A Density-Driven Publish Subscribe Service for Mobile Ad-Hoc Networks

Anna Kaplun
A Density-Driven Publish Subscribe Service for Mobile Ad-Hoc Networks

Research Thesis

In Partial Fulfillment of The Requirements
for the Degree of Master of Science in Computer Science

Anna Kaplun

Submitted to the Senate of
the Technion — Israel Institute of Technology
Nisan 5771 Haifa April 2011
The research thesis was done under the supervision of Assoc. Prof. Roy Friedman in the Computer Science Department.

I would like to thank Roy for his continuous support throughout all stages of this work, for his constructive ideas and excellent guidance.

Special thanks to my family for the great support during this work.

The generous financial support of the Technion is gratefully acknowledged.
# Contents

Abstract 1

1 Introduction 2

2 Preliminaries 5

3 Related Work 7
   3.1 Network Wide Broadcasting Based Pub/Sub 7
   3.2 Tree Based Pub/Sub 8
   3.3 Structureless Pub/Sub that Utilizes Network Mobility 9
   3.4 Utilizing an Underlying Routing Protocol 10
   3.5 Other 10

4 Density Driven Virtual Topography 11

5 The Density-Driven Publish/Subscribe Protocols 13
   5.1 Density Biased Walk 14
      5.1.1 Stopping a Message 14
      5.1.2 Calculation of the Next Hop 14
   5.2 Route Walk 16
   5.3 Data Handoff 16
   5.4 Network Estimation Mechanism 17
      5.4.1 Scheduling an Estimation Round 17
   5.5 3DLS-Pub/Sub 18
      5.5.1 Publishing 19
      5.5.2 Subscribing 19
      5.5.3 Handoff 20
   5.6 LCDD-Pub/Sub 20
      5.6.1 Publishing 21
      5.6.2 Subscribing 21
   5.7 LDDD-Pub/Sub 21
      5.7.1 Publishing 22
      5.7.2 Subscribing 23
List of Figures

4.1 The structure of node p’s hello message ........................................ 12
5.1 Density biased walk forwarding algorithm (as preformed by node p) ......... 15
5.2 Merging estimated hilltops number when receiving a regular message (as preformed by a hilltop) .................................................. 17
5.3 Processing of handoff messages with estimation, as preformed by a hilltop (estSchedTime: the time when an Estimation Round will be started, estSchedAt: the time when an Estimation Round was scheduled) ........................................... 18
5.4 3DLS-Pub/Sub: stopping condition of publicationMsg ............................. 19
5.5 LCDD-Pub/Sub: stopping condition of publicationMsg (avg: estimated number of hilltops) .......................................................... 21
5.6 LCDD-Pub/Sub: stopping condition of lookup message (avg: estimated number of hilltops) .......................................................... 22
5.7 LDDD-Pub/Sub: stopping condition of publicationMsg (avg: estimated number of hilltops) .......................................................... 23
6.1 LDDD-Pub/Sub, varying mulFactor ...................................................... 27
6.2 LDDD-Pub/Sub, varying mobility for mulFactor 1.5 and 2 values .................. 28
6.3 Varying subscriptions matching single publication ................................... 29
6.4 Varying subscribers number ............................................................. 30
6.5 Varying field size ........................................................................ 31
6.6 Varying mobility ........................................................................... 33
6.7 Varying density ............................................................................. 34
6.8 Varying lookup interval ................................................................... 35
6.9 Counter based flood vs. our proposed algorithms for varying field size ....... 37
6.10 Counter based flood vs. our proposed algorithms for varying mobility ....... 38
6.11 Counter based flood vs. our proposed algorithms for varying density ........ 40
6.12 Tree vs. our proposed algorithms for varying field size ......................... 42
6.13 Tree vs. our proposed algorithms for varying mobility .......................... 43
List of Tables

5.1 3DLS-Pub/Sub: lookup multiply factor ................................................. 20
6.1 Network parameters ................................................................. 26
6.2 Default publish/subscribe service parameters ................................. 26
6.3 Publish/subscribe algorithms default parameters ............................... 26
6.4 Counter based flood default parameters ......................................... 36
6.5 Tree default parameters .............................................................. 41
Abstract

We study a density driven virtual topography based publish/subscribe service for mobile ad-hoc networks.

Mobile Ad-Hoc Networks (MANETs) are formed by a collection of mobile nodes, each equipped with wireless communication capabilities, without relying on any fixed infrastructure or centralized administration. In order to maintain network connectivity, each node may act as an ad-hoc router, forwarding data packets for other mobile nodes that may not be within direct transmission range of each other.

We explore three variants of a publish/subscribe service that is based on the virtual topography. In the first, nicknamed 3DLS-Pub/Sub, nodes send their publications to the several hilltops using a local greedy walk. Based on the properties of the virtual topography, this walk usually follows a very short path. Then, periodically, each node sends a lookup message, which carries its subscriptions and traverses several hilltops. In each encountered hilltop, the lookup message checks whether the visited hilltop stores matching events. This way, it collects all matching events, and returns them to the initiator. The second approach, nicknamed LCDD-Pub/Sub, improves on 3DLS-Pub/Sub message count and message sizes by employing a network estimation mechanism.

The last scheme, called LDDD-Pub/Sub, utilizes the opposite approach. That is, subscriptions are forwarded to the nearest hilltop. Events are published to multiple hilltops. Each hilltop through which the publication passes, checks which nodes have matching subscriptions, and forwards the event to them using optimized routing. This results in much faster event notification, at the expense of a higher message count.

We perform an extensive simulation based study, in order to explore the operational environ and performance of our proposed schemes. We also compare them with the tree-based publish/subscribe mechanism and an optimized flooding based scheme. Our results show that the density driven approach is superior to the others in most scenarios for most metrics. The only exception is for applications that require extremely fast notifications (sub-second). For the latter, flooding based is the most adequate, yet this comes with a price of transmitting much more messages than the tree-based and density-based approaches.
Chapter 1

Introduction

Mobile Ad-Hoc Networks (MANETs) are formed by a collection of mobile nodes, each equipped with wireless communication capabilities, without relying on any fixed infrastructure or centralized administration. In order to maintain network connectivity, each node may act as an ad-hoc router, forwarding data packets for other mobile nodes that may not be within direct transmission range of each other\(^1\). This enables fast deployment of high-bandwidth infrastructure-independent networks that are free of airtime fees. Until recently, commercial usage of mobile ad-hoc networks has been mostly restricted to military applications. Yet, with the adoption of the Wi-Fi Direct specification by the Wi-Fi Alliance, wide adoption of ad-hoc networking looks promising. This will initially be only for single hop networks. Yet, we believe that it will gradually build demand also for multiple hop extensions, in order to extend the range and scale of such networks, once users get used to their benefits.

Publish/subscribe is a well known middleware service for large networks and large scale distributed applications [5]. In particular, it suits well with many of the target applications of ad-hoc networking. This includes search-and-rescue missions, military applications, file sharing, ad-hoc micro-blogging and twitting, chatting (instant messaging), collaborative caching, some forms of sensor networks, etc. Consequently, several schemes for implementing publish/subscribe in ad-hoc networks have been published, e.g., [2, 3, 8, 9, 12, 13, 14, 15, 17, 20].

Existing publish/subscribe techniques for ad-hoc networks can be classified into overlay based vs. structure-less approaches. In the former, a logical overlay, such as a tree of various sorts, is maintained in the network. Publications and subscriptions are disseminated along the tree edges and intermediate nodes may store subscriptions and filter or forward events based on subscriptions they are aware of. In the structure-less approach, events are usually flooded throughout the network, using some efficient flooding mechanism, and are filtered at the middleware level of each node in the network.

Overlay based approaches are often sensitive to mobility, due to the need to maintain the overlay structure in the face of frequent topology changes. On the other hand, flooding based approaches typically send much more messages, since little or no filtering is done at intermediate nodes.

In this work, we explore a novel density-driven mechanism for content based publish/subscribe.

\(^1\)In this work we target networks that are always connected w.h.p.
It is based on the dynamic virtual topography substrate developed in [6] (for location services). In particular, in the virtual topography of [6], some nodes are designated as hilltops and serve as rendez-vous points; a simple protocol is defined, which enables routing messages to hilltops in a local greedy manner. Moreover, hilltops tend to be well spread in the network, such that any node is at most a few hops away from the nearest hilltop. Further, when hilltops are changed due to topology changes (caused by mobility), a new hilltop is typically only one hop away from a retiring hilltop. This enables a retiring hilltop to convey the information it had, and which is required for state continuity, to the new hilltop with a single message transmission. These properties make the virtual topography of [6] a promising candidate for publish/subscribe.

Specifically, we explore three variants of a publish/subscribe service that is based on the virtual topography of [6]. In the first, nicknamed 3DLS-Pub/Sub, nodes send their publications to several hilltops using a local greedy walk. Based on the properties of the virtual topography, this walk usually follows a very short path. Then, periodically, each node sends a lookup message, which carries its subscriptions and traverses several hilltops. In each encountered hilltop, the lookup message checks whether the visited hilltop stores matching events. This way, it collects all matching events, and returns them to the initiator. The second approach, nicknamed LCDD-Pub/Sub improves on 3DLS-Pub/Sub message count and message sizes by employing a network estimation mechanism, also developed and described in this work.

The last scheme, called LDDD-Pub/Sub, utilizes the opposite approach. That is, subscriptions are forwarded to the nearest hilltop. Events are published to multiple hilltops. Each hilltop through which the publication passes, checks which nodes have matching subscriptions, and forward the event to them using optimized routing. This results in much faster event notification, at the expense of a higher message count.

We would like to emphasize that while the density driven topography was introduced in [6], it was used there to implement a location service. Utilizing it for pub/sub, and in particular, the two schemes - LCDD-Pub/Sub and LDDD-Pub/Sub - is completely new. 3DLS-Pub/Sub, which is a more straightforward attempt to adopt ideas from [6], resulted in poor performance. In particular, 3DLS-Pub/Sub suffers from very large messages, it has to be manually configured depending on the network size and from our experience its performance is very sensitive to the distribution of publications/subscriptions over time.

We perform an extensive simulation based study, in order to explore the operational envelop and performance of our proposed schemes. We also compare them with the tree-based publish/subscribe mechanism of [12] and an optimized flooding based scheme [20]. Our results show that the density driven approach is superior to the others in most scenarios for most metrics. The only exception is for applications that require extremely fast notifications (sub-second). For the latter, flooding based is the most adequate, yet this comes with a price of transmitting much more messages than the tree-based and density-based approaches.

The rest of this dissertation is organized as follows: The model and basic assumptions are described in Section 2. Section 3 surveys related work. A brief overview of the density based virtual topography and its maintenance is given in Section 4. The density driven publish/subscribe schemes are described in Section 5. The simulations are reported in Section 6, and we conclude.
with a discussion in Section 7.
Chapter 2

Preliminaries

In this work, we focus on mobile ad hoc networks. A mobile ad hoc network (MANET) consists of a collection of wireless mobile nodes dynamically forming a network without relying on any existing network infrastructure or centralized administration. Every node is equipped with an omni-directional antenna that enables wireless communication.

A node can receive data from another node if the destination node is located within the transmission range of the source node. We assume that all nodes have the same transmission range. Therefore, if a node $q$ is within the transmission range of node $p$, then $p$ is within the transmission range of node $q$. In this case, $p$ and $q$ are able to communicate directly and we call them direct neighbors. In the following, $N(1, p)$ refers to the set of direct neighbors of node $p$ (including itself).

When one node transmits, its direct neighbor may not be able to receive the data. This is due to other transmissions present in the network at the same time. According to the model we have used in the simulations, called Radio Noise Additive in the Swans/Jist simulator [1], when a signal arrives, its power is compared with the cumulative noise level sensed by this node’s radio and with a receive threshold. If the signal’s strength is above the cumulative noise level by some pre-defined threshold, and it is above the receive threshold, the radio locks on it and starts receiving. If the radio is unable to lock on the signal (the signal is too weak or the noise is too strong), the power level of the signal is added to the cumulative noise at this node. If a new signal arrives when the radio is locked on another signal, the node recalculates the cumulative noise including the new signal. If the old signal is still sufficiently strong, the radio stays locked on it. Otherwise, the old signal is added to the cumulative noise of this node and the node checks whether it can lock on the new signal (the cumulative noise includes the old signal, but not the new one). If it cannot lock on the new signal either, the new signal is considered as noise. We can see that if two nodes transmit within the range of a third node, the third node may not receive neither of them due to signal collision. A more detailed explanation of this model appears in [10].

A node can communicate with another node, which is not its direct neighbor, with the assistance of other nodes. The source node may send a message to its direct neighbor, which in turn will forward it to its direct neighbor. This process proceeds until the destination is reached. The set of nodes used to forward the message from source to destination is called a route from source to destination.
In MANETs, nodes can move. Therefore, if a node A has a node B as its direct neighbor at some moment, it is not guaranteed that they will remain direct neighbors as time passes. Consequently, routes between nodes may break and new routes may be constructed.

We assume that each node has a unique identifier. We denote the identifier of node $p$ by $ID_p$.

A publish/subscribe system is responsible for delivering data from a source to interested users. A user expresses interest in receiving certain data by submitting a predicate about corresponding data content. The predicate is called a subscription, and the node submitting the subscription is called a subscriber. The node that submits the data is called a publisher and the submitted data is called a publication.

In This work we target networks that are always connected w.h.p. We assume that nodes do not move very fast relative to the transmission range. In other words, neighborhood changes occur in a resolution of seconds (or longer). Nodes utilize a single shared wireless channel. Finally, the network size is constant, but its size is unknown to the participating nodes.
Chapter 3

Related Work

3.1 Network Wide Broadcasting Based Pub/Sub

A trivial solution for routing publications to subscribers is delivering each publication to all nodes in the system. Events are then filtered locally based on each node’s subscription. This may be achieved using various broadcast algorithms designed for MANETs [8, 14, 15, 20].

For example, in basic flooding [8], a source node broadcasts a message to all its neighbors. Every node, after receiving a message, rebroadcasts it exactly once. It is claimed in [8] that flooding is a good solution for highly dynamic networks.

The SBA [15] algorithm attempts to save unnecessary retransmissions made by the basic flooding protocol. A retransmission is redundant if all neighbors of the transmitting node have already received the message. In SBA, every node maintains a 2-hop neighborhood knowledge via periodic exchange of “hello” messages. When a node \( p \) receives a message from a neighbor \( q \), it determines which of its neighbors are not neighbors of \( q \). Those neighbors are called uncovered neighbors. If no such neighbors exist, \( p \) records the message, but does not retransmit it. If there are uncovered neighbors, then \( p \) schedules the retransmission with a random delay. For every redundant message received during the retransmission delay, the node recalculates the uncovered neighbors. If there are no uncovered neighbors, it cancels the message transmission.

OFP [14] utilizes location knowledge to save redundant packet retransmissions. OFP is based on the “covering problem”. It calculates the centers of circles of diameter “\( R \)” (“\( R \)” is a transmission range) where the center of each circle lies on the circumference of at least one other circle. The circles should cover the entire 2-dimensional space. When a node receives a message it first checks whether it is new and whether it received the message from a node that is located sufficiently far away. If not, the message will be recorded but not retransmitted. Otherwise, the node calculates its distance from the center of a circle and schedules a retransmission with a delay proportional to its distance. If during the delay it hears the packet being transmitted by another node, it checks whether the retransmitting node is closer than a predefined threshold. If yes, the retransmission is canceled.

The counter based flood [20] protocol optimizes many of these ideas. A description and discussion of this protocol is given in Section 6.3. We compare our approach with an implementation of
pub/sub over the optimized counter based flood protocol of [20].

3.2 Tree Based Pub/Sub

There have been several attempts to implement publish/subscribe in MANETs by routing publications to the subscribers over a tree substrate [9, 12, 13]. These ideas differ slightly in the way the tree is managed, and whether publications always have to first reach the root, as in [9, 12], or can be disseminated immediately by all internal nodes, as in [13].

Specifically, Huang and Garcia-Molina [9] construct a publish/subscribe tree (PST). The constructed PST is a spanning tree. One node in the system is designated the root of the publish/subscribe mechanism. Huang and Garcia-Molina assume that only the root node can publish new events. The tree is constructed via periodic exchange of “hello” messages. Every node independently decides who is its parent in the PST. When a node receives a “hello” message from its neighbor, it recalculates the routing metric, and selects a node with the best routing metric and connects to it as its son. When a node decides to join a new parent, it notifies the new and the old parents about it. Publications are routed from the root to all subscribers through tree brunches. A node will retransmit a publication only if there is a subscriber in its subtree interested in it. The paper proposes several routing metrics and compares the efficiency of the PST constructed by each of them. Huang and Garcia-Molina did not evaluate the proposed schema in a mobile network. They only moved one node to a random location and validated that the system stabilizes after the move and that a new PST is always constructed.

Moon, Ko and Lee [12] construct a PST via periodical messages, initiated by the root node, and forwarded by all other nodes. They claim that their approach is more adaptive to nodes’ mobility than the one proposed in [9]. A detailed overview of the protocol of [12] is given in Section 6.4, where we also compare it to our approach.

Mottola, Cugola and Picco constructed a spanning tree to forward publications to the subscribers [13]. Unlike previous approaches, publications are forwarded along the tree substrate rather than to the root and then down to the leaves. In order to route publications effectively along the tree, every node for every neighbor in the tree maintains a subscription list. If a node receives a publication, it checks in the subscription list whether one of its neighbors in the tree is interested in it. If yes, it forwards the publication. When a node subscribes, it floods its subscription to all nodes in the tree. The flood is performed by routing the subscription along tree branches. When a node receives a subscription, it adds the new subscription to the subscription list of the neighbor the subscription was received from. For every neighbor, except the one that sent the subscription, a node checks whether it already forwarded to this neighbor subscriptions covering the new one. If not, a node forwards the subscription to it. The tree structure is maintained like in MAODV [11, 18]. A downstream node, when realized that it lost a link to its parent, tries to reconnect to the tree. The paper describes which node to select as a parent in order to get more optimal PST and improves the MAODV reconnection process in order to reduce the delay until broken links are repaired.
3.3 Structureless Pub/Sub that Utilizes Network Mobility

There have been several works that utilize network mobility to reduce the number of sent messages that are used to route publications to subscribers [2, 3, 17].

Baldoni, Berlundd and Cugola [3] use network mobility to estimate the geographical distance between nodes. When forwarding publications, a node will retransmit the publication if its distance to at least one of the matching subscribers is lower than the distance of the node it received the publication from. The distance is estimated as follows: nodes periodically broadcast a “beacon” message. A node estimates the distance to the other node by measuring the time elapsed since it received the last “beacon” message from it. In addition, “beacon” messages are used to inform all nodes in the network about active subscriptions and their subscribers. Their proposed algorithm was evaluated in networks with high mobility (speed 5-45 m/sec) and it was shown that the best beaconing interval depends on the network speed. The study reported in [3] obtained only a moderate delivery ratio (≈ 85%).

Rezende, Rocha and Loureiro’s work is based on the idea that mobility increases the connectivity of MANETs [17]. The proposed solution is suitable for asynchronous applications that can tolerate delays of dozens of minutes. Nodes are responsible for disseminating locally received publications to different areas of the network after they move. Each publication is broadcasted by the publishing node at the moment it is created. When a node receives a new publication, it saves it in its buffer and rebroadcasts it immediately if it knows about subscriptions matching this publication. To make publications reach new areas, when a node stops, it broadcasts some of the publications saved in its buffer. Subscriptions are sent to neighbors in two scenarios: when they are created and every time a node stops moving. When a node receives a subscription, it saves it in a subscriptions buffer and broadcasts publications matching the new subscription. The subscriptions, initiated by other nodes, will stay for a limited time in the node’s buffer and will be deleted when the node starts moving again.

Baehni, Chhabra and Guerraoni [2] use nodes mobility to deliver publications to interested subscribers. In their algorithm, nodes that subscribed to some topic periodically send a “heartbeat” message. The “heartbeat” message informs neighbors about a node’s own subscriptions and matching publications the node has already received. When a node receives a publication, it drops the publication if it does not match any of the node’s subscriptions. If matched, the node saves the publication. A node rebroadcasts the publication if it knows about a neighbor that is interested in the publication too. When a node receives a “heartbeat” message from a neighbor and it realizes that one of the saved publications is matching the neighbor’s subscription, but the neighbor did not receive it yet, the node broadcasts the publication. The authors report high delivery ratio only when there are a lot of nodes that subscribe to the same topic. The algorithm’s performance improves with fast mobility.
3.4 Utilizing an Underlying Routing Protocol

Petrovic, Muthusamy and Jacobsen developed a reliable and fault-tolerant algorithm for publish/subscribe [16]. The algorithm requires an underlying routing protocol that maintains a route from every node to all others. A publisher knows all subscribers that should receive the publication. The publication is routed to subscribers as follows. Every publication carries a list of matching subscribers. When a node receives a publication (or a publisher itself when publishing), it queries the underlying routing protocol to determine the next hop to reach every subscriber in the list. Then, the node forwards the publication to each of these next hops, modifying the set of subscribers, in the publication, to include only those subscribers that a particular next hop is responsible for forwarding to. To achieve reliability, a publisher tags each publication with a unique sequence number. For each subscriber, the publisher holds the last sequence number for the publication that was forwarded to it. Subscribers once in a while, after receiving a publication, send an acknowledgment to the publisher specifying ranges of sequence numbers that the subscriber is missing. After receiving an acknowledgment, a publisher will forward undelivered publications to the subscriber. To make publishers know all subscribers interested in their publications, Petrovic, Muthusamy and Jacobsen propose that publishers periodically broadcast their advertisements, to which matching subscribers will reply with their subscriptions.

3.5 Other

Publish/subscribe can be seen as implementing the data-centric approach discussed in the Haggle papers, e.g., [19]. Most solutions to publish/subscribe in mobile ad-hoc networks, including ours, conform to some of the design principles detailed in the Haggle papers, such as empowering intermediate nodes, message switching, network visibility of user data, and request-response inside the network. Unlike Haggle, here we restrict the discussion to connected mobile ad-hoc networks. Also, our focus is less on the overall architectural design of the system, and more on the actual delivery protocols. Our work can be plugged into an architecture like Haggle in the case of connected networks.
Chapter 4

Density Driven Virtual Topography

We developed our algorithms based on the density driven virtual topography, described in detail in [6]. Briefly, a density driven virtual topography is a substrate for routing messages through the network. The virtual topography is constructed by assigning every node a virtual height. The height represents the density of the node’s neighborhood. Nodes located in a sparse area will have a lower height than nodes located in a dense area. The height of a node is calculated according to Equation 4.1.

\[ H_p = \sum_{q \in N(1,p)} |N(1, q)| \] (4.1)

After defining the height of a node we can define an order relation “higher than” on a set of nodes as a lexicographical order of their tuples \(< H, ID \rangle\), meaning that \( p \) is higher than \( q \) if and only if \((H_p > H_q \lor (H_p = H_q \land ID_p > ID_q)))\).

We define a hilltop to be a node that is currently higher than all of its direct neighbors (excluding itself). At every given time, we associate a single hilltop to each node recursively as follows: the hilltop of a hilltop node is itself and the hilltop of a node that is not a hilltop is the hilltop of its highest direct neighbor. We denote the hilltop of node \( p \) by \( \text{Hilltop}_p \). Intuitively, \( \text{Hilltop}_p \) is the node to which node \( p \) would eventually reach if it would follow a path greedily choosing the highest direct neighbor at each step until reaching the local maxima. Each node constantly remembers its currently assigned hilltop.

Nodes maintain the density driven virtual topography by periodically exchanging “hello” messages (the structure of such messages is presented in Figure 4.1). Every node associates with every neighbor a record of its direct neighbors, its height and its hilltop according to the latest “hello” message received from the neighbor. If a “hello” message from some neighbor \( q \) was not received for a predefined period of time, the node will assume that \( q \) is no longer its neighbor and will delete its record. Upon receiving a “hello” message and when deleting some neighbor’s record, the node has to recalculate its height according to Equation 4.1. After recalculating its height, the node also recalculates its hilltop. These hilltop and height recalculations are done asynchronously and independently by each node, and in particular might include some stale or inaccurate information for a short while.
<table>
<thead>
<tr>
<th>Name</th>
<th>Denoted</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>$ID_p$</td>
<td>The unique address of the node $p$</td>
</tr>
<tr>
<td>Height</td>
<td>$H_p$</td>
<td>The current height of node $p$</td>
</tr>
<tr>
<td>Hilltop</td>
<td>$\text{Hilltop}_p$</td>
<td>The current hilltop of node $p$</td>
</tr>
<tr>
<td>Direct neighbors</td>
<td>$N(1, p)$</td>
<td>The unique addresses of all of the direct neighbors of node $p$</td>
</tr>
</tbody>
</table>

Figure 4.1: The structure of node $p$'s hello message
Chapter 5

The Density-Driven Publish/Subscribe Protocols

In this chapter, we present the three protocols for publish subscribe that utilize the density driven virtual topography: 3DLS-Pub/Sub is based on the 3DLS algorithm of [6]. It has a low message cost, but suffers from very high delay and large packets; LCDD-Pub/Sub (low cost density driven pub/sub) improves 3DLS-Pub/Sub in terms of message size; LDDD-Pub/Sub (low delay density driven pub/sub) provides a low delivery delay compared to the other two algorithms, at the expense of a somewhat higher message cost.

In the 3DLS-Pub/Sub algorithm publishers forward their publications to several hilltops using a local greedy walk. Subscribers periodically collect matching publications from hilltops. The LCDD-Pub/Sub algorithm has a similar behavior to 3DLS-Pub/Sub except that it uses the network estimation mechanism to estimate the number of hilltops in the network. The estimated number of hilltops is used to determine the length of publication/lookup routes and enables significantly smaller messages, a more robust behavior and saves configuring the algorithm according to the network size. LDDD-Pub/Sub follows the opposite approach: subscribers periodically register at the closest hilltop, while publishers send their publications to relatively high number of hilltops. Then, hilltops forward publications to relevant subscribers. LDDD-Pub/Sub also utilizes the network estimation mechanism to determine the number of hilltops on the publication route. This results in much faster event notification, at the expense of a higher message count.

As shown in our simulations, described later in this dissertation, all three protocols have a very high delivery ratio, even in networks with medium mobility, and lower message cost compared to the CB-FLOOD [20] and TREE [12] algorithms; the latter representing flooding based protocols and tree based protocols, respectively.

Every subscription and publication has a unique identifier $< ID, sequence \ number >$ where $ID$ is the unique identifier of publishing/subscribing node. For simplicity of presentation, the identifier that we used in this work is the node’s MAC address, but it can be any other unique address. Before presenting our pub/sub protocols, we first describe the algorithms that are used as building blocks in all of them.
5.1 Density Biased Walk

In this section, we describe the algorithm used to forward a message from one node to another (the pseudo-code is presented in Figure 5.1). The algorithm utilizes information received from the virtual topography substrate (defined in Section 4) and information collected along the walk in order to navigate efficiently from hill to hill across the network. Every packet that advances by Density Biased Walk includes the following data structures:

- **route** - ID of nodes that forwarded the message.
- **trail** - ID of nodes that forwarded the message and their adjacent neighbors. A trail may be of two kinds: full or latest. A full trail contains all nodes that appear in the route and their neighbors. A latest trail is arranged in a FIFO order and has a size limit. When it exceeds the predefined size, the oldest addresses are removed.
- **hillTopCount** - number of hilltops that forwarded the message.

When a node receives a message, it has to decide whether to forward it or not (Section 5.1.1). If it decides to forward the message, it has to determine the next hop (Section 5.1.2).

5.1.1 Stopping a Message

A node may decide not to forward a message in the following cases:

- The node is a hilltop and the message has traversed through the required number of hilltops. The required number of hilltops is calculated dynamically depending on the message type and other parameters saved in the node. Note that every algorithm, presented later, calculates differently the number of hilltops that a message should pass through.
- The node cannot find an appropriate node to forward the message to (Section 5.1.2).

5.1.2 Calculation of the Next Hop

We now describe the main idea in determining the next hop. For a detailed overview refer to the “determineNextHop” method of the pseudocode in Figure 5.1.

The next hop is calculated differently if the node is a hilltop or not. Intuitively, if a node is a hilltop, the message should be forwarded roughly to the opposite direction from where it was received in order to reach a new hilltop (yet, without geographical knowledge). This kind of step is called pathFinderStep. When performing pathFinderStep, we are searching for a neighbor that:

- The message did not pass through yet. This means that it is not included in the route field.
- Has the largest fraction of neighbors that are not contained in the trail. This means that we are searching for the highest value of the following fraction: for a direct neighbor \( n \) and its direct neighbors \( E_n \)

\[
\frac{|E_n \in \neg \text{msg.trail}|}{|E_n \in \text{msg.trail}|}
\]  

(5.1)
Upon receive \((walk_msg)\) then

- handle\((walk_msg)\) /*according to the message's type */

- if \((timeToStop(walk_msg))\) then return; /* reached sufficient number of hilltops */

- nextHop := determineNextHop\((walk_msg)\);

- if nextHop = NULL then return; /* no next hop to forward the message to */

- update navigational information in walk_msg;

- send\((walk_msg, next_hop)\); /* forwarding message */

Upon determineNextHop\((walk_msg)\) then

- next_hop := NULL
- if \(p\) not hilltop then

- \(next_hop := getHighestNbr(walk_msg)\) ; /* gradientStep */

- endif

- if \((next_hop = NULL)\) then next_hop := getPathFinderHop\((walk_msg)\); /* pathFinderStep */

- if \((next_hop = NULL)\) then /* a dead end has been reached */

- if \((Route_Back.isEmpty())\) return NULL; /* the backtracking stack is empty - message returned to sender */

- next_hop := walk_msg.Route_Back.pop(); /* backStep */

- endif

- return next_hop;

Upon getHighestNbr\((walk_msg)\) do

- highest_nbr := NULL;

- foreach \(q \in (N(1,p)\setminus\{p\})\) do

- if \(|N(1,q)| \leq 2\) then continue; /* we are not considering dead ends*/

- if \(q \in walk_msg.route\) then continue;

- if \((Hilltops_{q} \in walk_msg.Trail)\) then continue; /* overcoming most hilltop misconceptions */

- if \((highest_nbr! = NULL \land highest_nbr is higher than q)\) then continue; /* select a neighbor with highest density */

- highest_nbr := q;

- end foreach;

- return highest_nbr;

Upon getPathFinderHop\((walk_msg)\) do

- best_exit_score := 0;

- best_exit := NULL;

- foreach \(q \in (N(1,p)\setminus\{p\})\) do

- \(#unseen_neighbors := |N(1,q)\setminus walk_msg.Trail|;\)

- \(#seen_neighbors := |N(1,q) \cap walk_msg.Trail|;\)

- \(exit_score := \#unseen_neighbors\#seen_neighbors;\)

- if \((exit_score <= best_exit_score)\) then continue;

- best_exit_score := exit_score;

- best_exit := q;

- end foreach;

- return best_exit;

Figure 5.1: Density biased walk forwarding algorithm (as performed by node \(p\)
If no appropriate neighbor is found (all neighbors that are not included in the route field have a value in Equation 5.1 equal to zero), the message will be forwarded to the last node on the route field. This step is called a backStep. If the route is empty, then the message advance is aborted, since no appropriate node to forward the message to was found.

If a node is not a hilltop, the message should be forwarded to the closest unvisited hilltop. This step is called a gradientStep. The best candidate to be the next hop is a direct neighbor which is a hilltop that the message did not pass through yet. If such a neighbor is not found, we search for neighbors that:

- The message did not pass through.
- The hilltop that is closest to this neighbor is not included in the trail.
- The neighbor has a highest density.

If no appropriate neighbor is found (all neighbors that are not included in the route have a closest hilltop contained in the trail field or a density equal to zero), a node searches for the next hop according to pathFinderStep. If this cannot be performed either, a node will search for a next hop according to backStep. If a next hop matching the backStep could not be found either, the message will not be forwarded to any additional node.

5.2 Route Walk

Here we describe how messages are forwarded from one node to another when there is a given route that the message has to pass through. In this case, normally a node should forward the message according to this route. But due to mobility, the route might break. We consider a route as broken when a node tries to send the message to the next hop, but the transmission failed. As assumed in Section 2, usually the unreachable node \( q \) has not moved too far from the current node \( p \). Hence, the transmitting node \( p \) looks for a neighbor \( r \) that has the unreachable node \( q \) as its neighbor (this data is retrieved from the density driven virtual topography, as described in Section 4). If such a neighbor \( r \) is found, then \( p \) forwards the message to it. This procedure is repeated if the transmission to the selected neighbor failed. To prevent loops, the selected neighbor may appear at most once on the route already traversed by the message.

5.3 Data Handoff

Since some data that was collected by a retiring hilltop must be available for at least one other hilltop in order for the pub/sub mechanism to work properly, this data should be transferred to a nearby hilltop. When a node decides that it is not a hilltop anymore (see Section 4), it issues a handoffMsg message. This message is forwarded according to the Density Biased Walk (Section 5.1) until it encounters the first hilltop. The data contained in the handoffMsg message depends on the protocol, as described individually in the following sections.
Upon receive(reg_msg) then
    if reg_msg.type ≠ EstimationMsg then
        merge(hilltopsEstNum, reg_msg.hilltopsEstNum);
        reg_msg.hilltopsEstNum := hilltopsEstNum; /*in case this message will be forwarded*/

Upon merge(win1,win2) do
    if win2 is empty then return;
    if win1 is empty then win1 := win2; return;
    win1.avg := (win1.avg * win1.count + win2.avg * win2.count)/(win1.count + win2.count);
    win2.count := win1.count + win2.count;
    if win1.count > 10 then win1.count := 10;
    return;

Figure 5.2: Merging estimated hilltops number when receiving a regular message (as performed by a hilltop)

5.4 Network Estimation Mechanism

This is a mechanism used by the LDDD-Pub/Sub and LCDD-Pub/Sub protocols to estimate the number of hilltops in the network. The mechanism relies on the density driven virtual topography property, by which the number of hilltops depends on the network size and deviates very slowly over time, and is usually only a marginal percentage of the total number of nodes in the system [6].

Every hilltop holds a weighted window representing the number of hilltops estimated by the mechanism, called estWin. To estimate the number of hilltops, a hilltop sends a message that should pass through all hilltops in the network. The message, called EstimationMsg, carries the number of hilltops it has passed through (initiated to 1 by the source node). The EstimationMsg is forwarded according to Density Biased Walk (Section 5.1). When a hilltop receives an EstimationMsg, it adds the number of traversed hilltops to its own estWin. We set the maximal number of visited hilltops to infinity since we want the message to pass through all hilltops (it will be stopped only when an appropriate next hop is not found). Since the maximal number of traversed hilltops is not limited and we do not want the EstimationMsg to wander around forever in the network, the trail type has to be full (see Section 5.1, message structure). We would like the value of estimated hilltops to be about the same in all hilltops. For this reason, every message sent by the publish/subscribe algorithm, except EstimationMsg, contains an average of the estimated hilltops number of all hilltops it has passed through. When a hilltop receives the message, it merges the average value with its own (pseudo code in Figure 5.2).

5.4.1 Scheduling an Estimation Round

We define an estimation round as the time since an EstimationMsg was sent by the source node until it was dropped by the last hop. An Estimation Round is started only by hilltop nodes. There are few issues that should be handled in scheduling the next estimation round. First, if every hilltop will start an estimation round periodically, in small networks we will see few estimation rounds.
handle_handoff_msg_with_estimation(msg)
    if (msg.estPassedTime > (currentTime - estSchedAt)) then
        /*schedule Estimation Round according to the received value*/
        estSchedTime = currentTime + msg.estSchedLeftTime;
        /*set the time when the Estimation Round was scheduled*/
        estSchedAt = currentTime - msg.estPassedTime;

Figure 5.3: Processing of handoff messages with estimation, as performed by a hilltop
(estSchedTime: the time when an Estimation Round will be started, estSchedAt: the time when
an Estimation Round was scheduled)

But in large networks that have a large number of hilltops, too many estimation rounds will be
started concurrently. This will dramatically increase the cost of the estimation mechanism in such
networks. The second issue that should be handled is the fact that in a mobile network, a node
may not remain a hilltop enough time to schedule an estimation round (for example, in a MANET
following the Random Walk mobility model in which nodes’ speed varies randomly between 0.5 and
2 m/sec, the average time a node serves as a hilltop before stepping down is less than 5 seconds).

For the number of estimation rounds to be independent of the network size, every hilltop
schedules the next estimation round according to the formula in Equation 5.2. When a hilltop
receives an EstimationMsg, it reschedules the next Estimation Round.

\[ estSchedTime = currentTime + estInterval + randomEstInterval \]  \hspace{1cm} (5.2)

Also, as mentioned above, in a mobile network, hilltops exchanges are common. Therefor, if
a hilltop schedules an estimation round, by the time the timer will expire, it probably would no
longer be a hilltop. To handle this issue, when a node decides that it is no longer a hilltop, it sends
a handoffMsg as described in Section 5.3, which in addition to other fields required by the protocol
also includes:

- time left until the estimation timer will expire
- time passed since the timer was scheduled

A hilltop that receives this handoff message will reschedule the start of the next estimation round
according to the received value if the time passed since it has scheduled the estimation round is
shorter than the one received in the message (pseudo-code in Figure 5.3).

5.5 3DLS-Pub/Sub

As mentioned before, 3DLS-Pub/Sub is a publish/subscribe protocol based on 3DLS [6]. The idea
of 3DLS-Pub/Sub is to store a publication in several hilltops. Then, each subscriber periodically
collects matching publications by querying a sufficient number of hilltops.
timeToStop(msg)
    if (msg.avgKnowWin \leq \text{PUB\_LIMIT}) then
        return true; /*stop forwarding the message*/
    return false;

Figure 5.4: 3DLS-Pub/Sub: stopping condition of \textit{publicationMsg}

5.5.1 Publishing

When a node wishes to publish some data, it has to disseminate it to several hilltops. To do so, it sends a \textit{publicationMsg} message. The \textit{publicationMsg} message is forwarded by the nodes according to \textit{Density Biased Walk} (Section 5.1). Every time the message passes through a hilltop, the publication is saved in the hilltop. Every publication has a predefined TTL specified in seconds; after this period of time, it will be deleted from hilltops’ memory. The number of hilltops that should forward the \textit{publicationMsg} is determined by the “knowledge level” of the network. The “knowledge level” of the network is defined as the average percentage of hilltops that hold in their cache a certain, unexpired, publication. The “knowledge level” is estimated by the subscribers (Section 5.5.2). For this, the \textit{publicationMsg} contains a field, which is a weighted average window, called \textit{avgKnowWin}. Every time the message is received in a hilltop, the hilltop’s \textit{knowWin} is merged into the message’s \textit{avgKnowWin}. The message traverses until the \textit{avgKnowWin} is higher than the desired value, which is a parameter of the algorithm, empirically set to 2 (Figure 5.4). In other words, the hilltop decides not to forward a \textit{publicationMsg} when it estimates that the “knowledge level” is high enough. In order to prevent rare situations were a message could follow an infinite route, the trail used by this kind of message should be full.

To have a better dissemination of publications among hilltops, when a publication message is forwarded by a hilltop, other publications already stored in the hilltop are piggybacked on the forwarded message. Yet, for space considerations, the number of piggybacked publications is limited. Hence, we select a constant number of freshest publications plus a constant number of randomly selected older publications. When a \textit{publicationMsg} is received at a hilltop, the piggybacked publications are stored locally.

5.5.2 Subscribing

A subscriber should collect matching publications that are stored in hilltops. Note that we are interested in collecting all relevant publications and not only the first matching one. For this, a subscriber sends a \textit{lookup message}, which is forwarded according to \textit{Density Biased Walk} (Section 5.1). When a message reaches a hilltop, all matching publications that are stored in the hilltop are saved in the message, and the message is forwarded to the next hop. When a node decides not to forward the message any longer (since it has visited enough hilltops or it cannot determine an appropriate next hop), it sends a \textit{response message} on the reverse path (the traversed route is saved in the \textit{route} field of the message). The \textit{response message} is forwarded according to \textit{Route}
<table>
<thead>
<tr>
<th>field size $[m^2]$</th>
<th>multiply factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000X1000</td>
<td>1.1</td>
</tr>
<tr>
<td>1580X1580</td>
<td>1.6</td>
</tr>
<tr>
<td>2000X2000</td>
<td>2</td>
</tr>
<tr>
<td>2450X2450</td>
<td>2.3</td>
</tr>
<tr>
<td>2830X2830</td>
<td>2.4</td>
</tr>
<tr>
<td>3160X3160</td>
<td>2.7</td>
</tr>
</tbody>
</table>

Table 5.1: 3DLS-Pub/Sub: lookup multiply factor

Walk (Section 5.2) and contains all matching publications that the lookup message has found.

The number of required hilltops that a message should visit is calculated dynamically. Every hilltop stores a weighted window indicating the “knowledge level” in the network, called $knowWin$. Every time the lookup message finds a matching publication that was not found in previously visited hilltops, it updates $knowWin$ with the number of hilltops it has visited so far. This means that $knowWin$ contains an average number of hilltops that the lookup message had to pass through until it has reached a new matching publication. This value represents the inverse of the “knowledge level” in the system; a high value of $knowWin$ indicates a low “knowledge level”. The propagation of a lookup message may be stopped according to the “knowledge level”. If the $knowWin$ is high, it means that a message should propagate through more hilltops to collect all publications. As mentioned before, the lookup message contains a field, which is a weighted average window, called $avgKnowWin$. Every time the message is received in a hilltop, the hilltop’s $knowWin$ is merged into $avgKnowWin$. The message traverses until the number of traversed hilltops is higher than the value of $avgKnowWin$ multiplied by a multiplication factor. The value of multiplication factor depends on the size of the network, and is set according to Table 5.1 (was calculated empirically).

The frequency of the lookups determines the delivery delay of the publication. The delay is about half the interval between sequential lookups. Note that in order to get a high hit rate, the publication’s TTL (time to live) should be at least twice the interval between sequential lookups.

5.5.3 Handoff

When a node stops acting as a hilltop, it has to pass the data in its memory to some other hilltop. A node that realized that it is no longer a hilltop, sends a handoffMsg containing all publications it is aware of and the value of the $knowWin$ (Section 5.3). When a hilltop receives a handoffMsg, it merges the new publications and the received $knowWin$ with its own.

5.6 LCDD-Pub/Sub

LCDD-Pub/Sub is designed to overcome the main shortcomings of 3DLS-Pub/Sub. The first is that 3DLS-Pub/Sub suffers from very large messages due to two reasons:
Figure 5.5: LCDD-Pub/Sub: stopping condition of publicationMsg (avg: estimated number of hilltops)

- The trail type of the publicationMsg has to be “full”.
- It is necessary to piggyback a large amount of publications on every publicationMsg to get a high hit rate.

Another problem is the multiply factor that has to be set according to the network size, which makes the algorithm less flexible.

LCDD-Pub/Sub utilizes the network estimation mechanism (Section 5.4) to overcome the above drawbacks. The idea behind LCDD-Pub/Sub is to determine the number of hilltops that a message should pass through according to the number of hilltops in the network (rather than the “knowledge level” as in 3DLS-Pub/Sub), which is estimated by the network estimation mechanism. Below, we list the differences between LCDD-Pub/Sub and 3DLS-Pub/Sub.

5.6.1 Publishing

When a node wishes to publish some data it sends a publicationMsg. The publicationMsg is identical to the one used in 3DLS-Pub/Sub except for the fact that the number of hilltops that the message should visit depends on the number of hilltops estimated by the network estimation mechanism. In Figure 5.5 we illustrate the calculation of the required number of hilltops that a message should visit. We can see that the message should pass through a finite number of hilltops. Thus, we can use a latest trail type to reduce the message size.

5.6.2 Subscribing

The subscriber periodically sends a lookup message to find matching publications at the hilltops. The lookup Message is identical to the one used in 3DLS-Pub/Sub except for the fact that the number of hilltops that the message should visit depends on the number of hilltops estimated by the network estimation mechanism (Figure 5.6). We use a latest trail type in lookup message to reduce the message size.

5.7 LDDD-Pub/Sub

LDDD-Pub/Sub has a low delivery delay compared to LCDD-Pub/Sub and 3DLS-Pub/Sub. As described before, the delay of LCDD-Pub/Sub and 3DLS-Pub/Sub depends on the lookup frequency.
In order to make the algorithm efficient, the interval between subsequent lookups should be on the order of a few minutes. The low lookup frequency causes the delay to be in the order of a few minutes, while LDDD-Pub/Sub has a delivery delay of several seconds. Yet, it comes at a cost of additional messages.

In LDDD-Pub/Sub, publishers send their publications to hilltops. Subscribers disseminate their subscriptions to the closest hilltop; the corresponding hilltop is responsible for forwarding matching publications to the subscriber. The publications are forwarded on the reverse path of an appropriate subscription message. The algorithm utilizes the property of the virtual topography substrate that every node is only a few hops away from the closest hilltop. Therefore, the routes from hilltops to subscribers are very short and do not require maintenance except for occasional refreshing.

### 5.7.1 Publishing

When a node wishes to publish some data, it sends a publicationMsg message. The publicationMsg message is forwarded by the nodes through the network according to Density Biased Walk (Section 5.1). When a hilltop receives a publication, it checks whether it has registered subscribers interested in this publication (Section 5.7.2). If yes, it forwards the publication to them (Section 5.7.3). Every time a publication passes through a hilltop, it is saved in its memory. Every publication has a predefined TTL; after this period of time, it will be deleted.

The number of hilltops that should forward the publicationMsg is determined by the number of hilltops estimated by the network estimation mechanism. Long publication routes will improve the delay until the publication is received by the subscribers and will increase the number of subscribers that receive the publication. But on the other hand, it will increase the number of messages sent for every publication. To balance the above tradeoff, every publication should traverse a moderate number of hilltops (number of hilltops that a publication should visit is illustrated in Figure 5.7). To make the publication reach other hilltops, it is assisted by subsequent publication messages.

When a publication message is forwarded by a hilltop, publications already stored in the hilltop are piggybacked on the forwarded message. The number of piggybacked publications is limited. Hence, we select a constant number of freshest publications plus a additional constant number of randomly selected older publications. When a publicationMsg is received at the hilltop, for every piggybacked publication the hilltop checks whether it is already stored in its memory. If not, it treats it like a new publication, saves it and forwards it to matching subscribers. Since the

```plaintext
timeToStop(msg)
    if (roundDown(√avg) ≤ msg.hillTopCount) then
        return true; /*stop forwarding the message*/
    return false;
```

Figure 5.6: LCDD-Pub/Sub: stopping condition of lookup message (avg: estimated number of hilltops)
timeToStop(msg)
    advMul := 1;
    if (roundDown(avg) > 2) then
        advMul := −0.025 * mulFactor * (roundDown(avg) − 2) + 1;
        if (advMul * roundDown(avg) ≤ msg.hillTopCount) then
            return true; /*stop forwarding the message*/
        return false;
    end

Figure 5.7: LDDD-Pub/Sub: stopping condition of publicationMsg (avg: estimated number of hilltops)

publicationMsg message should pass through a finite number of hilltops, we can use a latest trail type to reduce the message size. To increase the hit rate of LDDD-Pub/Sub, every time a node receives a publicationMsg message it matches the publications to its subscriptions.

5.7.2 Subscribing

When a node subscribes, it sends a subscriptionMsg message. The message is forwarded by Density Biased Walk (Section 5.1) until it reaches the first hilltop. When a hilltop receives the subscriptionMsg message, it associates the subscriber with the received subscription. In addition, it saves the route the subscriptionMsg message has followed. This route will be used to forward the matching publications to the subscriber. Every subscription registration has a timeout. If it is not refreshed until the timer expires, the registration is deleted. The subscription messages are sent periodically by the subscriber. This is required since a route to the hilltop may break due to nodes’ mobility.

Since the subscriptionMsg message is forwarded until it reaches the first hilltop, it is sufficient to use a latest trail type in subscriptionMsg to reduce the message size.

5.7.3 Forwarding the Publication

When a hilltop receives a publicationMsg message, for every publication it checks whether it is new (not stored in its memory). For every new publication, it saves the publication and looks for matching subscribers, to whom the hilltop has to forward the publication. As described before, this is done by utilizing the reverse path through which the subscription traversed from the subscriber to the hilltop. To forward the message to the subscriber, we use two types of messages: responseMsg and aggregatedResponseMsg. The responseMsg is a message carrying all new publications matching the subscriber’s interest and the route to the subscriber. The aggregatedResponseMsg consists of several responseMsg messages, all of which share the first hop on the route to the subscriber. That is, instead of sending multiple responseMsg messages to the same neighbor, a node sends only one aggregatedResponseMsg message. Both responseMsg and aggregatedResponseMsg are forwarded according to Route Walk (Section 5.2). The difference is that responseMsg route is the entire route to the subscriber, whereas in aggregatedResponseMsg the route consists of only one node, the first common node in all contained responseMsg messages. When a node receives an
aggregatedResponseMsg, it treats every responseMsg separately and if there are still several responseMsg messages that should be forwarded to the same neighbor, they are aggregated again into a single aggregatedResponseMsg message.

5.7.4 Handoff

The handoffMsg message in this protocol contains all publications stored in a node’s memory and all registered subscribers. When a hilltop \( p \) receives the handoffMsg message, it checks whether the message includes new publications. If yes, \( p \) saves them and forwards them to matching subscribers. After that, \( p \) adds the received subscribers to its own subscribers. Special care should be taken with the routes to the subscribers. In general, the route to a new subscriber consists of the old route appended with the route the handoff message has traversed. Yet, for efficiency reasons, we search from the end of the handoff route till we find a common node to both routes; denote this node commonNode. Then, a final route is constructed by concatenating the shortest prefix of the original route until commonNode with the suffix of the handoff route starting at commonNode.

24
Chapter 6

Simulations

In this chapter, we evaluate the performance of our protocols in a simulated environment. We conducted extensive simulations using the Swans/Jist simulator \[1\] to evaluate the performance of our protocol under different network characteristics and using different protocol parameters.

**Setup:** The default network characteristics and default publish/subscribe service parameters are presented in Tables 6.1 and 6.2 respectively. The default parameters we used in 3DLS-Pub/Sub, LCDD-Pub/Sub and LDDD-Pub/Sub are presented in Table 6.3. The simulation consists of three time periods:

- **Warmup:** this time period is used to stabilize the network. Messages sent during the warmup period are not counted in the simulation results. Publications issued during this period expire before warmup ends. Warmup lasts for 200 seconds in counter based flood simulations, and 60 minutes in all other simulations.

- **Simulation:** during this period, publications are sent and all subscriptions are active. The simulation period lasts for 1,200 seconds.

- **Epilogue:** This time period is used to finish collecting all publications. During epilogue, no publications are sent, but all subscriptions are still active. The epilogue period lasts for 1,300 seconds.

Each data point was generated as an average of 10 runs.

**Metrics:** During each simulation we tested the following measures:

- **Hit rate:** the percentage of published items that reached the matching subscribers. In particular, if some item was matched by several subscriptions, it is counted according to the number of nodes that it should be delivered to.

- **Message count:** number of messages sent during the simulation, not including hello messages.

\[1\]This is the overall rate of publications in the network. For example: if there are 1000 publishers in the network than every publisher will publish every 1000 sec.
<table>
<thead>
<tr>
<th>Network parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation area</td>
<td>3160X3160 m²</td>
</tr>
<tr>
<td>Node count</td>
<td>1000</td>
</tr>
<tr>
<td>Transmission range</td>
<td>198 m (each node has 12 neighbors on average)</td>
</tr>
<tr>
<td>Sensing range</td>
<td>376 m</td>
</tr>
<tr>
<td>Signal propagation model</td>
<td>RadioNoiseAdditive</td>
</tr>
<tr>
<td>PathLoss</td>
<td>two-ray</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>11 Mb/s</td>
</tr>
<tr>
<td>Mobility model</td>
<td>Random Walk (no pause time)</td>
</tr>
<tr>
<td>Speed</td>
<td>1.5-2.5 m/sec</td>
</tr>
<tr>
<td>Mac protocol</td>
<td>IEEE 802.11 with no message fragmentation</td>
</tr>
</tbody>
</table>

Table 6.1: Network parameters

<table>
<thead>
<tr>
<th>Publish/subscribe parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscribers number</td>
<td>20% of nodes</td>
</tr>
<tr>
<td>Subscriptions per subscriber</td>
<td>1</td>
</tr>
<tr>
<td>Subscriptions matching single publication</td>
<td>14.3% of subscriptions</td>
</tr>
<tr>
<td>Publication frequency</td>
<td>1 publication/sec</td>
</tr>
<tr>
<td>Number of publishing nodes</td>
<td>all nodes</td>
</tr>
</tbody>
</table>

Table 6.2: Default publish/subscribe service parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>3DLS-Pub/Sub</th>
<th>LCDD-Pub/Sub</th>
<th>LDDD-Pub/Sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello cycle time</td>
<td>10 sec</td>
<td>10 sec</td>
<td>10 sec</td>
</tr>
<tr>
<td>neighbor timeout</td>
<td>55 sec</td>
<td>55 sec</td>
<td>55 sec</td>
</tr>
<tr>
<td>publication TTL</td>
<td>1300 sec</td>
<td>1300 sec</td>
<td>300 sec</td>
</tr>
<tr>
<td>lookup cycle time</td>
<td>600 sec</td>
<td>600 sec</td>
<td>——</td>
</tr>
<tr>
<td>subscription registration cycle time</td>
<td>——</td>
<td>——</td>
<td>270 sec</td>
</tr>
<tr>
<td>subscription registration TTL</td>
<td>——</td>
<td>——</td>
<td>600 sec</td>
</tr>
<tr>
<td>network estimation cycle time</td>
<td>——</td>
<td>600 sec</td>
<td>600 sec</td>
</tr>
<tr>
<td>trail size</td>
<td>——</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>piggybacked new</td>
<td>100</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>piggybacked random</td>
<td>100</td>
<td>5</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 6.3: Publish/subscribe algorithms default parameters
• Delay: the average time that passed since the publication was published until it was received at a matched subscriber. In particular, if some item is matched by several subscriptions, its delay is counted separately for every node that it should be delivered to. We are not counting here publications that were not received.

• Data sent: Total number of bytes sent by the algorithm during the simulation, not including hello messages. We include here messages that were not received by any node. We do not count the MAC protocol overhead nor the overhead caused by message retransmission at the MAC layer.

### 6.1 LDDD-Pub/Sub \textit{mulFactor} Parameter

In this section, we explore the influence of the \textit{mulFactor} parameter, which determines the length of the publication path (Figure 5.7). A lower value of \textit{mulFactor} will result in longer publication routes. Recall that longer publication routes cause the message count metric to grow. But on the other hand, the hit rate would be higher and the delay would decrease.

Figure 6.1(a) illustrates the message count for different \textit{mulFactor} values. As expected, as the \textit{mulFactor} increases the message count decreases. Figure 6.1(b) illustrates the hit rate for different \textit{mulFactor} values. The hit rate is very high when \textit{mulFactor} is set to 1 but becomes very low when \textit{mulFactor}'s value is 3 or higher. Since we are interested in a high hit rate, the acceptable values for \textit{mulFactor} are 1.5 and 2.

Figure 6.2 presents the hit rate achieved by LDDD-Pub/Sub when \textit{mulFactor} is 1.5 and 2 for networks with different nodes’ speed. We can clearly see that setting the \textit{mulFactor} parameter to 2
will result in a low hit rate in static networks. Therefore, we set $mulFactor$ to 1.5 in all our following simulations.

![Hit rate for varying speed](image-a)

![Hit rate for varying speed (zoom in)](image-b)

Figure 6.2: LDDD-Pub/Sub, varying mobility for mulFactor 1.5 and 2 values

### 6.2 Density Driven Pub/Sub Performance

**Subscriptions matching a publication** We study the message cost, hit rate and delay metrics for various numbers of subscriptions matching a single publication. The number of subscribers is left constant and it is set to 200 nodes.

Figure 6.3(a) illustrates the number of messages sent as the number of subscriptions matching a single publication grows. The message count metric in LCDD-Pub/Sub is not influenced by the number of subscriptions matching a single publication since publication and lookup routes length is determined by the number of estimated hilltops. The number of messages sent by LDDD-Pub/Sub and by 3DLS-Pub/Sub grows as the number of matching subscriptions increases. This is an expected result since in LDDD-Pub/Sub more publications have to be forwarded from the hilltops to the subscribers. 3DLS-Pub/Sub sends more messages since the estimated knowledge level becomes lower as the number of matching publications increases. The value of the knowledge level become lower since every time the lookup visits a new node it has a higher chance of finding new matching publications. Notice that when updating the knowledge level we do not take into account the amount of new publications found in the current hilltop.

The hit rate (Figure 6.3(b)) is almost 100% in all proposed algorithms, and does not depend on the number of matching subscriptions. LDDD-Pub/Sub has about 99.2% hit rate, LCDD-Pub/Sub achieves 99.6% and 3DLS-Pub/Sub delivers 99.9% of matching publications.

The delay metric does not depend on the number of subscriptions matching a publication.
**LCDD-Pub/Sub** and **3DLS-Pub/Sub** have an average delay of 300 sec, which is half of the lookup interval. **LDDD-Pub/Sub** has a delay of 4.7 sec.

![Message count for varying subscriptions matching single publication](a)

![Hit rate for varying subscriptions matching single publication](b)

**Figure 6.3: Varying subscriptions matching single publication**

**Subscribers number** We examined the message cost and hit rate metrics for various number of subscribers (Figure 6.4). Every publication is matched by 14.3\% of subscribers. Hence, raising the number of subscribers proportionally raises the number of subscriptions matching a single publication. As a reference, we added the message cost of **counter based flood** to Figure 6.4(b) (for detailed description of **counter based flood** see Section 6.3). The curve matching the **counter based flood** algorithm is constant since the number of subscribers does not affect the amount of sent messages in a flooding based protocol.

As expected, as the number of subscribers increases, the total number of messages sent by the algorithms increases (Figure 6.4(b)). The message cost in **LDDD-Pub/Sub** rises since the number of matching publications grows with the number of subscribers. In **LDDD-Pub/Sub** what mostly influences the number of sent messages is the number of subscribers every publication should be forwarded to and not the total number of subscribers located in the network. **LCDD-Pub/Sub**, as was shown in the previous section, is not sensitive to the number of subscriptions matching a single publication, but as the number of subscribers rises, the number of lookups increases and this causes the increase in the message cost metric. In **3DLS-Pub/Sub**, the message cost increases with the number of subscribers in the same ratio as in **LCDD-Pub/Sub**. It is interesting to notice that when the number of subscribers grows, the length of publication and lookup routes in **3DLS-Pub/Sub** decreases. The reason why the message cost of **3DLS-Pub/Sub** and of **LCDD-Pub/Sub** grows in the same ratio is that **3DLS-Pub/Sub** has much longer lookup routes than **LCDD-Pub/Sub**. Note
that, in all proposed protocols, even when there are almost no subscribers the message cost is not zero since the publications are still forwarded to the hilltops as if there were many subscribers.

The hit rate of 3DLS-Pub/Sub (Figure 6.4(a)) is a bit lower (99.2%) when there is a very low number of subscribers since the “knowledge level” is rarely being recalculated.

![Figure 6.4: Varying subscribers number](image)

**Field size** We evaluate the proposed algorithms’ performance vs. the size of the simulated network while the node density is left constant. Notice that the number of subscribers is set to 20% of the nodes and every publication matches 14.3% of subscribers. Hence, increasing the size of the network results in a larger number of subscribers and a higher number of subscriptions matching a single publication.

The hit rates are reported in Figure 6.5(b). All three protocols achieve a high hit rate. Yet, LDDD-Pub/Sub has the lowest hit rate (about 99%).

The message cost (presented in Figure 6.5(a)) is about the same in 3DLS-Pub/Sub and in LCDD-Pub/Sub while LDDD-Pub/Sub has a significantly higher cost. This is an expected result since in 3DLS-Pub/Sub and LCDD-Pub/Sub the publications are aggregated in one response message while in LDDD-Pub/Sub, to achieve low delivery latencies, a publication is forwarded immediately when it is received at the hilltop.

The delay (presented in Figure 6.5(c)) is about the same in 3DLS-Pub/Sub and in LCDD-Pub/Sub (~ 5 minutes) and equals half of the lookup interval. The delay in LDDD-Pub/Sub is significantly lower (~ 4.7 sec). The delay in 3DLS-Pub/Sub and LCDD-Pub/Sub is constant and has no sensitivity to the network size. But, in LDDD-Pub/Sub it grows from 1.8 seconds in a network of 1580X1580 m² to 4.7 seconds in a network of 3160X3160 m².

The total amount of data sent (Figure 6.5(d)) by LCDD-Pub/Sub is lower than in the two other

30
protocols. This is since it has a very low number of piggybacked publications in publicationMsg messages and the trail size is limited in all messages except for the EstimationMsg message. 3DLS-Pub/Sub sends less data than LDDD-Pub/Sub in small networks, but as the network size increases, it suffers from very large messages due to the unlimited trail. We see that in large networks, even LDDD-Pub/Sub, which sends significantly more messages, outperforms 3DLS-Pub/Sub.

![Figure 6.5: Varying field size](image)

**Mobility** We evaluate the proposed algorithms’ performance while varying the nodes’ speed. Figure 6.6(a) illustrates the hit rate of the proposed protocols. All three protocols achieve a high hit rate when nodes’ speed is less or equal to 10 m/sec. At higher speeds, the hit rate of 3DLS-Pub/Sub drops to 87% while LCDD-Pub/Sub’s hit rate is 91% and LDDD-Pub/Sub still achieves
97% delivery ratio. The drop in hit rate of 3DLS-Pub/Sub is caused by the decrease in overall knowledge level. When a node starts to act as a hilltop it is created with an empty cache and this reduces the overall knowledge level in the network. When two hilltops merge, if at least one of them has had a non empty cache, the united hilltop would also have a non empty cache. It means that when merging two hilltops, one with an empty cache and the other not, the overall knowledge level rises. As described in [6], frequent hilltops’ splits and merges reduce the knowledge level of the network. This results in longer lookup routes that are required to find all matching publications. From the simulations, we see that 46% of the response routes are broken. This is since routes are more likely to be broken when node’s speed is high and the lookup route was very long. The other issue, is the fact that 20% of lookups were stopped since no appropriate next hop was found. This means that those lookups probably did not collect all required publications. LCDD-Pub/Sub also suffers from a decrease in the hit rate when nodes move at speeds of 15-20 m/sec. This is since 20% of the response routes are broken due to high nodes’ mobility.

The message cost (presented in Figure 6.6(b)) of all three algorithms grows with mobility. In LDDD-Pub/Sub the message cost grows mainly due to message retransmissions. Recall that the algorithm determines the next hop it should send the message to according to hello messages exchanged with its neighbors. But, when nodes move quickly the information is stale and the next hop is often no longer there. The fact that the next hop has moved away is detected only after a message transmission fails. When we examine the number of received messages in LDDD-Pub/Sub, we see that it does not depend on the node’s speed. The increased message cost in 3DLS-Pub/Sub is due to the long routes the publications and lookups have to follow until they are not forwarded anymore. The increase in message cost in LCDD-Pub/Sub is caused by the combination of two facts:

- lookup and publication routes become slightly longer when mobility increases
- message retransmission when the next hop has moved out of the transmission range.

The delay (presented Figure 6.6(c)) in 3DLS-Pub/Sub and in LCDD-Pub/Sub sharply increases when nodes mobility is higher than 10 m/sec. The delay in LDDD-Pub/Sub, Figure 6.6(d), varies from 4.7 seconds, in a network with nodes’ speed of 1.5-2.5 m/sec, to 29 seconds, when nodes’ speed is 15-20 m/sec.

**Density** We evaluate the proposed algorithms’ performance while varying the number of nodes located in the network. The network size is left constant and set to $3160 \times 3160 \, m^2$. The number of subscribers is left constant and set to 200 nodes. Every publication is matched by 14.3% of the subscribers.

The hit rate (Figure 6.7(a)) is high for all proposed algorithms for all tested nodes’ density. LDDD-Pub/Sub and LCDD-Pub/Sub achieve a hit rate of 93% and 3DLS-Pub/Sub achieves 98%, even when the density is 16 nodes/km$^2$ (according to Gupta and Kumar [7] this density is slightly lower than the minimum required density to achieve connectivity w.h.p.).

The message cost (Figure 6.7(b)) in LCDD-Pub/Sub and in 3DLS-Pub/Sub does not depend on density as long as the network is connected. It is a little bit higher when the density is 16 nodes/km$^2$. 

32
Figure 6.6: Varying mobility
LDLD-Pub/Sub’s message cost slightly increases with the density.

The delay (Figure 6.7(c)) in LCDD-Pub/Sub and in 3DLS-Pub/Sub is constant and does not depend on nodes’ density. LDDD-Pub/Sub has a higher delay of 15 sec when the nodes’ density is 16 nodes/km$^2$. It drops to 4 sec as density increases.

![Hit rate for different density](image1)
![Message count for different density](image2)
![Delay for different density](image3)

Figure 6.7: Varying density

**Lookup frequency** Next, we explore the tradeoff between the message cost and the delay in LCDD-Pub/Sub and in 3DLS-Pub/Sub. Recall that in LCDD-Pub/Sub and in 3DLS-Pub/Sub subscribers collect matching publications periodically. Decreasing the time interval between sequential lookups will decrease the delay, but will increase the message count metric. Figure 6.8 illustrates...
the message count and the delay metrics for varying lookup intervals. For comparison, we have added appropriate values of LDDD-Pub/Sub and counter based flood (for a detailed description of counter based flood see Section 6.3). Note that their curves are constant since the lookup interval is not applicable to those algorithms.

We see that the delay (Figure 6.8(b)) in LCDD-Pub/Sub and in 3DLS-Pub/Sub have the same value for the same lookup frequency, and it equals half the lookup time interval. The message cost increases with the lookup frequency, but it is still smaller than the one of counter based flood.

![Graphs showing message count and delay for varying lookup intervals.](image)

Figure 6.8: Varying lookup interval

6.3 Density driven Pub/Sub vs. Flooding-Based Protocols

In this section, we compare our proposed algorithms with counter based flood ([20]). In counter based flood, publications are flooded throughout the network. Every node, when receiving a publication, matches it to its local subscriptions. The flood is performed "wisely". When a node receives a message it first validates that it is new. Then it waits a random time interval before transmitting it. If during the waiting period the node hears this message being retransmitted at least a predefined number of times, it cancels its transmission. The number of times that the message should be heard in order for its transmission to be aborted depends on the number of node’s neighbors as described in [20]. To determine the number of neighbors, nodes exchange hello messages periodically. If a node does not hear from one of its neighbors for some time then it is not counted as a neighbor anymore. Counter based flood was implemented with the parameters listed in Table 6.4.

Field size We now compare the performance of counter based flood with our protocols for various network sizes. The nodes’ density is left constant and set to 100 nodes/km². The number of
<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello cycle time</td>
<td>10 sec</td>
</tr>
<tr>
<td>neighbor timeout</td>
<td>55 sec</td>
</tr>
<tr>
<td>transmission wait period</td>
<td>1 sec</td>
</tr>
</tbody>
</table>

Table 6.4: Counter based flood default parameters

subsribers is set to 20% of the nodes and every publication matches 14.3% of subscribers.

The hit rate is high in all algorithms (Figure 6.9(a)). But we can see that counter based flood has a slightly lower hit rate.

From Figure 6.9(b), we can see that the message count in counter based flood is even higher than in LDDD-Pub/Sub and the benefits of our proposed schemes become more significant as the network size grows. Figure 6.9(c) illustrates the message count metric of our algorithms vs. counter based flood when every publication is matched by 50% of subscribers, which is 10% of the nodes. We can see that when the number of subscriptions matching every publication is 10% of the nodes, it is better to use counter based flood rather than LDDD-Pub/Sub since counter based flood sends about the same number of messages as LDDD-Pub/Sub, but it has significantly smaller messages. LCDD-Pub/Sub and 3DLS-Pub/Sub still have a significantly lower message count metric than counter based flood. (Recall, the message cost metric of LCDD-Pub/Sub is not influenced by the number of subscriptions matching every publication.)

The amount of data sent (Figure 6.9(d)) by counter based flood is much lower than of our proposed schemes. This is expected, since counter based flood has no additional information that should be sent in publication messages. But, we should keep in mind that counter based flood utilizes a broadcast channel while all messages sent by LDDD-Pub/Sub and LCDD-Pub/Sub are unicast. Unicast messages are often sent using higher bandwidth than broadcast messages. For example, in many 802.11g cards, the difference can be be more than an order of magnitude.

The delay (Figure 6.9(e)) in LDDD-Pub/Sub is 2.4 times greater than the delay in counter based flood in small network. It becomes about the same when the network size varies from 1580X1580 m² to 2000X2000 m² and grows fast with the network size. We can see that the delay in LDDD-Pub/Sub grows faster than in counter based flood as the network size increases.

**Mobility** We compared the hit rate, message count and delay metrics of counter based flood with LDDD-Pub/Sub and with LCDD-Pub/Sub for various nodes’ speeds. From Figure 6.10(a) we can see that the hit rate of counter based flood only slightly decreases with nodes’ speed. In static networks, counter based flood outperforms our algorithms and achieves 99.3% hit rate while LCDD-Pub/Sub exhibits a 98.6% hit rate and LDDD-Pub/Sub delivers 95.4% of the publications. But when there is even slow mobility in the network, LDDD-Pub/Sub’s hit rate rises and has about the same value as the one of counter based flood. The hit rate of LCDD-Pub/Sub is higher than the one of counter based flood in networks with nodes’ speed of 0.5-10 m/sec.

The message count of counter based flood (Figure 6.10(b)) slightly decreases with speed, but it is
Figure 6.9: Counter based flood vs. our proposed algorithms for varying field size
higher than the one of LCDD-Pub/Sub. It is also higher than the message count of LDDD-Pub/Sub when nodes’ speed is lower or equal to 10 m/sec. The delay (Figure 6.10(c)) of counter based flood only slightly increases with nodes’ mobility, while the delay of LDDD-Pub/Sub reaches a high value when nodes’ speed is above 5 m/sec.

![Graphs showing hit rate, message count, and delay for different speeds](image)

Figure 6.10: Counter based flood vs. our proposed algorithms for varying mobility

**Density**  Figure 6.11 shows the performance of counter based flood, LDDD-Pub/Sub and LCDD-Pub/Sub in various network densities. The density has been changed by varying the number of nodes placed on the same 3160X3160 m² test area. The number of subscribers is left constant and set to 200 nodes.
From Figure 6.11(a) we see that counter based flood has a hit rate of 82% when there are 500 nodes placed in the field, while LDDD-Pub/Sub and LCDD-Pub/Sub achieve 93% hit rate in the same density. (Notice that 500 nodes located in 3160X3160 m\(^2\) test area is a slightly lower density than the required one, according to Gupta and Kumar [7], to achieve full network connectivity w.h.p.). When the density increases, counter based flood has a high hit rate that is only slightly lower than the one achieved by our proposed schemes.

The message count metric (Figure 6.11(b)) of counter based flood rises when the density is 66 nodes/km\(^2\) (750 nodes) and decreases back as the density increases. This is a result of counters values used in counter based flood that depend on number of node’s neighbors. Counter based flood has a higher cost than LDDD-Pub/Sub and LCDD-Pub/Sub for all tested node densities.

The delay (Figure 6.11(c)) of counter based flood, like the delay of LDDD-Pub/Sub, decreases with the density. Yet, it is lower than the delay of LDDD-Pub/Sub for all tested densities. When there are 500 nodes placed in the test area, the delay of LDDD-Pub/Sub is 15 seconds while counter based flood achieves a delay of 6 seconds. Yet, recall that counter based flood delivers only 82% of publications to the subscribers while LDDD-Pub/Sub has a 93% hit rate.

6.4 Density Driven Pub/Sub vs. Structured Protocols

In this section, we compare the tree based protocol of [12] for pub/sub with LDDD-Pub/Sub and LCDD-Pub/Sub. In the scheme of [12], some node is designated the root of the tree. Publishers forward their publications to the root along the tree. Publications are then forwarded down the tree to interested subscribers. Only nodes that have children matching the publication broadcast the publication. The tree is maintained as follows: the root periodically broadcast an advertisement message to its neighbors. Every advertisement message is associated with a sequence number to determine more fresh advertisement messages. An advertisement contains the number of nodes that forwarded this message, which measures the distance to the root. When a node receives an advertisement message, it checks whether it is new. Only then it may forward it. A message is forwarded after a random delay. If during the delay an advertisement message was received with a lower distance to the root, the advertisement message that was scheduled will be updated with the lower distance metric. To construct a tree, each node sends join messages to the node it received an advertisement message from with the lowest distance metric. In the paper it is not clear what happens if the lower metric is not the first one, whether a node should cancel previous join messages or not. Since Join is a very costly operation, we decided to implement it as follows: a node that should send a join message will send it together with the appropriate advertisement message to the neighbor it received the lowest distance metric from during the waiting period. Advertisement messages received after an appropriate advertisement was forwarded are discarded. The join message contains the identifier of the selected parent and the subscription information of the node and its children. Note that if a node received a join message from its neighbor with new subscriptions, it has to send an additional join message to its parent with an updated subscriptions list. Moon, Ko and Lee propose to use two optimizations:

- When receiving a join message, a node checks whether the message contains the same parent
Figure 6.11: Counter based flood vs. our proposed algorithms for varying density
Table 6.5: Tree default parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>root advertisement cycle time</td>
<td>10 sec</td>
</tr>
<tr>
<td>son timeout</td>
<td>11 sec</td>
</tr>
<tr>
<td>advertisement wait period (constant)</td>
<td>40 msec</td>
</tr>
<tr>
<td>advertisement wait period (random)</td>
<td>20 msec</td>
</tr>
<tr>
<td>publication wait period (random)</td>
<td>100 msec</td>
</tr>
</tbody>
</table>

identifier as its own parent and whether the subscriptions in the message can cover its own subscriptions. If both conditions are met, the node cancels the sending of any pending join message.

- A node cancels the forwarding of an advertisement if the number of received advertisement messages with the same sequence number exceeds a predefined threshold value.

We used only the second optimization, since we decided that it is better to send join messages with unicast, which is more reliable than broadcast. In addition, we do not think that the first optimization will significantly reduce the number of sent messages, but will increase the consumed calculation power since the node will have to find whether the forwarded join message fully covers itself and its children’s subscriptions.

The tree algorithm was implemented with the parameters listed in Table 6.5. Note that son timeout is much smaller than in previously analyzed algorithms since join messages are sent using unicasts, which is much more reliable than broadcast. Therefore, if we did not receive a join message from a child for a very short period of time, we may assume that it joined another parent. Another imported issue in our implementation is the way publications are forwarded along the tree to the root. In order to increase the hit rate, which is low when there is mobility in the network, publications are broadcasted, and nodes that have a lower distance to the root forward the publication.

When we compare the protocols, we did not take into account the advertisement messages. The number of advertisement messages is about 25% lower than the number of hello messages sent by previously described algorithms.

Field size We now compare the performance of the tree protocol with LDDD-Pub/Sub and with LCDD-Pub/Sub for various network sizes. We can see that tree suffers from a low hit rate (Figure 6.12(a)), and it decreases as the network size increases. The message cost of tree (Figure 6.12(b)) is higher than any of the proposed schemes. Note that the message cost is a little bit smaller than the cost of counter based flood. The delay (Figure 6.12(c)) of tree increases with the network size, but it is significantly lower than the delay of LDDD-Pub/Sub and of counter based flood.
Figure 6.12: Tree vs. our proposed algorithms for varying field size
Figure 6.13: Tree vs. our proposed algorithms for varying mobility
**Mobility**  We compared the hit rate and message count of *tree* with *LDDD-Pub/Sub* and with *LCDD-Pub/Sub* for various nodes’ speeds. We can see that as the nodes’ speed increases, the hit rate of *tree* significantly decreases (Figure 6.13(a)). The *tree* protocol is not able to reach a satisfying hit rate (above 90%) even when nodes’ speed is 2.5 m/sec. The message cost decreases as the speed rises. This is since there are much fewer publications that reach their destination (Figure 6.13(b)). The delay (Figure 6.13(c)) of the *tree* protocol is very low; its value is about 0.8 sec. for all tested nodes’ speeds.
Chapter 7

Discussion and Conclusions

In this dissertation, we have presented three novel protocols for pub/sub in mobile ad-hoc networks, namely 3DLS-Pub/Sub, LCDD-Pub/Sub and LDDD-Pub/Sub. The proposed protocols are built on top of the density-driven virtual topography substrate described in [6]. In the process, we have designed a network estimation mechanism that makes our protocols scalable and adaptive to all network sizes. Our simulations show that all our proposed schemes obtain a very high delivery rate. Yet, they differ in the tradeoff they make between message sizes and delivery delay with LCDD-Pub/Sub and LDDD-Pub/Sub usually outperforming 3DLS-Pub/Sub.

LCDD-Pub/Sub is suitable for applications that can tolerate delays of a few minutes. Our simulations (Section 6) demonstrate it to be highly frugal in terms of message count compared to the other schemes and robust to mobility.

LDDD-Pub/Sub on the other hand delivers publications to subscribers within a few seconds even in very large networks. It performs very well as long as nodes move in moderate speed (walking, running, and cycling, but not driving). LDDD-Pub/Sub is favorable to counter based flood [20] when a latency of several seconds is acceptable and each publication matches a relatively small number of subscriptions (below 10%-18% of the total nodes, depending on the scenario). We believe that this is a common case, as different people tend to have different interests. An open problem for future research is to combine our schemes with counter based flood as follows: If the publishers assumes that its event is of interest to a large fraction of the entire network, it would be disseminated using the efficient flooding mechanism. Otherwise, using one of our mechanisms. An interesting question is how a publisher can tell how many subscribers are interested in its event, especially in a content based pub/sub service.

Looking ahead, some of our ideas can be used to implement other services in MANETs like membership services [4]. A membership service has to maintain a list of currently active and connected nodes. This can be obtained as follows. Every hilltop maintains two lists: one is called global view and another is called local view. Global view is a list of nodes this hilltop thinks are active in the network. Every node connected to the system periodically sends a message similar to a “subscription” message in LDDD-Pub/Sub informing the closest hilltop of its existence. When a hilltop receives a “subscription” message, it adds the source node to the local view. When a hilltop receives a “subscription” from a node not included in global view it broadcasts the updated view.
to all nodes in the system. In order to handle situations when nodes leave the network without appropriate disconnection, hilltops periodically exchange their local views via messages like those used in the network estimation mechanism. The uniqueness of an “estimation message” is that it is routed through almost all hilltops in the network. This may be used to merge all local views into a global view as follows: some hilltop starts sending a message similar to an “estimation message”. Every time it reaches a hilltop, the local view of the hilltop is merged with local views of previously visited hilltops. Then, the message is forwarded with the updated view. When the message stops, it has visited almost all hilltops and contains an updated global view. To disseminate the new global view among all hilltops, another “estimation message”-like message is sent. Every time it reaches a hilltop, the new global view is recorded there. If some node does not appear in at least one local view for some time period, it is deleted from the global view, and the hilltop that realizes this broadcasts an updated view. The other option, in order to avoid broadcasting the view every time it is updated, is that every node will ask its closest hilltop for an updated view every time it is needed. In this case, when a hilltop receives a “subscription” from a previously unknown node, it only needs to inform other hilltops about this by sending an “estimation”-like message.
Bibliography


פרסום/מנרי מוכתרי צפיות

ברשתות אל-הוק ניידות

كلفנו כותב
פרסום/מנוי מווכנים כספים

ברשחתת אד-חק יידות

תורור על מחקר

ליש מיכלי חלקי של זריררות לקבלת החרוזה המגדור

למדעים במדעי המחשב

cplusplus אנה

دراج לเสนה הטכניות - מקוונ חדשני לישראלי

נסף תשע"א - חוף אפריל 2011
היבחר על מחקר בשדה בדדרכת פרו פורפ' בר רועי פרידמן
פוקרה למדעי המחשב

אני מודע ל tüכית ואלה המוחה הכספית והדריך הבודהmistי

הפקוד המתקדם

רשמה את-וק מודעה ממרב תמציתיות של תודות�יות, כל תודות ביכולת לוכלת הקודים, תוכל קרית שיתוף תכנית תוכן אל קיימ מצל שקית. לע

הקדמה כי-ווק, בורהשזת את-וק לא-קוד אוritosmaktadır של היקף לכל תודיה לכל מקוד הפוך להפריד בין המופיעה מבחר

הедакציה שלחן ב- Wi-Fi Direct Wi-Fi Alliance

продажתו של-וק של-וק של מנהיג השיטה, עם יד-וק בו פומשים

אצנים המנטה-ביסנג' ילי אורו-הורודות הדלות יד-וק יד של-

㈔ה מיהודית כי-מדגמלת להחקור את זה עם teg השיגRIES בא-וק.

פרובז立て עניית היד-וק המחוזות המחוזות מתאימה לימים עד הינן.

הקוד של-וק הנותנבה בوبة הודיע בה.activities, ב-וק מוכנים פעמים שני

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הפקוד המתקדם של ה-פקוד המתקדם של שהני

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

 pn hitch הערוך של היד-קוד המחוזות המחוזות מתאימה לימים עד הינן.

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

 pn hitch הערוך של היד-קוד מחוזות המחוזות מתאימה לימים עד הינן.

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

הקוד של-וק של מנהיג הורודות הדלות יד-וק יד של-

 pn hitch הערוך של היד-קוד מחוזות המחוזות מתאימה לימים עד הינן.

 pn hitch הערוך של היד-קוד מחוזות המחוזות מתאימה לימים עד הינן.