Multiplexing Content Exchange for Petition Drives in VANETs: Receiver Interest and Sender Utility

by
Osamah Abdulwahid Dhannoon

Bachelor of Education
Computers Sciences
Mosul University
2009

A thesis submitted to
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May, 2013

We the undersigned committee hereby recommends that the attached document be accepted as fulfilling in part the requirements for the degree of Master of Science in Computer Science.

"Multiplexing Content Exchange for Petition Drives in VANETs:
Receiver Interest and Sender Utility"
a thesis by Osamah Abdulwahid Dhannoon

Marius C. Silaghi, Ph.D.
Assistant Professor, Computer Science
Major Advisor

Jewgeni H. Dshalalow, Ph.D.
Professor, Mathematical Sciences
Committee Member

William H. Allen, Ph.D.
Associate Professor, Computer Sciences
Committee Member

William D. Shoaff, Ph.D.
Associate Professor and Program Chair
Computer Science

ABSTRACT

Multiplexing Content Exchange for Petition Drives in VANETs:

Receiver Interest and Sender Utility

by

Osamah Abdulwahid Dhannoon

Thesis Advisor: Marius C. Silaghi, Ph.D.

While drivers are not expected to vote while driving, VANETs can be an excellent media for dissemination of pre-recorded votes/opinions on regional issues in a decentralized opinion poll. We propose and evaluate heuristics for scheduling messages in a VANET broadcasting-based dissemination of decentralized opinion polling data among self-interested participants. The goal of the heuristics is to increase dissemination of the polling data under the given assumptions. The self-interest of participants is assumed to be manifested by selectivity in the storage and forwarding of topics and opinions for those topics.

The report starts by describing the concepts enabling the fully decentralized organization of the polls. The underlying protocol that we implemented for fully decentralized polling of opinions over VANETs is also introduced and evaluated.

Contents

List of Figures

List of Tables

Acknowledgements

Above all, I would like to thank my Professor PH.D. Marius C. Silaghi, for his support, his encouragement, and his valuable remarks that helped me throughout my courses and the process of writing my thesis. I am also obliged to my committee members, Dr.J.Dshalahow, and Dr.W.Allen. I am grateful to my colleague Rahul Vishen for his help with research experiments.

Dedication

I would like to thank my family - my father, my mother, my wife, my brothers, and my sister, - for their love and support during my study time in Florida. They have always encouraged me towards success and excellence, and have not for a moment forgot to support me with their prayers and lovely wishes. Finally, to all of my beloved Iraqi friends here and in Iraq, particularly those who helped me with and participated in my research studies, I say thank you from the bottom of my heart.

Chapter 1

Introduction

A protocol is proposed for dissemination of decentralized opinion polling over wireless, Vehicular Ad Hoc Networks (VANETs). When regional opinion polls are organized in a decentralized fashion, vehicle to vehicle (V2V) communication can be exploited for exchanging pre-recorded votes in neighborhoods (without the drivers being assumed to interact while driving).

VANETS are composed of wireless devices found in moving cars. Each of these devices can communicate with other devices found in its proximity. Common devices with powerful receivers can record messages sent from emitting devices found hundreds of meters away. A fully decentralized polling can be based on a decentralized authentication and census mechanism. Each device is owned by a self-interested user and we assume that the system is open, which implies that a user has full control over her device and its software.

Since they have full control, self-interested participants can refuse to store and forward information related to polls in which they are not interested. They can also refuse to store and disseminate opinions that they do not share. The communication model assumes that each device broadcasts data it wants to disseminate and simultaneously listens and processes data broadcast by passing-by devices. A challenge is to design heuristics for selecting what to emit, such as to maximize dissemination of polling data under working assumptions.

We evaluate heuristics that broadcast data either with uniform randomness, or favoring certain types of items such as: new votes, personal votes, votes similar to the personal votes or the intersection between the interests of the sender and the ones of potential receivers. Some input for these heuristics may come from information about interests of peers, and potentially their GPS location and velocity (bearing and speed). For efficiency, once packed, data can be broadcast several times. A set of queues are maintained to implement these heuristics.

To enable comparison between the described heuristics, a utility model is introduced where the dissemination of each item is associated with a numerical value. For example, the utility value for disseminating personal votes and opinions can be considered to be the highest, followed by the utility value for disseminating votes with choices similar to the personal ones. The average utility value for disseminating opposing opinions is assumed smaller, but for various

users it can be either positive or negative (based on whether they want their choice to succeed by any mean, or they are principled and ready to submit to the opinion of others, or they are open and even ready to learn from others justifications and to eventually change their minds). The utility for disseminating votes on which the current user abstains can be assumed in certain experiments to have an average value between the utility for similar opinions and opposing opinions. The impact of the actual numerical ranges of these utilities on results can also be evaluated.

After presenting the background and related work, we continue by introducing the concepts involved in decentralized polling and a sample data model for the storage of each node. Subsequently we present the protocol for broadcasting in terms of message components and their semantic. In chapter Heuristics we discuss the tested techniques and the involved data structures. After describing the experimental settings and results, we end with conclusions.

Chapter 2

Background

Common VANET applications are designed for traffic safety information sharing or file sharing. In both types of applications, each node needs to propagate data to the other nodes in the system. Communication between vehicles commonly demands a good routing protocol which is a common research interest in VANETs. In general, each pair of vehicles communicate using wireless broadcast. Using broadcast in high traffic areas may lead to transmission collisions between data packets. This problem is known as the *broadcast storm* problem [22]. In the following we describe some of the common protocols proposed to disseminate data in VANET scenarios. After that we discuss some of the most relevant applications in the literature.

2.1 Protocols

Mobile nodes connectivity in wireless Ad Hoc network is not constant. A proposed system for data dissemination based on delay-tolerant broadcast is presented in [10]. The system is designed as follows: nodes request their desire contents from encountered nodes on the network environment. The user can select a specific broadcast channel when operating. At the same time each user can be asked for contents on a different broadcasting channels. The contents are delivered in chunks in a random manner. Different data types can be used in the system such as still images, video clips, and voice tracks. The system can be implemented between users in high density places such as city sidewalks, public transportation areas, etc.

The system in [10] is evaluated based on an analytical model (street scenario) and simulation. In the street model a set of various measurements are used to compute the system performance using space-time queuing models. These measurements address the following: for an operating node in the system, what is the predicted number of contacts that a node establishes and what is the expected period of time for all the connections of that node while operating. In order to capture the total duration of the connections, two elements need to be computed: the ratio of nodes arriving into the studied environment (street), and the distribution of the velocities of the nodes. This analysis claims that the assumption of a Poisson arrival process is the most rational solution to predict

the probability of nodes arrival in a street. To measure the duration of the connections, the work considers two possibilities: nodes moving on the same and on the opposite paths. For a similar direction, the connection time for a node in an infinite street can be calculated by :

$$T\infty = \frac{\pm\bar{x_\Delta} + \Delta}{v_1 - v_2} \tag{2.1}$$

Where Δ is the transmission range in meters. X is a random variable that denote distance between two nodes. $\pm\bar{x_\Delta}$ is the average distance when contact occurs. It can be either positive, when the first node is faster than the second, or negative otherwise. The velocity is denoted by v. For nodes moving in opposite direction the duration of contact for a node operating in the two way pattern with fix velocity (v) is:

$$\bar{T} = \frac{1}{v}(r(\lambda_s)(L - \bar{x_\Delta}) + (1 - r(\lambda_s)) \times (1 - \acute{r}(\acute{\lambda_s}))\acute{\lambda_s}(\bar{x_\Delta} + \Delta)L). \tag{2.2}$$

Where λ_s is spatial rate, $r(\lambda_s)$ is the node probability to be connected with other nodes in the system. L is the street length. $\acute{r}(\acute{\lambda_s})\acute{\lambda_s}$ is the contact rate for nodes traveling in opposite direction. In this study, they do not measure the impact of aggregated data size in memory that each node prepare to broadcast. Three parameters are considered to be given in this model : length of the street, communication range, and data rate of the connection. Bluetella is a

prototype system that implements delay-tolerant broadcasting using cell phones. It uses Bluetooth technology as a communication method. Experiments with the Bluetella system showed that nodes spend a significantly long time for the setup of the connection. However, in comparison to this system, our implemented system exploits any devices with wireless LANs and nodes are pre-connected to the private network when operating (see Chapter 6). It is related to our problem, where we implement broadcasting data of interest, just that the interests in our case have a different nature and a more specific model. Nodes mobility in Bluetella address people walking, while our study is restricted to vehicles. In our system, we use queues of messages for broadcasting to increase the flexibility and choice of strategies, which in turn can be tuned for maximizing the efficiency of peers synchronization. Our analytical measures are used to calculate message utilities to test broadcasting approaches using data queues (see Chapter 5).

The VANET research is a sub-area of Mobile Ad Hoc Networks (MANETs), where communication is between wireless devices not necessarily found in vehicles. Many routing protocols have been proposed for MANETS, and are trying to address the issue of broadcast collisions, such as: the Dynamic Source Routing (DSR) algorithm [9], Ad Hoc on-Demand Distance Vector Routing (AODV) [20], Optimized Link State Routing protocol (OLSR) [8], etc.

In traffic safety applications, one disseminates vehicle speed and position information, besides other types of data that could be beneficial to road safety.

The dissemination is desired over a large number of nodes, using intermediate nodes to deliver information between devices that are not connected directly. This type of applications demand from the network nodes to be capable of sharing and processing the information in a highly efficient manner. The VANETs topology is changing depending on the nodes density in the environment. This creates frequent network disconnections which result in poor system performance. A statistical study for this problem is presented in [26]. This problem is considered irrelevant to our work. Our application is designed for synchronizing petition items in VANET. We do not need to disseminate real time emergency data.

The DV-CAST protocol is designed to perform efficiently in various traffic density scenarios [21]. Simulation results in [11] shows that DSR is more efficient when there is a small number of nodes. AODV performs similarly to DSR as the number of nodes increases. Some studies [23] have shown that using the AODV protocol in VANETs results in poor performance. Factors such as VANETs node mobility patterns (i.e., roads) and rapid network disconnections may be the cause of the slow response of AODV. Later studies [27] propose enhancements to AODV (AODV-VANET) that increase its performance. AODV-VANET introduces a Total Weight of the Route (TWR) to help decide the intermediary routing nodes between a sender and receiver. Many circumstances are reflected into the TWR value, such as vehicle speed, direction, and

the condition of its link to the recipient node. An example of VANET routing on city roads is proposed in [19]. It introduces a set of so called *Road-Based Vehicular Traffic (RBVT)* routing protocols. They use current traffic data to initiate the end-to-end communication paths.

Another big concern in VANETs traffic safety applications is security. Generally, traffic alert applications are disseminating data related to vehicle information. This data can be used by attackers for different purposes such as misleading the vehicle with false traffic information which in turn can endanger the life of the drivers. Authenticating broadcasted data is an important aspect in the security of traffic safety applications. A practical solution for this problem would be that all vehicles should be able to certify the received broadcasted messages. A known method for authenticating messages is based on using digital signatures. Flooding the system with signature verification is a common problem for this type of applications. Significant research have been proposed to handle this issue [7]. However, in our proposed system the goal is to receive votes from every node connected in the network. Verification in our system is to make sure that the received vote belongs to the claimed peer. Otherwise the received message is discarded.

2.2 Applications

In [2], a scalable algorithm for data dissemination is proposed to provide vehicles in VANETs with parking spots availability. A live video streaming service between vehicles is proposed in [6]. The Traffic View [16] project uses VANET communication to share information among cars moving on roads. Each vehicle collects traffic information from other vehicles and broadcast it to cars it encounter afterward. It can disseminate road assessments (such as foggy weather) to help find the best route to a destination. The system aggregates data in packets, to increase sending efficiency. The system is evaluated with simulation using the Network Simulator II (NS2) [12].

The Cross-Network Ad Hoc Dissemination project (X-NETAD) [5] proposes to use a VANET to share traffic alert data originally received via cellular phones, or more generally from Universal Mobile Telecommunications System (UMTS) networks. The intent is to have wireless devices inside vehicles that broadcast this data using Wi-Fi connections. Broadcasting data is based on the IF (Irresponsible Forwarding) protocol [1] which assigns transmission probabilities for each vehicle. Their prototype is built on a simulator. Figure 2.1 shows an illustration of how the X-NETAD system communicates with UTMSs.

The support of peer-to-peer applications in VANET provide new requirements on VANET protocols. The idea of deploying P2P applications in VANETS is introduced in [4]. VANET applications can benefit from the unique charac-

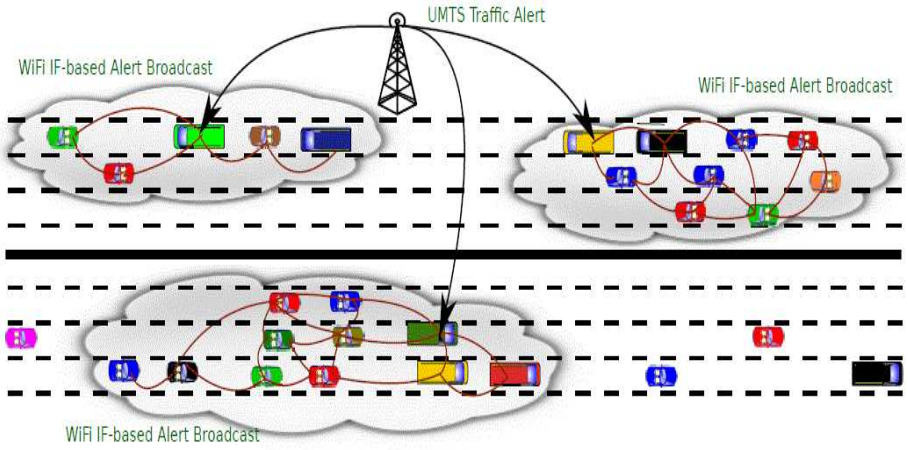

Figure 2.1: X-NETAD System.

teristics of P2P architectures such as: decentralization of information sharing, enhancement in content delivery, and VANET nodes can be more scalable and self-organizing. An example of applying this architecture is found in the Local Peer Group (LPG) [3], which clusters the neighboring nodes to restrict the dissemination range. This helps to avoid the broadcast storm. The clustering in LPG can be done either in a stationary or in a dynamic manner.

In [17], a swarming protocol (SPAWN) is introduced. It is used to implement a new method for downloading and sharing files over VANETs using the peer-to-peer architecture. The mechanisms used to implement data dissemination is called *gossiping*. This technique makes use of the position of the vehicles to select a peer for communication. The evaluation for this technique is done using simulation. Implementing CarTorrent in a real world scenario is reported in [13], reporting field tests for the SPAWN protocol. The communicating protocol per-

forms periodic gossiping to announce the availability of contents in each peer. If any node is interested in an available file, it will send a request for a particular piece of the file through a protocol called Ad Hoc On Demand Distance Vector (AODV). Then, upon receiving, the initiator sends back the specific file piece through AODV. Another example of content sharing is CodeTorrent [14], a protocol for P2P file sharing over VANETs that uses encoding. It aims to decrease the file downloading time. Ad Torrent [18] is a content sharing application for VANETs based on the idea of advertising boards.

P2P sharing contents over VANETs based on data popularity is introduced in the Roadcast project [28]. It delivers the most proper and relevant data (such as MP3 audio files) based on the peers queries by applying efficient Information Retrieval (IR) mechanisms. It also uses algorithms to make sure that popular data in the system is not replicated many times. Roadcast is evaluated through simulation.

Another method for data dissemination over VANETs is the Segment-Oriented Data Abstraction and Dissemination (SODAD) [24]. It aims to increase the communication range between connected vehicles, taking it beyond the wireless card range. It can be used for various types of applications such as: traffic safety data, utilities information (e.g. locating gas stations). A sample usage of SODAD is found in the Self-Organizing Traffic Information System (SOTIS) [25]. The SOITS system is based on Vehicle-to-Vehicle (V2V) communication. Each

car broadcasts traffic information which is collected externally or being received from others. A digital map and navigator is also included in the system.

Basically, each node broadcasts packets that contain traffic information based on the current position of the car. Each road is identified by a Road ID. The node will discard messages with expired time stamps.

Chapter 3

Concepts

In this section we introduce formally the concepts that can make possible a fully decentralized debate/petition/poll process. They define the data items exchanged by the proposed protocols.

Polling is a process whereby one gathers the opinion of a sample from a certain population on a given question. For the statistical relevance of the result it is important that only opinions of the members of the targeted population are recorded and that each of them is recorded only once. Minor errors can leave the results relevant in case the support of competing opinions differs by large margins. It should be difficult to systematically manipulate the outcomes.

Definition 1 (Peer) *The set of software agents that coordinate publicly to represent a given user is referred here as* peer*. A peer may have agents running on various devices (laptops, desktops, phone of a user) and which share the same*

public and secret key pair. The peer is globally identified by its public key.

Each semantically independent type of item (vote, justification, etc.) is uniquely identified by a *Global Identifier (GID)*. To guarantee that there is no voluntary or involuntary conflict between GIDs, these are built either as public keys for a secret, or as digests of the information that they represent.

A working definition of the population eligible for a set of polls, is captured in the concept of organization.

Definition 2 (Organization) *An organization is an entity defining the mechanism whereby an authority is defined for specifying and controlling eligibility for voting on a set of issues. An organization is defined by the unchangeable set of parameters describing its governance and function. This unchangeable characteristic is captured in its global identifier.*

The fact that an item is generated for this organization is specified by tagging the item using the *global identifier* (which is included in the digitally signed data for the item). The parameters of the organization can specify the criteria for eligibility to vote on items, and the ontology for the related communication.

Decentralized governance is currently rare. Some organizations are appropriate for a decentralized governance, such as: a diaspora, a union, or a club.

Definition 3 (Grassroot Organization) *A grassroot organization is a tuple* $\langle \mathcal{O}, p \rangle$*. The global identifier* $\mathcal{O}$ *is the digest of a set of parameters* p*, which can*

never change and are referred to as its constitution.

Other organizations are more appropriate for centralized governance, such as: a class of students specified by their instructor, a faculty coordinated by the department head, or a committee coordinated by its chair.

Definition 4 (Authoritarian Organization) *An authoritarian organization is a tuple* $\langle \mathcal{O}, p, r, d, s \rangle$. *The global identifier* $\mathcal{O}$ *is the public key of the authority, which can change any of its parameters, and is referred to as the* dictator *of the organization. The parameters of the organization are specified by* p*, the revocation status by* r*, and the date of this declaration by* d*. A signature* s *is used by the authority to certify this data:* $s = SIGN(SK(\mathcal{O}), \langle p, r, d \rangle)$.

The participants defined by an organization as eligible to vote on its issues, and whose votes matter with a predefined weight in decisions, are referred to as its *constituents*.

Definition 5 (Constituent) *A constituent of an organization is a person that is eligible to cast a vote with a predefined weight on the issues relevant to that organization. A constituent can be either* active *(i.e., generating data items) or* inactive *(i.e., its existence is discussed but it is not active). A constituent is defined by a tuple* $\langle \mathcal{C}, \mathcal{C}', \mathcal{O}, i, r, d, s \rangle$ *where* $\mathcal{C}$ *is the constituent GID,* $\mathcal{C}'$ *is the GID of the active constituent submitting this data,* $\mathcal{O}$ *is the GID of the organization,* i *contains the identity details (as defined by* $\mathcal{O}$*),* r *is the revocation*

status of the constituent, d is the date and time of the data creation, and s the signature generated by $\mathcal{C}'$.

The definition of identity details is specified by the parameter p of the organization refered by $\mathcal{O}$. An *active constituent* is identified by its public key, which is used as its global identifier. An *inactive constituent* is identified by its available description, and its global identifier is a digest of this data. The revocation status is a boolean flag that blocks any further update or usage of the constituent item, other than its dissemination. Items depending on it can be discarded. $\mathcal{C}'$ is redundant for an active constituent, being the same as its own GID $\mathcal{C}$, but for the conciseness of the description we prefer to use it, to unify the notation with the one of inactive constituents.

As data structure, a constituent is described by the set of parameters that enable his classification as to weather it is eligible to vote in the given organization.

Certain organizations can define voting eligibility based on a centralized mechanism, such as an authority that issues eligibility certificates according to its criteria. Decentralized organizations addressed here employ a mechanism that defines eligibility based on witnessing.

Definition 6 (Witnessing) *An act of witnessing is a process whereby a first participant states whether the conditions for eligibility in an organization are satisfied by a second participant. A witnessing act is described by a tuple*

$\langle W, \mathcal{O}, S, T, m, e, d, \sigma \rangle$ where W is the witnessing stance GID, $\mathcal{O}$ is on organization identifier, S is the constituent identifier of the witnessing participant, T is the constituent identifier of the witnessed participant, e is an explanation, d is the time and date of the witnessing and σ is a signature of the data. The mode m of the witnessing consists of a set of semantic statements, each of them being either positive *witnessing or* negative *witnessing about some quality of the target.*

For large decentralized organizations it is impractical to detect whether an isolated group of people witnessing for each other are really eligible or just elements of an attack. This problem is partly addressed by the concept of neighborhood.

The constituency of an organization is classified in a hierarchical tree structure where each constituent belongs to a single leaf node and each node of the tree is called a **neighborhood**.

Definition 7 (Leaf Neighborhood) *A leaf neighborhood is a sufficiently small subgroup of the constituents of an organization such that each constituent of a leaf neighborhood can verify the eligibility of the other members of the subgroup with reasonable effort, and where the existence of the subgroup as a whole can be verified by other members with reasonable effort. A leaf neighborhood is identified by its parameters and its global identifier is given by their digest. A neighborhood is a tuple $\langle \mathbb{N}, n, t, \mathcal{P}, c, \mathcal{C}, \sigma \rangle$ where $\mathbb{N}$ is the GID, n is the name*

of the neighborhood, t is its level (e.g. city,street), $\mathcal{P}$ is the GID of the parent neighborhood (empty for the root), c is the list of expected neighborhood levels under this neighborhood, $\mathcal{C}$ is the GID of a constituent supporting its existence and σ is her signature for the data.

We need to formalize the concept of reasonable effort and that can be done in terms of the cost it brings to a volunteer participant.

Definition 8 (Reasonable Effort) *In this paper we say that an effort is reasonable in the context of an organization if the majority of constituents of the given organization can perform it in a predefined unit of time (e.g., one day), fixed for that organization, without consuming extra resources besides their taxed revenue (given applicable laws) for that period of time.*

The neighborhood immediately above the current neighborhood in this tree is referred to as its *parent neighborhood*. The *set of ancestor neighborhoods* for a constituent contains the leaf neighborhood where the constituent is currently declared to belong, together with all the hierarchically higher neighborhoods containing this leaf neighborhood.

By convergence of polling data we understand that, once the generation of new data stops, eventually everybody that is connected and interested in a given topic sees the same items for that topic.

3.1 Communication Items

Besides the above mentioned items used for managing the organizations (organizations, constituents, neighborhoods and witnesses), agents communicate items related to polling. The question raised by a poll is captured by the concept of motion.

Definition 9 (Motion) *A motion is a formal question with a predefined set of possible responses on which each willing constituents is called to select an answer. A motion is defined by a tuple* $\langle \mathcal{M}, t, o, \mathcal{C}, \sigma \rangle$ *where* $\mathcal{C}$ *is the GID of a constituent supporting the motion,* t *is its text,* o *is a list of possible answers of the motion, and* σ *is the signature generated by* $\mathcal{C}$*. The motion is globally identified by the hash of its data,* $\mathcal{M}$*.*

Definition 10 (Justification) *A justification is an explanation that a constituent provides for his selection of an answer for a motion. It is defined by the tuple* $\langle \mathcal{J}, \mathcal{M}, t, \mathcal{C}, \sigma \rangle$ *where* $\mathcal{J}$ *is the justification GID,* $\mathcal{C}$ *is the GID of the constituent supporting the text of the justification,* $\mathcal{M}$ *is the motion GID,* t *is the text of the justification, and* σ *is its signature.* $\mathcal{J} = HASH(\mathcal{M}, t, \mathcal{C})$*.*

Definition 11 (Vote) *A vote is the selected answer to a motion, as submitted by a constituent. Each vote consists of a tuple* $\langle \mathcal{V}, \mathcal{M}, c, \mathcal{C}, \mathcal{J}, d, \sigma \rangle$ *where* $\mathcal{V}$ *is its GID,* $\mathcal{C}$ *is the GID of the constituent authoring the answer to the motion,* $\mathcal{M}$ *is*

the motion GID and c the selected answer. $\mathcal{J}$ is the GID of a cited justification, and can be empty, d is a date and time, and σ is the signature of $\mathcal{C}$.

3.2 Data Model

Each self-interested software agent stores the data of interest to it into a local database. The agent stores the received data if it refers to organizations, neighborhoods, constituents and motions of interest. By default, received definitions of peers and definitions of organizations received from non-blocked peers are stored to give users an opportunity to inspect and define their interest about them.

The database schema allows for storing the following types of items that have a stand-alone semantic and that are separately digitally signed by the entity generating them: peer, organization, neighborhood, witnessing, motion, justification, vote.

Each item, is tagged with three user controlled flags: `blocked, broadcastable, interest`. These flags control the communication as described in the next chapter. Each received data item is also associated with the arrival time, which is the date when a last change to the digitally signed parameters of the item was registered. The signed parameters of each item contain the creation time, which is the data when the signature was issued. The creation time is used to compare and select the newest item among items whose parameters change over time,

such as active constituent, vote, and authoritarian organization.

For the case where an attacker or mistake leads to two distinct versions of the same item claiming the same creation time, the comparison is made on the hash of the data. This is used to prove that at convergence all participants have coherent databases.

3.3 Resilient

Note that, for security reasons discussed below, we currently do not allow for the public key of the authority in an authoritarian organization to be automatically replaced in case it is revoked. Similarly, the constitution of a grassroot organization cannot change, without migrating to a completely new organization.

Attack on Global Identifiers The above mentioned security considerations refer to the attacks based on generating new organizations with the same global identifier as the attacked organization but with different parameters. Since the organization parameters define the ontology of the activity and the employed criteria for eligibility of constituents, such an attack can mislead the constituents of the attacked organization to generate data that is inconsistent with the constitution and rules of the organization. For example, a constituent A of an attacked grassroot organization O_1 could be misled by the constitution of the

organization O_2 of an attacker to support the eligibility of another participant B (eligible with O_2 but not with O_1), support that would be transferable as valid into the attacked organization, O_1.

Other Attacks on Authoritarian Organizations The *Freeze Attack* is an attack possible only on authoritarian organizations where the secret key of the authority is destroyed. The *State Coup Attack* is also an attack possible only on an authoritarian organizations, where an attacker steals the secret key of the authority while destroying all the copies available to the original authority. The *Confuse Attack* on an authoritarian organization is where the attacker gets a copy of the secret key of the authority and issues contradictory parameters for the organization.

In the case of the Confuse Attack, the original authority can issue a `revoke` message (a new definition of the organization with the r parameter set) and the organization is blocked, hinting the constituents that they have to move their activity to a new organization. For Freeze Attacks and State Coup Attacks, the constituents cannot be warned automatically.

Attack on Grassroot Organizations An *Identification Attack* is where an attacker creates an organization with similar but not identical parameters, to the attacked organization. Since not all parameters are easily visible in graphical widgets, attacker organizations where the most visible part (in some GUI) is

similar to the view of the attacked organizations can create confusion among users. This attack also works against authoritarian organization. The *Creator Attack* on a grassroot organization is an attack whereby the secret key of the creating peer of the organization is compromised. Such an attack can facilitate an *Identification Attack* by setting the name of the creator to values that are not recognizable to users.

To detect storage attacks, any received item is tagged with the GID of the peer from which it is received.

Chapter 4

Protocol

Let us now describe the structure of the exchanged messages. Software agents found on wireless enabled devices with Ad Hoc capabilities are assumed to broadcast messages continuously (potentially with short pauses).

4.1 Communication Control

The default settings of our current implementations assume that a self-interested receiver normally refuses to store items about unknown organizations, as well as items relating to organizations, constituents, neighborhoods or motions that are specifically blocked by the user. To refuse items about unknown organizations, newly received organizations are blocked by default. Organizations where the user registers are automatically unblocked.

By default, all the stored data about items that are not blocked is made

available for broadcasting, but that behavior can be manually controlled for each item using a flag called `broadcastable`.

For example, if an organization is blocked, then we store only its parameters but any extra data associated with it (e.g., constituents, neighborhoods, motions) are discarded. Similarly we handle blocked constituents, neighborhoods, or motions.

Messages received can refer to the GID of an unknown item (constituent, neighborhood, motion, justification). If users decide to store the item referring to unknown GIDs, then *temporary items* are created for each of the unknown GIDs, to enable their control (blocking, broadcastability). The enabling of certain temporary items, such as temporary constituents, open the door for *Storage Attacks*, namely where attackers attempt to fill users databases with data that is more difficult to verify. If temporary data is enabled, then remaining data for temporary items can be advertised as *requested* in subsequent broadcast messages. Various mechanisms (such as references to source peers) can be used to mitigate these attacks.

Items of particular interest to the user, such as motions, constituents or organizations that the user is particularly involved with, can be announced as *interests* in broadcast messages. This feature can inform cooperating peers, which can thereby give priority in sending such data back to the user. To enable this feature, each stored item is associated with the `interest` flag that the user

can manually set and that the system can use to generate the corresponding interest information in messages.

4.2 Messages

Each broadcast message contains a self-contained information. The two most complex types of messages are the ones carrying votes and the ones carrying witness acts (since they include data about many other types of items but are not included in other types of data).

A message containing a witness act consists of a tuple $\langle p, o, c_s, N_s, c_d, N_d, w \rangle$ describing the definition of the relevant organization o, the definition p of the peer that created the organization, the definition c_s of the constituent making the witness stance, the definition c_d of the constituent for which the witness stance is made, the definition w of the witness stance. It also contains the set of definitions of ancestor neighborhoods N_s of the neighborhood of c_s and the set of definitions of ancestor neighborhoods of the neighborhood of c_d.

A message containing a vote consists of a tuple $\langle p, o, c, N, m, j, v \rangle$ describing the definition of the relevant organization o, the definition p of the peer that created the organization, the definition c of the constituent making the witness stance, the definition m of the motion, the definition j of the justification and the definition v of the vote. It also contains the set of definitions of ancestor neighborhoods N of the neighborhood of the c.

Each broadcast message is also attaching a *set of interest hints*. This set contains some of the GIDs of the organizations, neighborhoods, constituents and motions that the user has marked with the `interest` flag.

Probabilistically, the data concerning the details of the organization, the peer or the constituent can be dropped from a vote message or a witness message to reduce some of the replication, with the risk of rendering some messages useless (as those messages may be dropped by receivers missing one of the items required for storing it: its organization, neighborhood, etc.).

4.3 Handling

Here we describe reference procedures for handling received messages. In Algorithm 1 we introduce the method used by a software agent to manage the knowledge it has about interests of peers found in passing-by cars. An interest consists of the GID of an organization, neighborhood, constituent, or motion. Whenever indication of a particular interest is received from a peer, it is stored locally, tagged with the GID of the sending peer and an expiration time. The expiration time is computed based on the arrival time of the message containing this interest, the available information about the relative speed between that peer and the vehicle of the users, and an estimation of the maximal distance within which the two devices can communicate.

When the devices are not equipped with GPS (as in the experiments reported

here), then the computation simply returns the estimated expiration time as the sum between the current time and a constant `life_span` (Line 1.3). In our experiments this constant is set to 6 seconds. Note that each time that a message is received from the same peer, the expiration time of its interests is updated, thereby accounting for devices that are reachable for a longer period of time than the selected `life_span` constant.

A variable `min_interest` stores the current time, updated on the clock (Line 1.5) and any interests with higher expiration time is removed at that moment (Line 1.6).

Algorithm 1: Management of interest without GPS

```
1.1 procedure handle_interests (Peer, interests) do
1.2     for i in interests do
1.3         set_interest-value(i, min_interest+life_span);
1.4 procedure on clock() do
1.5     min_interest++;
1.6     drop expired interests;
```

Next we describe the algorithms used to handle received **witness** and **vote** messages (Algorithms 2 and 3). Similar and simpler algorithms are used to handle messages carrying other types of items.

The algorithms for handling messages employ the procedure `handle_interests()` defined in Algorithm 1, and a procedure `verifySignature(item)` that is checks the signature of the item passed in parameter, quitting on failure. The procedure `store-or-update(item)` verifies whether a previous version of the item is

Algorithm 2: Receiving and Handling a Witness

```
2.1  on witness(Peer, interests, (p, o, c_s, N_s, c_d, N_d, w)) do
2.2      handle_interests(Peer, interests);
2.3      if !verifySignature(p) then return;
2.4      store-or-update(p);
2.5      if (blocked(p)) then return;
2.6      if !verifySignature(o) then return;
2.7      store-or-update(o);
2.8      if (blocked(o)) then return;
2.9      for n in N_s do
2.10         if verifySignature(n) then
2.11             store-or-update(n);
2.12             if (blocked(n)) then return;
2.13     for n in N_d do
2.14         if verifySignature(n) then store-or-update(n);
2.15     if !verifySignature(c_s) then return;
2.16     store-or-update(c_s);
2.17     if (blocked(c_s)) then return;
2.18     if !verifySignature(c_d) then return;
2.19     store-or-update(c_d);
2.20     if verifySignature(w) then store-or-update(w);
```

already available and whether its creation date is newer than the received item. On failure it store the item (if no other version was found), or updates it (if a version with earlier date or identical date but lexicographically smaller digest value is found);

Before handling any item, first the software agent checks whether the item is not `blocked` by the user (i.e., by being generated by a blocked peer, or constituent, or for a blocked organization, neighborhood, motion, justification, or choice for the motion).

The procedures to handle messages start by handling first the more basic

Algorithm 3: Receiving and Handling a Vote

```
3.1  on vote(Peer, interests, (p, o, c, N, m, j, v)) do
3.2      handle_interests(Peer, interests);
3.3      if !verifySignature(p) then return;
3.4      store-or-update(p);
3.5      if (blocked(p)) then return;
3.6      if !verifySignature(o) then return;
3.7      store-or-update(o);
3.8      if (blocked(o)) then return;
3.9      for n ∈ N do
3.10         if verifySignature(n) then
3.11             store-or-update(n);
3.12             if (blocked(n)) then return;
3.13     if !verifySignature(c) then return;
3.14     store-or-update(c);
3.15     if (blocked(c)) then return;
3.16     if !verifySignature(m) then return;
3.17     store-or-update(m);
3.18     if (blocked(m)) then return;
3.19     if !verifySignature(j) then return;
3.20     store-or-update(j);
3.21     if verifySignature(v) then store-or-update(v);
```

types of items before handling the ones that are based of the first. The typical order is: peer, organization, constituent, neighborhood, motion, justification, vote. Note that there can be a circular relation between constituent and neighborhood since a constituent may reside in a neighborhood and the neighborhood is supported/created by a constituent (potentially the same). In this case the two are stored only either if they are simultaneously available, or if storage of `temporary items` is enabled (as discussed earlier).

4.4 Messages Structure

We now describe the exchange messages contents. We implement five types of messages Peer, Organization, Constituent, Neighborhood, Witness, and Vote. Our reported experiments where done by giving witness and vote messages the highest broadcasting probability (see Chapter 7). These messages are encoded and decoded using ASN1 (see Appendix A). The ASN1 structure name for the messages are : Peer (D_PeerAddress), organization (D_Organization), constituent (D_Constituent), neighborhood (D_Neighborhood), witness (D_Witness), and vote (D_Vote).

Peer Message :

D_PeerAddress

D_PeerAddress

Organization Message:

D_PeerAddress

D_Organization

Constituent Message:

D_PeerAddress

D_Organization

D_Constituent

Neighborhood Message:

D_PeerAddress

D_Neighborhood

D_Constituent

Witness Message:

D_PeerAddress

D_Organization

D_Witness

Vote Message :

D_PeerAddress

D_Organization

D_Vote

Chapter 5

Heuristics

To model incentives and their relation with the behavior of the users, we formalize the utility of a message. In practice each item has its own utility, and a different utility for different users.

Definition 12 (Utility of messages) *Each user draws a certain utility for learning an item, depending on that item. A user also gains a given utility for disseminating an item.*

In the following we assume that the utility of storing items is flat for the items in an organization, while the utility of forwarding an item depends on its similarity with the items generated by the user (and therefore describing her values).

5.1 Uninformed Heuristics

Uninformed heuristics for broadcasting correspond to an assumption that no hints received from peers should be trusted and transmission is made based on an *a priori* model of frequency for vehicles with peers traveling in the two directions. With uninformed heuristics, all peers are supposed to be interested in all items that the current peer has, and to be able to store all messages that they receive from this user. Such a model assumes that a number of A reachable vehicles travel in the same direction with a relative speed v_A while a number of B reachable vehicles travel at each moment in opposite direction with relative speed v_B. The local computer is able to load new items from a local database with an efficiency of M messages a second. Messages (each with utility u_M) can be emitted at a speed of v_M messages a second from a sending queue of size B_s, the buffer of the queue being reloaded from database at a period of time:

$$P_{reload} \geq \frac{B_s}{\min(v_M, M)}. \tag{5.1}$$

If D is the communication range of the device then $T_A = \frac{D}{v_A}$ is the duration for which a car traveling in the same direction is reachable, and $T_B = \frac{D}{v_B}$ is the similar duration for the opposite direction. We also assume that the assumption that the queues of preloaded messages used for sending data are long enough to

provide data for the whole time T_B, i.e.,

$$\frac{B_s}{v_M} \geq \frac{D}{v_B}. \tag{5.2}$$

Then, the utility of sending data during time T_A is:

$$U_{T_A} = u_M \cdot A \cdot B_s \cdot \lceil \frac{T_A}{P_{reload}} \rceil + u_M \cdot B \cdot \frac{T_A}{T_B} \cdot T_B \cdot v_M$$

where the first part of the right hand expression refers to the utility obtained by sending items to cars in the same direction (cars that each receive the content of $\lceil \frac{T_A}{P_{reload}} \rceil$ full buffers of messages, each of size B_s). Note that in this equation we assume that the reminder of $P_{reload} : T_A$ is larger than $\frac{B_s}{v_M}$. The second part of the expression is the utility from the items transmitted to cars driving in opposite direction. There are $\frac{T_A}{T_B}$ road segments of size D with such cars that travel in opposite direction, each holding B cars, and each of these cars receives $v_M \cdot T_B$ messages.

If one set P_{reload} to the closest (smaller) divisor of T_A, then the utility rate per unit of time that the agent gets for broadcasting from a given queue of messages in this condition is approximated to (obtained by dividing U_{T_A} by T_A):

$$\frac{\partial U}{\partial t} = u_M \left(\frac{A \cdot B_s}{P_{reload}} + B \cdot v_M \right) \tag{5.3}$$

The current peer has a number N_P of personal items, a number N_S of similar items, a number N_O of other items and a number N_F of opposing opinions of positive utility (opposing opinions of negative utility are not sent). An assumption is that $N_P \ll N_O$. Based on this model we search for the best policy in terms of number of times that items with high utility should be broadcast before broadcasting some items with a lower utility.

5.2 Informed Heuristics

Informed heuristics assume that peers announce their interests as sets of GIDs for organizations, constituents, neighborhoods, motions or justifications for which they want to get related items, and that they drop any other messages. Senders thereby build special queues with data of interest to these peers and give these messages priority over other items. In our experiments, agents broadcast only data relevant to current peers and found in current queues.

Each message loaded in sending queues is tagged with information about contained organizations, constituents, neighborhoods, motions, justification (and potentially vote choice), to help dynamically retrieve those of interest to new detected peers.

While our experiments were run with laptops that were not provided with GPS sensors, such sensors can provide extra information as to when the peers travel in the same direction or in opposite direction, and for how long the peer

be reachable.

Our utility model can be combined with the statistical model of the efficiency of communication described at uninformed heuristics (as shown in the Experiments section), to decide the policy of transmission for each type of data (what percentage of each type of data should be sent at each moment of time). One can select the ratio of data of each type such as to maximize the expected utility of the sender. Rather than using the model resulting in Equation 5.3, one can introduce utilities in decisions based on the statistical models in [10].

Chapter 6

Agent Architecture Details

In this section we describe the architecture and implementation challenges encountered while developing the described protocols. Our implementation can run on Linux, Windows, and Mac OS. The network configuration is automated on Linux and Windows while the Mac OS network configuration has to be performed manually. We performed experiments with our implementation of a VANET platform, based on agents running on laptops that are located in moving vehicles. We allocate an Ad Hoc wireless cell based on the open (unencrypted) SSID `DirectDemocracy` at Frequency 2.462 GHz resulting in the cell `46:32:D1:F2:88:67`.

6.1 Portable Detection of Wireless Cards

Each device in the system could have more than one wireless card (Interface). This may be a useful feature in case users want to be simultaneously connected to the Internet via one wireless card and to the "**DirectDemocracy**" Ad Hoc network via a 2nd wireless card. In order to run the **POLLER** on the Ad Hoc network, the user should select one of the wireless interfaces for this purpose. The available wireless interfaces are automatically detected. The detection of wireless interfaces is not yet well supported by Java in a OS independent manner. Currently the Java application calls OS-dependent external scripts to learn the wireless interface names. For example in Windows one gets wireless card interface names and IP addresses using "`ipconfig /all`", while in Linux "`iwconfig`". Operating systems vary in how they name the wireless interfaces. For example, Windows 8 refers to a wireless interface as "Wi-Fi". From the second wireless card on, the string is followed by a number starting with 2 and separated with a space. Windows 7 refers to a wireless interface as "Wireless Network Connection". Subsequent interfaces will have a number after the string. In Linux operating system, the interface names is configured by the distribution, where the common names are "wlan" or "eth" followed by a number. The wireless interfaces names and their configuration are stored in the database, to be re-instated when the application is restarted. The format of the retrieved information is `(Interface Name, IP, SSID)`. In the following

section we describe how the software agent detect wireless configuration and connection process to **DirectDemocracy** Ad Hoc network.

6.2 Global Wireless Cell

We designed a GUI widget Figure 6.1 that provides access to the database used to store the configuration of the wireless interfaces. The user may use this widget to select the interface to be used for the Ad Hoc network. When a particular interface is selected, the **POLLER** automatically configures it to connect to **DirectDemocracy**. The process in which the system detect current connection information and connects to this Ad Hoc network differs between operating systems. A set of OS-dependent external scripts are used for this purpose.

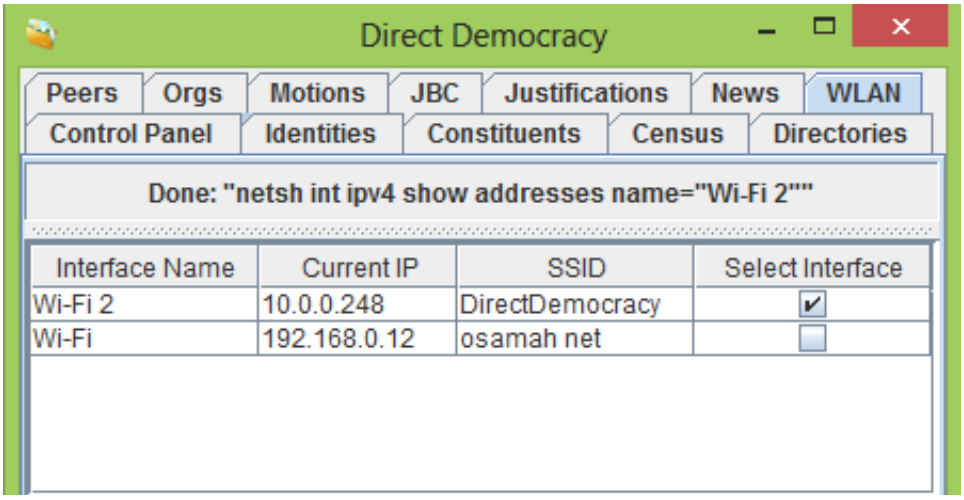

Figure 6.1: Wireless Widget

6.2.1 Windows

Detect Wireless Configuration

In order to capture connection information in Windows we rely on a system command utility "netsh". To obtain Wireless interface name and current SSID we use the following command:

`netsh wlan show interface`.

"netsh" is used to display static IP addresses, while for the DHCP configured interfaces we use "ipconfig" [15]. If we obtain **DirectDemocracy** as SSID (means that the wireless card is already connected to **POLLER** network) then we get the IP using "netsh" to retrieve the IP using the command:

`netsh int ipv4 show addresses name="Interface name"`.

Otherwise we use `ipconfig /all`.

Connect to DirectDemocracy

To connect to a new Ad Hoc network, a profile for that network can be added to the desired interface. A profile is a XML file that contain the specifications of the network. Figure 6.2 shows the **DirectDemocracy** network profile. A new profile is added with the command:

`netsh wlan add profile filename="DirectDemocracy.XML" interface= "Interface name"`.

The system connect to this network using this command:

```
netsh wlan connect name="DirectDemocracy" interface="Interface name".
```

```
-<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/profile/v1">
    <name>DirectDemocracy</name>
   -<SSIDConfig>
     -<SSID>
         <hex>44697265637444656D6F6372616379</hex>
         <name>DirectDemocracy</name>
      </SSID>
      <nonBroadcast>false</nonBroadcast>
    </SSIDConfig>
    <connectionType>IBSS</connectionType>
    <connectionMode>manual</connectionMode>
   -<MSM>
     -<security>
       -<authEncryption>
           <authentication>open</authentication>
           <encryption>none</encryption>
           <useOneX>false</useOneX>
        </authEncryption>
      </security>
    </MSM>
 </WLANProfile>
```

Figure 6.2: DirectDemocracy.XML Ad Hoc Network Profile

Setting IP Address

When the agent connect to **DirectDemocracy** it will set the IP of the network. The command used to change the IP is :`netsh interface IP set address InterfaceName static IPAddress SubnetMask`. This command demand an administrative permission in order to run it. In some versions of Windows (such as Windows 7 Professional) the command runs successfully in the following format:

`RUNAS /savecred /USER:UserName "netsh interface IP set address InterfaceName static IPAddress SubnetMask"`.

In order to run the command in other types of Windows versions we provide the command in the widget to be copied by the user. This command should be implemented in an administrator command prompt window. In Windows, the IP is assigned only when two devices come into close proximity. If the IP is not assigned to the network, the software will not broadcast data until a valid broadcast address is obtained. We can learn that other peers are reachable on the Ad Hoc network by checking whether the IP of the network is displayed by `ipconfig`. When the user start broadcasting the system perform "ipconfig" command every 1 second.

Disconnect from DirectDemocracy

The user can disconnect from **DirectDemocracy** by deselecting it from the wireless widget. The command is:

`netsh wlan disconnect interface=''InterfaceName''`.

Setting the IP from static to DHCP require administrator privilege as with setting IP value. The following command will run on some versions (Windows 7 Professional):

`RUNAS /savecred /USER:UserName "netsh int IP set address InterfaceName DHCP"`.

In general, wireless cards are configured to automatically reconnect to the user preferred network. Note that further scripts can be added to capture user

wireless configuration before connecting to **DirectDemocracy** to be used when user deselect from it.

6.2.2 Linux

Detect Wireless Configuration

We use the command:

`IP link show`.

It gets all available Interfaces names. For each Interface we use the command `iwconfig` to get current SSID. Lastly, to obtain IP address we use the command: `IP addr list InterfaceName`.

Connect to DirectDemocracy

Figure 6.3 shows the script used in Linux to connect to a new Ad Hoc network.

```
service network-manager stop
ip link set InterfaceName down
iwconfig InterfaceName mode ad-hoc
iwconfig InterfaceName channel 11
iwconfig InterfaceName essid "SSID"
iwconfig InterfaceName key off
ip link set InterfaceName up
ip addr add IP
/sbin/ifconfig InterfaceName IP broadcast BroadcastAddress
```

Figure 6.3: Script to connect to a new Ad Hoc network in Linux

Disconnect from DirectDemocracy

The user can disconnect from **DirectDemocracy** by deselecting it from the wireless widget. The command in Linux is :

`service-network manager start`.

6.3 Broadcasting Server and Client

The architecture of the system is depicted in Figure 6.4. Each agent starts a server bound to local port UDP/54321 and accepting broadcast messaged. Our experiments were done with the Network ID of the network card set to "10/8". The local Host ID part of the IP is set to a random value. If the random part of the IP is considered insufficient to avoid IP collisions between peers, an additional *random identifier* is also generated to uniquely detect the agent, and messages tagged with this identifier can be discarded assuming that their source is the agent of the server. When the device has more than one wireless card, the agent can be configured to only use a subset of them for this protocol. Each agent has a client that broadcasts messages on the network interfaces allocated to our protocol, sending them to the address "10.255.255.255:54321" from a set of queues prepared with preloaded messages. A small pause (e.g. 5 ms) can be introduced between the transmission of packets, as this was found to slightly improve transmission rates as well as CPU load (see Chapter 7).

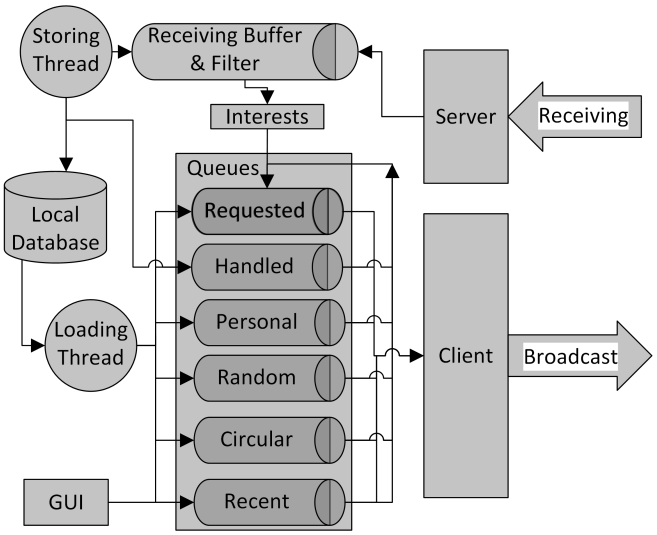

Figure 6.4: Architecture of the Peer

Each of the queues with preloaded messages has a special policy as to the type of contained items (*personal, similar to personal, recent, random, round-robin, requested*) and its mechanisms for loading and reloading. The broadcast client picks items from the various existing queues based on a probability distribution that can be specified by the user. We experiment with various heuristics for specifying these probabilities. To maximize its dissemination efficiency, the probability of sending items of interest must grow with the number of receivers having expressed that interest (potentially serving only the items of interest to most current peers). Before broadcasting a message, the client prepends to it a header describing: *the interests of the current user*, its *random identifier*, and *available GPS data about current location and velocity*. Potentially this header can include extra information about the content of the body of the message

(such as GIDs of organizations, motions, etc) to help receivers decide faster on storing or dropping messages that are not of interest. The existence of peers that drop messages not tagged with interest could push self-interested agents to provide this extra information (which otherwise reduces their bandwidth).

The servers may not be fast enough in handling and storing all the data they can receive in real time and therefore incoming data is stored in buffers. Our server has a receiving buffer of size B_r set to 20000 messages (average message size being measured to be 5KB in the current experiments). The server extracts the interests advertised by peers from the header of received messages and enqueues all the message bodies deemed new based on their size (or hash). A separate *storing thread* is used to dequeue received messages and to store their data based on the aforementioned algorithms.

If the receiving buffer is full, until the internal *storing thread* frees some entries, the server drops new incoming messages except if they are tagged in their header with information specifying that they contain items of interest to the receiver (in which case these messages are used to replace untagged messages from the buffer).

Chapter 7

Experiments

For the reported measurements, the databases of the agents were filled with 60000 votes for 10 organizations (O_1 to O_{10}) and 3816 motions, 9094 justifications, 629 constituents, and 4486 witness stances. These numbers were chosen based on our estimation of the ratio of the various types of items in a deployed system. To generate these items we implement a simulator that allocates each new generated vote probabilistically. First we manually generated a certain number of organizations. Then, each generated vote is allocated to a new organization with probability 10^{-5}, otherwise it is uniformly assigned to one of the existing organizations. Similarly, each vote is allocated with probability 10^{-2} to a new constituent. The size of the text of each artificially generated motion is 1000 characters and the size of each justification is 300 characters.

We performed experiments with transitive dissemination across several ve-

hicles, validating the fact that data can be disseminated between cars that do not have direct contact. First we report numerical results about the measured characteristics of the communication between immediately connected nodes.

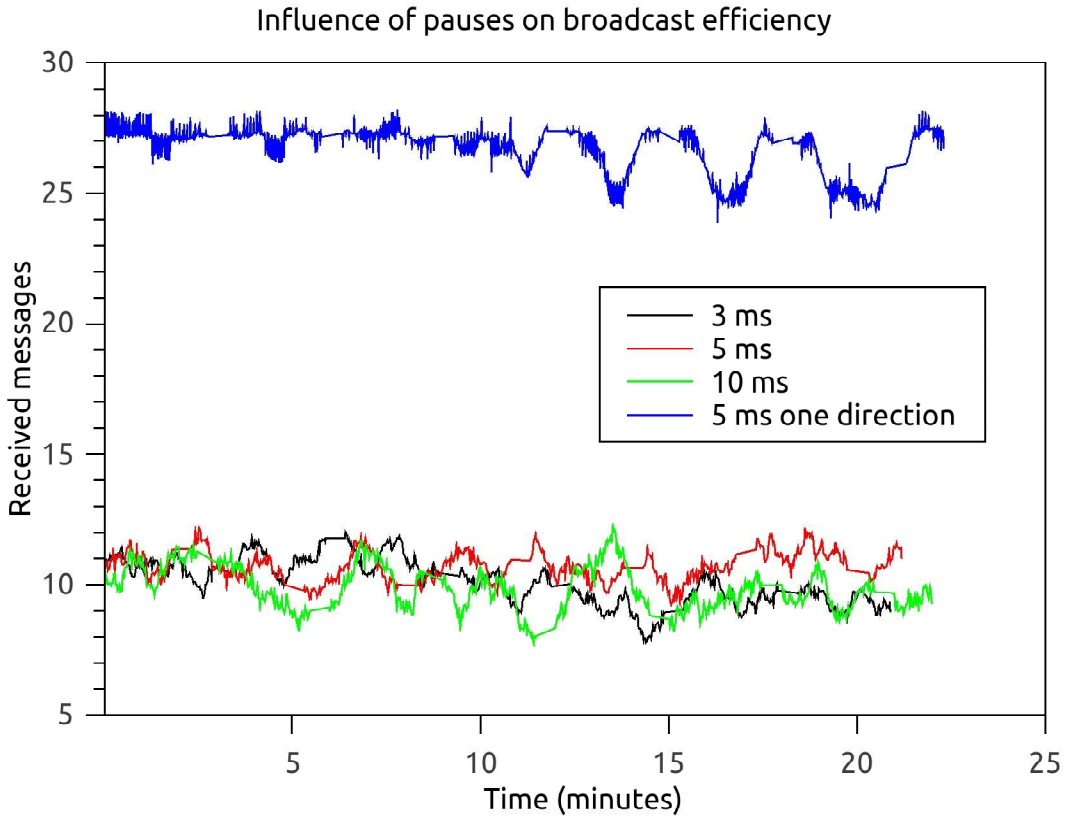

Figure 7.1: Experiments measuring the speed of messages transmitted (v_M). Averages for duplex communication is: 3ms pauses at 10.2 $\frac{\text{msg}}{sec}$, 5ms pauses at 10.78 $\frac{\text{msg}}{sec}$, and 10ms pauses at 9.97 $\frac{\text{msg}}{sec}$

We measure the speed of communication v_M between two nodes in ideal conditions (when the nodes are placed far from other wireless devices). Communication is measured between an HP G62-111EE with 3GB RAM and an Acer Aspire P5WE0 with 4GB RAM running Ubuntu 12.04 on an I3 proces-

sor. Preliminary measurements were made with different pause duration (0, 3, 5, 10, 15, 250, 500, 750, 1000 ms) between transmitted packets. This pause impacts on the number of packet collisions, and therefore on the transmission efficiency. More extensive measurements were performed on the values that showed promise (3, 5, 10 ms). Measurements were taken over 25 minutes of communication for each pause duration and for each of the following two cases: when both devices transmit data. and when only one device transmits data. The results, averaged over a sliding window of size 30 seconds, are displayed in Figure 7.1. The maximum value of 26.7 messages per second for one direction broadcasting at 5 ms pause duration is used as reference.

roads	speed	T_B	$M = v_M \cdot T_B$
Parking lot – crowded	15	15	158
Street – open area	40	4.3	50
Street – school area	35	2.6	15
Highway – free	70	6.3	91
Highway – trucks	70	4.5	34

Table 7.1: Average time of encounter (seconds) and number of exchanged messages in this time for various vehicle speeds (mph) and environments, with communication in one direction (5 ms pauses)

We measure an estimate of the range of communication D and of the time T_B during which two devices are able to communicate. These measurements are performed with laptops found in two vehicles moving in opposite direction in several scenarios: *in a parking lot (crowded) at 15 mph, on a city street in an open area (10 wireless networks) with median strip at 40 mph, on a city street*

close to a school (35 wireless networks) with median strip at 35 mph, on an empty highway with median strip at 70 mph, and on the same highway (with trucks separating the communicating cars). The measurement in the parking lot and on the city street were averaged over 10 encounters. The numbers of messages successfully transmitted in the three scenarios are shown in Table 7.1, as well as the duration T_B estimated from logs. It can be noticed that the speed of communication between devices is strongly influenced by the number of wireless networks in that area.

To have all messages available for a peer encountered while driving in opposite direction in a crowded parking lot, the sender needs queues of size $B_s \geq \frac{D \cdot v_M}{v_B}$, which correspond to the maximum number of messages M in Table 7.1.

Dissemination over Chains of Vehicles To evaluate and confirm empirically the dissemination between vehicles that do not meet each other but communicate via other intermediary vehicles, we run experiments with three cars: A, B, and C. The car C contains a device with a preloaded database (as per the previous experiments) while the devices in the other two cars are initially empty. We evaluate two topologies of communication patterns between these vehicles: *chain* and *triangle*. For each topology the vehicles have a fix trajectory that they repeat 20 times, synchronized in such a way that pairs of vehicles meet at the same location. Also we evaluate the impact of the studied heuristics and of

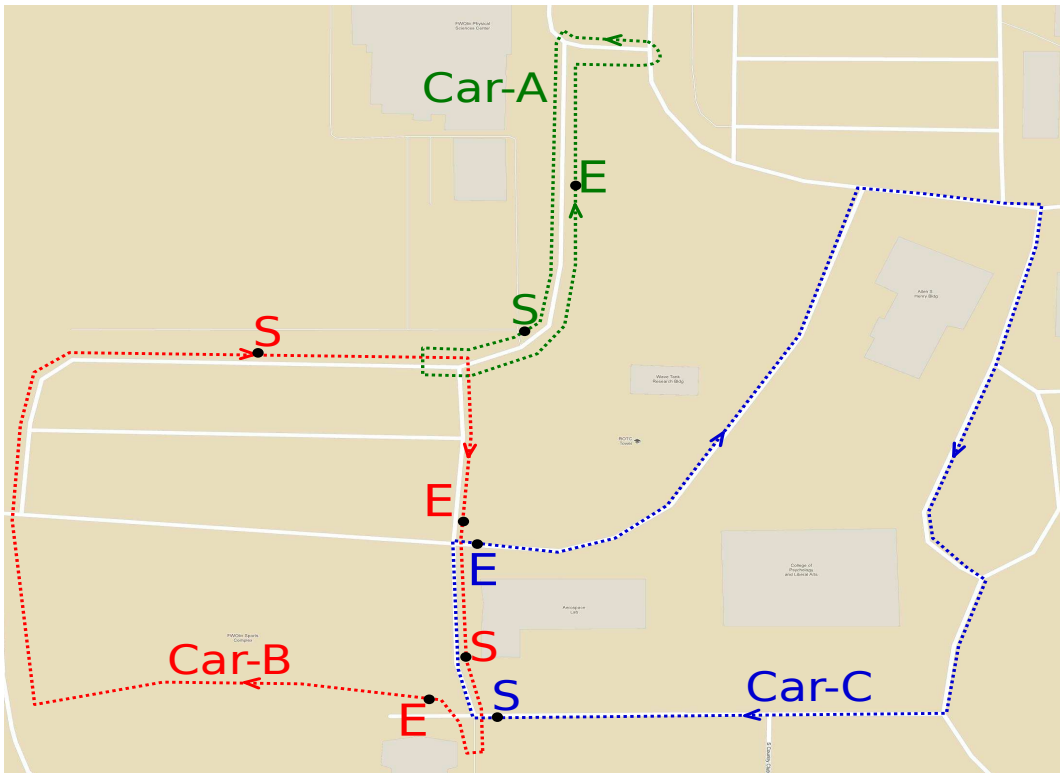

Figure 7.2: Trajectories in the chain topology. Areas of communication for each meeting point start at the corresponding S point and end at the corresponding E point, for each car.

the user types (interests) on the efficiency of dissemination.

The trajectory of these cars on a map in the chain topology is shown in Figure 7.2. Two curves in the diagram in Figure 7.3 shows the number of new data items received and stored in each of the two cars during 20 rounds of encounters with this topology. We remark that the *Car A chain* curve shows that its device receives approximately 60% of what is received by the device in car B (see *Car B chain*). It is nevertheless logical to expect that the ratio would decrease with time and rounds due to the expected decrease in overlap between

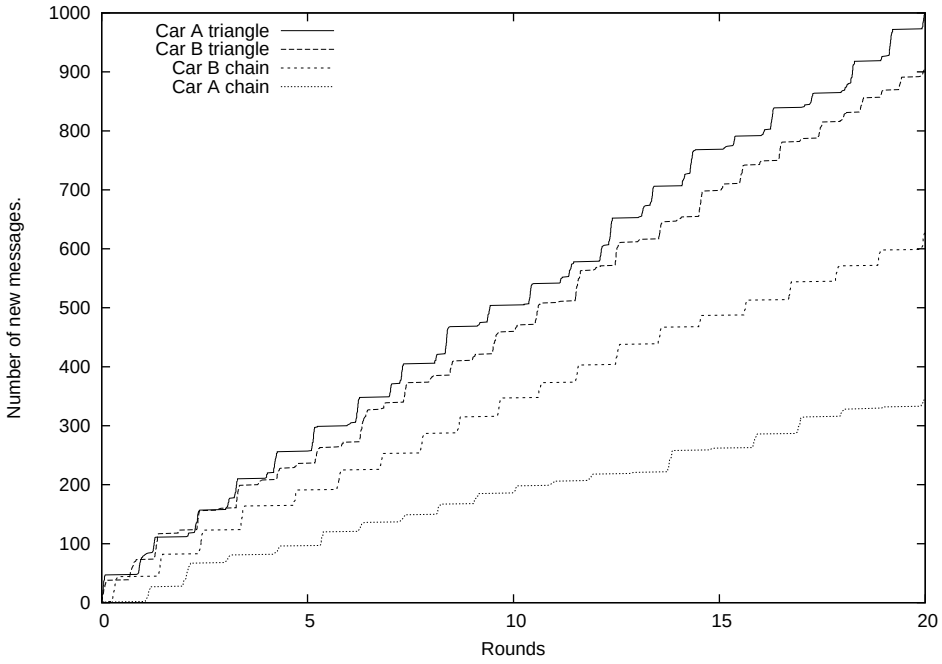

Figure 7.3: Received items (votes and witness stances) for cars A and B in the chain and triangle topologies. The ratio votes to witness stances is approx 2:1.

messages received by B from C, and messages sent by B when her database increases. The usage of queue *handled* (containing data recently received from other peers) is meant to mitigate this effect.

A comparison is made with the situation when the three cars communicate according to a triangular topology (see Figure 7.4). We see that the number of messages received by the car B (and car A) in this topology is approximately 50% more than the number of messages received by car B in the chain topology.

Impact of Interests on Efficiency We count the number of messages of interest to the receiver, successfully transmitted to a given peer, in scenarios with

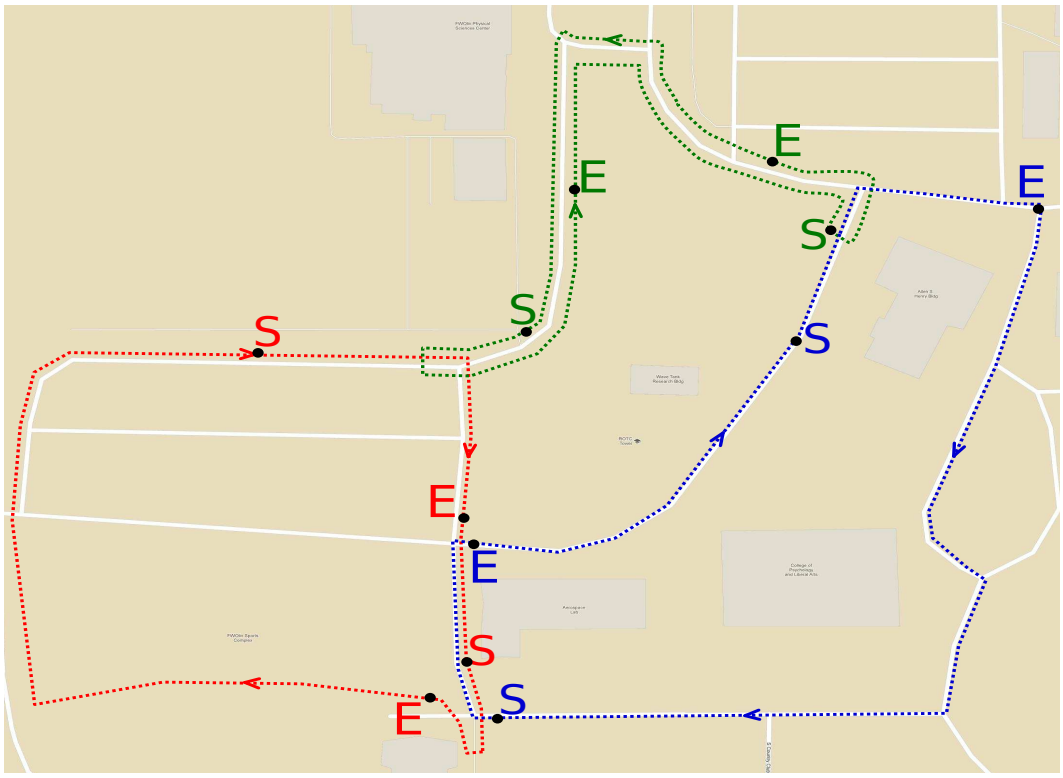

Figure 7.4: Trajectories in the triangle topology. Areas of communication for each meeting point start at the corresponding S point and end at the corresponding E point, for each car.

the studied peer expressing interests in two organizations, while other peers also express their interests. The graph in Figure 7.5 shows the number of received messages given the number of different interests considered by the sender. It can be observed that the efficiency for the receiver decreases with the number of interests submitted by neighboring peers.The other straight horizontal line in the graph shows the efficiency of the receiver when no interests are advertised by anybody and the sender transmits randomly data from its 10 organizations. Note that the efficiency of the server is given by the sum of the efficiency of

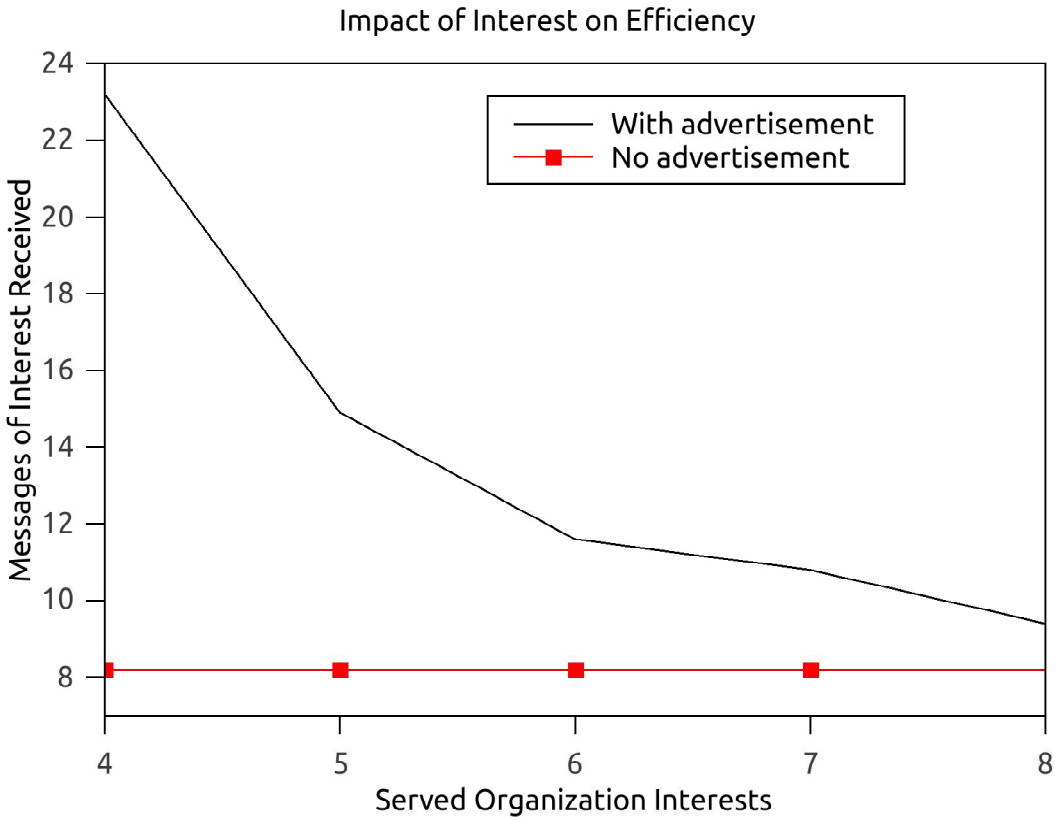

Figure 7.5: Comparison of efficiency with and without advertisement of interests.

its receivers, being expected to grow monotonically with the number of peer vehicles receiving its data. The efficiency of the sender in disseminating its data without advertisement of interest be smaller than with advertisement of interest except when all the available data of equal interest to receivers.

With the computed parameters, when we use a single sending queue with randomly picked data or with round-robin transmission, the occurrence of personally generated items has a negligible probability and the utility is practically equivalent to sending only messages of type other. Assuming that the trans-

mission of each item has a utility of 1¢ for the sender and the utility of a personally generated item is 10¢, the obtained utility per second with $A = 2$ vehicles driving in the same direction and $B = 2$ vehicles traveling in opposite direction on a highway is $\approx 107\frac{¢}{s}$ (based on Equation 5.3). For the case $N_P = 10$, on a highway, the speed of sending messages with personal items has to be $v_M^P \geq \frac{B_s}{T_B} = \frac{10}{4.5} = 2.2$. Therefore the speed of sending the other types of messages (assumed to be all of type "other") can be $v_M^{max} - v_M^P \approx 24.5$. The total utility with this configuration is $88 + 98 = 186\frac{¢}{s}$ ($88\frac{¢}{s}$ for personal messages). This proves that it is useful to separate messages into queues of specialized types (gaining $186\frac{¢}{s}$ rather than $107\frac{¢}{s}$).

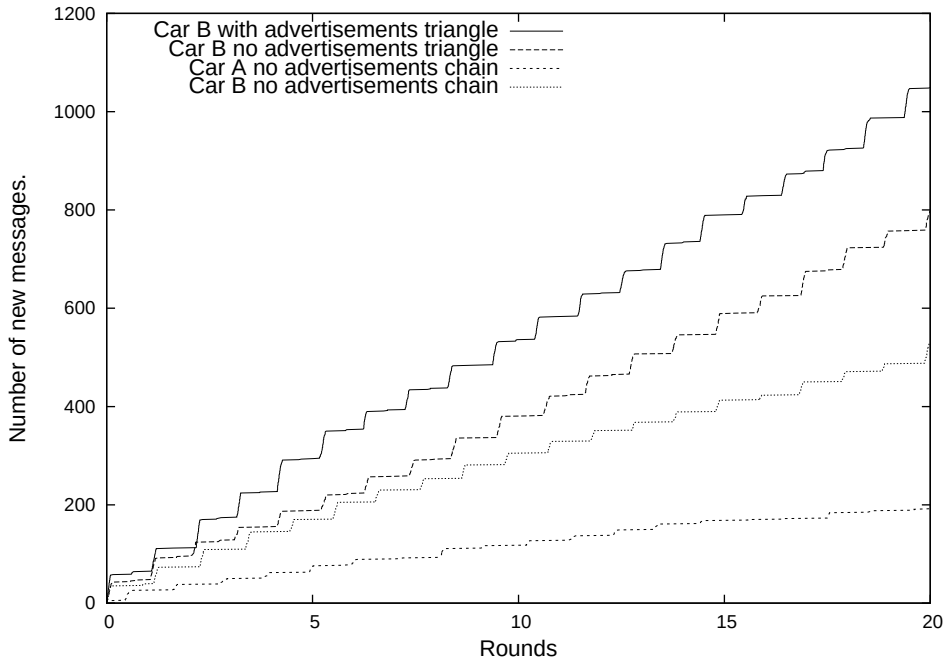

Figure 7.6: Impact of interest advertisement.

Empirical Results with Interests We ran experiments with the three cars (A, B, and C) where A is only interested in storing and forwarding the organizations O_1 to O_7, and B is only interested in storing and forwarding organizations O_4 to O_{10}. The impact of advertising their interests is shown in Figure 7.6, with an improvement of 28%, proportional with the ratio of interest in the available organizations. It can be seen that, when devices filter received data based on their interests, car A eventually receives a lower fraction (36%) of the data received by B than in the absence of such filtering (54%, see Figure 7.3). Advertisement of interests compensates for this difference.

Chapter 8

Conclusion

A set of techniques for dissemination of data in decentralized opinion polls via Vehicular wireless Ad Hoc Networks of self-interested peers is proposed and evaluated. For comparing heuristics we compute the utility of achieved dissemination from the perspective of the given sender. The long term goal is to find the behavior at equilibrium of self-interested senders. A utility model is discussed where the highest utility is for items generated by the sender, followed by items with similar opinion, while the least utility is assigned to items of opposing opinion (which may even have a negative utility for the sender).

Based on a set of experiments with our VANET implementation we compute the parameters of a model for the vehicle to vehicle interaction. Strategies for broadcasting based on several queues are evaluated as well as percentages of broadcast time to allocate to different types of data items. The tested heuristics

can be uninformed or informed with data received from peers such as their interests, identity, position and relative speed and bearing. Interests of peers are expressed in terms such as opinion (vote choice), issues (motions), voters (constituents), or topics (organization).

Separate outgoing queues can be maintained for data of different types (random, generated by sender, similar with sender, opposing senders, others). Cars traveling in opposite direction should get the most valuable data (generated by this sender) while cars traveling in the same direction and in contact for a long time should eventually fully synchronize with the sender on all items with positive utility and of interest to them.

Bibliography

[1] S. Busanelli, G. Ferrari, and S. Panichpapiboon. Efficient broadcasting in ieee 802.11 networks through irresponsible forwarding. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –6, 30 2009-dec. 4 2009.

[2] Murat Caliskan, Daniel Graupner, and Martin Mauve. Decentralized discovery of free parking places. In *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, VANET '06, pages 30–39, New York, NY, USA, 2006. ACM.

[3] Wai Chen and Shengwei Cai. Ad hoc peer-to-peer network architecture for vehicle safety communications. *Communications Magazine, IEEE*, 43(4):100 – 107, april 2005.

[4] L. Chisalita and N. Shahmehri. A peer-to-peer approach to vehicular communication for the support of traffic safety applications. In *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pages 336 – 341, 2002.

[5] EUREKA. `http://www.eurekanetwork.org/project/-/id/6252`, 2010.

[6] Meng Guo, M.H. Ammar, and E.W. Zegura. V3: a vehicle-to-vehicle live video streaming architecture. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 171 – 180, march 2005.

[7] Hsu-Chun Hsiao, Ahren Studer, Chen Chen, Adrian Perrig, Fan Bai, Bhargav Bellur, and Aravind Iyer. Flooding-resilient broadcast authentication for vanets. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 193–204. ACM, 2011.

[8] P. Jacquet, P. Mhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. pages 62–68, 2001.

[9] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[10] Gunnar Karlsson, Vincent Lenders, and Martin May. Delay-tolerant broadcasting. *IEEE Transactions on Broadcasting*, 53(1):369–381, March 2007.

[11] T. Kaur, A. Malhi, and A.K. Verma. Simulation and comparision of aodv and dsr routing protocols in vanets. *International Journal of Computing, Intelligent and Communication Technologies (IJCICT)*, 1(1):51–56, Oct 2012.

[12] Srinivasan Keshav. The network simulator ii. `http://www.isi.edu/nsnam/ns/`, 2013.

[13] K.C. Lee, Seung-Hoon Lee, Ryan Cheung, Uichin Lee, and M. Gerla. First experience with cartorrent in a real vehicular ad hoc network testbed. In *2007 Mobile Networking for Vehicular Environments*, pages 109 –114, may 2007.

[14] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, and Mario Gerla. Code torrent: content distribution using network coding in vanet. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, MobiShare '06, pages 1–5, New York, NY, USA, 2006. ACM.

[15] Technet Microsoft. `http://technet.microsoft.com/en-us/library/bb490943.aspx`, 2013.

[16] Tamer Nadeem, Sasan Dashtinezhad, Chunyuan Liao, and Liviu Iftode. Trafficview: traffic data dissemination using car-to-car communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3):6–19, July 2004.

[17] A. Nandan, S. Das, G. Pau, M. Gerla, and M.Y. Sanadidi. Co-operative downloading in vehicular ad-hoc wireless networks. In *Wireless On-demand Network Systems and Services, 2005. WONS 2005. Second Annual Conference on*, pages 32 – 41, jan. 2005.

[18] Alok Nandan, Alok N, Saurabh Tewari, Shirshanka Das, Mario Gerla, and Leonard Kleinrock. Adtorrent: Delivering location cognizant advertisements to car networks, 2006.

[19] J. Nzouonta, N. Rajgure, Guiling Wang, and C. Borcea. Vanet routing on city roads using real-time vehicular traffic information. *Vehicular Technology, IEEE Transactions on*, 58(7):3609 –3626, sept. 2009.

[20] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90 –100, feb 1999.

[21] O. Tonguz, N. Wisitpongphan, F. Bai, P. Mudalige, and V. Sadekar. Broadcasting in vanet. In *2007 Mobile Networking for Vehicular Environments*, pages 7 –12, may 2007.

[22] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wirel. Netw.*, 8(2/3):153–167, March 2002.

[23] S.Y. Wang, C.C. Lin, Y.W. Hwang, K.C. Tao, and C.L. Chou. A practical routing protocol for vehicle-formed mobile ad hoc networks on the roads. In *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pages 161 – 166, sept. 2005.

[24] L. Wischhof, A. Ebner, and H. Rohling. Information dissemination in self-organizing intervehicle networks. *Intelligent Transportation Systems, IEEE Transactions on*, 6(1):90 – 101, march 2005.

[25] Lars Wischhof, Andr Ebner, and Hermann Rohling. Self-organizing traffic information system based on car-to-car, 2004.

[26] N. Wisitpongphan, Fan Bai, P. Mudalige, and O.K. Tonguz. On the routing problem in disconnected vehicular ad-hoc networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2291 –2295, may 2007.

[27] Xi Yu, Huaqun Guo, and Wai-Choong Wong. A reliable routing protocol for vanet communications. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1748 –1753, july 2011.

[28] Yang Zhang, Jing Zhao, and Guohong Cao. Roadcast: A popularity aware content sharing scheme in vanets. In *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on*, pages 223 –230, june 2009.

Appendix

ASN1 Definitions

```
D_Message ::= IMPLICIT SEQUENCE {
        sender [APPLICATION 0] D_PeerAddress OPTIONAL,
        Peer [APPLICATION 1] D_PeerAddress OPTIONAL,
        interest WB_ASN_Interest [APPLICATION 2] OPTIONAL,
        organization [APPLICATION 3] D_Organization OPTIONAL,
        motion [APPLICATION 4] D_Motion OPTIONAL,
        constituent [APPLICATION 5] D_Constituent OPTIONAL,
        witness [APPLICATION 6] D_Witness OPTIONAL,
        vote [APPLICATION 7] D_Vote OPTIONAL,
        signature [APPLICATION 8] OCTET_STRING OPTIONAL,
        neighborhoods D_Neighborhood [APPLICATION 9] OPTIONAL,
        recent_senders [APPLICATION 10] SEQUENCE OF UTF8String OPTIONAL,
        dictionary_GIDs [APPLICATION 17] UTF8String OPTIONAL
}

D_PeerAddress ::= IMPLICIT SEQUENCE {
        globalID PrintableString,
        name UTF8String OPTIONAL,
        slogan [APPLICATION 0] UTF8String OPTIONAL,
```

```
        creation_date GeneralizedDate OPTIONAL,
        address SEQUENCE OF TypedAddress OPTIONAL,
        broadcastable BOOLEAN,
        signature_alg PrintableString OPTIONAL,
        served_orgs [APPLICATION 12] SEQUENCE OF D_PeerOrgs OPTIONAL,
        signature OCTET_STRING
}

D_Organization ::= IMPLICIT SEQUENCE {
        global_organization_ID PrintableString,
        name UTF8String OPTIONAL,
        last_sync_date GeneralizedDate OPTIONAL,
        params [APPLICATION 1] D_OrgParams OPTIONAL,
        concepts [APPLICATION 2] D_OrgConcepts OPTIONAL,
        signature OCTET_STRING OPTIONAL,
        signature_initiator [APPLICATION 14] OCTET_STRING OPTIONAL,
        creator [APPLICATION 0] D_PeerAddress OPTIONAL,
        neighborhoods [APPLICATION 3] SEQUENCE OF ASNNeighborhoodOP OPTIONAL,
        constituents [APPLICATION 4] SEQUENCE OF ASNConstituentOP OPTIONAL,
        witnesses [APPLICATION 5] SEQUENCE OF D_Witness OPTIONAL,
        motions [APPLICATION 6] SEQUENCE OF D_Motion OPTIONAL,
        justifications [APPLICATION 7] SEQUENCE OF D_Justification OPTIONAL,
        signatures [APPLICATION 8] SEQUENCE OF D_Vote OPTIONAL,
        translations [APPLICATION 9] SEQUENCE OF D_Translations OPTIONAL,
        news [APPLICATION 10] SEQUENCE OF D_News OPTIONAL,
        requested_data [APPLICATION 11] SEQUENCE OF D_Message OPTIONAL
}

D_Constituent ::= [APPLICATION 48] IMPLICIT SEQUENCE {
        global_organization_ID PrintableString OPTIONAL,
        global_constituent_id [APPLICATION 0] PrintableString OPTIONAL,
        surname [APPLICATION 1] UTF8String OPTIONAL,
        forename [APPLICATION 15] UTF8String OPTIONAL,
```

```
        address [APPLICATION 2] SEQUENCE OF D_FieldValue OPTIONAL,
        email [APPLICATION 3] PrintableString OPTIONAL,
        creation_date [APPLICATION 4] GeneralizedDate OPTIONAL,
        global_neighborhood_ID [APPLICATION 10] PrintableString OPTIONAL,
        neighborhood [APPLICATION 5] SEQUENCE OF D_Neighborhood OPTIONAL,
        picture [APPLICATION 6] OCTET_STRING OPTIONAL,
        hash_alg PrintableString OPTIONAL,
        signature [APPLICATION 7] OCTET_STRING OPTIONAL,
        global_constituent_id_hash [APPLICATION 8] PrintableString OPTIONAL,
        certificate [APPLICATION 9] OCTET_STRING OPTIONAL,
        languages [APPLICATION 11] SEQUENCE OF PrintableString OPTIONAL,
        global_submitter_id [APPLICATION 12] PrintableString OPTIONAL,
        slogan [APPLICATION 13] UTF8String OPTIONAL,
        weight [APPLICATION 14] UTF8String OPTIONAL,
        submitter [APPLICATION 15] D_Constituent OPTIONAL,
        external BOOLEAN,
        revoked BOOLEAN
}

D_Neighborhood ::= IMPLICIT SEQUENCE {
        global_organization_ID PrintableString OPTIONAL,
        global_neighborhood_ID [APPLICATION 0] PrintableString OPTIONAL,
        name [APPLICATION 1] UTF8String OPTIONAL,
        name_lang [APPLICATION 2] UTF8String OPTIONAL,
        description [APPLICATION 3] UTF8String OPTIONAL,
        boundary SEQUENCE OF ASNPoint OPTIONAL,
        name_division [APPLICATION 4] UTF8String OPTIONAL,
        names_subdivisions [APPLICATION 6] SEQUENCE OF UTF8String OPTIONAL,
        parent_global_ID [APPLICATION 7] PrintableString OPTIONAL,
        parent [APPLICATION 8] D_Neighborhood OPTIONAL,
        submitter_global_ID [APPLICATION 9] PrintableString OPTIONAL,
        submitter [APPLICATION 10] D_Constituent OPTIONAL,
```

```
        creation_date [APPLICATION 11] GeneralizedDate OPTIONAL,
        picture [APPLICATION 12] OCTET_STRING OPTIONAL,
        signature [APPLICATION 13] OCTET_STRING OPTIONAL
}

D_Witness ::= IMPLICIT SEQUENCE {
        hash_alg PrintableString,
        global_witness_ID PrintableString,
        witness_category UTF8String,
        witnessed_global_neighborhoodID PrintableString,
        witnessed_global_constituentID PrintableString,
        witnessing_global_constituentID PrintableString,
        global_organization_ID PrintableString,
        creation_date GeneralizedDate,
        signature OCTET_STRING,
        witnessing [APPLICATION 0] D_Constituent OPTIONAL,
        witnessed [APPLICATION 1] D_Constituent OPTIONAL
}

D_Vote ::= IMPLICIT SEQUENCE {
        hash_alg [APPLICATION 0] PrintableString OPTIONAL,
        global_vote_ID [APPLICATION 1] PrintableString OPTIONAL,
        global_constituent_ID [APPLICATION 2] PrintableString OPTIONAL,
        global_motion_ID [APPLICATION 3] PrintableString OPTIONAL,
        global_justification_ID [APPLICATION 4] PrintableString OPTIONAL,
        choice [APPLICATION 5] UTF8String OPTIONAL,
        format [APPLICATION 6] UTF8String OPTIONAL,
        creation_date [APPLICATION 7] GeneralizedDate OPTIONAL,
        signature [APPLICATION 8] OCTET_STRING OPTIONAL,
        constituent [APPLICATION 9] D_Constituent OPTIONAL,
        motion [APPLICATION 10] D_Motion OPTIONAL,
        justification [APPLICATION 11] D_Justification OPTIONAL,
        global_organization_ID [APPLICATION 12] PrintableString OPTIONAL
```

```
}

D_Motion ::= IMPLICIT SEQUENCE {
        hash_alg [APPLICATION 0] PrintableString OPTIONAL,
        global_motionID [APPLICATION 1] PrintableString OPTIONAL,
        motion_title [APPLICATION 2] D_Document_Title OPTIONAL,
        motion_text [APPLICATION 3] D_Document OPTIONAL,
        global_constituent_ID [APPLICATION 4] PrintableString OPTIONAL,
        global_enhanced_motionID [APPLICATION 5] PrintableString OPTIONAL,
        global_organization_ID [APPLICATION 6] PrintableString OPTIONAL,
        creation_date [APPLICATION 7] GeneralizedDate OPTIONAL,
        signature [APPLICATION 8] OCTET_STRING OPTIONAL,
        choices [APPLICATION 9] SEQUENCE OF D_MotionChoice OPTIONAL,
        constituent [APPLICATION 10] D_Constituent OPTIONAL,
        enhanced [APPLICATION 11] D_Motion OPTIONAL,
        organization [APPLICATION 12] D_Organization OPTIONAL,
        category [APPLICATION 13] UTF8String OPTIONAL
}

D_Justification ::= IMPLICIT SEQUENCE {
        hash_alg [APPLICATION 0] PrintableString OPTIONAL,
        global_justificationID [APPLICATION 1] PrintableString OPTIONAL,
        global_motionID [APPLICATION 2] PrintableString OPTIONAL,
        global_answerTo_ID [APPLICATION 3] PrintableString OPTIONAL,
        global_constituent_ID [APPLICATION 4] PrintableString OPTIONAL,
        justification_title [APPLICATION 5] D_Document_Title OPTIONAL,
        justification_text [APPLICATION 6] D_Document OPTIONAL,
        creation_date [APPLICATION 9] GeneralizedDate OPTIONAL,
        signature [APPLICATION 10] OCTET_STRING OPTIONAL,
        motion [APPLICATION 11] D_Motion OPTIONAL,
        constituent [APPLICATION 12] D_Constituent OPTIONAL,
        answerTo [APPLICATION 13] D_Justification OPTIONAL,
        global_organization_ID [APPLICATION 14] PrintableString OPTIONAL
```

```
}

WB_ASN_Interest ::=IMPLICIT SEQUENCE {
                        organizations [APPLICATION 0] SEQUENCE OF PrintableString OPTIONAL,
                        motions [APPLICATION 1] SEQUENCE OF PrintableString OPTIONAL,
                        constituents [APPLICATION 2] SEQUENCE OF PrintableString OPTIONAL,
                        justifications [APPLICATION 3] SEQUENCE OF PrintableString OPTIONAL
}

D_Document_Title ::=IMPLICIT SEQUENCE {
                         title_document [APPLICATION 0] D_Document OPTIONAL
}

D_Document ::= IMPLICIT SEQUENCE {
          format [APPLICATION 0] PrintableString OPTIONAL,
          document [APPLICATION 1] OCTET_STRING OPTIONAL
}

D_News ::= IMPLICIT SEQUENCE {
         hash_alg [APPLICATION 0] PrintableString OPTIONAL,
         global_news_ID [APPLICATION 1] PrintableString OPTIONAL,
         title [APPLICATION 2] D_Document_Title OPTIONAL,
         news [APPLICATION 3] D_Document OPTIONAL,
         global_constituent_ID [APPLICATION 4] PrintableString OPTIONAL,
         global_organization_ID [APPLICATION 6] PrintableString OPTIONAL,
         creation_date [APPLICATION 7] GeneralizedDate OPTIONAL,
         signature [APPLICATION 8] OCTET_STRING OPTIONAL,
         constituent [APPLICATION 10] D_Constituent OPTIONAL,
         organization [APPLICATION 12] D_Organization OPTIONAL,
         global_motion_ID [APPLICATION 14] PrintableString OPTIONAL,
         motion [APPLICATION 13] D_Motion OPTIONAL
}

D_Translations ::= IMPLICIT SEQUENCE {
```

```
        hash_alg PrintableString,
        global_translation_ID PrintableString,
        value PrintableString,
        value_lang PrintableString,
        value_ctx PrintableString,
        translation PrintableString,
        translation_lang PrintableString,
        translation_charset PrintableString,
        translation_flavor PrintableString,
        global_constituent_ID PrintableString,
        global_organization_ID PrintableString,
        creation_date GeneralizedDate,
        signature OCTET_STRING
}

D_OrgConcepts ::= IMPLICIT SEQUENCE {
         name_forum [APPLICATION 0] SEQUENCE OF UTF8String OPTIONAL,
         name_justification [APPLICATION 1] SEQUENCE OF UTF8String OPTIONAL,
         name_motion [APPLICATION 2] SEQUENCE OF UTF8String OPTIONAL,
         name_organization [APPLICATION 3] SEQUENCE OF UTF8String OPTIONAL
}

D_PeerOrgs ::= SEQUENCE {
         org_name [APPLICATION 0] UTF8String OPTIONAL,
         global_organization_IDhash [APPLICATION 1] PrintableString OPTIONAL,
         global_organization_ID [APPLICATION 2] PrintableString OPTIONAL
}

D_FieldValue ::= IMPLICIT SEQUENCE {
        field_extra_GID PrintableString,
        value OCTET_STRING,
        field_GID_above PrintableString,
        field_GID_default_next PrintableString,
```

```
        value_lang PrintableString,
        neighborhood D_Neighborhood OPTIONAL
}

D_OrgParams ::= IMPLICIT SEQUENCE {
        certifMethods ENUMERATED,
        hash_org_alg [APPLICATION 0] PrintableString OPTIONAL,
        creation_time [APPLICATION 1] GeneralizedDate OPTIONAL,
        creator_global_ID [APPLICATION 2] PrintableString OPTIONAL,
        category [APPLICATION 3] UTF8String OPTIONAL,
        certificate [APPLICATION 4] OCTET_STRING OPTIONAL,
        default_scoring_options [APPLICATION 5] SEQUENCE OF UTF8String OPTIONAL,
        instructions_new_motions [APPLICATION 6] UTF8String OPTIONAL,
        instructions_registration [APPLICATION 7] UTF8String OPTIONAL,
        description [APPLICATION 10] UTF8String OPTIONAL,
        languages [APPLICATION 8] SEQUENCE OF PrintableString OPTIONAL,
        orgParam [APPLICATION 9] SEQUENCE OF D_OrgParam OPTIONAL
}

orgParam::= IMPLICIT SEQUENCE {
        global_field_extra_ID PrintableString,
        can_be_provided_later BOOLEAN,
        certificated BOOLEAN,
        entry_size INTEGER,
        partNeigh INTEGER,
        required BOOLEAN,
        label [APPLICATION 0] UTF8String OPTIONAL,
        label_lang [APPLICATION 1] PrintableString OPTIONAL,
        default_value [APPLICATION 2] UTF8String OPTIONAL,
        default_value_lang [APPLICATION 3] PrintableString OPTIONAL,
        list_of_values [APPLICATION 4] UTF8String OPTIONAL,
        list_of_values_lang [APPLICATION 5] PrintableString OPTIONAL,
        tip [APPLICATION 6] UTF8String OPTIONAL,
```

```
        tip_lang [APPLICATION 7] PrintableString OPTIONAL
}

ASNNeighborhoodOP ::= IMPLICIT SEQUENCE {
        neighborhood D_Neighborhood,
        op INTEGER
}

ASNConstituentOP ::= IMPLICIT SEQUENCE {
        constituent D_Constituent,
        op INTEGER
}

TypedAddress ::= SEQUENCE {
        address UTF8String,
        type PrintableString
}

ASNPoint::= SEQUENCE {
        latitude REAL,
        longtitude REAL
}

D_MotionChoice ::= IMPLICIT SEQUENCE {
      name [APPLICATION 0] UTF8String,
      short_name [APPLICATION 1] UTF8String
}
```