Volume 1, No. 2, July-August 2010

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Complete Design & Implementation of Replica Middleware Strategy in Dmanets

Pankaj Kumar Verma*
I.T. Dept.
Haryana College of Tech. & Management
Kaithal, India
Justpankajverma@gmail.com

Dr. Vijay Lamba
E.C.E. Dept.
Haryana College of Tech. & Management
Kaithal, India
lamba_vj@hotmail.com

Abstract: Mobile Ad Hoc Network is an infrastructure less network where there is no need of any base station where Every node is responsible to handle the data & transmit the data. The world called dense Mobile Ad-hoc NETworks (MANETs), i.e., limited spatial regions, such as shopping malls and airports, where a high number of mobile peers can separately cooperate without a statically deployed network infrastructure. This paper will provide the complete detailed study of Replica middleware strategy architecture [1] and all its functions and methods to manage, retrieve, and distribute replicas of resources to cooperating nodes in a dense MANET. The guideline is to exploit high node population to enable hopeful lightweight resource replication capable of tolerating node exits/failures. In our paper we have adopted some original approximated solutions, exclusively designed for DMANET that have confirmed good scalability and limited overhead for DMD, for RD/RR, and for RLM.

Keywords: Replica Middleware, DMANET Design, Replica Monitor, J2ME, Replica DMANET Design & Implementation, Resource Delgate.

I. INTRODUCTION

To support the operations in replication strategy, we are designing and implementing the Replica Middleware architecture of DMANET for Entertainment Applications as shown in above figure 1. transparently disseminates, retrieves, and manages replicas of common interest resources among cooperating nodes in dense MANETs [2].

Basically we have divided this architecture in three macro-modules, according to the role of the entity they support and all the macro contains four major components. The four components are DMD, RD, RR and RLM.

First module discussed about the Monitor who only performs RLM operations; in particular, they decide resource replication Level as a new delegate enters the dense MANET, and counteract Level inconsistencies, by invoking new replications. To this end, they need a Shared Resource Table verifier, periodically checking actual vs. established Level. Second Module discussed about the Delegates they perform resource dissemination and retrieval, and are also partly responsible of Level maintenance; in fact, in case of exit from the dense MANET, they are in charge of notifying the Monitor, and getting free of their shared resources via neighbor uploads. The last module is DMD who work is divide in two parts Boundary identification and the Monitor election. The rest detail will be provided in next section.

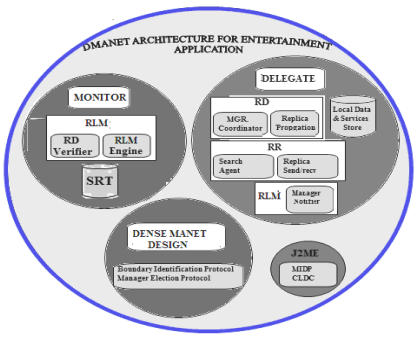

Figure 1. A DMANET Replica Middleware Strategy architecture for Entertainment areas

II. DESIGN

In design part we will discuss the replica middleware strategy diagram. We can see the replica strategy middleware architecture describe in the figure 1 has not shown the complete details about all the components: So in this section we have provided the complete working of all macro-modules (DMD, delegate and Monitor), components (RD, RR, RLM, DMD) of the middleware architecture.

A. Monitor

Basically the role of Monitor is to manage the DMANET replica Level. It includes three main components. RLM Engine, SRT Monitor & RD verifier. Figure 2 displays all the components of first Monitor activity and all its agents. Whenever a new delegate tries to enter in DMANET boundary, then he decides the level of Resource replication & then cancel lout the level of inconsistency. It has a RD verifier who checks the desired Replica Level level consistency & adds the resource into resource table.

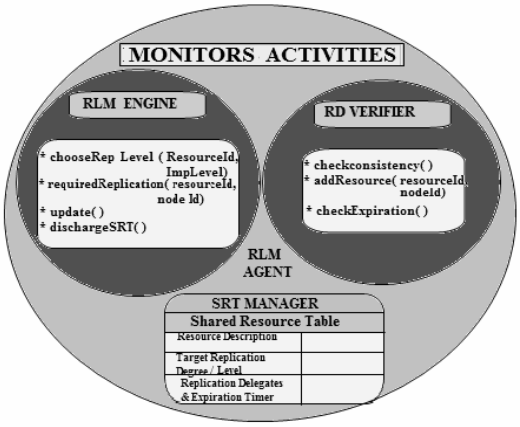

Figure 2.Modular architecture for Replica strategy Monitor

RLM Engine performs the Monitor function it determines the Target Replication Level and enter its value in the SRT. Monitor operates on SRT by updating its contents periodi-

cally by checking replication Level consistency. If Actual Replication Level of any resource is below the target Level then SRT Monitor invokes required Replica on REM Engine.

B. Delegate

Delegate comes in second Layer of Replication Middleware architecture. It includes RD and RR components. Delegate is responsible for all work of replica distribution and replica retrieval. They also have some part of RLM as a Monitor Notifier. When a node is leaving the DMANET then it is the duty of Delegate through RLM to notify the existence of the leaving Monitor node. Whenever a node leaves the DMANET then delegate relives all the resources having by the Monitor.

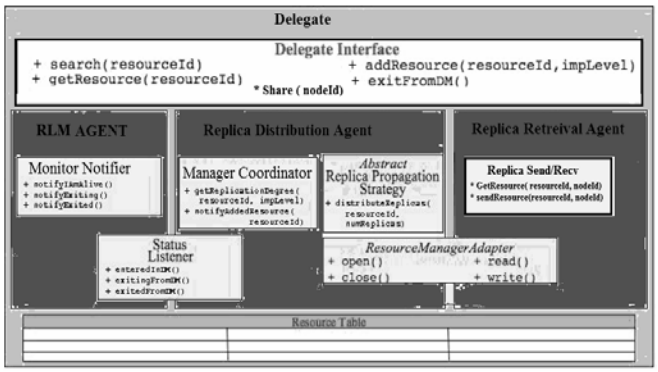

Figure 3. Modular architecture for Replica strategy Delegate

Figure 3 shows the Middleware Architecture of Delegate. This is more complex then the DMD layer. There is a Delegate interface, RLM agent replica Distribution agent, Replica retrieval agent and a Resource table. All the components of Delegates work through their agents.

Replica distribution is divided into two parts Monitor coordinator and abstract replica propagation strategy. Whenever a new node entered in DMANET the manger coordinator delivers a list of carried resource with an RDF descriptor [3], to the network Monitor to get suitable replica Level. When the proper replica Level is found or maintained. Then on the basis of this maintained Level RPS (replica Propagation Strategy) agent perform the required distribution. RPS is provided as an abstract since it is representing only the interface implemented by the actual agent (Installation takes place through factory pattern)[4]. RPS agent need to shift full resource along the distribution path.

Delegate Interface exports two methods of Replica Retrieval.

- Share *( )*: To find a node sharing the needed resource.
- getResource *( )*: To effectively command the download.

In Replica Retrieval component we have RR agent and it includes a Replica send/recv component. The method getResource and the SendResource perform Replica download and upload work. Both methods deals with Resource Monitor Adapter to extract and store the resource. If a node decide on its turn to share the downloaded resource. Method getResource interface with Monitor coordinator via addResource method.

Status Listener is only notify the entering and exiting of a node from DMANET. To perform these activities it has three functions.

- EnteredInDMANET( )
- ExitingFromDMANET( )
- ExitedFromDMANET( )

C. Dense Manet Design(DMD)

DMD is the third layer of Replica Strategy Middleware Architecture. It allows upper (RR, RD) layer to access topological information. It is the basic module for operating DMANET. It performs two main activities Boundary Identification Protocol and Monitor Election Protocol through *SetupDMANET* function.

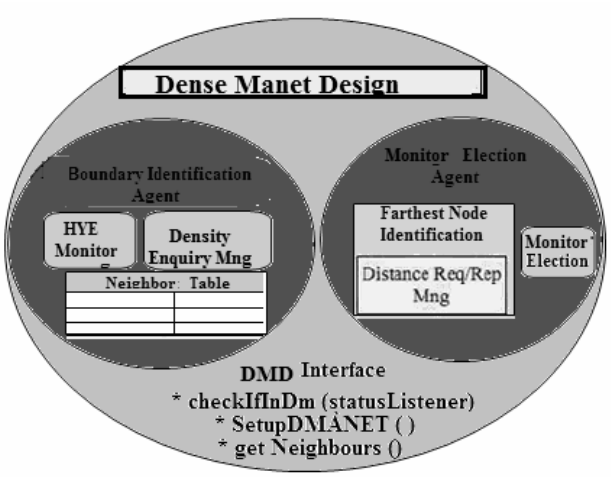

Figure 4. Modular architecture for Replica strategy DMD

When the time expired the **Density Enquiry Monitor** broadcast a discovery message and starts Monitor election which in turn start the Farthest Node Identification. When the operation end the new elected Monitor is stored in the table and the control returns.

Boundary identification agent continuously performs Monitor role maintenance operation and creates a neighbor table via Hye Monitor which is needed in RR and RD protocol.

III. IMPLEMENTATION

According to the guidelines and key requirements stated in the introduction section we have implemented our Replica Strategy Middleware Architecture on J2ME. Here we have discussed the main implementation issues encountered during the development process of our architecture. But before starting the implementation of architecture, we should first understand about J2ME and its uses in the implementation process of or work. How it will help us in ad hoc networking.

A. J2ME Plateform

The Java 2 Micro Edition (J2ME) platform gives developers breakthrough tools for building advanced wireless applications. Now, one of the leading wireless application consultants at Sun has written a step-by-step guide to successful wireless development using the J2ME SDK. Vartan Piroumian illuminates every key feature of the J2ME platform and every step of the development process: planning, architecture, design, coding, compilation, execution, debugging, and deployment. Wireless J2ME Platform Programming [5] covers all this, and more:

- Using the Mobile Information Device Profile's (MIDP) high-level and low-level APIs Building effective wireless user interfaces with the J2ME platform
- Leveraging the J2ME platform's facilities for persistent storage

- Managing, provisioning, and internationalizing wireless applications.
- Integrating complete wireless solutions: Internet portal interfaces, wireless application interfaces, and the wireless Internet environment

J2ME modular design permits to combine Designs and Profiles, the former essentially defining virtual machines and basic Java classes, the latter extending Designs with additional libraries. In this thesis work, we focus on CLDC, i.e., the Design with the lowest hardware/software requirements, and MIDP (Mobile Information Device Profile's), providing in particular UDP support, persistent storage access, and a model for the development of applications (Midlet). CLDC's Generic Connection Framework (GCF) jointly supports communication as well as I/O operations with a common set of abstractions. Essentially, GCF is an API refining a generic Connection into more specific abstractions: DatagramConnection defines packet I/O; InputConnection, and OutputConnection stream-based I/O, which is further refined in ContentConnection. GCF does not implement any protocol; MIDP (Mobile Information Device Profile's) extends GCF by implementing TCP and UDP classes. GCF abstraction can be seamlessly implemented to provide file support: FileConnection [6] permits to access files stored on device filesystem as well as removable memory cards. By exploiting MIDP's RecordStore. Differently from filesystem, RecordStore is strictly tied to the instantiating application and hardly permits resource sharing between a well-defined group of applications. RecordStore abstraction is by no means connected to GCF, but is separately defined [7].

B. Implementation Issue

This section reports some interesting issues we encountered and solved during REPLICA IN DMANET implementation. In particular, we focus on four main areas: i) Packet Dealing; ii) Message Passing; iii) Resource packetization; iv) Routing protocol interactions.

1) Packet Dealing: Every packet in REPLICA IN DMANET contains a common header including Type field, SrcAddress/DestAddress, DatagramId etc. The actual Sender and the receiver of the packets are identified through SrcAddress/DestAddress. When DatagramId is combined with the SrcAddress, then it provides a unique identifier for the packet. As soon as a packet is received, a common REPLICA IN DMANET Dispatcher is in charge of determining how it should be managed by inspecting the Type field. Different choices are possible: the Dispatcher could sequentially elaborate the packet, or it can notify the packet to a single waiting Thread, or it can activate a brand new Thread for every received packet. Ideally, to maximum parallelize the execution; the last solution would be the best; however, constrained devices implicitly limit the maximum number of active Threads, by degrading performance as more Threads are activated. Thus, we choose to differentiate packets requiring a complex (and generally blocking) elaboration, e.g., those delegating the execution of Monitor election or resource dissemination protocols, from those expecting a quick reply, e.g., shared neighbors probe in SID. The Dispatcher activates brand new Threads for the former, while it only notifies existing Threads for the latter, by placing the packet in the respective waiting queue.

2) Message Passing: Most packets are delivered with local broadcasts, e.g., hyes, neighbor probes, farthest node determination relaying. However, we found that limited broadcast (with destination "255.255.255.255") is not supported on J2ME. We collected the same experience on a number of different implementations: Palm OS and Windows Mobile versions of IBM Websphere, and Sun and IBM Wireless Toolkits. In particular, the limited broadcast destination address is not recognized as a valid argument in the Connector's open method. This problem could be solved by replacing limited broadcast with direct broadcast (with destination "X.Y.Z.255").

3) Resource Packetization: Resources are locally accessed via FileConnection GCF APIs where a filesystem is supported, via Record Store elsewhere. During upload/download phases, resources need to be carried in Datagram packets. Unfortunately, they often exceed datagram sizes; thus, they need to be split into a sequence of packets. REPLICA IN DMANET implements automatic methods to fragment resources at sender, and recompose at destination. In this case, it is important to determine the packet size allowing the best performance. We experimentally proved that, as expected, the biggest packet size supported by the communication device always leads to best performance (i.e., because this choice minimizes the communication overhead).

4) Routing Protocol communications: Even if we have not implemented any routing protocol yet, we realized that some of the operations we support would be identically repeated at the network layer. For instance, many multi-hop routing protocols exchange Hye packets to monitor local connectivity, and maintain a neighbor table. REPLICA IN DMANET repeats the same operations/data structures at an upper level. Interestingly, cross-layer design could avoid this unnecessary communication/memory waste, by allowing REPLICA IN DMANET to directly access network-layer information.

IV. EXPERIMENTAL RESULT

We have earlier stated that J2ME is designed for wireless devices, So We have tested REPLICA IN DMANET prototype in small PDA (personal digital assistants) and laptop network setups, by using J2ME. CLDC and MIDP provided by IBM J9Websphere (for PDAs) and by Sun Wireless Toolkit (for laptops). We utilized different types of PDAs (Compaq, Palm, HP) with 400MHz Intel CPU, [64MB-128MB] RAM and PalmOS or Pocket PC operating systems. As for the laptops, we ran REPLICA IN DMANET on Dell Latitudes D600, equipped with Pentium M1.4GHz, 512MB RAM and Windows XP operating system. Due to the limited number of available devices, we only aimed at evaluating the basic mechanism of REPLICA IN DMANET protocols. In particular, we instantiated a 1 to 2-hop (i.e., 2 to 3- node) network and evaluated latency during basic DMD operations. Here we present only a subset of the results we obtained (we are still working to extend our evaluation).

We measured DMD farthest node identification latency in 1 and 2-hop networks. This step of the election algorithm is highly influenced by two timers: one on the flooding forwarder determining whether the node is the farthest in its direction or not (FarthestReplyTimer), one on the current initiator to stop the farthest identification process and proclaim its INvalue (StopFarthTimer). Since we were interested only in the communication latencies, we set FarthestReplyTimer = 0 to collect replies from all the few nodes, while

we did not take into account StopFarthTimer at all, but we measured only the time needed to obtain all the replies. The first row of Table 1 shows that this value is on the order of few hundred milliseconds.

Table I. DMD Protocol Latency

Network hop diameter	*1 Hop*	*2 Hop*
Farthest Node Determination [ms]	204.5	228.1
Monitor Election [ms]	1146.2	1871

Then, we measured the election latency in the same network setup. In this case, the results are highly influenced by StopFarthTimer, which is set equal to 500ms, and by MaxConsecutiveEqualSolutions, which is set equal to 2. We observe that the election latency is lower than a couple of seconds for 1-hop as well as 2-hop networks (second row of Table 1). As expected, the difference for the two cases approximates the value StopFarthTimer meaning that for the second case one more election iteration is needed.

V. ACKNOWLEDGMENTS

I would like to thanks Dr. Vijay Lamba Associate Prof. In ECE Deptt. HCTM, Kaithal, Haryana (INDIA) for his gentle guidance and also thankful to my friends they prompt me to do research work. I also thankful to all staff members of I.T Deptt. HCTM, Kaithal who provide me help and resources for my research work.

VI. CONCLUSION

In particular, middleware typically increases service portability over different lower-layer communication protocols. In this paper work, we identified two primary challenges hampering effective remote resource access in MANET. First, mobile nodes can leave network area without any notice, disrupting availability of common interest resources. Second, lacking centralized server authorities, resources need to be distributed discovered and located. This issue is aggravated in wide-scale, sparse network, such as VANET. By applying the REPLICA IN DMANET middleware it will possible to provide the distribution of resource replicas, to improve their availability. In particular, it proposes effective distributed protocols, to locate close resources and to maintain established replication Levels in spite of possible node mobility outside the service area

VII. REFERENCES

[1]. A. Kumar, Pankaj Kumar Verma, Dr. Vijay Lamba "Concept of Middleware Services in Mobile Ad-hoc Networks" IJCA (0975 – 8887) Volume 2 – No.8, June 2010.

[2]. P. Bellavista, A. Corradi, E. Magistretti, "Lightweight Replication Middleware for data and Service Components in Dense MANETs", 1st IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), June 2005.

[3]. S. Decker, P. Mitra, and S. Melnik. Framework for the semantic web: an rdf tutorial. IEEE Internet Computing, 4(6), 2000.

[4]. E. Gamm, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publisher, 1994.

[5]. Wireless J2ME Platform Programming (http://books.internet.com/ books/0130449148)

[6]. Java Specification Requests75. http://jcp.org/en/jsr/detail?id=75.

[7]. Java Record Management System. http://developers.sun.com/techtopics/mobility/midp/articles/databaserms/.