Towards Efficient Group Management and Communication for Large-Scale Mobile Applications

Rafael Oliveira Vasconcelos, Lincoln David Nery e Silva, Markus Endler
Department of Informatics
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio
Rio de Janeiro, Brazil
{rvasconcelos, lnsilva, endler}@inf.puc-rio.br

Abstract—Applications such as fleet management and logistics, emergency response, public security and surveillance or mobile workforce management use geo-positioning and mobile networks as means of enabling real-time monitoring, communication and collaboration among possibly large sets of mobile nodes. The majority of those systems require real-time tracking of mobile nodes (e.g., vehicles, people or mobile robots), reliable communication to/from the nodes, as well as group communication among the mobile nodes. In this paper we describe a distributed middleware with focus on management of context-defined groups of mobile nodes, and group communication with large sets of nodes. We also present a prototype Fleet Tracking and Management system based on our middleware, give an example of how context-specific group communication can enhance the node’s mutual awareness, and show initial performance results that indicate small overhead and latency of the group communication and management.

Keywords—group communication, group management, context-defined groups, middleware, mobile systems, DDS.

I. INTRODUCTION

Applications such as fleet management and logistics, emergency response, public security or mobile workforce management require real-time dissemination of context/position information of the mobiles, as well as efficient point-to-point and group communication with/among hundreds or thousands of nodes, which are usually connected through 3G/4G networks subject to intermittent connectivity. Many such applications also require means of dynamic grouping of mobile nodes, i.e., according to their current location or other context information, in order to enable group messages to (or within) such groups. As an example, consider the following hypothetic application and scenario: Door2Door (D2D) is a parcel delivery service that operates on wide-scale, and has 5000+ vehicles, each equipped with an on-board positioning and cellular (3G/4G) communication device. Through their Fleet Management system, D2D can track its vehicles in real-time, as well as send instant messages to drivers individually, or to groups of drivers, to give them instructions, ask them some questions, or send them specific alerts. In addition, the service also needs to group its vehicles according to their current position (e.g., all the vehicles within a given metropolitan area, or at a specific highway), according to the transported load (e.g., perishable goods), or current operating conditions of the vehicles (e.g., with high speed, or high engine RPM), so as to be able to send specific instructions or alerts to the drivers of each of these vehicle subgroups. Unlike pre-defined and explicit groups, these groups have a dynamic membership set, as vehicles may frequently enter or leave each group. Finally, D2D also uses mutual tracking and communication capabilities among the vehicles. This allows each driver to receive updated information of all the other vehicles that are current members of the driver’s vehicle group, (e.g., all the vehicles in the same region), enabling the drivers to help each other and to stand in when some problem with a nearby vehicle of the fleet is detected.

Although such dynamic group communication capability opens up an array of new communication and collaboration possibilities in many mobile management application, there is still the question of how context-defined groups can be maintained in an efficient and scalable way.

To facilitate the development of applications with such complex group communication requirements, we have developed the Scalable Data Delivery Layer (SSDL). SSDL employs the OMG Data Distribution Service for Real-time Communication (DDS) [1] standard for communication among stationary nodes in its core, and implements a scalable gateway approach to bridge the gap between the high-performance communication within DDS and the potentially unreliable and disconnection-prone communication with thousands of mobile nodes. While in a previous papers [2,3] we presented performance results of SDDL’s mobile unicast and broadcast communication, this paper describes the design and implementation of asynchronous group communication and management capabilities in SSDL and how it can be used for instantaneous sharing of application data and context among many mobile nodes.

Overall, this paper makes the following contributions: First, we present an approach for efficient management of dynamic membership in context/location-defined groups of mobile nodes, for the purpose of group cast communication. Second, we present the SSDL middleware for mobile communication that uses this group management for enabling efficient sharing of application messages and context updates within context-defined groups, Third, we present a prototype of a Fleet Tracking and Management system based on our middleware where it is possible to create region-specific groups, and let all the vehicles within each group to share messages and location information in almost real-time. Finally, we present extensive performance tests that show the low overhead incurred by SDDL’s group management and that group cast message delivery is efficient even for thousands of nodes.
The rest of the paper is organized as follows: next section gives an overview of the SDDL middleware and section 3 explains its group management and communication approach. In section 4 we explain the group subscription capabilities for mobile nodes and how they are implemented. In section 5 we present a prototype Fleet Management System, and how it uses position-defined group. Sections 6 and 7 present initial results of performance and related work, respectively. Finally, in section 8 we draw some conclusions and point to future work.

II. SDDL OVERVIEW

Scalable Data Distribution Layer (SDDL) is a communication middleware that connects stationary DDS nodes in wired “core” network to mobile nodes with an IP-based wireless data connection. Some of the stationary nodes are information and context data processing nodes, others are gateways for communication with the mobile nodes, and yet others are monitoring and control nodes operated by humans, and capable of displaying the mobile node’s current position (or any other context information), managing groups, and sending message to the mobile nodes (MN). SDDL employs two communication protocols: DDS’s (Distribution Service Service) Real Time Publish/Subscribe RTPS [1] for the wired communication within the SDDL core, and the Mobile Reliable UDP (MR-UDP) [2,3] for the inbound and outbound communication between the core network and the mobile nodes. DDS [1] is a standard from the OMG, which specifies a peer-to-peer middleware architecture for real time and high-performance data distribution, with Quality of Service (QoS) contracts between producers and consumers of data (e.g. reliable communication, data persistency, priority lanes, etc.).

As part of the SDDL core, two kinds of nodes play an important role in regard to group communication:

The Gateway (GW) defines a unique Point of Attachment (PoA), for connection with the mobile nodes (MN). The Gateway is thus responsible for managing a separate MR-UDP connection with each of these MN, forwarding any application-specific message or context information into the core network, and in the opposite direction, converting DDS messages to MR-UDP messages and delivering them reliably to the corresponding MN(s).

GroupDefiners are responsible for establishing the group membership of all MNs. To do so, they subscribe to the DDS topic where any application message or Context Update (CxtU) is disseminated, e.g. messages originated from the MNs and relayed by the corresponding GW, and map each node to one or more groups, according to some application-specific grouping logic. This group membership information is then shared with all Gateways in the SDDL core network, to which all Gateways subscribe so that they can update their cached MN’s group membership information.

III. SDDL GROUP MANAGEMENT

In SDDL, a mobile node $n$ may be member of one or more groups. Groups of nodes may be either long-lived or context-defined/implicit. In the former category they are explicitly defined by the application developer/operator, e.g. nodes belonging to a certain type of vehicle or company. In Context-defined groups, membership of a node is determined implicitly by its most recently produced context data. For example, if we consider the context “position”, then all nodes located within a certain geographic region (e.g. a metropolitan area or within the boundaries of a state), can form a context-defined group. However, any other mobile-sensed & produced context data may be used as well for group membership determination, such as: the node’s local weather/temperature condition, its speed, residual energy (fuel or the user/driver’s energy), or current mobile telephony company connected with. In any case, such context-defined groups have to be continuously updated according to the most recent context update sent by the nodes, and this will be done by specific nodes in the SDDL core, called GroupDefiners.

A. GroupDefiner

The main task of a GroupDefiner, a fixed node of the SDDL core, is to continuously check and update the MN’s group membership by subscribing to Context Update (CxtU) messages generated by the MNs. Whenever a GroupDefiner detects some group membership change of a MN it will publish a Group-membership-diff message (shortly, G-diff) related to the MN into the DDS domain, i.e. requesting to include the MN into some groups, and/or to remove it from other groups. For example, consider a mobile node X moving along Highway 66. At one point in time it may be member of the position-related groups “Highway 66” and “California”, but after moving on some miles and crossing the state border, it leaves group “California” and joins group “Nevada”, while remaining in group “Highway 66”. In this case, the G-diff message would conceptually consist of commands {add(X,"Nevada"), rem(X,"California")}. All these G-diff messages will be used by the corresponding Gateway to update its Group-to-MN mapping. For long-lived groups, the GroupDefiner will only publish a G-diff message if the node is explicitly removed from, or added to, a group.

Each GroupDefiner internally consists of a generic CxtU message processing part, and an application-specific, Group selection module. The generic part is responsible for reading CxtU messages from the DDS domain, recording the current groups related to the message, and handing the CxtU object to the Group Selection module. This module will execute a specific mapping algorithm to determine the context-defined group(s) that correspond to the received context data, e.g. if the CxtU data is within a certain range or region. This result is then handed back to the generic part of GroupDefiner, which will calculate the G-diff message and publish it to GroupAdvertisement topic in the DDS domain. Fig. 1 illustrates the GroupDefiner’s architecture and its interaction with the Gateways.

This split between the generic and the specific group membership processing parts has some advantages: (i) it is possible to deploy several GroupDefiners in the SDDL core, each of which executing a Group selection module that examines a certain type of the CxtU object independently of the other modules, and (ii) Group selection modules may be
easily exchanged in the GroupDefiner, without compromising the remaining function of the SDDL group management and communication capabilities.

**Fig. 1. Interaction between GroupDefiners and Gateways**

### B. Gateway

The Gateway holds two mappings named “MN-to-Groups” and “Group-to-MNs” to support the Group Management and Communication. The first mapping, MN-to-Groups, is used for setting group tags to each inbound MN message, which will be checked by the GroupDefiner. The second, Group-to-MNs mapping, allows the Gateways to efficiently select all the MNs that belong to a group, so as to efficiently forward groupcasts to them. Thus, the main task of Gateways in regard to Group Management and Communication is to keep its mappings up to date, which is done by subscribing to G-Diff messages from the GroupDefiner.

Another responsibility of the Gateway is to receive and manage MNs’ subscriptions, which could be application message subscriptions or context update subscriptions. The application message subscription informs the Gateway that the MN is interested in receiving all application messages sent to the specified group (that differs from the groups the MN is already member). When receiving such a subscription from a MN, the Gateway stores it, and from now on, all groupcast messages to the group informed in the subscription will be forwarded by the Gateway also to this MN. The message subscriptions are treated differently from the MN’s group membership. While MN’s group membership is determined by the Gateway’s group tagging of CxtU messages, and by the G-diff messages disseminated by the GroupDefiners, the message subscriptions are stored locally in the Gateway to enable the correct forwarding of groupcast messages to MNs based on group membership and its subscriptions. In case of mobile node handover, the MN has always to re-submit its subscriptions to the new Gateway.

The MN is also capable of expressing interest of CxtU messages produced by others MNs. After receiving a context update subscription from one of its MNs, the Gateway forwards this CxtU-subscription to a Global Subscription Manager (GSub), which is responsible for manage the CxtU-subscriptions and relaying the CxtU messages of a group to all the subscribed MNs, through the corresponding Gateways. Different from application message subscriptions, MNs do not need to re-submit their subscriptions after a handover.

Further explanations about asynchronous communication can be found in section IV.

### IV. ASYNCHRONOUS COMMUNICATION AT THE MOBILES

SDDL also includes a client Group API which provides to the mobile client developer the capability for realizing asynchronous communication based on groups. Hence, mobile nodes can subscribe and unsubscribe to data published in some existing groups, that may be both explicit or context-defined.

The API for asynchronous group communication includes methods `getSubscribedGroups` and `getMyImplicitGroups` that allow the client on the MN to get the list of groups that it has explicitly subscribed to, or where it is member due to its current context/location data, respectively. Using methods `subscribe` or `unsubscribe`, a MN may register (or unregister) its interest in delivery of application-specific or CxtU messages sent to a set of groups. Thus, invoking `subscribe(getMyImplicitGroups)`, a MN may receive continuously CxtU from all other MNs that are currently in its groups. And with method `publish(msg, groups)` it may send a message to a list of groups. Finally, using `addMembershipListener` / `removeMembershipListener` a mobile node can receive notifications about membership changes of its groups. Whenever such a change happens, either `onGroupEntry` or `onGroupLeave` callback methods (of `MembershipListener`) are called. These have to be implemented by the application programmer, for the mobile client to properly handle the entering or leaving of groups.

#### A. Managing Subscription at the Gateways

When a mobile node (MN) issues a `subscribe()` to some groups, a subscription control message is sent to the corresponding Gateway, which stores this subscription information locally and marks this MN as being interested in groupcast messages of each subscribed group. Thus, every DDS message tagged with the corresponding GroupID that this Gateway receives is replicated and forwarded to the mobile nodes subscribed to the group.

On the other hand, the Gateway will not tag inbound CxtU messages (from this MN) as being associated to this subscribed group(s), since the node is not a producer of data for this group, but only a consumer. Neither will the Gateway update the mobile node’s group subscription record upon reception of a G-diff message from a GroupDefiner.

Whenever a mobile node performs a handover to a new Gateway, the mobile client will have to re-issue all the subscriptions to the new Gateway, and the former Gateway will discard the node’s subscription records as soon as the MR-UDP connection from the mobile node is broken.

Recall that a MN will automatically receive all groupcast messages for all the nodes in the long-lived and implicit groups that it is member of, including their Context Updates. Now, through by issuing a subscription, a MN may also receive continuous CxtUs produced by any nodes that are members of implicit groups, for example, allowing it to receive instantaneous position updates of MNs that are within a geographic region associated to the group.
V. FLEET TRACKING AND MANAGEMENT SYSTEM

SDDL has been used within a Fleet Tracking and Management application of a major Brazilian gas distribution company. Using this application, the company's Operations Center is able to track trajectories of its trucks in real-time, in order to optimize the trucks' itineraries, to detect and notify obstructions or jams on roads, and to monitor the truck driver’s actions (e.g. elapsed time on both planned and involuntary stops). In this application, we have implemented the following context-based group communication capability: for an important highway in Brazil, BR-116 (Rodovia Pres. Dutra), we have created location-based groups for each highway section, which are defined by perimeter bands along the highway spanning a few meters on each side of the highway. We have named the groups using the highway number and the closest city name, e.g. BR-116-Guarulhos, BR-116-Resende, etc. Hence, each truck that is moving on such section of the highway is automatically associated with the corresponding group, and all the position updates and application messages sent by this vehicle (e.g. Instant message about a spotted accident on the highway) are immediately disseminated to all members of the corresponding group. This allows truck drivers (subscribers) of a highway section to get up-to-date information about possible problems (potholes, accident reports) on the highway. This dynamic group membership is processed in a Group Selection Module at one GroupDefiner, which holds all the rectangles/polygons defining each road section. It uses the JavaTopologySuite library (JTS) [6] to check if nearby nodes/trucks are inside or outside any of the rectangles, as shown in Figure 3 for the BR-116-Guarulhos section.

VI. GROUP-CAST AND GROUP MANAGEMENT PERFORMANCE

In this section, we will focus exclusively on performance tests which access the round trip delay of application groupcast messages originating at SDDL Controller, and the overhead incurred in the processing and dissemination the G-diff messages (for group management). In the sections D, E and F, we then present performance results of groupcasts originated at the MNs and delivered to subscribed MNs. Performance results of unicast and broadcast message delivery can be found in [2].

A. Basic Groupcast and Group Management Performance

All the tests were executed with following system configurations and simulation parameters: (a) 1,000, 2,000, 3,000, 4,000 simulated MNs per Gateway; (b) 2 Gateways; (c) CxTU frequency by each MN was twice per minute; (d) 1 GroupDefiner; (e) high and low probability of MNs group membership change (i.e. by dynamic grouping based on different parts of the generated position data by the simulated MNs). Furthermore, we evaluated two distinct implementations of GroupDefiner and Gateway: one using G-Diff message and other disseminating the whole MN’s group membership information.

B. Setup of the Experiment

To test the communication performance, in each test round we split all simulated MNs across all Gatewaysx2tu and then sent unicast and groupcast “ping” messages to approximately 25% of all MNs. For each type of message we calculated the round trip delay (RTD), i.e. the difference between the instant of time the message was sent and the instant when the confirmation was received, from all the group members. We also measured: (i) the RTD between the instant the Gateway published the CxTu in the DDS domain and when it updated its mapping after receiving the MNs’ group memberships and (ii) the group membership processing time, i.e. the Gateway internal processing time to update the mapping’s data structures. Recall that so far, our group management performance tests are restricted to the SDDL core elements, i.e. so far we have not evaluated the delays of the callback invocations OnGroupEntry() and OnGroupLeave() at the mobile nodes.

Our hardware test setup was composed of 5 desktop computers of our lab, interconnected through a 10/100/1000 Mbps switch: two of them running Gateways, one simulating all the MNs, one executing a GroupDefiner and the last one running our Control and Visualization Program, used to send the “ping” unicast and groupcast messages.

C. Results

The results are presented in Fig. 4, Fig. 5 and Fig.6. All times are shown in milliseconds. For the sake of better legibility, subtitles were abbreviated as follows: RTD unicast/G-cast means the RTD of a unicast/groupcast message; RTD G-Diff GW-GD means the RTD between the publication of a new CxTu and the corresponding reception of the G-Diff message from the GroupDefiner (measured at the Gateway); RTD No G-Diff GW-GD is also a RTD between the Gateway and GroupDefiner but now for transmitting the whole MN group membership information; G-Diff Process time means the group membership processing time by Gateway using G-Diff messages and No G-Diff Process time is the same as before but using the entire group membership information. All results are the mean value of 6 measurements. The RTD for unicast “ping” messages are presented here for the sake of better time comparison.
As Fig. 4 shows, the RTD G-Diff GW-GD times are lower than the other implementation that does not use G-Diff message. As expected, the network bandwidth saved when disseminating only the group membership changes caused better response times for the Gateways’ update of the MNs’ group membership. In a scenario with higher group change probability, times are a little bit higher when compared with a low group change probability. This is explained by the higher processing and network overhead that Gateways and GroupDefiner experience when MNs switch groups more frequently.

**TABLE I. RTD results for 10% of all MNs in SG**

<table>
<thead>
<tr>
<th>Mobile Nodes</th>
<th>Smallest RTD</th>
<th>Mean RTD</th>
<th>Highest RTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>34 ms</td>
<td>1,482 ms</td>
<td>30,288 ms</td>
</tr>
<tr>
<td>2,000</td>
<td>33 ms</td>
<td>1,655 ms</td>
<td>32,569 ms</td>
</tr>
<tr>
<td>3,000</td>
<td>36 ms</td>
<td>8,365 ms</td>
<td>44,627 ms</td>
</tr>
</tbody>
</table>

Table I shows that on average, a group cast (with all associated replies) to/from 3,000 MN SG members (last row) has a RTD of 8,365 ms - while other 3,000 simulated MNs were producing background CxtU traffic load. In worst case, the group cast plus replies to 300 SG members had a RTD of 44,627 ms. This huge difference between the mean and highest RTD may be due to the occasional overload at some MN-Set-Simulator programs and/or due to the congestion of the shared WiFi network, which took some simulated MNs longer to reply.

**TABLE II. RTD results for 25% all MNs in SG**

<table>
<thead>
<tr>
<th>Mobile Nodes</th>
<th>Smallest RTD</th>
<th>Mean RTD</th>
<th>Highest RTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>31 ms</td>
<td>2,704 ms</td>
<td>34,268 ms</td>
</tr>
<tr>
<td>2,000</td>
<td>33 ms</td>
<td>7,136 ms</td>
<td>42,214 ms</td>
</tr>
<tr>
<td>3,000</td>
<td>37 ms</td>
<td>20,094 ms</td>
<td>91,210 ms</td>
</tr>
</tbody>
</table>

Table II shows that increasing the percentage of SG subscribers from 10% to 25% does not increase significantly the Mean and Highest RTDs. Regarding the latter, group cast plus reply to 750 SG subscribers had a RTD of 91,210 ms against RTD of 44,627 ms for 300 SG subscribers, which is an
increase of approx. 104%. The mean RTD had a similar proportional increase (140%) from 300 to 750 simulated MNs.

Table III. RTD results for 50% all MNs in SG

<table>
<thead>
<tr>
<th>Mobile Nodes</th>
<th>Smallest RTD</th>
<th>Mean RTD</th>
<th>Highest RTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>31 ms</td>
<td>5,363 ms</td>
<td>40,139 ms</td>
</tr>
<tr>
<td>2,000</td>
<td>30 ms</td>
<td>16,834 ms</td>
<td>56,453 ms</td>
</tr>
<tr>
<td>3,000</td>
<td>43 ms</td>
<td>24,785 ms</td>
<td>102,592 ms</td>
</tr>
</tbody>
</table>

Table III shows the situation where 50% of the MNs are members of SG. Mean group cast RTD to 1,000 MNs was measured less than 17 seconds and group cast RTD to 1,500 is less than 25 seconds, recalling that these values are also affected by the access latency due to the shared WiFi AP and the overload at MN-Set-Simulator. And since the mean and highest RTD increased only slightly (compared to the results of Table II, but now for 25% more MNs in SG), it seems to indicate that the Gateway and the GroupDefiner introduce negligible overhead to the group cast.

VII. RELATED WORK

Cugola et al [5] propose a context-aware extension to the publish-subscribe model that combines content and context-awareness, and present a routing protocol to efficiently implement it in a distributed system. Similar to our context-defined group cast their middleware also supports the delivery of messages according to the publisher's context. The main differences of their work are: that the context has to be explicitly informed rather than implicitly evaluated by the system, their protocol is suited only for static nodes, and their performance evaluation was limited to only hundred clients.

In [7], a DDS-based middleware is proposed for real-time data distribution, with a specific architectural element to support adaptation to mobile networks. This element, the Mobile DDS Client, is a lightweight version of DDS executed on all mobile devices. But due to DDS connectivity and Firewall/NAT traversal restrictions, all these Mobile DDS Clients must run in single network domain and rely on stable wireless connectivity. Moreover, in [7] there is no support for context-defined groups and groupcast communication.

REVENCE [10] is a DDS-compliant news dispatching infrastructure for mobile nodes and which is capable of transparently and autonomously balancing the data distribution load. It implements a P2P routing substrate - deployed on a LAN - that is fault tolerant and self-organizing. More specifically, it is able to detect crashed nodes, and to reorganize the routing paths from any source node to any mobile sink nodes. Concerning asynchronous communication capabilities at the mobile nodes, this system provides full DDS-based Pub/Sub support, while SDDL implements only a restricted form of group subscription, but which has the advantage of high performance and scalability.

SALES [8] is a middleware for data distribution, aimed at large-scale mobile systems. Unlike our work, SALES relies solely on pure UDP for inter-node communication, and hence does not take advantage of all real-time and QoS support of DDS. Although SALES probably supports grouping of nodes and group-based communication, its hierarchical architecture suggests that efficient group communication is only possible if all MNs that are group members are within a branch of the tree.

VIII. CONCLUSION

We presented a DDS-based middleware that implements efficient group cast communication and management of context-defined groups, that is practical for large sets of mobile nodes. Our evaluation has shown that SDDL has a small group communication latency (e.g. group cast to approx. 1,500 MNs in less than 12 secs) and the management overhead for such groups is quite small.

As future work, we will investigate how the groupcast support can be extended to enable also conjunctive groups (e.g. delivery to all MN ∈ GA ∩ Gb) and how to implement other complex group subscriptions.

REFERENCES