Chapter 9

Ad hoc mobile OLAP

This chapter deals with an entirely different aspect of the mOLAP domain. So far, this dissertation has examined conventional server-client architectures. In this chapter, we extend the examined scenarios, in which clients exclusively rely on a server for query answering, by additionally incorporating ad hoc connections, and present the case of ad hoc mOLAP [110].

The chapter is structured as follows: In Section 9.2, we present the background of this approach, namely the network architecture, the data model and the emerging issues in multidimensional querying. Section 9.3 presents related work. In Section 9.4, we formalize the optimization problem. Section 9.5 introduces our query propagation and dissemination protocol, while in Section 9.6, we present the results of the protocol evaluation. Section 9.7 highlights yet unaddressed issues.

9.1 Introduction

Mobile ad hoc networks (MANETs) are self-configuring networks of mobile routers (and associated hosts) connected by wireless links, the union of which form an arbitrary topology. The routers are free to move randomly and organize themselves arbitrarily. Thus, the network's wireless topology may change rapidly and unpredictably. Such networks may operate in a standalone fashion, or may be connected to the larger internet.

MANETs have become increasingly popular in recent years, primarily due to the fact that they can be deployed without any infrastructure cost. Nevertheless, it appears more realistic for MANETs to comprise an extension of existing infrastructure than to operate completely independently.

In the main part of this dissertation, we introduced *FCLOS*, a mOLAP architecture explicitly designed for infrastructure based wireless networks. The main motivation of this chapter is initiated by observing infrastructure mOLAP clients remaining idle for periods of time, while at the same time they could have been serving neighboring clients' queries. We believe that ad-hoc mOLAP is capable of

assisting infrastructure mOLAP applications. Therefore, the proposed protocol, despite being able to operate in an entirely ad hoc way, is designed in order to assist existing infrastructure.

Unfortunately, as it shall be shown in the following sections, the fundamental ideas behind not only *FCLOS*, but infrastructure mOLAP in general as well, are not applicable for ad-hoc mOLAP. In this context, we present the case of ad-hoc mOLAP. To the very best of our knowledge, there is no existing approach dealing with mOLAP in MANETs. We introduce *QDGV*, a simple and robust query propagation and data dissemination protocol, which according to our experimental results, reduces the amount of generated traffic, query access time and energy consumption overhead. *QDGV* does not employ any cross-layer technique and is completely independent of the underlying MAC and ad hoc routing protocol.

It is imperative to underline that *QDGV* is designed to perform in realistic mOLAP application scenarios because we don't expect such scenarios to take place under high velocity or in large scale MANETs. *QDGV* is by no means an all purpose protocol, neither in regard to the running application nor in regard to the network topology.

9.2 Background

9.2.1 Architecture

Wireless networks can be categorized according to their connectivity mode into infrastructure based and ad hoc. In infrastructure wireless networks, nodes operate independently of each other, using the base station to communicate with other nodes and access services. Management strategies therefore only need to consider the node and its local applications. In ad hoc networks, nodes are highly interdependent, do not rely on any infrastructure (access points, wired servers, etc.), and must cooperate to provide routing and other services.

We discuss conventional server-client mOLAP architectures in Chapters 6, 7, 8. The corresponding architecture consists of a server that is responsible for handling incoming queries from multiple wireless nodes, like in Fig. 6.1. The server retrieves the appropriate data from a backend data source and runs a scheduling algorithm to serve queries. The main idea is to exploit the derivation semantics among sub-cubes in order to simultaneously serve multiple queries. Clients share a downlink channel that the server uses to deliver query answers. They continuously monitor this channel after making a request in order to tune in when appropriate data is transmitted.

Founded on this architecture, we now underline the focus of this chapter. Assume a setting with n mobile nodes $\{M_1, M_2, \ldots M_n\}$ each of which contains a portion P_i of the data cube, where there is no restriction between P_i and P_j if $i \neq j$. This would be a typical situation in infrastructure mOLAP, after a given running time i.e., the server has broadcast several sub-cubes. In infrastructure scenarios, there is no direct communication between clients. Enabling ad hoc con-

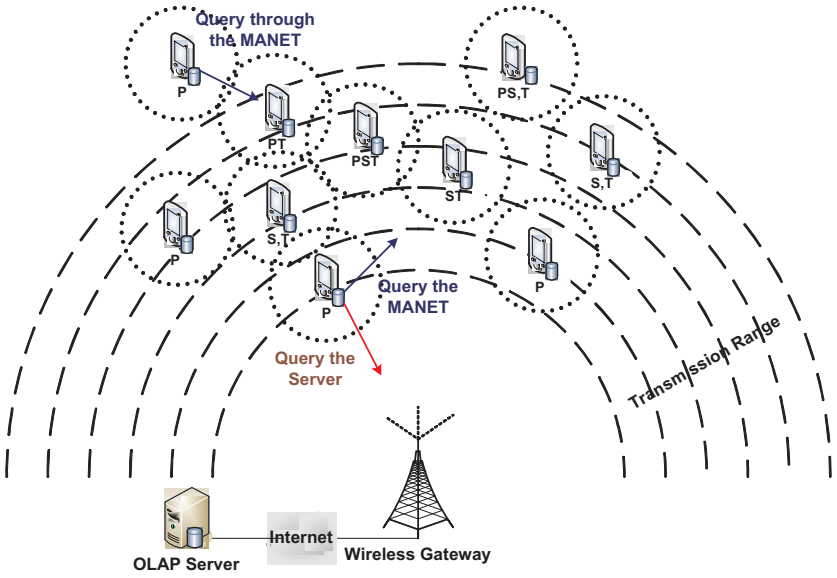

Figure 9.1: Extended mOLAP architecture

nectivity between clients leads to an architecture like the one depicted in Fig. 9.1. We argue that such an architecture, where mobile clients can additionally query one another and not necessarily the server, should be considered for the following reasons:

1. When clients exclusively rely on a server, they may have to remain idle for fractions of time, waiting for an answer. Meanwhile, the requested data could also exist in the local storage of a neighboring client. Apparently, a direct, ad hoc communication between these two clients could accelerate the execution of the query. Indeed, similar observations can be found in [167], where the authors deal with cooperative caching in mobile environments.

2. Clients located (either permanently or transitionally) outside of the transmission range of the wireless gateway, have to anyway route their queries through their neighbors. Since requested data may exist in nodes that propagate the query to the server, the request can be served faster.

3. Utilization of the MANET leads to an indirect load balancing. The more ad hoc connections are established, the more alleviated the server and the wireless gateway become.

In order to isolate the problem, throughout the rest of the chapter we will no longer assume the existence of any infrastructure. Query propagation and data dissemination is exclusively based on direct communication between mobile nodes.

9.2.2 Ad hoc vs. Infrastructure based

The fundamental ideas behind not only *FCLOS*, but infrastructure mOLAP in general as well, are not applicable for ad-hoc mOLAP, primarily due to the following reasons:

- *Shared wireless medium*: Unlike infrastructure scenarios, in which transmissions take place in separately allocated wireless channels, the participating nodes must use a common resource and additionally ensure that bandwidth is optimally exploited.
- *Query propagation*: Efficiently locating a node that is able to answer a query, while minimizing the incurred overhead, is a major concern.
- *Node roles*: There is no role distinction between participating entities. Apart from the fact that nodes will act both as a server and as a client, they may also act as intermediate hops, plainly relaying data.
- *Scheduling*: It is unrealistic to expect accumulated requests in one node, namely that one single node will have to handle so many concurrent requests, that a scheduler is necessary. In other words, intelligent scheduling is not the solution. Maintaining long queues must be avoided, since having many requests waiting in a single node's queue, when at the same time some other nodes are sitting idle, deteriorates the performance.

9.2.3 Requirements

The requirements for an ad hoc mOLAP protocol are directly dictated by the assumed application scenario. Since the ultimate objective is to assist infrastructure mOLAP, we restrict the requirements to the following:

1. **Reactiveness**: Since the protocol is supposed to extend existing infrastructure, it should be employed only when necessary (in reactive mode). Therefore, it should avoid unnecessary utilization of bandwidth through control messages.
2. **Generality**: The protocol should be independent of lower layers (MAC and routing). In this sense, no cross-layer techniques should be assumed.

9.3 Related Work

9.3.1 Distributed Data in MANETs

A huge amount of research papers in the area of MANETs has appeared in the last decade. Although to the very best of our knowledge, there is no related work explicitly dealing with ad hoc mOLAP, considering that ad hoc mOLAP is just an application running in MANETs, it should be first examined if more general

approaches can be utilized. It is clearly beyond the scope of this chapter and this document to delve into specific approaches. Instead, we outline the main research problems, which are relevant to ad hoc mOLAP.

Service Discovery

What we define as query propagation in Section 9.2.2, is known as *service discovery* in the MANET domain. In protocols based on *broadcast*, such as the ones proposed in [50, 117, 161], the main idea is either to always broadcast requests and use the underlying routing protocol to propagate replies or to integrate the routing with the service discovery protocol. *Multicast* service discovery protocols, such as the ones in [61, 87], essentially try to limit the overhead incurred by broadcast protocols. They pursue this by establishing multicast trees, along which services are advertised and requested. *Cluster* service discovery protocols, such as the ones in [132, 170], divide the network into logical clusters. Typically, there is one node (directory node) in the cluster responsible for the service directory. Clusters are selected so that each node has at least one such directory node within a radius of a limited number of hops. Services from normal nodes are registered in the directory node. Directory nodes periodically advertise their content. Finally, *geographical* service discovery protocols are discussed in [40]. Such protocols use the geographical position of the nodes to assist service discovery and ad hoc routing.

Peer-to-Peer Systems

Peer-to-Peer systems were designed for the internet. Therefore, their implementation in ad hoc networks is not straightforward, since the inherent shortcomings of MANETs must be considered and reflected on the protocol design. In the area of unstructured P2P systems, *ORION* [83] is a file sharing protocol, combining broadcast based request propagation and the *AODV* [121] routing protocol for propagation of replies. The Mobile Peer-to-Peer (*MPP*) protocol [136] is an approach similar to *ORION*, which uses the *DSR* routing protocol [78] instead of *AODV*.

The area of structured P2P systems has been dominated by approaches based on Distributed Hash Tables (DHTs). DHT functionality is particularly useful in MANETs because it can be used to implement scalable network services in the absence of servers [42, 171, 129]. Arguably the most promising approach of this area is the Virtual Ring Routing (*VRR*) [30]. *VRR* is inspired by overlay routing algorithms in DHTs, but unlike DHTs, *VRR* is implemented directly on top of the link layer and does not rely on an underlying network routing protocol. *VRR* provides both traditional point-to-point network routing and DHT routing to the node closest to a key.

9.3.2 Ad hoc mOLAP Suitability

Obviously, the systems described in the previous paragraphs could be used to build ad hoc mOLAP applications. However, service discovery protocols are rather ill-suited for the purpose of ad hoc mOLAP. Their main objective is to serve as building blocks for large-scale distributed network applications in MANETs. To achieve that, they employ complex mechanisms that typically found on cross-layering and proactiveness.

P2P systems in MANETs primarily cope with the inefficient usage of link-layer connections (differences between overlay and physical route), the high routing overhead and the frequent loss of application-layer connections. While these problems do exist in MANETs, their effect is particularly problematic in large-scale and highly dynamic topologies. Apart from the fact that ad hoc mOLAP does not take place under such conditions, these systems resort to cross-layering techniques to address these issues.

In addition to that, we argue that ad hoc mOLAP requires a unique approach for the following reasons:

1. *Semantics*: Service discovery in all aforementioned efforts, targets unstructured data objects with no direct semantic connection. Therefore, a major concern is to match requests with available services, which is fundamentally different than assuming data objects belonging to a database model.

2. *Subsumption based advertisement*: Cube querying is more coarse-grained than traditional service lookup, as a result of the query mapping to the corresponding aggregation lattice. As a consequence, employing complex service discovery mechanisms would add a truly unnecessary overhead. Query mapping enables a significant reduction of the objects to be advertised. Moreover, according to subsumptions an advertisement for the sub-cube n_a is automatically an advertisement for every other sub-cube n_b for which it is $n_a \succeq n_b$.

3. *Cross-layering*: mOLAP is just an application, not a network service. In this sense, it would be rational to utilize existing protocols and not resort to cross-layer approaches. While it is acceptable to consider cross-layering for lower layers (e.g., MAC and routing), the application layer should be excluded, considering that more applications should be able to run in parallel. In this sense, the ad hoc mOLAP design should be restricted to application layer packets, especially given the fact that its eventual goal is to be integrated in existing infrastructures, without any major engineering required.

4. *Aggregation*: In service discovery protocols, service is an abstract notion, i.e., the protocol is responsible for informing the initiator about the location of the service. In P2P networks, service translates into file transfer. In mOLAP, the service is not necessarily a unicast file transfer, but dissemination of sub-cubes, which might have to be aggregated and then multicast or broadcast.

5. *OLAP operations*: Typical end user behavior includes navigating through the requested data set. Assume a very simple example using the *DCL* of Fig. 2.4. If a client requests the sub-cube P, but then decides to add dimension S, which means that sub-cube PS should also be fetched, it would obviously have been better off fetching the sub-cube PS from the beginning in order to avoid an additional connection.

9.4 Problem Formalization

The optimization problem for a given ad hoc mOLAP query consists of two sub-problems: query propagation and data dissemination. Optimization of ad hoc scenarios is much more complex than the one of infrastructure scenarios, for the reasons explained in the previous section.

A mobile node M_c issues a multidimensional query q, which is eventually answered by a mobile node M_s. This procedure involves two phases: the query propagation phase, which refers to the process that M_c locates M_s, and the data dissemination phase, which is the actual data transfer. This means:

$$Cost_{Comm} = Cost_{QProp} + Cost_{DataDissem} \tag{9.1}$$

The goal is the reduction of total generated traffic, average access time and energy consumption overhead. Evidently, these metrics are not uncorrelated, since generated traffic directly influences access time and energy consumption.

9.4.1 Generated Traffic

In MANETs, the wireless medium is shared, therefore the amount of generated traffic is a crucial metric, not only for the application itself, but also for any other application utilizing the network. In our scenario, we distinguish between traffic generated for locating a node able to answer the query (Tr_{QProp}), and traffic for the actual data dissemination ($Tr_{DataDissem}$). $Tr_{DataDissem}$ is not stable and can be influenced by the degree of the exploitation of the subsumption property among sub-cubes. Obviously, the number of generated messages is directly connected to the access time and the energy consumption. For reasons of clarity let it be denoted that:

$$Tr_{all} = Tr_{QProp} + Tr_{DataDissem} \tag{9.2}$$

9.4.2 Query Access Time

The experienced query access time (T_{all}) is the total period of time M_c spends since issuing a query until the requested subset is actually fetched in its local storage:

$$T_{all} = T_{QProp} + T_Q + T_{DataDissem} + T_L \tag{9.3}$$

where T_{QProp} represents the total period of time until the request reaches M_s, T_Q the time the request might have to wait queued, $T_{DataDissem}$ the time the actual multidimensional data is transferred and T_L the time M_c might spend to process them (local computations).

9.4.3 Energy Consumption

The network interface represents a significant fraction of the energy consumed by the device, especially in PDAs. Let it here be underlined that our cost model computes the energy consumption *overhead* imposed by the application. In MANETs, the idle energy consumption is quite high, comparable to that of receiving, and an order of magnitude higher than that of sleeping. Conserving energy in MANETs is usually a task of lower stack layers. Since our work is on top of such protocols, we will isolate the energy problem at an application layer perspective. Therefore, we measure the energy overhead caused by queries and not the total energy consumed by the nodes. The goal is to optimize the total energy overhead E_{all}:

$$E_{all} = E_{QTransmit} + E_{QFwd} + E_{QRcv} + E_{DataSend} + E_{DataFwd} + E_{DataRcv} + E_L \quad (9.4)$$

where $E_{QTransmit}$ is the energy spent by M_c transmitting the query request, E_{QFwd} the energy consumed by intermediate nodes forwarding the request, E_{QRcv} and $E_{DataSend}$ the energy spent by M_s receiving the request and actually sending the data, $E_{DataFwd}$ the energy consumed by intermediate nodes forwarding this data, $E_{DataRcv}$ the energy consumed by M_c receiving the data and finally E_L the energy M_c might spend to process them.

9.5 Query and Disseminate under Global View

The optimization problem involves several tradeoffs. In order to exploit properties of multidimensionality and simultaneously apply a simple and collaborative scheme, we propose a query propagation and data dissemination protocol called *QDGV*. *QDGV's* fundamental design concepts are simplicity and independence of the underlying ad hoc routing and MAC protocol. The fundamental elements comprising *QDGV* are presented in the following paragraphs.

9.5.1 Data Model and Query Mapping

As far as the data model is concerned, we continue assuming the same as throughout this thesis, and which is thoroughly described in Chapter 2. However, there is a fundamental difference in the query mapping mode.

Query mapping is a critical aspect. mOLAP in infrastructure networks assumes a coarse-grained query mapping. Every user query is mapped to the corresponding *DCL* node. This design principle is analytically justified in Chapter 7. Nevertheless, the analytical model explicitly refers to infrastructure mOLAP, and

thus cannot be exploited in ad hoc scenarios. In ad hoc scenarios, there is no server facility to make use of the analytical information.

In MANETs, resources are particularly scarce and bandwidth needs to be consumed carefully. Transferring fact table data, despite achieving increased offline capabilities, might prove disastrous for the system performance. We therefore use the respective *hDCL* instead. This enables data exchange in several granularities. It should be noted though, that our proposed architecture can handle both query mappings without any modification.

Finally, it is assumed that clients are aware of the schema metadata and have dimension table values stored.

9.5.2 View Exchange

The query mechanism uses four types of messages: MSG_{QUERY}, MSG_{REPLY}, MSG_{FETCH} and MSG_{OLAP}, which are explained in the following paragraphs.

Each node M_i possesses a subset of the cube P_i. A service advertisement, or in our case a sub-cube advertisement, mechanism is necessary to avoid costly query propagations. We propose a *piggybacking* approach, where advertisements and queries take place simultaneously. By enforcing nodes to piggyback information about the data (the portions of the sub-cube) they possess, the query propagation process is substantially accelerated. In *QDGV*, the advertisement does not rely on periodical broadcasts or on nodes acting as service brokers. Instead, nodes are forced to give information about the data they posses, every time they issue a query.

The data advertisement content is called *view*, *V*. The view contains information about not only the status of its owner, but also the state of other nodes, thus constituting a global sub-view.

The *hDCL* depicts the relationships between all $\prod_{n=1}^{D}(gr_i+1)$ sub-cubes, given a D-dimensional cube and the number of grouping attributes gr_i of each dimension. A *view* is a structure containing i indexes so that $0 \leq i \leq \prod_{n=1}^{D}(gr_i+1)$ and each index V_i points to maximum UB entries. An elementary example of a view is depicted in Fig. 9.2, corresponding to the *hDCL* of Fig. 2.5, with UB=3.

A *view entry* contains information about a node possessing a sub-cube. For example, the node with id 15 possesses the sub-cube with the identifier 111. A very important element is the time span, since the actuality of the entry is a critical factor. The mechanism is naturally extendable, open to include any additional information, which could assist the systems performance. Namely, the actual content of the view can be defined according to a specific environment.

Initially, a node's view contains exclusively personal information. Each view reception from any other node causes a *view merge*, to produce a more complete view. Gradually, nodes gain a better and more complete view of the network. The formalism of the view exchange mechanism is depicted through its two protocols *Query/View Send* and *Query/View Receive*.

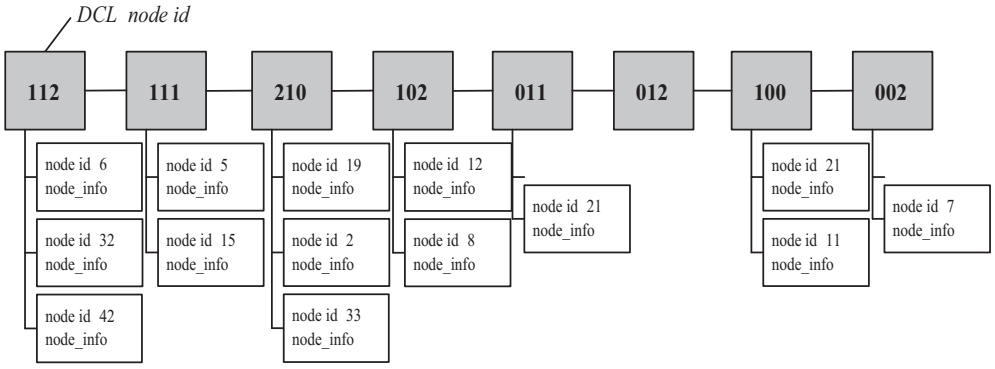

Figure 9.2: Elementary view for a 3-dimensional schema

Protocol Query/View Send
1: Create Query/View packet (MSG_{QUERY})
2: Fetch local view V
3: If $size(V) > UB$ discard obsolete entries so that $size(V) \leq UB$
4: *store(V)*
5: *Piggyback* V in MSG_{QUERY}
6: *propagate*(MSG_{QUERY})

Protocol Query/View Receive
1: For each incoming MSG_{QUERY}:
2: Extract sender's $V_{incoming}$
3: *merge*(V_{local},$V_{incoming}$)
4: *store(V)*
5: *process(Query)*

Letting this procedure to continue without control, would cause scalability problems, since inevitably the views would inflate in terms of size. Moreover, due to node mobility and churn, information can become obsolete pretty fast. There is absolutely no reason to maintain big views. This is how the upper bound UB in the view definition is justified, so as to control the view's size. When this bound is reached, the node discards view information according to their time stamp, on a FIFO basis. Naturally, the actuality of the information is highly dependent on the node mobility and the frequency of the view exchange.

Moreover, *QDGV* exploits the relationships between sub-cubes in order to keep the view's size small. Each client does not insert view entries for all of its available sub-cubes, but only for a *minimal set*. Consider the *hDCL* of Fig. 9.3 depicting a client's data. All locally available sub-cubes are marked with gray. The nodes comprising the minimal set are marked with darker gray. The client does not have

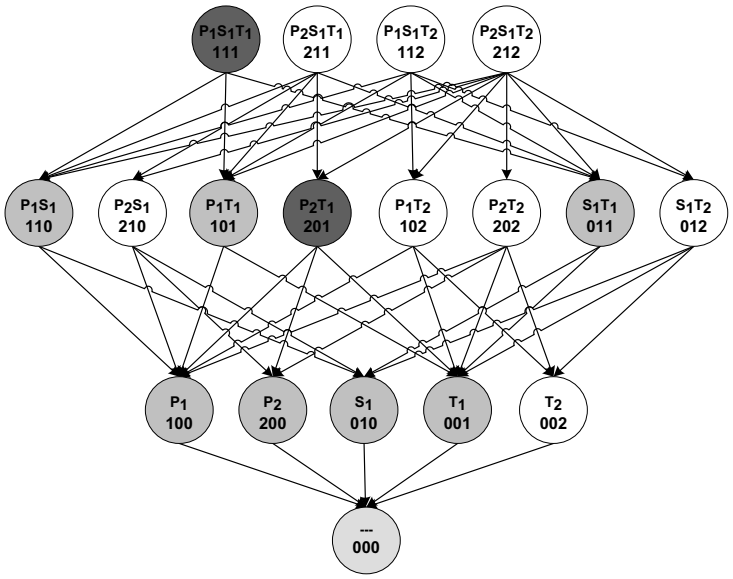

Figure 9.3: An example of a QDGV client's local data. All locally available sub-cubes are marked with gray. The nodes comprising the minimal set are marked with darker gray

to insert view entries for every gray node. It suffices to insert only two entries; in this example for the sub-cubes with id 111 and 201. In this way, view size can be kept relatively small. Otherwise, for the real dataset used in this thesis, which consists of 1200 *hDCL* sub-cubes, the size of the view would be prohibitive.

The view exchange mechanism introduces a communication and energy overhead. Since this overhead is proportionate to the request rate, a fair amount of adaptiveness is achieved. However, in MANETs each application packet transmission imposes a significantly bigger control overhead upon the underlying network layers, than in the case of wired networks. Therefore, piggybacking the views in querying packets is preferable than sending exclusive view packets, in spite of the fact that the application packets will grow bigger in size. As the experimental results show, in most cases the view exchange mechanism will enhance the system's performance.

9.5.3 Query Propagation

We now present the query mechanism of *QDGV*. The query mechanism is extremely simple. When a node M_c wants to pose a query targeting sub-cube *sc*, then it consults its current view. Then there are the following options according to the view information:

- If M_c's view V^C contains no information about *sc*, then M_c has no other option but to broadcast a MSG_{QUERY} message.

- If according to V^C, there is only one node able to serve the query (node M_s), an explicit MSG_{QUERY} is unicast to M_s.
- If according to V^C, there is more than one node able to serve the query, the node chooses the node whose view entry is more recent. Keeping older view entries serves as a backup mechanism, in case that the initially selected node fails to answer the request.

When M_s receives a MSG_{QUERY} message then:

- If M_s's view V^S contains no information about *sc*, then M_s has no other option but to (re-)broadcast a MSG_{QUERY} message.
- If V^S contains one or more entries about *sc* (node $M_{s'}$), then M_s unicasts a MSG_{QUERY} message to $M_{s'}$ informing about M_c's query.
- If *sc* is locally available, M_s sends a unicast message MSG_{REPLY} to M_c.

When M_c receives a MSG_{REPLY} from M_s, it unicasts a message MSG_{FETCH} to establish the connection. M_s then responds with MSG_{OLAP} messages, containing the sub-cube. If M_c does not receive any MSG_{REPLY} within a time bound, it repeats the abovementioned procedure.

In parallel, nodes function as servers as well. Dealing with requests in the queue, the server exploits the subsumption property, every time that this is possible. If two or more requests can be served together, the server uses a *multicast*. The benefit of this multicast is twofold. On the one hand, for the server node as it obviously does not have to establish multiple connections. On the other hand, client nodes that receive a superset of their initial request, locally store the additionally received data, and consequently may not have to transmit new queries, alleviating in this way the network. Moreover, more data redundancy is achieved, which eventually leads to better system performance.

9.5.4 Runtime Connection Handling

In this paragraph, we propose a runtime connection handling method that is based on the idea of breaking a connection and retransmitting, should this be beneficial. The idea is straightforward and could be characterized as a special type of preemption. Let us assume that a server M_s transmits the sub-cube sc_a to client M_{c1}. t_{break} time units after the beginning of the transmission, a new request for a sub-cube sc_b comes from client M_{c2}. Assume that the transmission of a sub-cube sc_i requires t_i time units. If sc_a can be derived by sc_b or vice versa, then we have the situation depicted in Table 9.1 (rows labeled with *break* refer to our proposal), where P_{tr} is the energy power for the transmission and P_{rcv} for the reception:

According to Table 9.1, in order to reduce access time (aggregated access time for both queries), it must be:

$$t_{break} + t_b + t_b < t_a + t_a - t_{break} + t_b \Rightarrow t_{break} < t_a - \frac{t_b}{2} \tag{9.5}$$

Table 9.1: Breaking Consequences

Metric	**Value**
Access Time (no break)	t_a+t_a-t_{break}+t_b
Access Time (break)	t_{break}+ t_b + t_b
Client Energy (no break)	$t_a \times P_{rcv} + t_b \times P_{rcv}$
Client Energy (break)	$(t_{break}+t_b)\times P_{rcv}+t_b \times P_{rcv}$
Server Energy (no break)	$(t_a+t_b)\times P_{tr}$
Server Energy (break)	$(t_{break}+t_b)\times P_{tr}$

while in order to reduce energy consumption there must be:

$$\begin{aligned}(t_{break} + t_b) \times P_{rcv} + t_b \times P_{rcv} + (t_{break} + t_b) \times P_{tr} \\ < \quad t_a \times P_{rcv} + t_b \times P_{rcv} + (t_a + t_b) \times P_{tr}\end{aligned}$$

which eventually leads to (from [46] there is $\frac{P_{tr}}{P_{rcv}} \simeq 2$) :

$$t_{break} < t_a - \frac{t_b}{\frac{P_{tr}}{P_{rcv}} + 1} \Rightarrow t_{break} < t_a - \frac{t_b}{3} \tag{9.6}$$

From Eq. 9.5, 9.6 we can directly obtain the cases in which breaking and retransmitting makes sense. However, in an environment with exclusive channels computing the t_i time units would not be problematic, since this value is directly connected with the size of the respective sub-cube sc_i. Nevertheless, in MANETs where the medium is shared, the estimation of any t_i is subject to the network's current state. The available bandwidth is necessary in order to compute t_i. Available bandwidth estimation is beyond the scope of this work. The authors in [36, 133] have coped with this problem using intrusive or non-intrusive mechanisms. We chose to use the non-intrusive mechanism proposed in [133], where each node uses only its local perception to evaluate the proportion of time the medium is free, thus not introducing any communication overhead.

9.5.5 Data Dissemination

QDGV employs a simple transfer protocol. To manage the relatively large size of sub-cubes, *QDGV* divides them into fixed-size pages. The page is the basic unit of transfer and provides the advantage of limiting the amount of state a receiver must maintain while receiving data. Sub-cubes are divided into pages of fixed number of tuples. Receiver node M_c is informed through MSG_{REPLY} about the number of pages to expect. Naturally, one page corresponds to more than one MSG_{OLAP} packet. In this context, *QDGV* employs a NACK (negative acknowledgment) based transport mechanism for every page and not for every MSG_{OLAP}.

In case of permanent disconnection, M_c can try either to re-contact node B or simply discover a new source. In the latter case, the MSG_{REPLY} message indicates already received pages in order to avoid redundant transfers.

9.6 Experimental Evaluation

9.6.1 Simulation Environment

In order to analyze the performance of *QDGV*, we performed extensive performance evaluation in the ns-2 network simulator [3]. As performance measures we used the average access time, the energy consumption overhead, the generated traffic and the discovery success rate for a multidimensional query. We run simulations with various network sizes, network densities, node velocities as well as with different cube dimensionalities and query skewness. The most important simulation parameters are shown in Table 9.2.

Nodes are placed randomly in space, in a square topology. We use the freeway mobility model [20]. Mobility and mobility models are critical parameters in MANETs. Since it is clearly beyond the scope of this thesis to delve into mobility patterns, a detailed analysis can be found in [164]. Nodes are using the 802.11 communication standard with a transmission range of 250m. Each node is randomly allocated with a subset of the cube. In order to set up a rather adversary scenario, nodes are initially allocated with only a very small subset of the cube, namely only 10% of the *hDCL* nodes. Once a query answer is successfully received, a new query is submitted according to the query interval of the scenario.

In this set of experiments, we used a synthetic dataset consisting of 300K tuples. As seen in Table 8.2, an average sub-cube occupies 1239 KB if stored as *m-Dwarf*.

The default values were chosen in order to represent a realistic scenario. Therefore, the velocity of the mobile clients is deliberately not too high because we believe that such scenarios will typically take place inside of buildings, with maximum moderate velocity. In addition to that, due to the limited capabilities of mobile devices, we assume data marts rather than big DWs.

In all set of experiments, requests reach nodes that are maximum 2 hops away. In *QDGV*, the size of the views is controlled with a value UB=2.

9.6.2 Reference Application

As a reference application we implemented a simple protocol based on broadcasting query requests, and which does not consider at all the multidimensionality of data. Locating and exchanging data take place without considering the type of the handled data. Let us name this approach Broadcast Query and Direct Answer (BQDA). *BQDA* uses *AODV* for routing.

BQDA issues queries by broadcasting MSG_{QUERY} messages, since there is no information about which node could serve its request. These messages reach

Table 9.2: Simulation parameters

Simulation Parameter	**Range**	**Default**
Wireless Network	802.11	–
Bandwidth	1-54 Mbit/s	11 Mbit/s
Routing Algorithm	AODV/MAODV	–
Client Population	20-100	50
Density	50-150 nodes/km^2	100
Velocity	0-5 m/s	1.4
Query Interval	1-10s	5

only its neighbors, namely the nodes within its transmission range. If one or more of them are able to answer this query, an appropriate MSG_{REPLY} message is sent back to the query initiator to inform it that its request can be served. Then the query initiator establishes a connection with the first node that has sent such a message. If a neighbor cannot answer the request, the query is forwarded, according to a probabilistic scheme [116] in order to avoid the broadcast storm problem. *BQDA*, exactly as *QDGV*, transmits sub-cubes and not exact answers. In *BQDA*, requests reach nodes that are 2 hops away, and are forwarded with a probability of 0.4. The same NACK based transfer protocol of *QDGV* is also used for *BQDA*. *BQDA*, exactly as our proposal, is a reactive protocol that does not employ any cross-layering.

Naturally, we do not claim that this is the only reference application that can be used. However, generally in MANETs, there is no standard reference application.

9.6.3 Basic Evaluation

Query Access Time

In the first set of experiments, we measure the mean query access time, as defined in Section 9.4.2. The results are shown in Fig. 9.4. Evidently, both protocols exhibit a linear behavior. Growing client population results in access time increase. This is quite rational, since more nodes have to share the wireless medium. Moreover, the number of requests propagated in the network is higher as well. Nevertheless, *QDGV* evidently outperforms *BQDA*, whose lack of any information about the state of other nodes prolongs T_{QProp}. On the contrary, *QDGV* manages to locate sub-cubes substantially faster, and by additionally exploiting the subsumption property among them reduces the experienced access time by almost 50%.

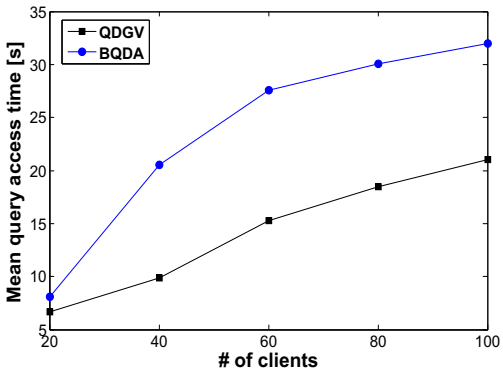

Figure 9.4: Mean query access time (T_{all})

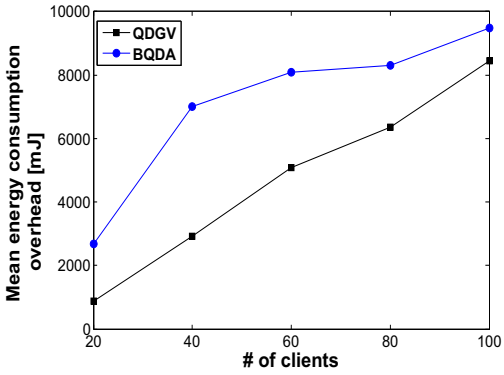

Figure 9.5: Mean energy consumption overhead (E_{all})

Energy Consumption Overhead

Energy consumption overhead behavior is quite similar. Growing client population increases the energy consumption overhead. Both schemes experience a linear growth. However again, *QDGV* exhibits a better performance. *BQDA*'s performance deteriorates because the broadcast based query propagation, despite the probabilistic scheme, still incurs transmissions and receptions of many unnecessary MSG_{QUERY} messages. Let it here be denoted that the overhead produced by the aggregations, necessary for a client receiving a superset of its initial request, is essentially negligible, since an aggregation eventually corresponds to a simple scan of every tuple. It is here too confirmed, exactly like in infrastructure mOLAP, how closely related access time and energy consumption overhead are.

Query Propagation traffic

We now evaluate the total amount of traffic generated by the query propagation. Remember that *BQDA* uses a probabilistic broadcast to reach the rest of the nodes, whereas *QDGV* depends on the information provided by the views. We capture this by measuring the generated traffic and not the number of generated messages, since *QDGV* query packets additionally carry the view structure, hence being bigger. Therefore, Fig. 9.6 depicts the number of transmitted bytes. The behavior of the two protocols is different. As expected, *BQDA*'s broadcast based scheme incurs increased traffic as the network grows. Although *QDGV* messages are bigger in size, the minimization of the total number generated is so big, that eventually the generated traffic is reduced by 90%. Relying on the view information, *QDGV* exhibits only a slight increase as the network grows.

Query interval is a very important system parameter. Since the view exchange mechanism imposes an indirect overhead to the network, it should be examined if this is equally beneficial with lower request rates, which corresponds to higher query interval. In Fig. 9.7, we examine how the two protocols react. Let it

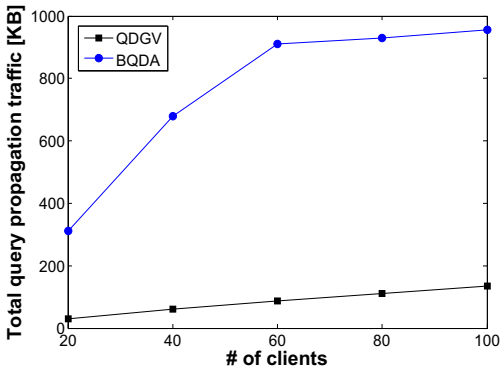

Figure 9.6: Query propagation traffic (Tr_{QProp})

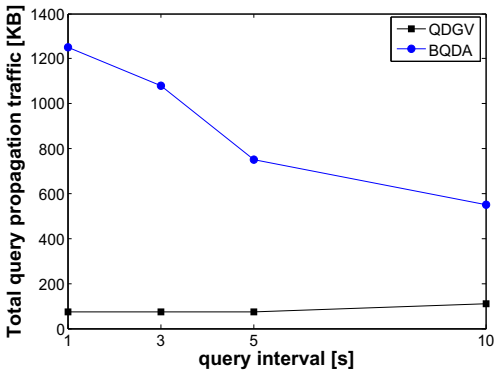

Figure 9.7: Query propagation traffic (Tr_{QProp}) with query interval

be reminded that upon reception of an answer, clients issue a new query after a uniformly distributed time interval of $[\frac{t}{2}, t]$s. *BQDA* performs better in less congested networks. When the query interval decreases, nodes are alleviated and the generated, broadcast based, traffic becomes less. However, it fails to scale as the query rate increases. *QDGV* on the contrary, is resilient. The reason for this is that the overhead posed by the view mechanism is piggybacked in the queries. Therefore, when the query rate is low, the overhead is low as well. There is no concern about unnecessary network overhead when the request rate is low. Indeed, a fair amount of adaptiveness is achieved, due to the protocol's reactiveness.

Discovery success

In the next set of experiments, we evaluate the efficiency of the query propagation mechanisms. In order to capture this, each request is deliberately transmitted only once. Figure 9.8 shows the percentage of the sub-cubes successfully located by the discovery mechanism. Naturally, in these numbers we do not include requests that were not served due to the fact that no participating node had the requested sub-cube. *QDGV*, despite generating much less traffic than *BQDA*, manages to be more efficient in discovering sub-cubes. This is quite impressive, since it becomes evident that in *QDGV* more data connections can in average be established. As far as *BQDA* is concerned, it should be clear that its poor performance is the consequence of a given compromise. A flooding broadcast would produce much better discovery success, but at the cost of incurring a huge overhead at the same time. *BQDA*'s probabilistic broadcast, reaching nodes maximum two hops away, restricts flooding on the one hand, but has to live with inferior discovery success on the other.

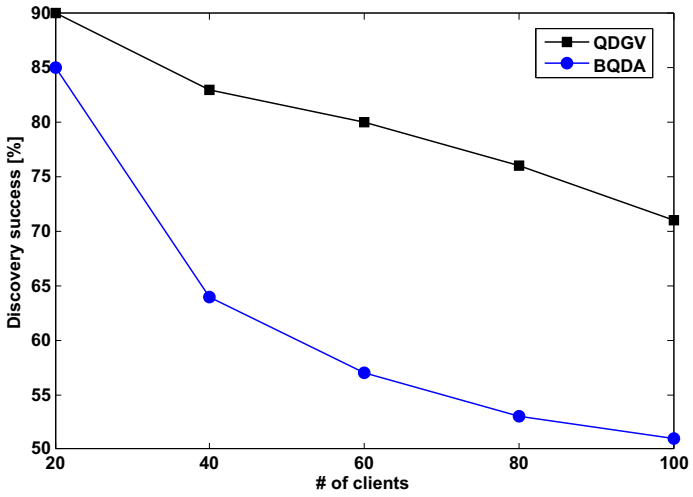

Figure 9.8: Discovery success

9.6.4 Further Evaluation

Network Density

MANETs are highly dynamic networks and a complete evaluation requires experiments in several network topologies. In the next set of experiments, we test the two protocols against different network densities, which is also an important parameter in MANETs indicating the number of end users in a given square plane. Network density is a particularly important factor when many broadcast operations are involved. Fig. 9.9 reveals two completely different behaviors. On the one side, in networks with higher density *BQDA*'s performance substantially deteriorates. This is caused by the huge number of messages transmitted for sub-cube discovery, which are now received and forwarded by more nodes. Essentially, query messages overwhelm the network, harassing the data exchange. On the other side, *QDGV* appears unaffected by the node density due to the limited number of MSG_{QUERY} messages. As a matter of fact, *QDGV* rarely needs to resort to a broadcast for a request, since the view typically provides the necessary information to directly reach an appropriate node.

Node Velocity

Finally, we examine networks with various node velocities: 0.1 m/s, 0.6 m/s (slow walking speed), 1.4 m/s (fast walking speed), 2.5 m/s and 5 m/s. Beyond the evident superiority of *QDGV*, Fig. 9.10 reveals that both protocols are practically unable to handle high mobility. As explained in Section 9.3, in order to cope with highly dynamic networks, ad hoc protocols have to resort to cross-layer techniques that actively detect link failures and accordingly cooperate with higher network

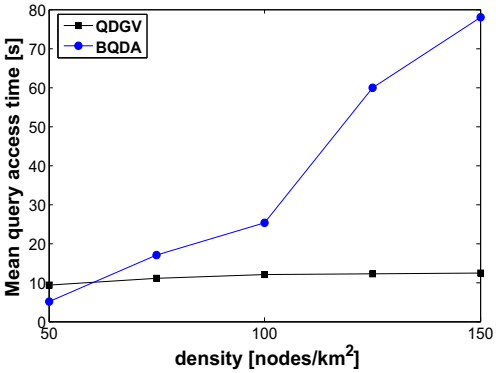

Figure 9.9: Mean query access time with density (T_{all})

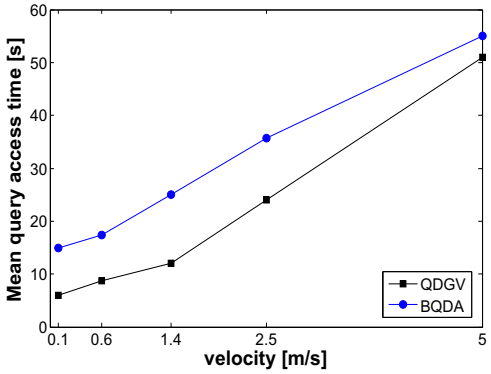

Figure 9.10: Mean query access time with velocity (T_{all})

stack layers. Even with cross-layering, it is still difficult to handle such networks and broadcast is the only option. Neither of the examined protocols employs such techniques, and thus the results come unsurprisingly.

9.7 Open Issues

Obviously, this chapter leaves several issues unaddressed, and which we consider future work. *QDGV* was tested using the reactive routing protocol *AODV*. It would be very interesting to investigate its performance with proactive routing protocols, such as *OLSR* [38], as well, especially considering that we expect ad hoc mOLAP to take place in not too dynamic environments.

Moreover, although *QDGV* does not rely on a specific transport protocol, it was evaluated using a simple NACK based protocol. Considering the research work done in transport protocols for MANETs (e.g., [31]), alternatives can be considered.

P2P systems for MANETs such as *ORION* [83] or *MPP* [136] do not match the requirements defined in Section 9.2.3. In spite of that, it is worthwhile comparing them against *QDGV*. While they should be expected to perform better in highly dynamic environments, it is unclear what to expect in less dynamic environments.

Finally and most importantly, while *QDGV* can assist infrastructure mOLAP architectures such as *FCLOS*, the degree of this assistance remains to be quantified.

9.8 Summary

This chapter presents the case of mOLAP in ad hoc networks. We envision ad hoc mOLAP scenarios to take place in conjunction with infrastructure mOLAP, assisting rather than replacing infrastructure applications. In this context, we define the requirements for a lightweight ad hoc mOLAP protocol. Unfortunately, key

ideas from conventional infrastructure mOLAP architectures are not applicable in ad hoc mode. After formalizing the involved optimization problem, we introduced *QDGV*, a simple query propagation and data dissemination protocol for ad hoc mOLAP.

QDGV is by no means an all purpose data dissemination protocol, neither in regard to the running application nor in regard to the network topology. It is explicitly designed for ad hoc mOLAP querying, under realistic application domain assumptions. *QDGV* does not employ any cross-layer technique and is completely independent of the underlying MAC and ad hoc routing protocol. *QDGV's* main component is a reactive view exchange mechanism. Views, which contain information about the locations of sub-cubes in the network, are piggybacked in query messages. Experimental results show that *QDGV* outperforms a probabilistic broadcast based scheme on all relevant criteria. For our default scenario mean query access time is reduced by 50%.