An Extendible Emulator for Investigating Configuration Algorithms in Mobile Ad-hoc Networks

Omid Bushehrian
Computer Engineering and IT Department, Shiraz University of Technology, Shiraz, Iran

Mojtaba Shahkolahi
Department of e-Education, Shiraz University, Shiraz, Iran

ABSTRACT
In this paper, an extendible mobile ad hoc network emulator for investigating different configuration algorithms, for service assignment, is proposed. In this emulator, a configuration algorithm can be plugged as a Java class at runtime. The proposed emulator enables the designer to define different service failure schemes, node movement strategies and routing protocols. The experimental results show the usefulness of the proposed emulator in evaluating a service discovery and assignment protocol from reliability viewpoint.

Keywords
Mobile Ad Hoc Network, configuration algorithm, emulator, redeployment.

1. INTRODUCTION
Mobile ad-hoc networks (MANET) are P2P networks of mobile nodes with the ability of communicating to each other without an underlying infrastructure [2]. MANET's are used in a wide range of applications such as communication between team members in a survival operation at disastrous situations or between military automotives in battle field during the missions. One of the main characteristics of MANETs is the instability of the underlying infrastructure which makes the service providing and access over these networks very challenging. MANETs allow ubiquitous service access, anywhere, anytime without any fixed infrastructure and its applications such as audio/video conferencing, webcasting requires very stringent and inflexible QoS. The provision of QoS guarantees is much more challenging in MANETs than wired networks due to node mobility, limited power supply and a lack of centralized control [1].

For example in a military operation assume that cars always require submitting their local observations to a central planner node frequently. The planner node is a powerful node with the high computational and storage capacity which is able to run complex AI algorithms to plan the subsequent actions of other nodes (cars).

Therefore, the planner node is providing a service and other nodes are consuming that. Generally every node can be a service provider, a service consumer or both. Assuming that a service can be provided by more than one node (most likely) with different qualities (i.e. different response times or reliabilities), the question is how to assign services to consumers such that the quality of service over the network is always retained in an acceptable level according to a predefined objective criterion. A *configuration algorithm* is an algorithm that recommends the best service-to-consumer assignment according to the criteria defined for the network. This algorithm can either be invoked once a service is requested by a consumer to recommend it the best provider or upon a specific event is perceived by a node (service provider or consumer) such as consumer SLA violation. The configuration algorithm can be formulated as an optimization algorithm that aims to find a service-to-consumer assignment which minimizes the total consumer SLA violations subject to the fact that the service assignments satisfy the reliability requirements of the service consumers:

$$Minimize \sum v_i , subject\ to \forall\ c_i : MTTF_i \geq t_i \quad (1)$$

Where c_i denotes the i^{th} service consumer and v_i denotes the amount of SLA violation of c_i which is the difference between the requested response time by the service consumer and the response time provided by the service provider. The reliability requirement of c_i is defined in terms of MTTF measure (mean time to failure) which is constrained to be higher than a predefined threshold t_i. Centralized or de-centralized algorithms with various cost and optimality degrees can be proposed to solve this problem. Due to the deficiency of configurable and extendible emulators to investigate different configuration algorithms, in this paper an extendible emulator architecture is presented which enables the researchers to evaluate their configuration algorithms from different aspects.

The remaining parts of this paper are organized as follows: In section 2 the related works are presented. Section 3 explains the architecture and features of the proposed emulator. The design of the emulator using software design patterns is presented in section 4. The implementation and evaluation of configuration algorithms is explained in section 5. The paper is concluded in section 6.

2. RELATED WORKS
Dynamic (re)configuration and re-deployment algorithms which aim to improve the quality aspects (such as reliability or performance) of service providing over MANET's, have been studied thoroughly in previous papers: One of them is the algorithm designed by Poladian[3] which provides a system infrastructure independent of, and external to, applications. This algorithm works in three phases: The First phase is querying all providers for their services. The second phase is generating all possible configurations and specifying the user utility of providers and the third phase is exploring the quality space of the configurations. The user utility is evaluated based on QoS parameters such as processing power, network bandwidth and battery capacity. The goal of the algorithm is to make a tradeoff between user's acceptable service level and the user utilities. Greedy [1] is another configuration algorithm proposed for the systems with changeable parameters such as network disconnection rates or bandwidth. A fitness function is calculated for each service on each host and the service with maximum fitness is selected and assigned. This function represents the overall satisfaction of the users with the QoS delivered by the system. The last algorithm is based on genetic algorithms [1]. Each individual

is a solution which represents a service assignment. In this algorithm the fitness of each individual is computed considering QoS metrics. There are also previous researches on MANET emulators:

OCTUPOS [2] is one of the MANET emulators that enable the designer to model interactively the network topology, the environmental characteristics (packet loss rates, environment obstacles) and the node movement strategies. The designer can use this emulator to evaluate the movement strategies of nodes. It is used for testing packet loss or disconnection rate for a given node moving plan. It also allows includes every type of device and applications. Possible obstacles, packet losses and enhanced movement models are supported by this emulator. However, in this emulator the configuration algorithms are not supported. This emulator is written in java. It has a GUI for designing nodes and obstacle positions and also modeling node movements. OCTOPUS emulator works fine on Windows and Linux and the only requirements is an IP network. It uses a TCP server, listening on port 8888 to receive commands from applications and reply to them. Furthermore OCTOPUS can be used in robotic planning and sensor networks.

SOAMANET [4] is another emulator which can provide different reusable modeling and analysis capabilities that facilitates the model-based construction of these systems and can be used to evaluate large-scale and highly dynamic service-oriented architecture over MANET's. Service assignment, ad-hoc routing protocols and its parameters are supported by this emulator. To use this emulator we need to define a workflow. A workflow describes that which node provides which services and which node requests for these services. Besides, a workflow is described using a DSML (Domain Specific Modeling Language). The number of completed or failed service requests can be measured during the simulation. Most of the network modeling is done using network simulation tools such as OMNeT++[9] or NS-2[5]. These tools can be integrated to SOAMANET DSML to configure SOA application, device, mobility, and routing modules.

NS2[5] on its own can emulate only wired networks. But a patch was developed by Magdeburg University to support wireless emulation. This simulation tools has some drawback: 1) client hosts should be Linux-based. 2) It is needed to write complex TCL scripts to configure all emulated aspects of wireless links. 3) Clients cannot affect any changes in nodes topology because of possible events during emulation are decided at batch time in TCL scripts.

EMWIN[6] is an IP-based MANET emulation system. It is used to test and evaluate mobile wireless network protocols, such as multi-hop mobile ad hoc routing protocols, directly. It also supports wireless mobility emulation. JEmu[7] is a MANET emulator that can test and evaluate Mobile ad-hoc routing protocols. It supports node movements and radio range of each node.

3. THE ARCHITECTURE

The emulator architecture is shown in figure 1. The proposed emulator enables the designer to model both the network and the application characteristics of a service providing MANET. Here a MANET is defined in terms of a set of nodes communicating in a virtual environment. The architecture consists of three layers: First, the application layer which consists of *Client* and *Service Manager (SM)* concepts. The Client concept represents the service consumer program installed on a node and the SM represents a management agent which should be installed on every service provider or service consumer node in the MANET. Each SM keeps track of both the provided services by the node on which it is installed and the list of current consumers corresponding to each provided service. SM is also responsible for invoking the configuration algorithm plugged to the emulator at a predefined rate. The second layer is the *Node* which consists of a *Processor* concept, a *Power-Consumption* scheme, a *Planner* scheme and a *Routing* scheme. The processor picks the service requests consecutively from its queue and sends back a reply message after a specific time delay to the corresponding client. The power-consumption scheme, the planner scheme and the routing scheme are user defined strategies for the node battery life-time, the node movement and the routing mechanism respectively. The last layer models the environment in which the nodes are moving. The environment is nothing but a *Frame Dispatcher* which is aware of the global image of node locations the Frame. As shown in figure 1, different layers communicate to each other using *packet channels*.

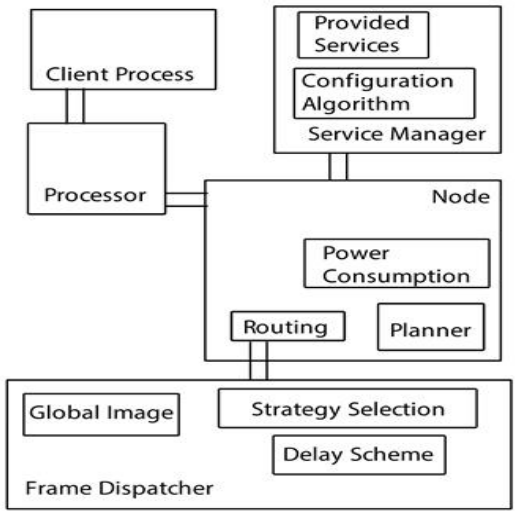


Figure 1: the Emulator Architecture

In the following subsections the detailed design for each concept is explained.

3.1 Channels

Channels are used to transfer packets between node components. A packet is defined as follows:

```
class Packet {
   public int sourceNode;
   public int destNode;
   public Protocol protocol;
   public object obj;
}
```

Where *sourceNode* and *destNode* attributes are identifiers of the sender and the receiver of the packet respectively, the protocol attribute keeps the protocol name of the packet. There are six protocols defined in our emulator: 1)*ServiceQuery*, which is used by a client process to ask its local service manager to start a service query, 2)*SearchService* which is used by service managers to search for a service in the network, 3)*FoundService* which is the reply of the *SearchService* protocol, after a service provider is found, 4)*SelectedService* which is used to notify a client process about the found service provider, 5)*Request* which is used to send a request packet to a selected service provider by the client and 6)*Reply* for replying a client request.

3.2 Client Process

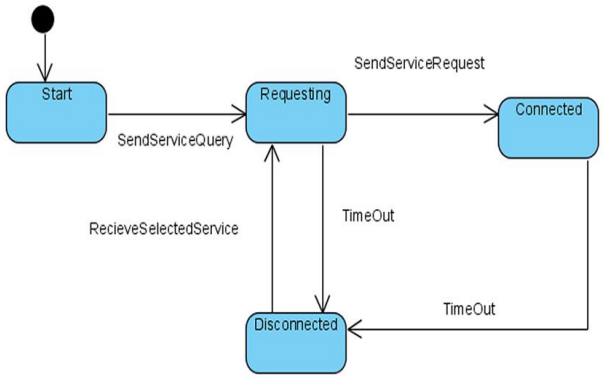


Figure 2: State Diagram of Client Process

The client process is implemented as a state machine as shown in Figure 2. It starts by sending a service query to the service manager and changes its state to "Requesting". Once the client process is notified about the found service it changes the current state to the "connected" and starts communicating to the service provider. Upon a time-out, the state is changed to the "disconnected".

3.3 Service Manager

Service manager is responsible for selecting the best service providers for a service request. This selection is performed by a configuration algorithm that user implements. When a client process asks for a service, service manager broadcast a packet to all nodes to find the available service providers. The service selection strategy in the service manager is implemented as a configuration algorithm which is called by the service manager. The use of the adapter design pattern [8] enables the designer for plugging different configuration algorithms at the simulation time (See Figure 3). To create and plug a new configuration algorithm, the designer first should implement the following standard interface:

```
public interface ConfigurationAlgorithm {
public abstract Packet[] Configure(ArrayList<Object> Temp,int clock, ServiceManager sm,Packet pks);
  }
```

Where the first parameter is a temporary collection of objects which can be used by the algorithm to store and retrieve session data during successive invocations of the algorithm. For example a node needs to store the identifier of its parent node after it receives a Search Service packet. The second parameter is the clock value of the service manager, the third parameter is the object of the current service manager which can be used by the algorithm to extract the list of provided services by the current node and the last parameter is the current incoming packet to be processed. The steps of the service manager thread are as follows:

1- Pick a packet from the node buffer
2- Invoke the *configure*() method of the current *configuration algorithm* object
3- Send the returned packet list to the node to be routed to the destination client

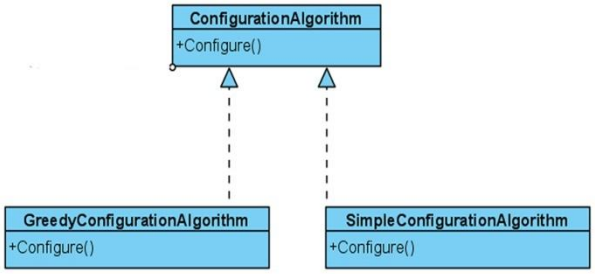


Figure 3: Adapter Pattern of Configuration Algorithm

3.4 Node

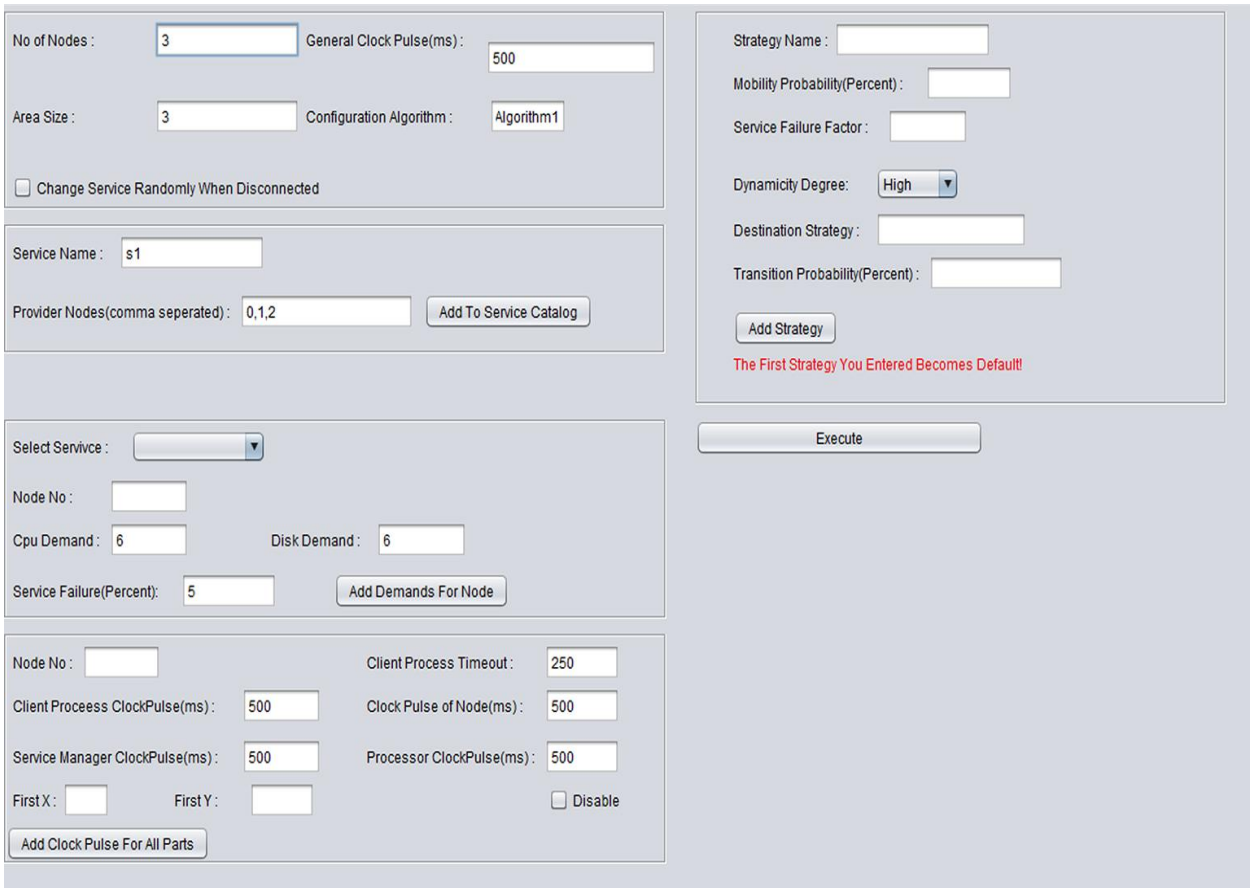
The movement field of mobile nodes is implemented as an n*n grid in which the initial node placements are defined by the user. Each node may move to one of its neighbor's squares with a predefined probability (*mobility probability*). Besides, when there are multiple free squares around a node to move to, one of them is selected based on the value of *dynamicity degree* parameter which also should be set by the user before

Figure 4: Emulator GUI

starting the simulation. This parameter indicates the tendency of nodes to move away each other's signal coverage during the simulation. The Planner component is responsible for node movement regarding the above two mentioned parameters. This component is associated with the *Node* component and after selecting the next adjacent square to move, it will notify the *FrameDispatcher* component the new position of the node.

The main task of the Node component is to transfer packets among channels or creating frames to be sent via the *FrameDispatcher* component. The steps of the Node thread are as follows:

1. Receiving a *ServiceQuery* packet and sending it to its local Service Manager channel.
2. Receiving a packet from the Service Manager channel.
3. Sending a packet to its local Client Process channel or creating a frame and sending it to the *FrameDispatcher*.



4. Receiving a *ServiceRequest* packet from the local Client Process and sending it to the *FrameDispatcher*.
5. Receiving a *ServiceReply* from the Processor channel and sending it to *FrameDispatcher*.

3.5 Failure Scheme

Two failure schemes have been implemented: *packet delivery failure* between nodes due to the network failure and *service failure* at the Processor component. The packet delivery failure probability is calculated by the *FrameDispatcher* component once a frame has to be delivered using the following formula:

$$delivery\ failure\ probability = \frac{D}{N \times m} \quad (2)$$

Where, D is the longest path length (in terms of hop-count) between the source node and the destination node, N is the number of mobile nodes and m denotes the magnifier coefficient. The *service failure rate* is determined by the user and is used by the Processor component to discard an incoming request according to that rate.

4. IMPLEMENTATION AND EXPREMENTAL RESULTS

The user interface of the proposed emulator is shown in Figure 4. To start the emulation, first the number of nodes, the logical clock pulse duration, the configuration algorithm and the client timeout values should be specified by the user. Then an execution strategy should be defined. This strategy is used by the emulator to change the network state during the execution. A strategy is defined in terms of
its *states* and the transition probabilities between states. Each state has its own *mobility probability* and *dynamicity degree* as defined in the previous part. It also includes service failure rate.

For example in Figure 5 below we have defined a two state strategy. The first state indicates a stable network for which the mobility probability, dynamicity degree and service failure rate are 20%, low and 5% respectively.

The second state represents a dynamic network for which the mobility probability, dynamicity degree and service failure rate are 20%, high and 5% respectively. The transition probabilities between states are shown in Figure 5.

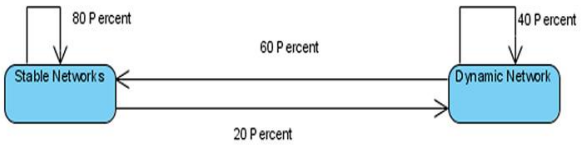


Figure 5: an example emulation strategy

To evaluate the proposed emulator, we have implemented a configuration algorithm to find the fastest service provider among the available service providers in the network and assign it to the client process. This algorithm has been implemented as a Java class which conforms to the defined standard interface for the configuration algorithm.

The emulation was performed with 4, 8 and 12 nodes among which three service providers and one service client were selected. The service failure rate was chosen 5%. The emulation was repeated 5 times with a one-state strategy, each time with a different mobility probability and dynamicity degree value. The emulation trace was inserted into a MySQL log-table with the following schema:

CPLogs		
NodeNo	integer(10)	Nullable = true
CPTime	integer(10)	Nullable = true
State	varchar(255)	Nullable = true
ServiceName	varchar(255)	Nullable = true
Provider	integer(10)	Nullable = true

Figure 6: client process log table

Where *NodeNo* is the identifier of the client process, *CP time* is the value of the client process clock, *State* is the client process state, *ServiceName* is the requested service of the client and *Provider* is the identifier of the service provider node. For each experiment, the fraction of emulation time in which the client process was in the disconnected state, was measured. The experimental results are shown in Figure 7. It was observed that the more the dynamicity degree or mobility rates are, the higher disconnection rate for the client process is resulted.

5. CONCLUSION

In this paper a novel extendible MANET emulator was proposed. The architecture of this emulator was designed using Adapter design pattern, hence different configuration algorithms could be plugged into the emulator at runtime easily. The designer then is able to investigate and compare different configuration algorithms from the reliability or SLA violation rate viewpoints. The experimental results showed the usefulness of the proposed emulator in comparing the reliability of a service assignment protocol in different network conditions.

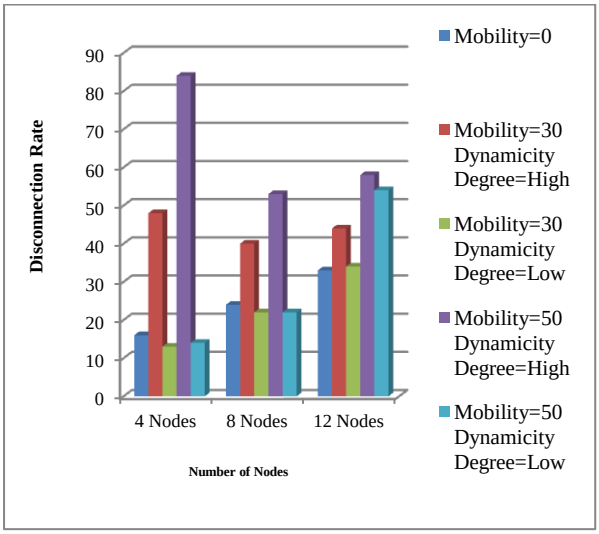


Figure 7: experimental results

6. REFERENCES

[1] Malek S., Medvidovic N., and Mikic-Rakic, M. "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture", IEEE Transactions on Software Engineering, 2012, in press.

[2] D'Aprano, F., Leoni, M., and Mecella, M. 2007. Emulating Mobile Ad-hoc Networks of Hand-held Devices The OCTOPUS Virtual Environment. In Proceedings of MobiEval '07 Proceedings of the 1st international workshop on System evaluation for mobile platforms, USA , New York.

[3] Poladian, V., Sousa, J. P., Garlan, D., and Shaw, M. May 2004 Dynamic Configuration of Resource-Aware Services. In Proceedings of the 26th International Conference on Software Engineering (ICSE'04), Edinburgh, Scotland.

[4] Neema, H., Kashyap, A., Kereskenyi, R., Xue, Y., and Karsai, G. 2010. SOAMANET: A Tool for Evaluating Service-Oriented Architectures on Mobile Ad-hoc Networks. In Proceedings of 14th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications, Fairfax, VA.

[5] The Network Simulator – ns-2. http://isi.edu/nsnam/ns/

[6] Zheng, P., and Lionel, M. Ni. 2002. EMWIN: Emulating a Mobile Wireless Network using a Wired Network. In Proceedings 5th ACM International Workshop on Wireless Mobile Multimedia.

[7] Flynn, J., Tewari, H., and O'Mahony, D. 2001. JEmu: A Real Time Emulation System for Mobile Ad Hoc Networks. In Proceddings of 1st Joint IEI/IEE Symposium on Telecommunications Systems Research.

[8] Larman C. 2004 Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Addison Wesley Professional Ltd.

[9] OMNeT++ Community Site, OMNeT++, July 30, 2010, ht/tp://www.emonetpp.org