SDP-Implementation in Opnet® v.1

MANETs' Service Replication and Load Balancing

- Release Notes -User Manual –Programmer Guide

FRIEDRICH-SCHILLER-UNIVERSITY OF JENA

2010

Acknowledgements

Many thanks are due to Alexander Klein from University of Wurzburg for his early support.

Table of Contents

SDP-Implementation in Opnet® v.1

- Release Notes -User Manual –Programmer Guide

Introduction

Enabling service oriented application on MANETs represents a common motivation for many research efforts during the last few years. MANET introduces many challenges. The main challenge for a service application there is the availability. There is no guarantee that any of the network participants will be able to access the service at any time of the network operation time. SDP (the **S**ervice **D**istribution **P**rotocol for MANETS) [1] has been introduced as an approach with many innovative mechanisms to increase the mobile service availability. It is mainly based on a new technique in servicer replication called "replication-hibernation-restoration". SDP can achieve a very efficient service distribution of the generated replicas (Service copies) over the MANET. In this report, the implementation of SDP in the Opnet® simulator is being described. More details about how SDP is functioning, its performance and comparisons can be found in [2].

Contents:

This documentation consists of 3 parts: I. Release Notes, II. User Manual and III. Programmer guide. In the first part a short description about the delivered software features is presented. Part II shows an easy user manual which introduces how to use the delivered software under the case tool of Opnet® Modeler® with Wireless. The last part show a detailed programmer guide to make it easy to implement your special extensions for the SDP-implementation in Opnet®.

Who is supposed to read this documentation? which part?

Mainly, apart from the release notes part, two categories of users may read this documentation. The purpose of reading such a documentation is varying between evaluating the SDP protocol under different specifications and settings and modifying SDP and its mechanisms. We guess that the first category, which we name "user category", is associated to the first purpose. If you are a user, then it is better, if you have some knowledge about:

- Opnet® Modeler® Wireless.
- Simulation Concepts.
- MANETs:
  - Routing protocols.
  - Data Transpiration protocols.
  - MANET addressing protocols.
- SOA Applications.

On the other hand, we call the second category the "programmer category" which is associated to the second purpose. Besides the knowledge of the required mentioned points in the previous category, the programmer should have knowledge about the following:

- C++.
- Automata.
- Modeling concepts in Opnet® Modeler®.
- Network architectures.
- Queuing techniques.

The first user category will easily read and understand the first two parts of this documentation. While the programmer category will be able to understand the whole documentation.

What to download? How to Install?

What to download?

Please find and download the SDP compressed model file in the Opnet contributed models at "https://enterprise1.opnet.com/tsts/support/cont_models.shtml".

The model compressed file is contacting the following components:

- A channel matching pipeline:
  - pipeline stage: dlvr chanmatch75m.ps.c
- The mobile node:
  - Process model: dlvr SerExe SDP Application Process Model.pr.m
  - Node architecture: dlvr ServExe SDP node arch.nd.m
- The repository node
  - Process model:dlvr SerEXE SDP global_db process model.pr.m
  - Node:dlvr SerEXE SDP global_db.nd.m
- A sample project:
  - Project1 SerEXE SDP 50 node.project.rar

How to install?

- Install Microsoft® Visual Studio® with VisualC++® 2005 or higher.
- Get a suitable Opnet® Modeler® license and install Opnet® Modeler® Wireless v.15 is preferred.
- Associate the C++ compiler to Opnet® Modeler®. Please have a look on www.opnet.com/support, if you have a question.
- Download the previously mentioned files in the "What to download?" section.
- Remove the "dlvr " token part from the naming of the downloaded files.
- Make a specified folder "SDP-Home" to host the downloaded files.
- Add the "SDP-Home" folder to the Opnet® Modeler® model directory by:

- From the project menu-bar, select: File→Manage model File→Add model directory.
- From the project menu-bar, select: File→Manage model File→Refresh model directories.

Start evaluating the SDP-implementation in Opnet®.

Part I: Release Notes:

In this part a short description about the requirements, abilities and limitations of the delivered software (SDP-Implementation in Opnet®) is given. This version is our initial release version, so the delivered release notes do not refer to any previous versions.

It requires:

- Software:
  - Opnet Modeler® with Wireless version 15 and its default delivered model libraries.
  - Microsoft® Windows XP® or any compatible version.
  - Microsoft® VisualC++® compiler with Visual studio® suite 2005 or higher.

- Hardware:
  - Suggested machines by:
    - Opnet®, for Opnet® Modeler®.
    - Microsoft®, for the operating system and Microsoft® Visual Studio®.

Abilities:

The provided SDP-Implementation in Opnet® present an Opnet® module which can perform all the functionalities of service distribution and the related mechanism with a user friendly Opnet® interface.

It can:

- Deploy either one or different services in a MANET
- Model a variant Gross Interest [2] for a specified service or set of services based on a very fixable modeling approach .
- Associate the client Gross Interests to the offered services.
- Distribute a set of initial replicas (the original service and a set of initial copies) of each service on the mobile nodes automatically.
- An easy and clear SDP-modes and specifications acquisition.
- Perform the service replication-hibernation-restoration mechanisms.
- Expressive collecting of different statistics for measuring the performance of SDP such as: Service availability, SDP availability, execution success ratio, replica allocation correctness, replication cost, typical network partition, …etc.

Limitations:

- Regarding Opnet® Modeler with Wireless® version15 release notes, the delivered software inherits the limitations described there. For example, when deploying a set of mobile nodes in a MANET (apart from their types), Opnet® can only operate only one routing protocol for the network.
- The default version of SDP-Implementation in Opnet® supports at most 10 running simultaneous different original initial services (apart from their replicas or initial sites). If you have a simulation specification which needs more number of initial services, please contact us.

- Network operation time is supposed not to exceed 24 hours (one day). If you have a longer-time scenario, please contact us to supply a solution for that.

Part II: User Manual:

Basic Concepts

SOA

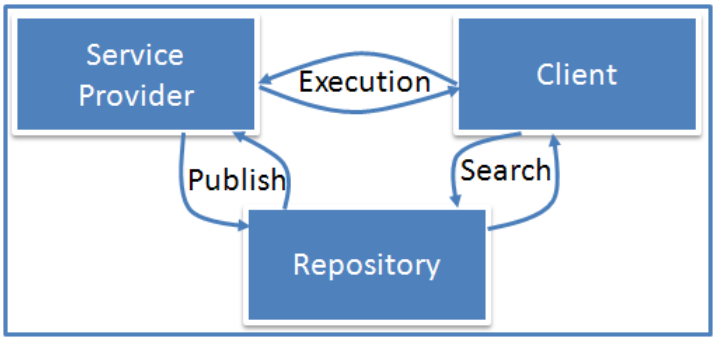

Figure 1: A general SOA architecture

SDP assumes that the services are organized and handled under the standards of SOA (**S**ervice **O**riented **A**rchitecture). Many proposals for achieving SOAs for MANETs are introduced and discussed as in [3]. Figure 1 shows the main three parties in any SOA. A Client is supposed to find out its required service form a centralized service repository through the search phase. The repository contains a set of different offers for many service which are provided by the different service providers.

For different reasons, in a MANET, the three parties are required to be implemented together on the mobile node. The mobile node may host a service or one of its replicas, so it is considered as a "Service Provider". The same mobile host may represent a client for the hosted service, so it should considered as a "Client". Replicating the repository upon many mobile nodes is highly required in the MANET's unstable environment. So the three parties are required to be implemented in the same mobile node.

Replicable services? Which services? Composite services?

The delivered software deals with atomic services which supply specific functions and requires occupying some resources on the hosting mobile nodes. Moreover, the service proposes connected sessions for each of the requesting clients. The session holds the required information which is being processed till it is sent to the client as a response.

The service replication does not only require the executable files of the service to be replicated but also the required resources which enable this service to be functioning. A "meta" file which contains the service status and the required instructions for consistency is assumed to be included in the replication process.

Some services are not replicable by definition, for example, imagine a print out service. Here, service replication has no meaning however we do not mean installing a new printer somewhere in the network. On the other hand, we can imagine a set of replicable service like gateway service,

security-key distribution services, DNS and DHCP services which can be replicated over the network participants.

About composite services, this version does not include the module of composite service execution over MANET based on SDP. This proposed module is included on an ongoing work and will be released sooner.

Service-Client Requesting Behavior (Requesting Model)

Each client has a behavior of generating the requests regarding a specific service. The number and frequency of the service requests are varying during the client lifetime.

Initially, all mobile nodes seek the initial (original) service provider node. The request frequency is the main output of the requesting model, the number of requests per a certain time interval dominates the request behavior of the clients toward a specified service. Of course, many other parameters could replace or be combined with the calling frequency like the call length. For simplicity, the proposed requesting model is based on the requesting frequency (calling rate).

Figure 2 shows how SDP models the requesting behavior –requesting model- from a client regarding a specified service, more about this model is shown in [2].

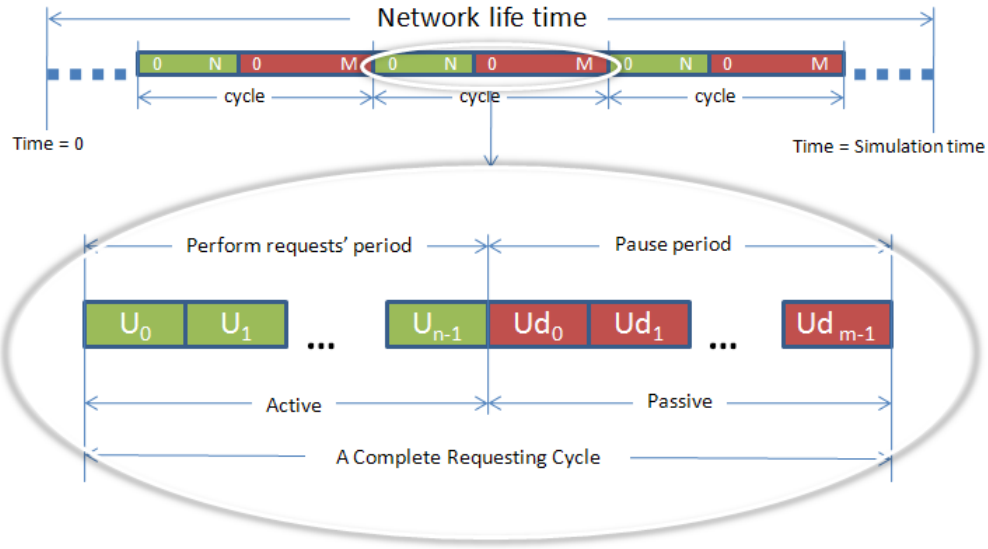

Figure 2: Requesting Model of SDP

As we can see in Figure 2, the network life time is divided into cycle. Each cycle is consist of two time-length variant periods; the requests' and pause periods. In the request period the mobile client is active and perform the service requests under the influence of the calling rate while it stops of requesting the service during the pause period. M and N specify the number of allowed U and Ud units. Specifying the values and their distributions of M, N, U and Ud conform the requesting model of a client regarding a specified service.

We claim that by using such a model, we can model whatever requesting behavior. Please check [2].

Service Gross Interest

The requesting behavior of a group of clients regarding a specified service generates a common interest for this service which is the gross interest. The service replication-hibernation-restoration processes are dependent on the gross interest. [2] presents a way to quantify the gross interest and introduce different (rich/poor) gross interests based on some criteria.

Assumptions

This version of SDP-Implementation in Opnet® introduce the set of SDP core functions. The motivation behind the work is to evaluate the SDP in powerful simulation environment. Therefore, a set of assumptions are drawn here.

Repository

Services' offers are published by the provider in the available repositories in the network. A powerful protocol for service publishing and consistent offer updating is assumed to be found.

Service Discovery

Clients are supposed to search in the available repositories to find their desired offers. So, a service discovery protocol is assumed to be used by the clients.

Service Choreography

If there exists a client who is interested in a specified service, the client knows well the required inputs, outputs and their formats of this service.

Service Ranking

The different replicas of a specified service need to be ranked based on many criteria in the service repository. Request price, delay and trust represent a set of many criteria to be taken into consideration in any service ranking process. Available ranking mechanism for the different replicas is assumed to be found. All the service instances will be ranked based on some "general requirement" index.

Next ongoing versions will contain some features of the previously assumed to be found.

Minimum Components to run a SDP simulation

After Setup of Opnet® project including the MANET library. You need to include the following components in the project to be able to run a SDP simulation safely and correctly.

Radio Range - Pipeline Stage

Specifying the radio transmission range of a mobile node depends on many varying factors like the transmission power. Practically, It is hard to draw a real fixed transmission range of a wireless

node. But, in simulation it is wide accepted to determine a certain fixed transmission range for the network participants.

In Opnet® Modeler® with Wireless, it is not easy to determine such a criterion. We do not consider this issue as a normal user issue. It requires some programming abilities to be modified from the delivered C pipeline file. So, for a normal user we can deliver upon your request the required custom C pipeline file and the required procedures. For the programmer who read this part more details are available in the programmer guide about that.

Important node:- The transmission range at this stage is supposed to be fixed to 75 meters (if changed, the C pipeline stage should be changed accordingly).

Wireless Domain

Specifies the characteristics of the area that you would like to deploy the network. A terrain profiles may be combined.

Mobility Config Module

Specifies the mobility model and its characteristics. Custom trajectory motion paths can be also defined. At the end of specifying your simulation environment, do not forget to associate the mobility config to your mobile nodes.

Repository Node

Only one module is required. Make sure that there is at least one and only one node of the repository module in your simulation. This node is not a real mobile node. It contributes as a virtual node in your simulation.

It represents the repository module which is supposed to be found over the mobile nodes. It performs the required functionalities of publishing, holding, updating the service/replicas' offers. It performs the required responses for the offer queries posed by the client. The statistics of SDP performance are processed, updated and stored in this module.

Moreover, it enables the user to set the specifications of the original service required to be deployed in the network.

Where to find?

From the object palette, search by the name for "SerEXE SDP global_db (Fixed node)". Add just one instance as the only one repository node required to be available in the project.

How to specify?

Right click on the repository node and choose edit where you get the menu in Figure 3. The shown menu of attributes in Figure 3 contains a set of unused attributes (they are proposed-reserved for later usage in next versions) like (Network size and delta Time()). All time values in this menu are in seconds.

The most important settings here are Radio range in meters(here, you should specify the radio range of your mobile nodes in order to enable the repository node to compute the general performance statistics) and the "Service Distribution Analysis each" in seconds (it is the time interval of collecting the statistics about the current service distribution over the network).

The "Start Time" attribute represents in seconds the time when the repository start collecting the statistics about the service distribution. The "Stop Time" attribute set in seconds the maximum allowed time for the repository to collect the statistics about the service distribution (it can be longer than the simulation time but not shorter).

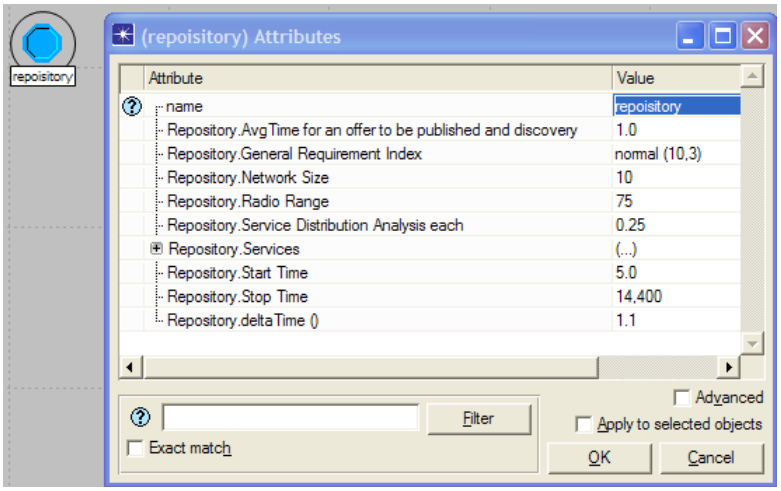

Figure 3: A repository node and its specifications

By extending the attribute of "Services", you get the menu in Figure 4 which enables you to specify your original services in the network. For instance, in Figure 4, we have two different services; "Serv123" and "ServABC". Initially, the two services will be deployed on a mobile node.

In order to deploy a service or one of its replica at any mobile node, it should facilitate the required disk and memory spaces to be run. The life time of a request-session (a session holds the information of the query till the processing ends at the provider side and a response is generated) is an output of the given distributions.

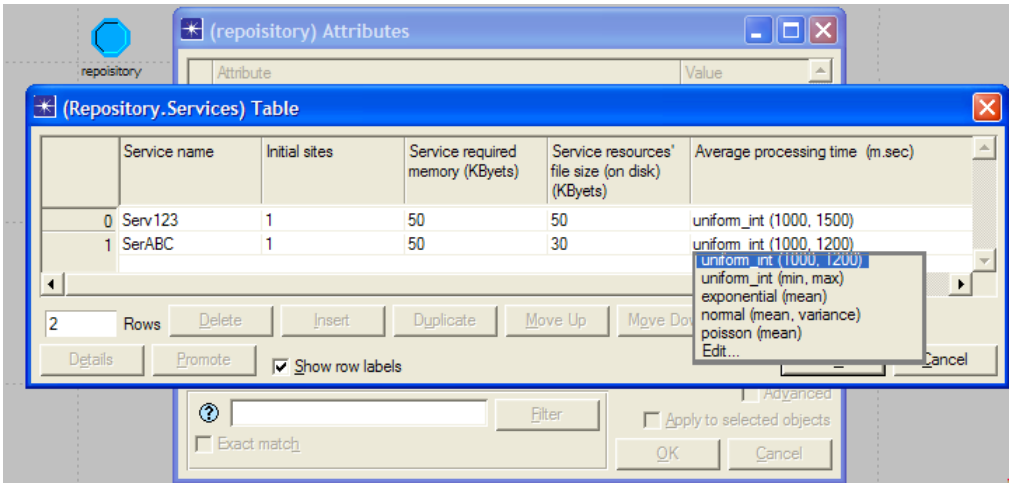

Figure 4: Specifications of the different services at the repository node

Wireless Mobile Node

The mobile node presents a network participant which may be a client or provider for the service. This node is built on the basis of an Opnet® delivered library of the wireless workstation of "manet_station_adv (Mobile node)" module. The modifications there are introduced in the programmer guide in this documentation.

Where to find?

From the object palette, search by the name for "ServExe SDP node arch (Mobile node)". Add as many instances as you want inside your wireless domain. Associate the mobility to the deployed mobile nodes.

How to specify?

Figure 5 shows the attribute menu of the mobile node. The prefix "application." refers to some required settings for the hosted application(s) by this node which is SDP-based.

1. Service Gross Interest (Requesting Model) menu:

   As in Figure 6, multiple service interests are allowed to be entered. Each of these interests (rows) indicates the requesting behavior of the application regarding a specified service.

   The interest(s) and the original available service will be associated with respect to their order. If the number of interests is greater than the number of the original available services, then the last exceeded interest(s) will be ignored. But, if the number of interests is less than the number of the original available services, then the last interest specifications will be copied to satisfy the service definitions.

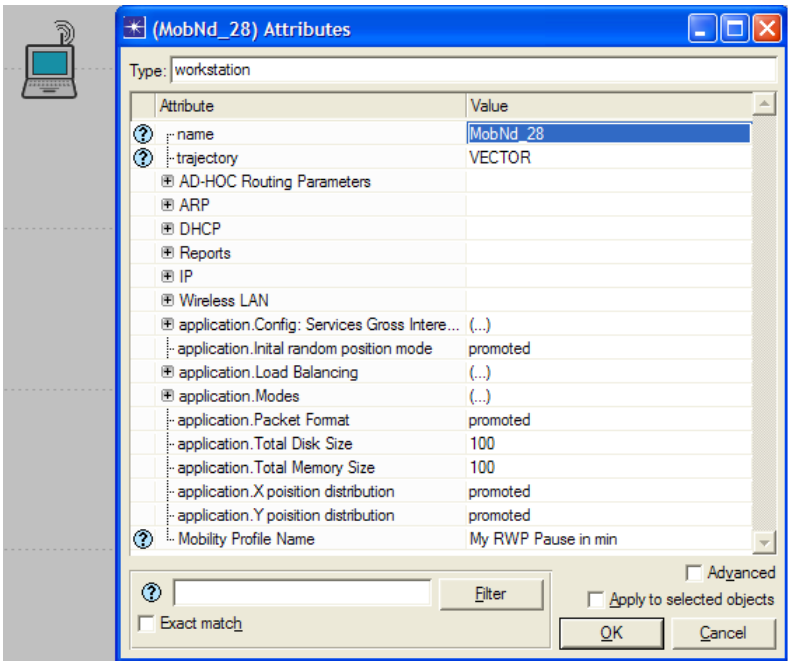

Figure 5: Specifications of the different application attributes at a mobile node

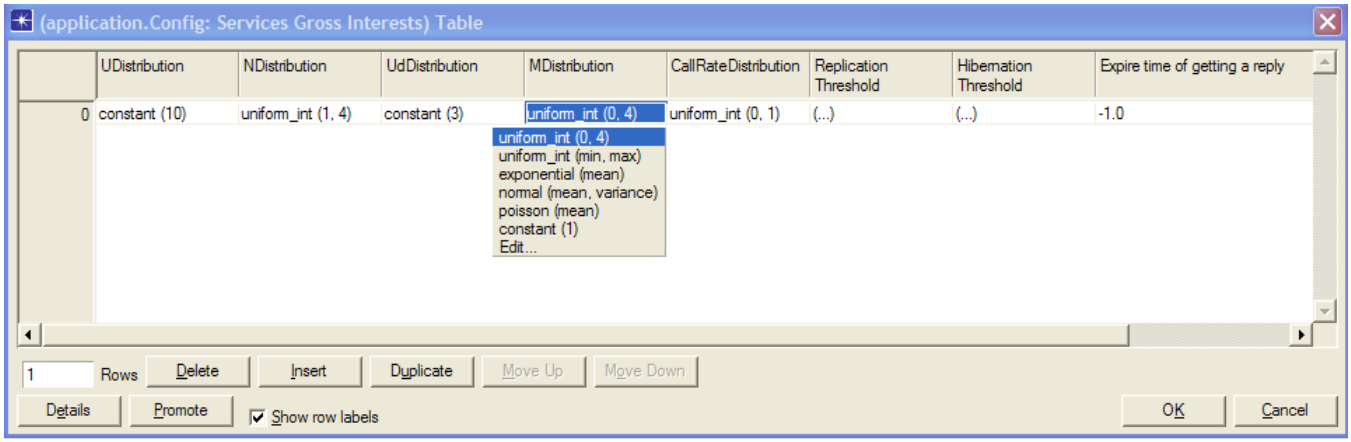

Figure 6: Specifications of the different service interests at a mobile node

2. Initial random position modes:

   This attribute is proposed-reserved for ongoing use in next versions.

3. Load Balancing

   As in Figure 7, SDP can perform a load balanced service distribution based on either the number of connected sessions at the provider side or the response time of certain request at the client side. Both of "Number of sessions" and "Response time" attribute may have -1 as a value to represent the infinity value.

Important note:- If any of the load balancing modes are turned on, then automatically the mode of "Wakeup Neighbors" of SDP [4] is turned on, too.

In this version the functioning "Load Balancing" is the "Mode: Number of sessions". The other one is proposed-reserved for ongoing use in next versions.

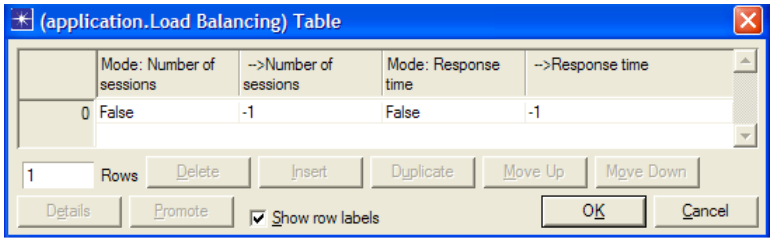

Figure 7: Specifications of the Load Balancing mode at a mobile node

4. Modes

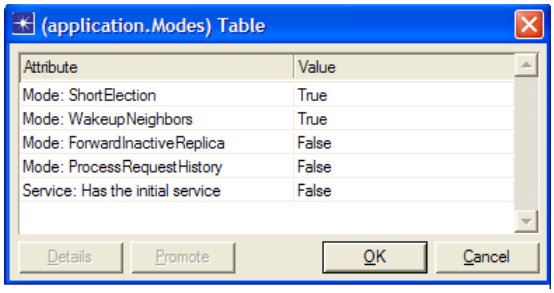

Figure 8: Specifications of SDP modes at a mobile node

In this attribute menu, shown in Figure 8, the service selection modes [4] can be specified here.

5. Packet Format

   Opnet® delivered attribute. Not suggested to be used.

6. Total Disk Size and Total Memory Size:

   As in Figure 5, these attributes specify the offered memory and disk space at the mobile node in KBytes.

7. X Position Distribution and Y Position Distribution

   These attributes are proposed-reserved for ongoing use in next versions.

Selecting the required Statistics

As mentioned before, the main statistics for evaluating the SDP performance are stored in the repository model. You may select them by:

1. Right click on the "repository" node.

2. Choose “Choose individual DES Statistics”.
3. Navigate in the “Module Statistics” to “Repository”
4. Select your desired matrix from the delivered different categories of SDP statistics.

More about the meanings of the delivered performance statistics are mentioned in [2].

If you have more than one original service defined in your experiment, then the statistic will be show with respect of the order of these services’ definitions as mentioned in Figure 4. The service names will be replaced by an index. For example, the “Success Ratio [0]” means the success ratio of the first mentioned service, regarding in Figure 4, and Success Ratio [0]” means the success ratio of the second mentioned service and so on.

Notes on the results

Regarding the following SDP performance criteria, please be aware about the fact that these statistics are accumulators. So, their meaningful readings will be the last value not the average

- SDP-Availability.
- SDP-Success Ratio.

Do not Forget

You should find out a good set of specifications of the routing protocol which suits your scenario, otherwise you may get some meaningless results. For example, as in the supplied sample project, if you use the AODV routing protocol, you should specify correctly some parameters like the network diameter.

Part III: Programmer Guide

SDP Packets' standard

Getting a look on how the SDP application deals with the packet sending process, message structure and interrupts will be useful in understanding of the SDP packets' standard.

Sending Packet Mechanisms

SDP-Implementation in Opnet® employs a packet based information exchange simulation. There are two categories of messaging mechanisms have been used in the delivered software.

- Send-to-inside category.

  This category contains the mechanisms that enable the mobile node to send its packet stream to itself or exchanging the queries and responses with the global repository node. This mechanism depends on the Opnet® packet sending function of "op_pk_deliver(..args..)" and its derivatives like "op_pk_deliver_forced (..args..)". Review Opnet® Modeler® v.15 documentation. The current implementation considers that the global repository is supposed to be found in each mobile node, so exchanging packets here is done with the "inside" category.

- Send-to-outside category.

  The mechanisms in this category facilitate exchanging information, mainly the service requests and responses, between the network participants. Packet streams are supposed not only to be handled inside the sending participants but also through the network till reaching the destination participants. So, the function of "op_pk_send(..args..)" and its derivatives like "op_pk_sen_forced (..args..)" represent the core functionalities of these mechanisms. Review Opnet® Modeler® v.15 documentation.

SDP show an architecture independent concepts [1] which mean that it can be applied as an application processor in any node architecture of the supplied Opnet® libraries for MANET. But, it is very important to pay attention to the applied node addressing protocol in the network. The current version presents a solution which has been merged with the delivered Opnet® IPV4 and IPV6. In the SDP supplied node architecture, which has been shown in the User-Manual part of this documentation, the SDP application uses directly the auto addressing attribute of the mobile node in both mentioned packet sending categories.

Moreover, the current implementation employ the original coupling mechanism of sending packets between the application layer "traf_src" processor and the "udp" processor in the Opnet® delivered library of the wireless workstation of "manet_station_adv (Mobile node)". So, please plane for modifications in the "send-to-outside" packet sending mechanisms if you are planning to use it with another node architectures.

Message Structure

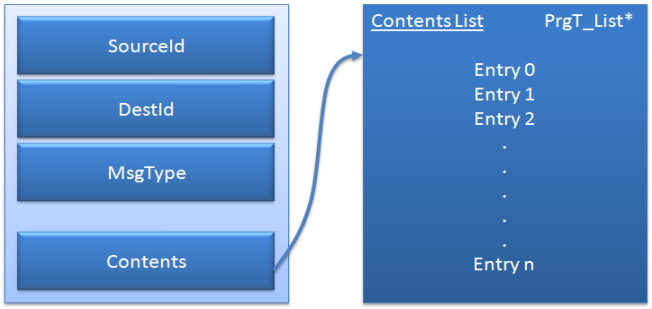

Figure 9: The general message structure of SDP.

Figure 9 shows the common structure of SDP messages which are always attached with the sent packets. The structure encapsulates the source id "SourceId", destination id "DestId", and a unique message type identifier "MsgType" with a list of a message record "Contents" which holds the different parameters regarding each message type. The "Contents" list is defined from the generic Opnet® list class "PrgT_List" which is able to hold different data types for the different entries.

Message Types and Interrupts

Based on the value of the "MsgType" identifier, Table 1 shows and describes the different message types and the involved sending and receiving parties of each of them.

MsgType	Interrupt	Sent by	Received at	Description
1	SDP_Init_Service	Repository	Host	At initialization time, the repository initiates service on the different hosts.
2	SDP_Request	Host	Host	Service query.
3	SDP_Reponse	Host	Host	Service response.
4	SDP_Request_Update	Host	Repository	Reports a(n) (un)succeeded and responded request at the repository.
5	SDP_Offer_Update	Host	Repository	Service/replica new status update control msg
6	SDP_List_Act_Rchble	Host	Repository	Lists the most interesting active replica in the local partition.
7	SDP_List_Act_Rchble_Rspns	Repository	Host	
8	SDP_List_Hib_Rchble	Host	Repository	Lists the most interesting hibernated replica in the local partition.
9	SDP_List_Hib_Rchble_Rspns	Repository	Host	
10	SDP_List_Original_Services	Host	Repository	Lists the original services Ids/names at network initialization.
11	SDP_List_Original_Services _Rspns	Repository	Host	
12	SDP_Resources_Qry	Host	Repository	A mobile asks for the required resources and "reqIndex" of a specified replica.
13	SDP_Resources_Rspns	Repository	Host	
14	SDP_Sataus_Qry	Host	Repository	A mobile node asks for the current status of a specified replica.

15	SDP_Sataus_Rspns	Repository	Host	
16	SDP_Wakeup_Rwquest	Host	Host	A “Wakeup” request from a client to a provider with an inactive replica.
17	SDP_Ask_For_Replica	Host	Host	A request to host a replica
18	SDP_Replica_Forward	Host	Host	A packet containing the replica
19	SDP_Find_next_provider	Host	Repository	A request for the Repository to find the next suitable provider.
20	SDP_Find_next_provider _Rspns	Repository	Host	A response to a host with the next valid provider in the local partition.

Table 1: Different SDP message types and descriptions.

MsgType	Interrupt	Contents’ list (parameters are separated by "|")
1	SDP_Init_Service	|SerID|Name|RepID|ReqSizeDisk| ReqSizeMemory|ProcTime|ReqIndex|
2	SDP_Request	|ClientID|SerID|RepID|Time of request| +|for_quality_reasons?(T/F)|
3	SDP_Reponse	|ProviderID|SerID|RepID|Time of request|
4	SDP_Request_Update	|ClientID|SerID|RepID|HostID|Suceeded?|
5	SDP_Offer_Update	|SerID|RepID|ProviderID|NewStatus|reqIndex| +|quality_reason (T/F)|
6	SDP_List_Act_Rchble	|ClientID|ServiceID|RequestNo|
7	SDP_List_Act_Rchble_Rspns	|ClientID|ServiceID|ReplicaID|ProviderID|PartitionNo| +|RequestNo|
8	SDP_List_Hib_Rchble	|ClientID|ServiceID|RequestNo|
9	SDP_List_Hib_Rchble_Rspns	|ClientID|ServiceID|ReplicaID|ProviderID|PartitionNo |RequestNo|
10	SDP_List_Original_Services	|ClientID|
11	SDP_List_Original_Services _Rspns	|ClientID|N:No.Services|Ser1ID|Ser1Name|…… +|SerN-11ID|SerN-11Name|
12	SDP_Resources_Qry	|ClientID|ServiceID|ReplicaID|ProviderID|RequestNo|
13	SDP_Resources_Rspns	|ClientID|ServiceID|ReplicaID|ProviderID|SizeDisk| +|SizeMemory|ProcTime|ReqIndex|RequestNo|
14	SDP_Sataus_Qry	|ClientID|ServiceID|ReplicaID|ProviderID|RequestNo|
15	SDP_Sataus_Rspns	|ClientID|ServiceID|ReplicaID|ProviderID|Status| +|RequestNo|
16	SDP_Wakeup_Rwquest	|ClientID|ServiceID|ReplicaID|Wakeup for load balance|
17	SDP_Ask_For_Replica	|ClientID|ServiceID|ReplicaID|
18	SDP_Replica_Forward	|ProviderID|ServiceID|NewReplicaID|ServiceFile|
19	SDP_Find_next_provider	|ClientID|ServiceID|LastProviderID|statusOfNextProvider|
20	SDP_Find_next_provider_Rspns	ServiceID|NextProviderID|status| NextProviderID = -1, if there is no next provider

Table 2: the Contents’ list structure w.r.t. the message type and interrupt.

Table 2 presents the Contents’ list structure with respect to the message type or interrupt. Note that the abbreviation used here are as followed:

SerID = service id, Name = service name, RepID = replica id, ReqSizeDisk= required size on disk for a service to be hosted (excutable and configuration files), ReqSizeMemory = required memory size for a service to be run, ReqIndex = service requirements' index value review [1], Ser1..N = Service id, and status= status of the service provider (active/iactive), and (T/F) indicates a Boolean value.

SDP Packets

The current version uses unformatted Opnet ® style of packets which can satisfy easily the numerous number of message types and their related message structures, review Opnet® Modeler® documentations. SDP deals only with the packet streams at the application level. As mentioned before, SDP is supposed to run only on the client /provider application layers.

Repository Node

The node architecture of the repository is simply represented as a processor. As motioned, this processor is supposed to be implemented in the mobile nodes. For simplicity, as we assumed, the repository is implemented independently. Considerations about more sophisticated implementations of this processor are issued for next versions.

Repository architecture is shown in file "SerEXE SDP global_db.nd"

Process Model

The presented process model of the repository node processor in Figure 9, see file "SerEXE SDP global_db process.pr", shows the states of:

- init: which is a state for initialization of the repository node. In this state, the following procedures take place:
  1. Reading the model attributes.
  2. Initialization for the statistics handlers.
  3. Scheduling for the interrupts of the "test" state, where the network is examined and the statistics are collected.
  4. Specifying an one meter altitude for are the mobile nodes. Actually, this step is required to enable the MANET participants to achieve a line of sight communication among them.
  5. Announcing the repository ID for the network participants.

  Two transitions are available here from the "init" state. The "default" transition and the "SART" transition. Upon the condition of "START" , the process control should move to the "deploy" state. Reviewing the source code in the entry part of the "init" state can give more details about that.

- deploy: which is a state for preparing for the deploying process of the original services on the remote providers. In this state, the following interrupt is directly scheduled to trigger the service deploying process. The deploying process is done while the transition from the "deploy" state to the "idle" state takes place.

Deploying of the original service is done in order based on the available resources at the hosting mobile node. A greedy deployment algorithm is proposed there but not included in this version. So, if your scenario is resource-restrictive, you may add your own original service deployment routine at the function implementation of "initiate_original_services_on_hosts(PrgT_List* Ser_metadata)" at the process function block.

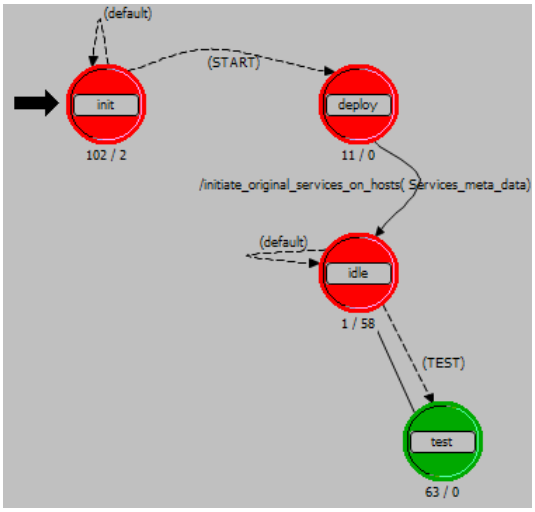

Figure 10: The repository process model.

- idle: this state has two main functions. The first important function is hosting the repository node listener in the exit part of the state. More details about the listener implementation are given sooner. The second important function is to trigger the state of "test" upon realizing the transition condition of "TEST". The required interrupts are set in the "init" state.
- test: the statistics are performed and collected here based on the given model attribute of "Service Distribution Analysis each:", review the "Manual" part of this documentation.

Repository Listener implementation

The following code portion, shown in Figure 11, presents the listener implementation of the repository node. In the first rectangle, the listener checks the inwards packet stream of the repository processor and gets the received packet if any. In the second rectangle, the listener extracts the message record of the arrived packet. In the last rectangle, based on the message type, a suitable routine will be triggered to meet the matched message interrupt then the message will be handled. Finally, after handling the message, it will be destroyed and its recourses will be released.

Statistics and Readings

As mentioned in the release notes of SDP-Implementation in Opnet®, the current number of supported original services are ten. In order to increase this number, the programmer is asked to

modify the constant of "MaxNr_Original_services" at the header block of the "SerEXE SDP global_db process.pr".

```
if (op_intrpt_type()==OPC_INTRPT_STRM)
{
    Packet* rcv_pkt_ptr = OPC_NIL;
    rcv_pkt_ptr = op_pk_get (op_intrpt_strm () );
    if (rcv_pkt_ptr != OPC_NIL)
    {
        Message_record* tmp = new Message_record();
        op_pk_fd_get (rcv_pkt_ptr, 0, &tmp);
        switch  (tmp->MsgType)
            {
                case    SDP_Request_Update  :
                    //Service_Request_Report();
                    break;
                case    5   :   //SDP_offer_update
                    Update_Service_Offer(tmp);

                    break;
                case    SDP_List_Act_Rchble :
                    List_Act_HIB_Rchble(true, tmp);
                    break;
                case    SDP_List_Hib_Rchble :
                    List_Act_HIB_Rchble(false, tmp);
                    break;
                case    SDP_List_Original_Services:
                    List_originals(tmp);
                    break;
                case    SDP_Resources_Qry   :
                    //Resources_Query();
                    break;
                case    SDP_Sataus_Qry  :
                    //Status_Query();
                    break;
                case    SDP_Find_next_provider :
                    SDP_Find_next_provider_fn(tmp);
                    break;
                default:
                    break;
            }

        delete tmp;
        op_pk_destroy (rcv_pkt_ptr);

    }
}

intrpt_code = op_intrpt_code();
```

Figure 11: The listener implementation of the repository node.

SDP Mobile Node

In the file "ServExe SDP node arch.nd", the node architecture of a mobile node is implemented (to be easily included in any SDP project). Figure 12 shows that architecture. The important note here to be taken into considerations is that the SDP application processor is the only processor involved by SDP processes. This architecture is the same architecture of the Opnet® Modeler® library module of manet_station_adv (Mobile node)". The "application" processor replaced the "traf_src" processor. Review Opnet® Modeler® documentations for more details about this architecture.

SDP-Application-Fitting in the Node Architecture

Although, SDP is an architecture-independent protocol, the SDP-Implementation required some installments to be fitted in the node architecture. These installments can be concluded in the following:

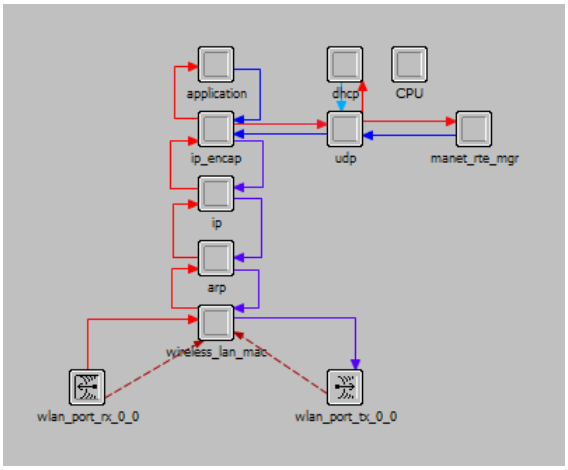

Figure 12: The SDP mobile node architecture.

- SDP uses the auto addressing mode of the mobile nodes.
- SDP uses the transportation standards in the architecture. Please review the source code of the function implementation "send_to_outside (int dest_node_id, Packet* pkt)" at the function block of the application process model at the file of "SDPrun Application.pr".

Process model

The process model of SDP application processor is presented in Figure 13. The states are mentioned as follows:

- init: in this state, some important flags are set such as "all_events_scheduled" state variable. Both of the node "Client" and "Provider" databases (defined from class PrgT_List) are initiated with some other important lists. Both of client and provider database entries are defined in the structures of "Client_db_record_structure" and "service" respectively in the header block of the process model.
- wait, wait_0, wait1: these state organize the node address registration process for all of the network mobile nodes.
- discover: in this state, at the state enter part, the next transistion to the "wait_2 SDP_init" state is scheduled. At the exit part of the state, The IP module is discovered and registered for the current mobile node.
- wait_2 SDP_init: the function of "init_SDP()", review the process function block, is triggered to perform:
    1. Reading the model attributes, review the "User-Manual" part of this documentation.
    2. Associating the local gross interest defined in the model attribute with the original service meta-definitions at the repository node.
    3. Starting scheduling all of the client database update, service replication and request generation interrupts.
    4. Scheduling the first hibernation check interrupt.

5. Triggering the next transition to move the process control to the “dispatch” state.

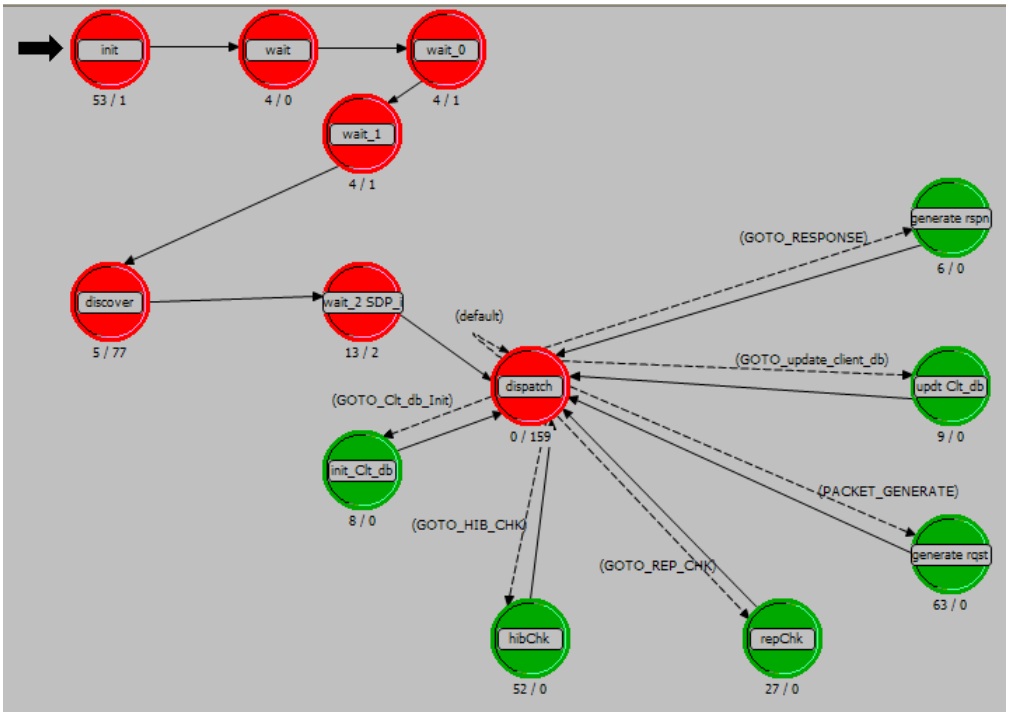

Figure 13: The process model of a SDP application.

- dispatch: on the same fashion of the repository node listener, the dispatcher states presents a listener handler stage for the inwards packet streams and arrived SDP messages. Based upon the extracted message interpret, the message is handled.
- init_Clt_db: the client database list is initiated and packed with the target service providers depending on the relevant defined gross interest. An only one transition will activate the state at the beginning of the run time.
- hibChk and repChk states: are activated based on the specifications of the hibernation and replication thresholds taken into consideration the service selection modes.
- generate_rqst: activated when the time of a request for a specific service is now.
- update_Clt_db : activated periodically (before a request generation) to keep the client database list updated about the most interesting services or replicas which are available and reached.
- generate_rspns: for a provider mobile node, the service requests are received regarding a specific offered service. The provider prepares a response for the arrived service request. To achieve the required queries in the service request, a time interval-request-session is required to hold the response information till it is sent. This state is called when one of the current request sessions has a time out and the response is ready to be sent.

Interactions among the Mobile nodes and the Repository

Service initiation process

The meta information about the given original services are defined as shown in the repository's attributes. Each of the defined services has a number of initial sites. At the beginning of the simulation the repository node performs a test at each candidate mobile node to host a service (or one of its initial copies). Based on the required service recourses, the repository determines if the mobile node will receive a service copy or not. Listing 1 shows a pseudo code for the logic behind the service initiation deployment.

```
For  all Service: Sx
        For all of Sx initial sites: Ix
              For All Mobile Nodes: Nx
                    If  ( the resources required for Ix are available on Nx)
                    then
                          Deploy Sx on Nx as an intial site
                          Break
```

Listing 1: Pseudo code of the initial deployment of services.

Figure 14 show how the repository interacts with a mobile node during the service deploying process .*

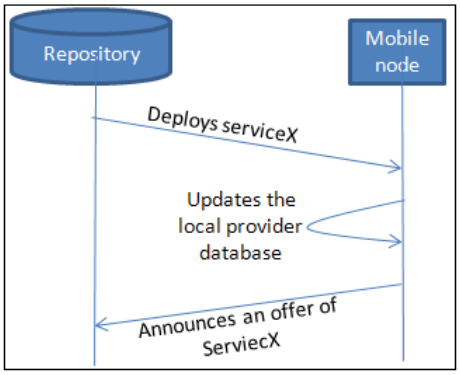

Figure 14: Interactions between the repository and a node during the service initial deployment.

Service request-response process

As mentioned in Figure 15, the service request-response process is taken place over two provider and client mobile nodes. Of course, the provider and client is supposed to be connected or at least there is a route to the client node. The shown communication lines in Figure 14 refers to streams at the service application layers.

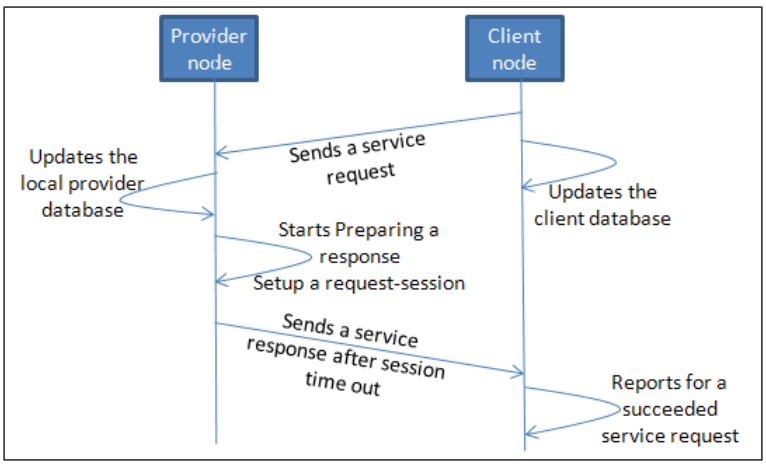

Figure 15: Interactions of a service request-response-provider-client.

Replica forward (service replication) process

The interactions in this process involve a provider and client nodes with the repository as shown in Figure 16. The client node after this process is supposed to become an active service provider.

Service hibernation process

As shown in Figure 17, an active service provider performs this process upon the low gained service interest to shut down the offered service, update the repository node and free the service occupied resources.

Service restoration process

As shown in Figure 18, an inactive service provider performs this process upon the noticed high gained service interest to activate his offered service, update the repository node and allocate the required resources.

Neighbor wakeup process (Wakeup neighbors' mode)

As shown in Figure 19, in the "Wakeup neighbor" mode, after a client database is updated, no active providers have been supplied, in this case, the client starts searching the repository for another inactive service provider in the neighborhood and to send a wakeup request to .

Forward inactive replica process (Forward Inactive replicas' mode)

As shown in Figure 20, after the client node receives his own replica of the provider node, the client is not allowed to activate the received replica. But it should update the repository database with a new offer about the inactive replica.

Load balanced service request process

Figure 21 shows a load balanced service request process in which there are two service clients. Assume the provider can only serve one client at a time. Client A posed a service request to the service provider. The provider establishes a connected session with client A. Once the session terminates, a response is sent to client A. During the session being opened for the service request of client A, another request of client B arrived. Since, the service provider, as assumed, can only serve one opened session at a time, a "busy"

message will be sent to client B. In this case, client B is supposed to send a "find a next provider" to the repository. This request holds the last busy provider which the service request has been directed to.

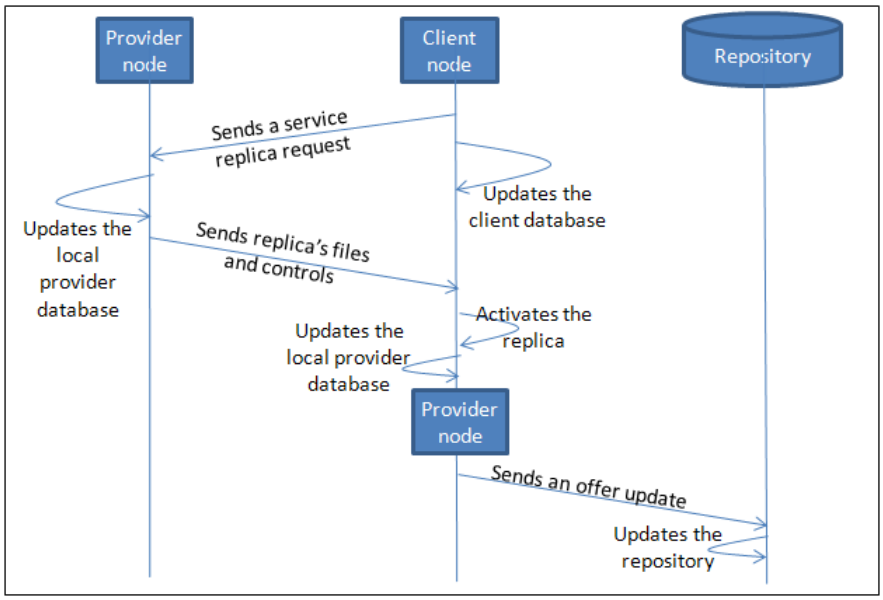

Figure 16: Interactions in a replica forward process.

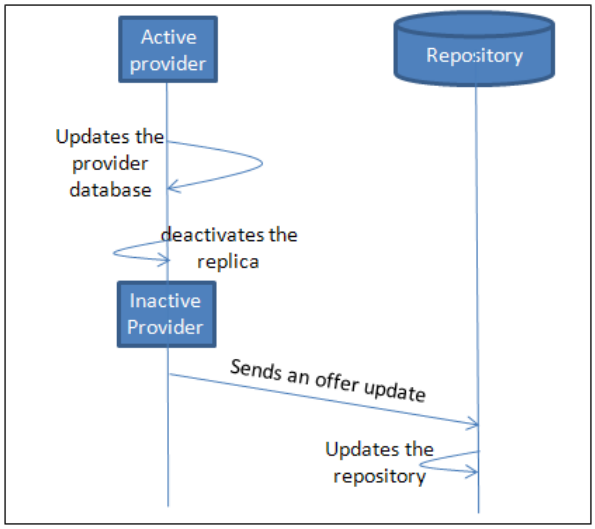

Figure 17: Interactions during a process of service hibernation.

The response of the repository contains the next active provider to the last busy provider based on the requirement index. If there is no active replica inside client B partition, the response will contain the next inactive provider to the last busy provider based on the requirement index. If there is no active or inactive providers next to the last busy provider, the , the response will contain a "-1" provider id.

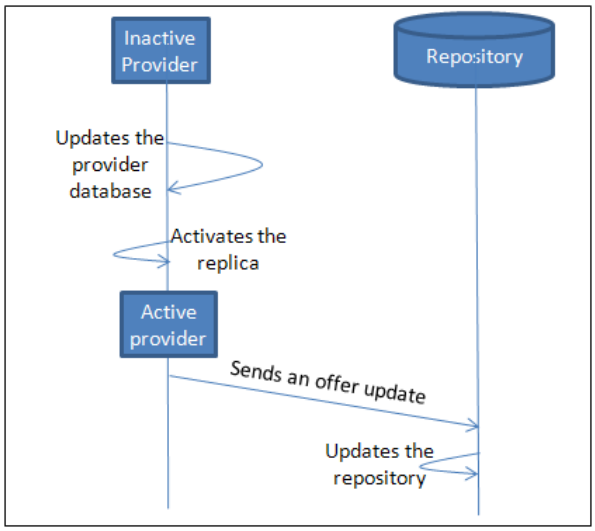

Figure 18: Interactions of a process of service restoration.

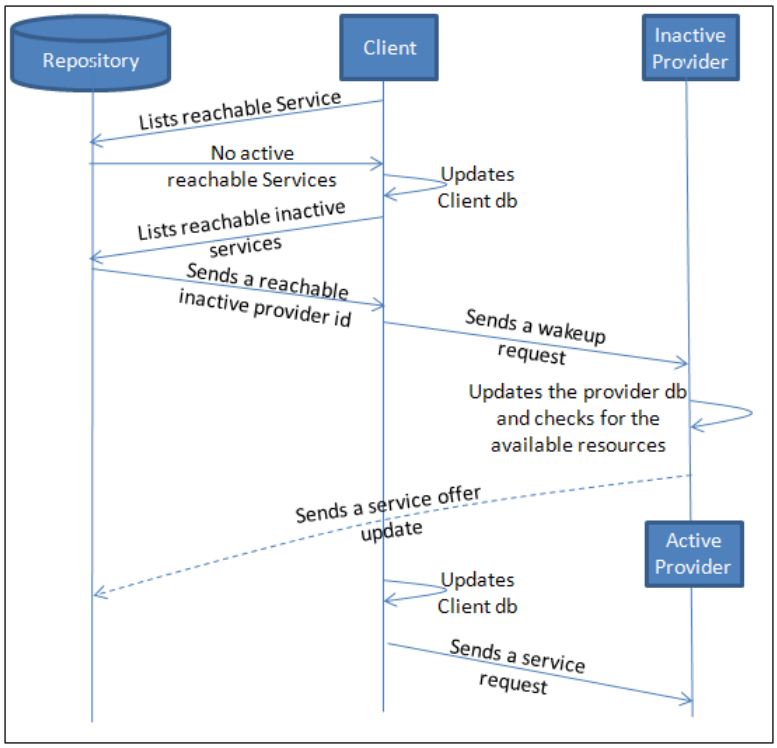

Figure 19: Interactions during a "wakeup neighbor" process.

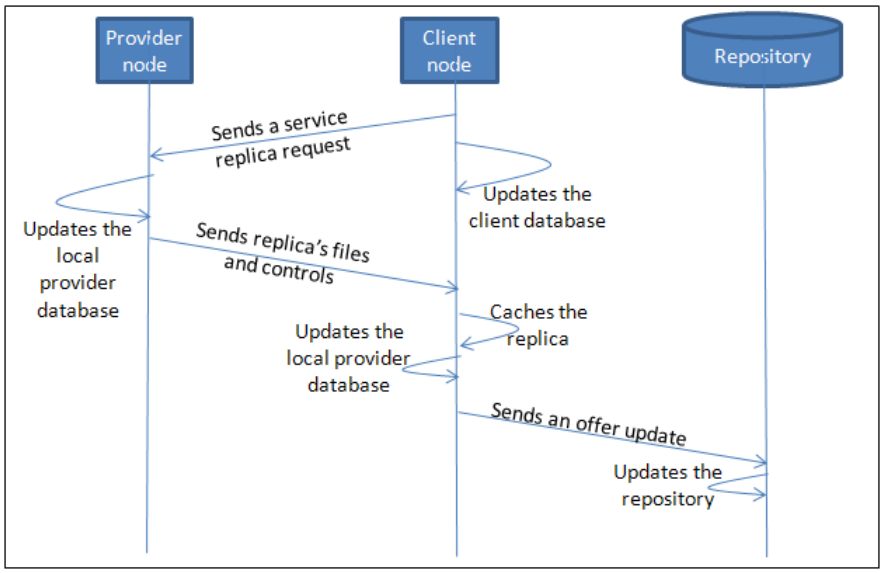

Figure 20: The interactions during a "Forward Inactive replicas" process

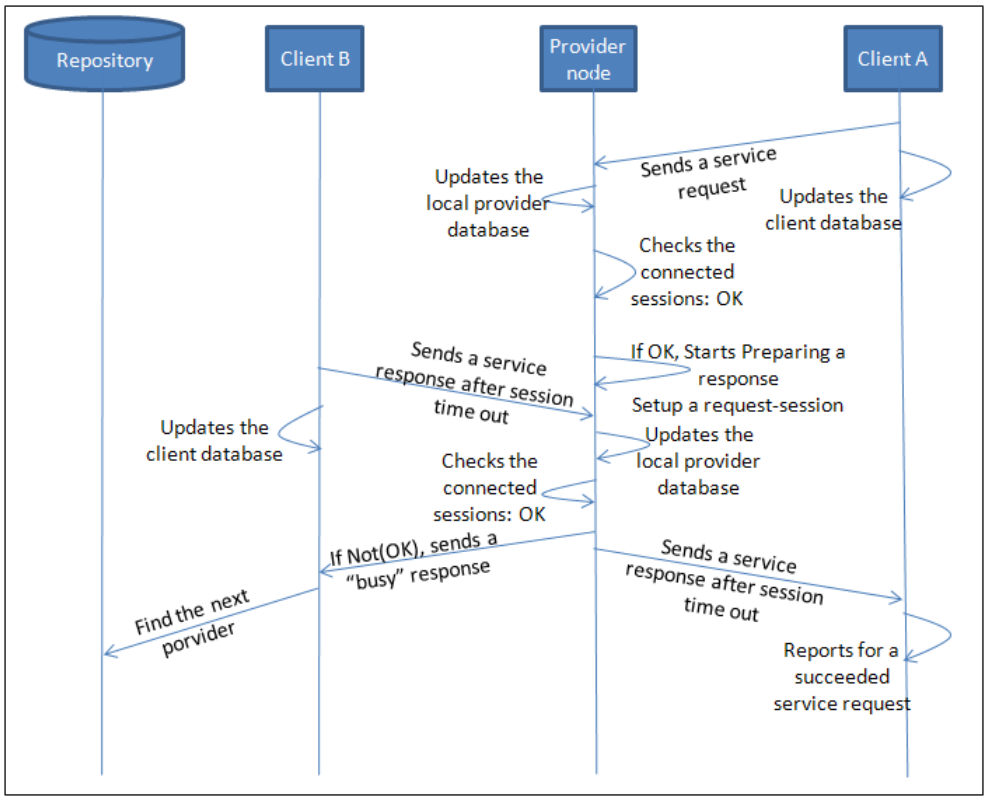

Figure 21: The interactions during a "load balanced" service request process

Service Selection Modes

The service selection modes can be set from the attribute of "Service.Modes" in the mobile node attributes. They offer a wide range of service selection mechanisms. The service selection processes are not only

responsible for keeping the replication cost or prevalence as minimum as possible but also increasing the service availability in cases of disappeared active service providers such as in "Wakeup neighbor" mode.

Pipeline stage

Although, specifying the transmission range in the repository module is enough and may avoid many complications regarding dealing with the low level physical layer, we recommend applying some modification at the physical layer pipeline stage of the "chanmatch model" of the radio transmitter. The provided new pipeline stage C file, review the downloads' section in the release notes part in this documentation, prevents the transmitter of proceeding if the receiving party is away greater than a fixed length (75 meters).

Source code function prototype explanation and Lists-structures prototypes

The next section contains the most important functions and structure which are implemented in both of repository and mobile node process models. The un-mentioned functions or structures here may be test ,"to be implemented", called internally or not important to be mentioned.

Repository

1- General and topological manipulation functions

void init_partition_list(PrgT_List* net_partitions) ;

Inputs: a partition list.

Outputs: void.

Comments: initiates the partition lists ate the beginning of the simulation.

---

bool neighbors(int i, int j, double RadioRange);

Inputs: node_i and node _j ids with the radio transmission range.

Outputs: Boolean value.

Comments: True, if the nodes are neighbors .

---

void pack_sevices_meta_data(PrgT_List* Ser_meta_data);

Inputs: a list of the original services' metadata.

Outputs: void.

Comments: reads the original services' metadata specifications from the model attribute and pack then into the list.

---

void update_partition_information(PrgT_List* net_partitions, double RadioRange);

Inputs: a partition list and the radio transmission range of them.

Outputs: void.

Comments: update the partition list information regarding the current positions of the network participants.

2- Messaging Functions

void List_Act_HIB_Rchble(bool Act_Hib,int node_id, int serId, double RadioRange,PrgT_List* MyOffers,PrgT_List* net_partitions,int requestNr);

Inputs: status of the required offer, the requesting node id, the service id, transmission range of the node, the current offers ate the repository and a network partition list.

Outputs: void.

Comments: returns, as a massage to the requesting node, the most interesting (active/inactive) service in the offers database (list) and of course, regarding the requesting node partition .

---

void List_originals(Message_record* msg);

Inputs: a message record which contains the requesting node id.

Outputs: void

Comments: packs and send back a response for the requesting node with a list of the original services from their metadata.

---

void SDP_Find_next_provider_fn (Message_record* msg);

Inputs: a message record which contains the requesting node id.

Outputs: void.

Comments: sends pack to the requesting node a message with the next available provider (active/inactive). It sends a "-1-provider-id" if there is no (active/inactive) providers in the requesting node partition.

---

void Update_Service_Offer(Message_record* msg);

Inputs: a message record which contains the requesting node id.

Outputs: void.

Comments: unpacks the sent message record and initiate/update an new/old service offer.

3- Performance matrix functions

void compute_statisticsGrIGrIII (PrgT_List* net_partitions);

Inputs: a partition list .

Outputs: void.

Comments: computes and write on the statistics' handlers the service distribution processes performance of SDP for all services.

4- Important lists and structures prototypes

PrgT_List* Offers; //list of the offered services and their information.

PrgT_List* Partitions; //holds the list of nodes' ids and their partitions' ids.

PrgT_List* \Services_meta_data; //stores the original service specifications

struct offers_list_entry; // "Offers" list entry

struct Message_record; //message record structure to be attached to the packets

struct Partition_list_entry; // "Partitions" list entry

struct Service_metadata_entry; // "Services_meta_data" list entry

Mobile Node

1- Initialization and tests' functions

void ask_for_a_replica(int ServiceID, int ReplicaID, int provider_id);

Inputs: the desired service id , replica id, and the hosting provider id.

Outputs: void.

Comments: composes a request for the passed service and replica ids.

---

bool available_resources(int ServiceID);

Inputs: a service id.

Outputs: Boolean .

Comments: Searchs in the service metadata at the repository database for the required resources of a specified service and compare it to the available service of the current node. Returns true if the resources are available.

---

bool chk_hibernate(int ServiceID);

Inputs: a service id.

Outputs: Boolean.

Comments: if the service gained low interest (request, regarding the hibernation threshold), it returns true .

---

bool chk_replicate(int ServiceID);

Inputs: a service id.

Outputs: Boolean.

Comments: if the service gained

---

void hibernate(int serId);

Inputs: the service id.

Outputs: void.

Comments: hibernates a specific active service/replica using the service id

---

static void init_SDP();

Inputs: void.

Outputs: void.

Comments: reads the SDP attributes of the current mobile node and extract the defined service(s) interests(s) from the "gross interest" attribute and save then in the "interest_list" list. Moreover, it initiates the requesting (calling) model specifications.

---

static void initialize_client_db();

Inputs: void.

Outputs: void.

Comments: initialize the client database.

---

void schd_the_events_for_all();

Inputs: void.

Outputs: void.

Comments: schedule all interrupts/events of request generation, service replication tests, and the initial hibernation test of the current mobile node.

high interest (request, regarding the replication threshold), it returns true

---

static bool update_a_new_calling_cycle(PrgT_List* interest_list, int s_service_id, double which_time, bool &continue_schd);

Inputs: the interest list, a service id, the time of simulation, and a flage used to toggle the scheduling processing in the pause period of the calling model (with calling rates > 0 call/time unite).

Outputs: Boolean.

Comments: Based on a service interest, regarding a specified service, the calling (requesting) model parameters are updated.

2- Client and Provider databases' manipulation functions

void activate_a_local_offer(service* local_offer, bool quality_reason);

Inputs: a local offer by the current mobile node and if it ist status is toggle because of a quality reason.

Outputs: void.

Comments: activates the local offer at the local provider database and sends an update for the repository node.

Client_db_record_structure* Get_Client_db_entrs(int ServieId);

Inputs: a service id.

Outputs: a client database entry.

Comments: returns the relevant entry of a specific service from the client database.

---

double get_request_response_travel_time(Message_record* msg);

Inputs: a received message record .

Outputs: Double value (time in seconds).

Comments: compute the response time of the received message.

---

bool has_an_active_service(int ServiceID);

Inputs: a service id.

Outputs: Boolean.

Comments: returns true if the current mobile node has an active replica of a specific service.

---

service* retrieve_provider_db_entry_service(int SerId);

Inputs: a service id.

Outputs: an offer entry from the provider database.

Comments: retrieves the relevant service record from a provider database.

---

bool retrieve_status_of_a_replica_in_my_Provider_db(int SerId);

Inputs: a service id.

Outputs: Boolean.

Comments: returns the status of a specified replica from the current node provider database.

---

void update_a_local_offer(service* local_offer);

Inputs: a local offer in the provider database.

Outputs: void.

Comments: updates the local offer at the local provider database and sends an update for the repository node.

---

void update_the_local_client_db(Client_db_record_structure* target);

Inputs: a specific client database entry.

Outputs: void.

Comments: update a specific client database entry

3- General manipulation functions

int check_number_of_currently_opened_sessions_of_service(int servId);

Inputs: a service id.

Outputs: Integer.

Comments: regarding a specific service, computes the number of the opened sessions for the currently served request.

---

void increase_total_nr_of_requests(int ServiceId);

Inputs: a service id.

Outputs: void.

Comments: increases at the repository node the total number of the succeeded request (to compute the success ratio).

4- Messaging functions

bool busy_rsponse (Message_record* msg);

inputs: a received message record.

outputs: Boolean.

comments: returns true, if the message record is a "busy" response.

---

void compose_wakeupNeighbors_request_for(Message_record* rspns, bool for_quality_reason);

inputs: a received message record and a flage to indicate if the generated "wakeup" request is done because a quality reason.

outputs: void.

comments: a wakeup request packet will be generated for the inactive provider id which is contained in the received message record.

---

void indicate_the_offer_for_quality(int serid,int repid);

inputs: a service id and replica id.

outputs: void.

comments: performs the required updated after indicating that the local offer was activated for a quality reason.

---

void Init_Service_Received(int GlobalDBModule,PrgT_List* Contents, PrgT_List* ProviderDB),

inputs: the repository node id, the contents of a message record and the provider database.

outputs: void.

comments: creates a service and save it in the provider db and sends a SDP_offer_update message to the global db.

---

void List_Act_Rchble_rspns(Message_record* msg);

inputs:a message record.

outputs: void.

comments: unpacks the received packet parameters and searchs for and store/update the service-replica-provider record in the client database

---

void List_originals();

inputs: void.

outputs: void.

comments: makes a query for the original services at the repository.

---

void List_originals_Rspns(Message_record* msg);

inputs: a message record.

outputs: void.

comments: associates (just with services' ids) the gross interests to the given services. if the number of services > the number of gross interests, it produces an equivalent number of gross interests to meet the number of services. The last gross interest will be used to be copied for the list of original services ids.

---

void manage_find_next_provider_for_load_balance(int ServiceId, int lastProviderId);

inputs: a service id and the last provider id, review the load balanced service request process.

outputs: void.

comments: sens a “find next provider” request for the repository node.

---

static void manage_sending_a_replica (int dest_node_id, Packet* ReplicaPkt);

inputs: a destination mobile node id and a packet containing a replica.

outputs: void.

comments: manages sending the replica packet to its destination node.

---

static void manage_sending_a_service_requets(int SerId, int replicaId, int dest_node_id, bool for_quality_reason);

inputs: a desired service id, replica id, destination node and a flag for the quality reason.

outputs: void.

comments: manages sends a service request to a specific service.

---

static void manage_sending_a_service_rsponse(Message_record* tmp_rkrd);

inputs: a message record of a service response.

outputs: void.

comments: manages sends a service response to its destination.

Packet* prepare_a_service_request_packet(int dest_id, int ServiceId, int RepId, bool for_quality_reasons);

inputs: a destination node id, a service id, a replica id and a flag for the quality reason.

outputs: a packet.

comments: generates a service request packet to be sent to a provider node.

Packet* prepare_a_service_response_packet(Message_record* tmp_rkrd);

inputs: a message record.

outputs: a packet.

comments: generates a service response packet to be sent to a provider node.

---

void SDP_A_Replica_rcvd(Message_record* rcvdReplica);

inputs: a received message record.

outputs: void.

comments: triggers the procedures of a receiving replica file.

---

void SDP_Ask_For_Replica_rcvd(Message_record* tmp);

inputs: a message record.

outputs: void.

comments: triggers the procedures of a receiving a request for a replica to be hosted by the sending node.

---

void SDP_Find_next_provider_rspns_procedure(Message_record* tmp);

inputs: a message record.

outputs: void.

comments: starts the unpack the received message record and test if to (a)sends a message request, (b)sends a "wakeup request" or (c) halt with the suggested received provider id.

---

void SDP_Wakeup_Request_rcvd(Message_record* rqst);

inputs: a message record.

outputs: void.

comments: activates the received required service based on the available resources.

---

void Send_a_service_rsponse();

inputs: void.

outputs: void.

comments: examines the opened session and sends the next scheduled response to its destination:

---

void Service_Request_Rcvd(Message_record* msg);

inputs: a message record.

outputs: void.

comments: starts the procedure of sending a service response. Opens the request-response sessions

---

void Service_Response_Rcvd(int ServiceID, double request_response_travel_time);

inputs: a service id and time in seconds

outputs: void.

comments: if the received response is valid (with time less than the defined time for receiving a response in the equiv. GI) then, updates remotely the number of the succeeded request at the repository.

---

void update_client_db();

inputs: void.

outputs: void.

comments: trace the whole entries of the Client_db and update the reachable services.

---

void wakeupNeighbors_for_service_chk(int ServiceId,Client_db_record_structure* target);

inputs: a service id and a specific client database entry.

outputs: void.

comments: tests if there is a hibernated service to compose a wakeup request for. (by sending a packet for the repository). Forwards the request for the waked-up replica.

5- Important lists' and structures' prototypes

PrgT_List*	\Provider_db;	\\ the provider database list.
PrgT_List*	\Client_db;	\\ the client database list.
PrgT_List*	\Services_interests;	\\ the service gross interests' set list.
PrgT_List*	\requests_replies;	\\the opened sessions' list.

struct	service;	\\ a provider database entry.
struct	Client_db_record_structure;	\\ a client database entry.
struct	Service_Interests_entry;	\\ a gross interests' set list entry.
struct	requests_replies_entry;	\\ the opened sessions' list entry.

References

[1] Mohamed Hamdy and Birgitta König-Ries. Book of Communications in Computer and Information Science, Book of the selected papers of the ICETE 2008, volume 48 of CCIS 48, chapter: The Service Distribution Protocol for MANETs- Criteria and Performance Analysis, pages 467-479. Springer Berlin Heidelberg, 2009.

[2] Mohamed Hamdy and Birgitta König-Ries, "Service Availability, Success Ratio, Prevalence, Replica Allocation Correctness, Replication Degree, and Effects of Different Replication/Hibernation Behavior Effects of the Service Distribution Protocol for Mobile Ad Hoc Networks -A Detailed Study-," JENAER SCHRIFTEN ZUR MATHEMATIK UND INFORMATIK, Technical Report: Math/Inf/08/08, Friedrich-Schiller-University Jena, December 2008.

[3] D I A N E - Project: Services in ad hoc Networks, http://fusion.cs.uni-jena.de/DIANE/

[4] Mohamed Hamdy and Birgitta König-Ries, "New Service Selection Strategies for the Service Distribution Protocol for MANETs," the IADIS International Conference on Wireless Applications and Computing (WAC 2010), Freiburg, Germany.