while more and more nodes are shared, which represents the worst case in terms of compression ratio. As we show in Section V-C, when the voting distribution follows a different distribution, the achievable compression ratio increases.

TABLE II
AVERAGE AND MAXIMUM DEVIATION Δ, AND INFORMATION OVERHEAD (IO) FOR THE ÖSTERMALM SCENARIO WITH DIFFERENT ARRIVAL RATES λ.

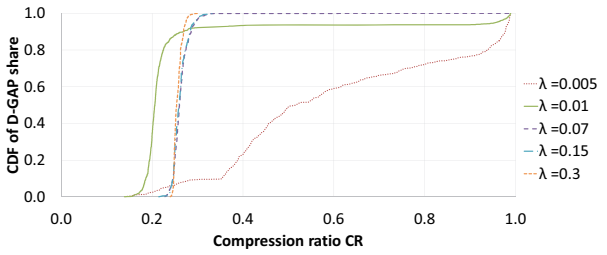
ARRIVAL RATE λ [NODES/S]	AVG. Δ [%]	MAX. Δ [%]	IO [%]
0.005	14.91±1.19	100	93.30±0.4
0.01	7.19±0.73	100	94.05±0.2
0.07	2.06±0.08	6.3	93.98±0.3
0.15	1.01±0.05	3.07	89.55±0.2
0.3	0.79±0.04	2.43	66.28±0.5

B. *Effect of scenario*

We now investigate the impact of different topologies by comparing the performance of the Östermalm and the Subway scenario. Table III shows the average and maximum deviation Δ including their confidence intervals, as well as the information overhead in steady state. Again, the minimum deviation values are omitted as they are zero in all cases. These results clearly show that both scenarios exhibit high accuracy, namely achieving an average deviation of 1 % for the Östermalm and the Subway scenario. The lower deviation in the Subway scenario is due to the bursty arrivals in the system. This also results in a lower information overhead of approximately 77 % compared to 90 % in case of the Östermalm scenario as node



(a)



(b)

Fig. 7. CDF of the compression ratio for storing (a) shared nodes and (b) votes in the Östermalm scenario under different arrival rates λ.

TABLE III
COMPARISON: AVERAGE AND MAXIMUM DEVIATION Δ AND INFORMATION OVERHEAD (IO) FOR THE ÖSTERMALM WITH $\lambda = 0.15$ NODES/S AND SUBWAY SCENARIOS.

SCENARIO	AVG. Δ [%]	MAX. Δ [%]	IO [%]
Östermalm	1.01±0.05	3.07	89.55±0.2
Subway	0.79±0.04	3.27	76.80±0.8

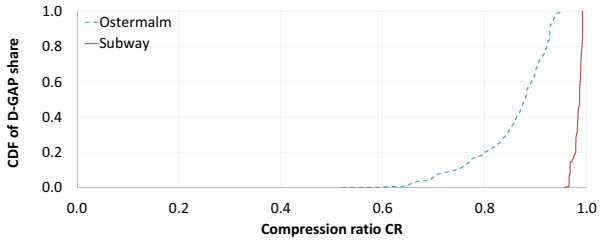
departures are also bursty. As the sojourn time of nodes is lower in the Subway than in the Östermalm scenario, nodes exchange less messages but those exchanged are useful for advancing the knowledge of other nodes.

Figure 8 shows the CDF of the compression ratio for storing shared nodes and votes in both scenarios. The Subway scenario exhibits higher compression ratio as nodes are quicker to disclose all other nodes in the system due to the smaller and more confined area, in which mobility occurs, Figure 8(a). Due to the same fact, the compression ratio of the D-GAPs for storing votes more closely approaches a value of 0.25 in the Subway scenario as apparent from Figure 8(b).

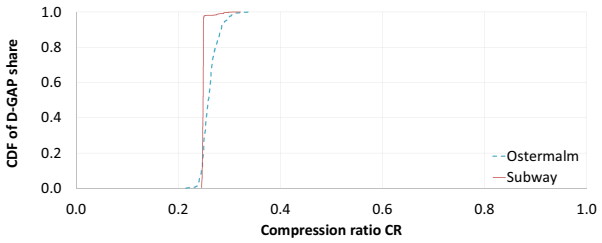
C. Effect of voting distribution

Finally, we compare the performance of DiVote for different voting distributions. We consider a deterministic as well as a uniform voting distribution, which can be seen as a more realistic representation of votes of pedestrians in an area. We choose three different mean values of the distribution ($\mathbb{E}(x)$ = 0.25, 0.5, and 0.75), and we perform five simulation runs for each mean value.

Table IV shows the average compression ratio with 95 % confidence intervals for $\mathbb{E}(x)$ = 0.25, 0.5, and 0.75 in the



(a)



(b)

Fig. 8. Comparison: CDF of the compression ratio for storing (a) shared nodes and (b) votes in the Östermalm scenario with for $\lambda = 0.15$ nodes/s and the Subway scenario.

TABLE IV
COMPARISON OF D-GAP COMPRESSION RATIOS (CRS) FOR THE ÖSTERMALM SCENARIO WITH $\lambda = 0.15$ NODES/S. UNIFORM AND DETERMINISTIC DISTRIBUTION ARE CONTRASTED WITH EACH OTHER.

	UNIFORM DISTRIBUTION		DETERMINISTIC DISTRIBUTION	
$\mathbb{E}(x)$	AVG. CR OF D-GAPS		AVG. CR OF D-GAPS	
	SHARED NODES	VOTES	SHARED NODES	VOTES
0.25	0.86±0.0003	0.38±0.019	0.86±0.004	0.32±0.002
0.5		0.23±0.005		0.08±0.002
0.75		0.34±0.012		0.26±0.001

Östermalm scenario. These results obtained for the uniform distribution are contrasted with the deterministic distribution. As the mobility trace is unchanged during simulation runs, the compression ratio for the D-GAP vector for shared nodes is independent of the proportion of nodes that vote for one. However, uniformly distributing the votes leads to higher compression ratios compared to the deterministic distribution as longer sequences of 1s and 0s are possible.

The gain in terms of achievable compression ratio becomes more apparent from Table V, which shows the average compression ratio and their 95 % confidence intervals for the Subway scenario. As expected, in case of the deterministic distribution the compression ratio approaches a value of 0 % for $\mathbb{E}(x)$ = 0.5 as the D-GAP expands to a bit vector of alternating 0 and 1. However, when votes are uniformly distributed, longer sequences of 1s and 0s become possible resulting in a compression ratio of 19 % even for an average global result of $\mathbb{E}(x)$ = 0.5.

TABLE V
COMPARISON OF D-GAP COMPRESSION RATIOS (CRS) FOR THE SUBWAY SCENARIO. UNIFORM AND DETERMINISTIC DISTRIBUTION ARE CONTRASTED WITH EACH OTHER.

	UNIFORM DISTRIBUTION		DETERMINISTIC DISTRIBUTION	
$\mathbb{E}(x)$	AVG. CR OF D-GAPS		AVG. CR OF D-GAPS	
	SHARED NODES	VOTES	SHARED NODES	VOTES
0.25	0.98±0.00001	0.33±0.01	0.98±0.0005	0.25±0.0004
0.5		0.19±0.01		0.001±0.0005
0.75		0.33±0.01		0.25±0.0003

VI. CONCLUSION

In this paper, we presented DiVote, a distributed voting protocol in the context of urban polling, which is suitable for environments, in which nodes exhibit high mobility. DiVote relies on device-to-device communication to exchange voting information. The proposed DiVote protocol exhibits the following main features:

- *Privacy*: By using a cryptographic hash function, votes cannot be related to the corresponding node identities.
- *Convergence speed*: The dynamism due to mobility imposes tight constraints on the convergence speed of the algorithm. Consequently, DiVote immediately updates the local estimate. Simulation results obtained when applying DiVote to realistic pedestrian mobility traces show that even in sparse scenarios local estimates quickly converge to the global result after having shared 30 % of all nodes in the system.
- *Accuracy*: At the same time, accuracy of local estimates is ensured as DiVote avoids to erroneously count votes multiple times, which is of decisive importance as the same node may be encountered several times. In dense scenarios, the local estimate does not deviate by more than 3 % from the global result after the system reaches the steady state.
- *Scalability*: Rather than storing shared nodes and their votes in plain bit vectors, DiVote uses D-GAP compression to be scalable in terms of required storage capacity. For realistic voting distributions, at least 19 % compression is achieved. Furthermore, the processing overhead introduced at the application layer is very low as only a fraction of the received messages (approximately 30 %) has to be processed even in the densely populated scenarios.

Prospectively, we intend to theoretically analyze DiVote and compare it with other existing schemes. We will further investigate more voting distributions and compare the D-GAP compression with other compression algorithms.

ACKNOWLEDGMENT

The authors would like to thank the German Research Foundation (DFG) (research fellowship, GZ: DA 1687/2-1) for their financial support.

REFERENCES

[1] S. Gambs, R. Guerraoui, H. Harkous, F. Huc, and A.-M. Kermarrec, "Scalable and secure polling in dynamic distributed networks," in *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, Oct 2012, pp. 181–190.

[2] L. Koeman, V. Kalnikaité, and Y. Rogers, ""everyone is talking about it!": A distributed approach to urban voting technology and visualisations," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3127–3136.

[3] Y. Benkaouz, R. Guerraoui, M. Erradi, and F. Huc, "A distributed polling with probabilistic privacy," in *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, Sept 2013, pp. 41–50.

[4] L. Nyers and M. Jelasity, "A comparative study of spanning tree and gossip protocols for aggregation," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4091–4106, 2015, cpe.3549.

[5] Ó. Helgason, E. A. Yavuz, S. Kouyoumdjieva, L. Pajevic, and G. Karlsson, "A mobile peer-to-peer system for opportunistic content-centric networking," in *Proc. ACM SIGCOMM MobiHeld workshop*, 2010.

[6] A. Kuznetsov, "D-gap compression," 2002. [Online]. Available: http://bmagic.sourceforge.net/dGap.html

[7] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, Oct 2003, pp. 482–491.

[8] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, Aug. 2005.

[9] J.-Y. Chen, G. Pandurangan, and D. Xu, "Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 987–1000, Sept 2006.

[10] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," in *Proceedings of the 7th International Conference on Peer-to-peer Systems*, ser. IPTPS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 13–13.

[11] J. Sacha, J. Napper, C. Stratan, and G. Pierre, "Adam2: Reliable distribution estimation in decentralised environments," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, June 2010, pp. 697–707.

[12] M. Borges, P. Jesus, C. Baquero, and P. S. Almeida, "Spectra: Robust estimation of distribution functions in networks," *CoRR*, vol. abs/1204.1373, 2012.

[13] F. Wuhib, M. Dam, R. Stadler, and A. Clem, "Robust monitoring of network-wide aggregates through gossiping," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 2, pp. 95–109, June 2009.

[14] I. Eyal, I. Keidar, and R. Rom, "Limosense: Live monitoring in dynamic sensor networks," *Distrib. Comput.*, vol. 27, no. 5, pp. 313–328, Oct. 2014.

[15] P. Jesus, C. Baquero, and P. S. Almeida, "Flow updating: Fault-tolerant aggregation for dynamic networks," *Journal of Parallel and Distributed Computing*, vol. 78, pp. 53 – 64, 2015.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*. MIT Press, 2001.

[17] S. T. Kouyoumdjieva, Ó. R. Helgason, and G. Karlsson, "CRAWDAD data set kth/walkers (v. 2014-05-05)," Downloaded from http://crawdad.org/kth/walkers/, May 2014.

[18] "Legion Studio," http://www.legion.com/.

[19] Ó. Helgason, S. T. Kouyoumdjieva, and G. Karlsson, "Opportunistic communication and human mobility," *Mobile Computing, IEEE Transactions on*, vol. 13, no. 7, pp. 1597–1610, July 2014.

[20] Ó. R. Helgason and K. V. Jónsson, "Opportunistic networking in OMNeT++," in *Proc. SIMUTools, OMNeT++ workshop*, 2008.