# Simulating Services-Based Systems Hosted in Networks with Dynamic Topology

Petr Novotny and Alexander L. Wolf

Imperial College London
London, UK

Imperial College London
Department of Computing
Technical Report DTR-2016-02 January 2016



## ABSTRACT

***Abstract***

*The emerging use of mobile ad hoc networks combined with current trends in the use of service-based systems pose new challenges to accurate simulation of these systems. Current network simulators lack the ability to replicate the complex message exchange behaviour of services, while service simulators do not accurately capture of mobile network properties. In this paper we provide an overview of a framework for simulating both a service behavioural model and a mobile network. The framework is implemented as an extension of the NS-3 network simulator.*

# 1 Introduction

Service-based systems are becoming integral elements of tactical wireless, mobile networks. Structured as a complex of interdependent, interrelated services, they can provide end users with a rich set of information combined from multiple sources. This model of information delivery is in sharp contrast to the flow-based, point-to-point model that has been traditionally supported in tactical networks. Analysing the behaviour of such complex applications is a challenging task. The challenge is compounded when services run on highly dynamic networks such as mobile, ad hoc networks (MANETs), in which the fluidity of the underlying network greatly impacts performance and availability.

Service-based systems consist of some number of services that interact with one another in order to complete client requests. Each service provides a set of methods for use by other services or clients. A method may use any number of other methods provided by other services to carry out its functionality. Thus, services are interconnected with each other. Clients initiate the flow of service requests by sending messages that request method executions, and then wait for some response.

A simulator that can closely replicate the behaviours of service-based systems running on MANETs can be a valuable analysis tool. In particular, it can provide a means to predict performance when designing, deploying, or managing the system. Furthermore, it can model network traffic workloads that are characteristic of MANETs. Existing simulation tools fall short of providing such capabilities. Packet-level network simulators, such as NS-3[1] and QualNet[2] provide detailed implementations of mobile, wireless networks, but lack the ability to replicate complex behavioural aspects of service-based systems. These aspects are addressed in high-level service simulators [5], which unfortunately do not provide a means to simulate a complex network layer.

In this paper we introduce a new simulation tool for service-based systems hosted on MANETs. With the system and behavioural models of services built on top of a packet-based simulator, our approach allows the replication of various critical aspects, such as the cascading flows of messages in complex conversations, comprehensive client-driven workload profiles, and the propagation of faults through services. Furthermore, the simulator provides generic and easily extended models that can be used to capture modern service-based platforms, such as SOA, operating in MANET or hybrid networks.

We have used the simulator for evaluation of several system management methods such as in method of discovery of software service dependencies in MANETs [4, 2], method of fault localization in service-based systems hosted in MANETs [3, 1], and in analysis of behavior of hybrid wireless networks [6].

A source code of the simulator is available for download at the following address:
https://github.com/jcmldev/service-simulator-ns3

# 2 Architecture

The simulator engine is built on top of the discrete event network simulator NS-3, extended with additional higher-level abstraction layers and components for simulating service entities and their interactions. NS-3 provides a comprehensive network simulation with detailed implementation of low-level network protocols. However, NS-3 provides only a simple mechanism for simulating the flow of packets from point to point. At the highest abstraction level, NS-3 provides sockets and packets as a basic network data transfer mechanism.

Figure 1 illustrates the architecture of our simulator engine. The simulation engine encapsulates the socket layer into a *messaging layer* that provides the abstraction of messages exchanged between end-points. The messaging layer is then encapsulated into a *service layer* that provides abstractions for entities (services and clients) and their interconnection models. Finally, the simulator provides methods for engineers to configure the simulation scenarios and their parameters, to run the simulation, and to generate output traces. In what follows, we describe these layers, models and the current implementation.

[1] `http://www.nsnam.org/`
[2] `http://www.scalable-networks.com/products/qualnet/`

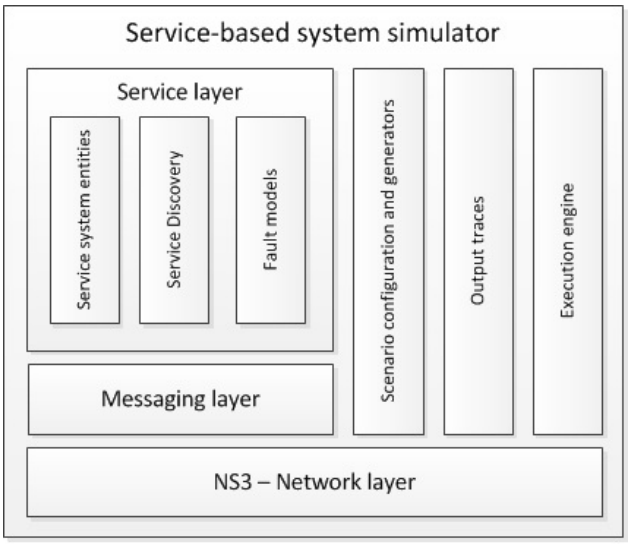


Figure 1: Architecture of the NS-3 based service-based system simulator

# 3 Messaging layer

The messaging layer provides abstractions for exchange of messages between end-points used by service-based system entities. The messaging layer is built directly on top of the socket layer of NS-3.

The abstractions provided by NS-3 for transferring data over the network are that of sockets and packets. The NS-3 socket is an asynchronous implementation of BSD socket API. The sockets are used to send and receive data in NS-3 packets. The NS-3 packet is transferred over the network in series of network packets. The NS-3 provides suit of socket types including one for each transport layer protocol implemented in NS-3 (i.e. TCP and UDP). The messaging layer encapsulates the sockets of the specific transport protocol and provides abstractions for exchange of messages.

Figure 2 illustrates the high level structure of the messaging layer. The messaging layer has two main objectives.

First, the messaging layer defines the abstractions used for exchange of messages over the network in unicast mode. A message represents remote method invocation mechanism such as a message in SOAP protocol. The message caries reference to the service and method to invoke and additional identification data. Additionally, the message caries certain amount of dummy data to represent the size the message would have in a real system. The messages are exchanged over the network between two types of end-points. A client end-point is used to send messages and a server end-point is used to receive messages. The client-end point is thus used by the client applications and services to send request messages to other services and to receive response messages. Furthermore, the client-end point is also used to send response messages from services back to clients and services. The server end-point is used by services to listen for and to receive invocation request messages.

The messaging layer further introduces semantics into the message exchange. The end-points thus support two message exchange patterns; namely the request-response pattern and the send-only pattern. In order to support these exchange patterns the end-points report on whether a message was successfully sent between end-points or failed (i.e. send failure) and whether a response message was received on time or not (i.e. response timeout).

The second objective of the messaging layer is to provide implementation of the specific transport protocols for exchanging messages over the dynamic networks. In our current implementation, the messaging layer provides implementation of the UDP transport layer protocol. The UDP is commonly used in MANET environments because it is a lightweight stateless protocol. Thus, it allows sending data between nodes

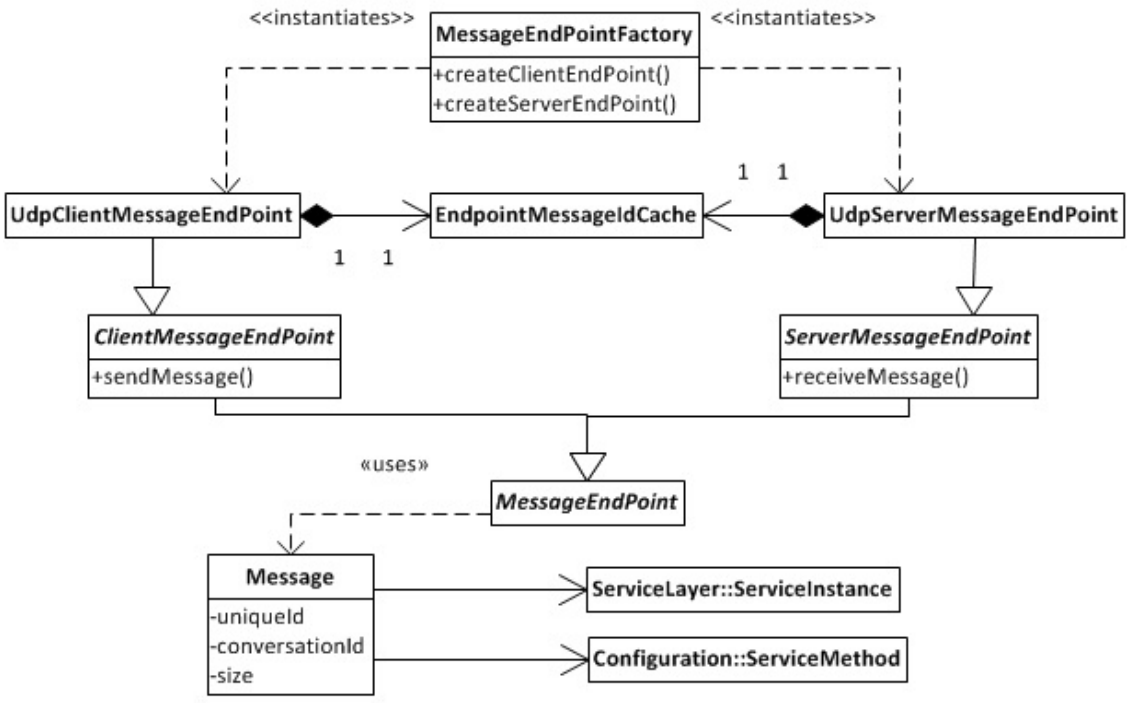


Figure 2: Class diagram of the messaging layer

without maintaining connection or other mechanisms which are problematic or ineffective in networks with dynamic topology. However, the UDP protocol is unreliable because it does not guarantee delivery of packets or notification of failure. Thus, the UDP based end-point implementation provides additional lightweight mechanism for semi-reliable message delivery. This mechanism also provides the necessary functionality for the end-point reporting of status of message exchange. The protocol uses acknowledgement messages (ACK) to verify if payload message was successfully received.

The acknowledgement protocol works as following: an end-point will wait for certain period of time after it sent payload message to receive ACK message. If the ACK message is received in predefined period of time, the end-point will confirm successful sending of the payload message. However, if the ACK message was not received, the end-point will re-send the payload message again and again wait of the ACK message. The re-send cycle will be repeated for certain number of times. On the receiving side, as a response to any payload message received, the end-point will automatically send ACK message. The acknowledgement protocol is illustrated in Figure 3 for client end-point and Figure 4 for server end-point. The client end-point is used for either to send request message and wait for response or to send single message without response.

Due to peculiarities of the dynamic networks, the messages get frequently lost between nodes. Hence, using the acknowledgement and re-send mechanism may lead to repeated sending and receiving of the same messages. To eliminate consequences of such a condition, the UDP end-points use two strategies. First, whenever end-point receives payload message, it always sends back ACK message to confirm reception of the payload message (even if the message was already received before). Second, the end-point uses mechanism of dropping redundant payload messages already received i.e. preventing redundant invocation of services. This is an important feature ensuring integrity of the computations, required in environment built on top of non-reliable transport protocols.

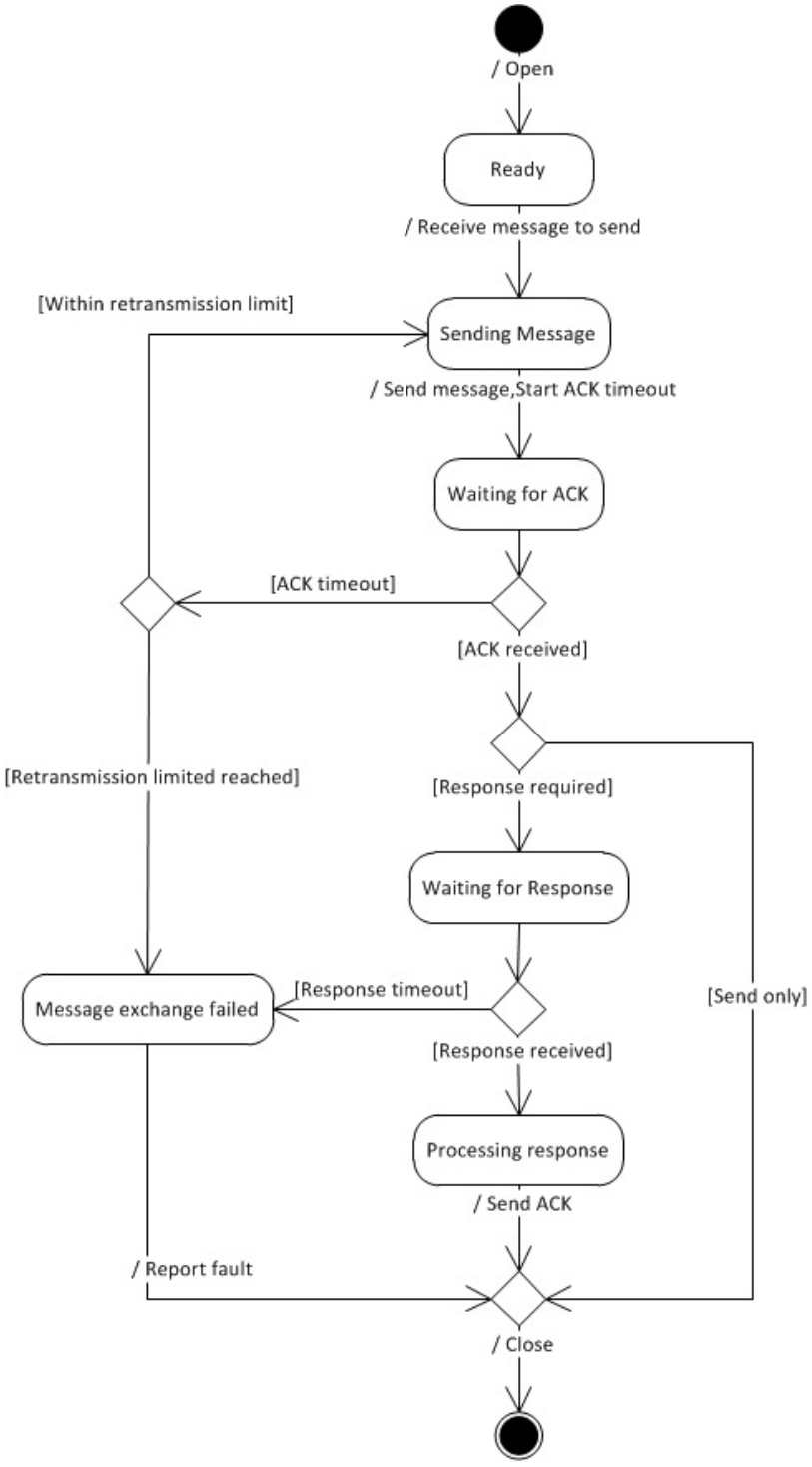


Figure 3: State-machine diagram of the UDP client end-point protocol

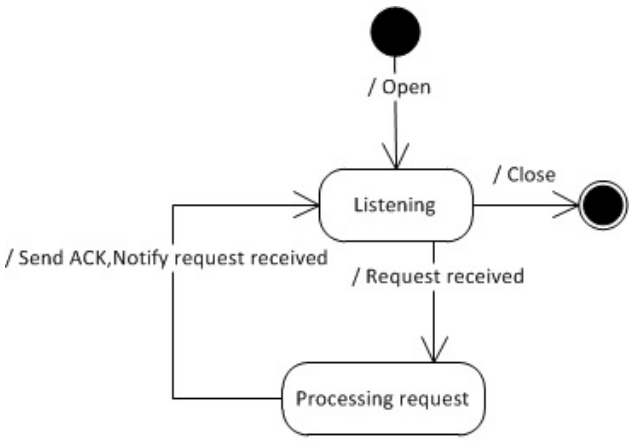


Figure 4: State-machine diagram of the UDP server end-point protocol

# 4 Service layer

The service layer of the simulator consists of several abstraction models: entities, interconnections, workloads, faults, messages and deployment.

(1) *Entity model:* Entity models provide the building blocks of the service-based system simulation.

- *Clients* represent applications used by end users. Each client behaves as an autonomous entity that contacts a set of services at times (random or deterministic) configured by the engineer.
- *Contracts* represent definitions of interfaces of services. Each contract defines an interface of a type of service as a set of methods. A contract is provided by (i.e. implemented by) one or more services.
- *Services* represent autonomous self-contained functional units. Each service adheres to (i.e. implements) one contract and has a set of methods that are available to be used by clients and other services. Each method contains an abstract definition of its computation consisting of delays to simulate processing time, and a set of steps that send requests to other services.

In the Figure 5 are illustrated the configuration elements of the service-based system entities. Furthermore, in the Figure 6 are shown the runtime elements of the service-based system entities. In the Table 1 are provided typical values used in configuration of the Entity model in our experiments.

(2) *Interconnection model:* The interconnection model defines the methods in other service contracts with which each entity in the service-based system interacts (i.e., sends service requests and receives responses). Two types of interconnections are defined: client-to-service, and service-to-service. The simulator provides probabilistic as well as deterministic generators of the service-based system interconnections. The probabilistic generator creates a randomized configuration with predefined connection probabilities. The deterministic generator allows the engineer to have control over the specific interactions in the system.

The interconnection model also defines the *service discovery* mechanism used during the system runtime. The service discovery mechanism provides functionality to the clients and services to discover service instances to send messages to during the system runtime. It is an essential component in service-based system where interconnections are defined between entities and contracts (i.e. types of services) but not services themselves. Thus, during the system runtime, the entities have to discover which actual service instance they should interact with based on the contract the service instance provides. This task is particularly important in MANETs with the dynamic topology continuously altering connectivity between nodes and consequently the availability of services.

In general, before every request, service-based system entity will query a service registry for a service instance to send the request to. The service registry will select the most appropriate service instance based

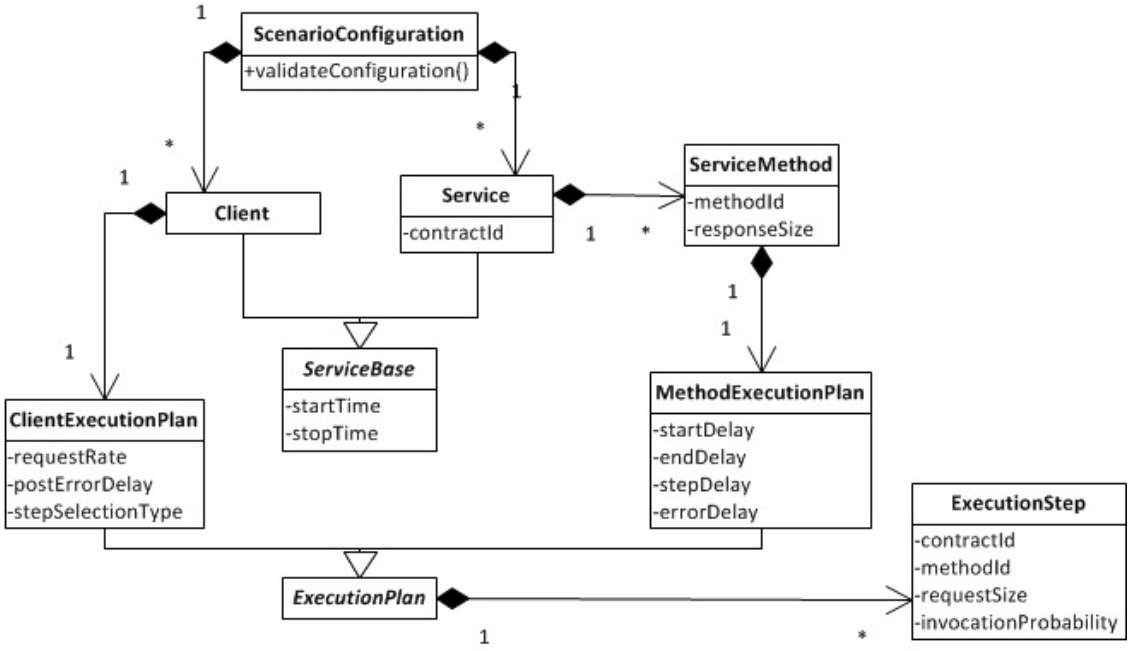


Figure 5: Class diagram of the configuration elements of the service-based system entities

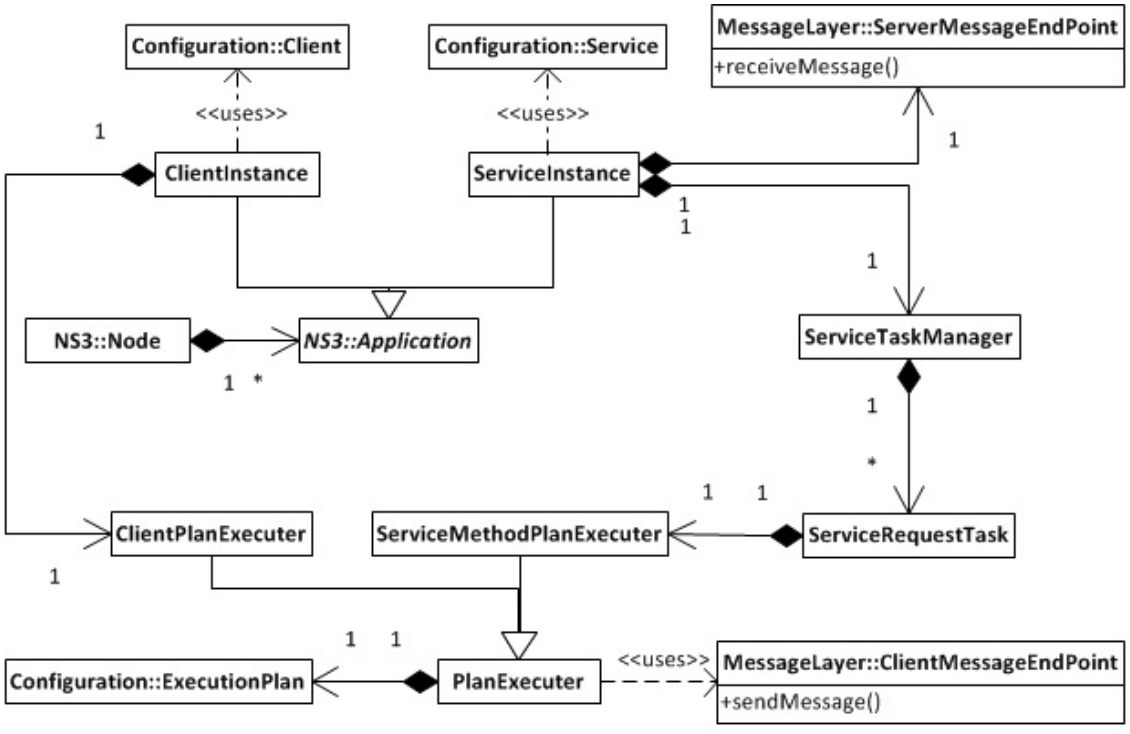


Figure 6: Class diagram of the runtime elements of the service-based system entities

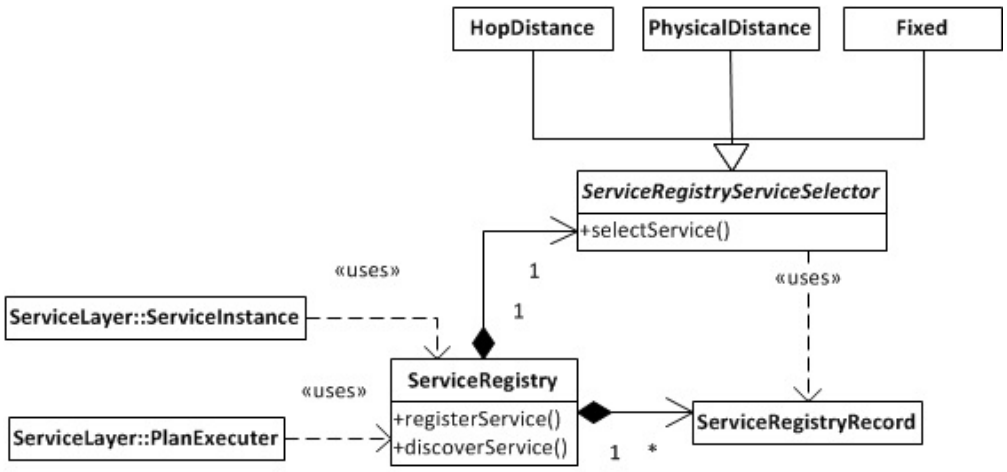


Figure 7: Class diagram of the service discovery components of the service-based system

on its availability or some other metric. The simulator provides three types of service discovery mechanisms; based on physical distance, based on metric from routing tables (i.e. hop distance) and based on fixed configuration. Figure 7 illustrates the service discovery components.

The interconnection model allows configuring specific system topologies. For example, a frequently used topology in the service-based systems is a 2-tiered topology. In this type of system, the first tier consists of the connections between the clients and a set of "front-end" services, while the second tier consists of the connections between the services themselves. In our experiments we make frequently use of the specific service topologies.

In the Table 1 are provided typical values used in configuration of the Interconnection model in our experiments. In this example configuration, the interconnections are defined probabilistically.

(3) *Message model*: There are three types of messages exchanged between entities: requests, responses, and exceptions. Request messages are used to invoke methods in other services, while response messages are sent by services back to the requesting entity upon the completion of the requested method. Exception messages are used to propagate fault symptoms caused by network or service faults. The flow of messages exchanged between services during the processing of a client request is called a *conversation*. In the simulator, all messages contain information about the conversation to which they belong. The conversation information is designed to replicate the behavior of WS-* standards such as WS-Addressing.[3]

(4) *Fault model*: Services running on MANETs are exposed to potentially frequent faults in the network due to network instability and in the services themselves due to resource constraints (and bugs). While network failures are immediately captured by the network simulation layer, we must include a service fault model that defines the failure behavior of the services.

The network faults are configured directly within configuration of the physical network. In MANETs, the essential configuration of the network includes the propagation loss model which defines the quality of wireless links between nodes. The simulator provides series of deterministic and probabilistic service fault models as well as the capability to configure composite fault model behavior. The fault models are injected into the services and into the service methods to allow fine grained configuration of the fault behavior. Figure 8 illustrates the components of the fault model.

In the Table 1 are provided typical values used in configuration of the Fault model in our experiments. In this example configuration, we use a probabilistic On/Off fault model to fail the services.

(5) *Workload model*: In service-based systems the workload is initiated by clients sending requests to services. The workload model defines the rates of such requests. In our current implementation, clients

[3] `http://www.w3.org/Submission/ws-addressing/`

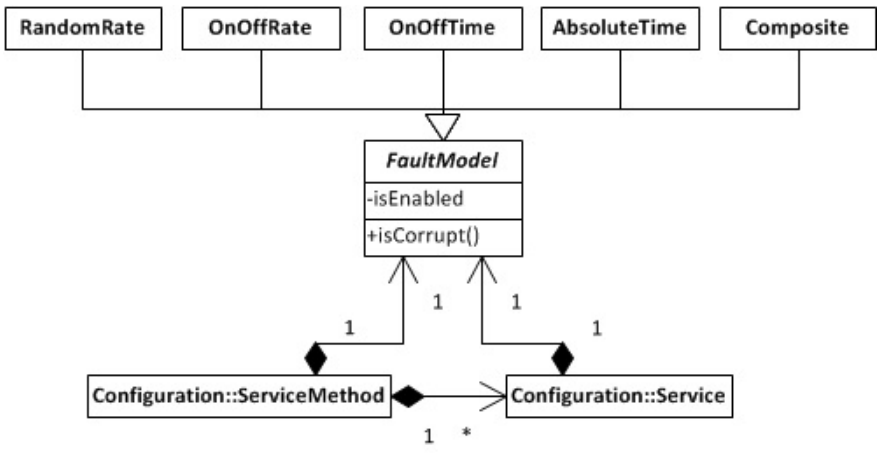


Figure 8: Class diagram of the fault model components of the service-based system

repeatedly, and at pre-configured random times select one method to request out of the set of available service methods, and then waits for a response.

In Figure 9 is shown client workload algorithm. The simulator provides two techniques to select a method to send the request to. The method is either selected randomly from set of available methods or the method is selected based on individual probability of each method to be invoked. For each client, the set of the available service methods is defined by the interconnection model.

Upon reception of a request, the requested service method is invoked and depending on the configuration of interconnections, further messages will be sent to other services. In Figure 10 is shown the processing of a request by a service.

In the Table 1 are provided typical values used in configuration of the Workload model in our experiments. In section *Workload model - client* are defined parameters of client request rate and the selection of the service method to send request to. In section *Workload model - service* are defined parameters of delays representing configuration of processing of requests by services.

(6) *Deployment model:* The deployment model specifies the mapping between physical network nodes and the instances of entities of the service-based system (i.e. clients and services). The simulator provides probabilistic as well as deterministic deployment methods.

In the Table 1 are provided typical values used in configuration of the Deployment model in our experiments. In this example, on each node is deployed one client and the services are distributed randomly across all of the nodes.

# 5 Simulation scenarios

The simulation scenarios are created by a configuration generator that creates scenario configurations based on a set of parameters of system characteristics. In addition to the network parameters configured through NS-3's configuration (e.g., number of nodes, mobility, wireless link characteristics, etc.), the services, clients, and their interactions and behaviors, including how and where the services are hosted, are configured for the above models. During the simulation run, the simulator records events, such as service message exchanges and fault symptoms, into trace files for analysis. Figure 11 illustrates the configuration and runtime components of the simulator.

In the Table 1 are provided typical values used in configuration of our experiments. Aside of configuration parameters of the service layer models, the table also contains configuration parameters of NS-3 defining the underlying network layer.

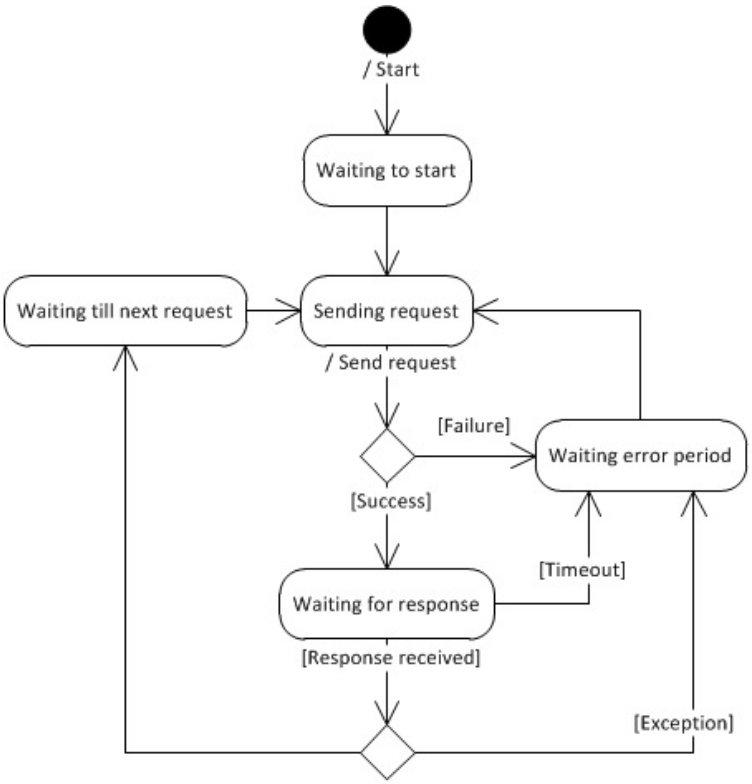


Figure 9: State-machine of the client workload algorithm

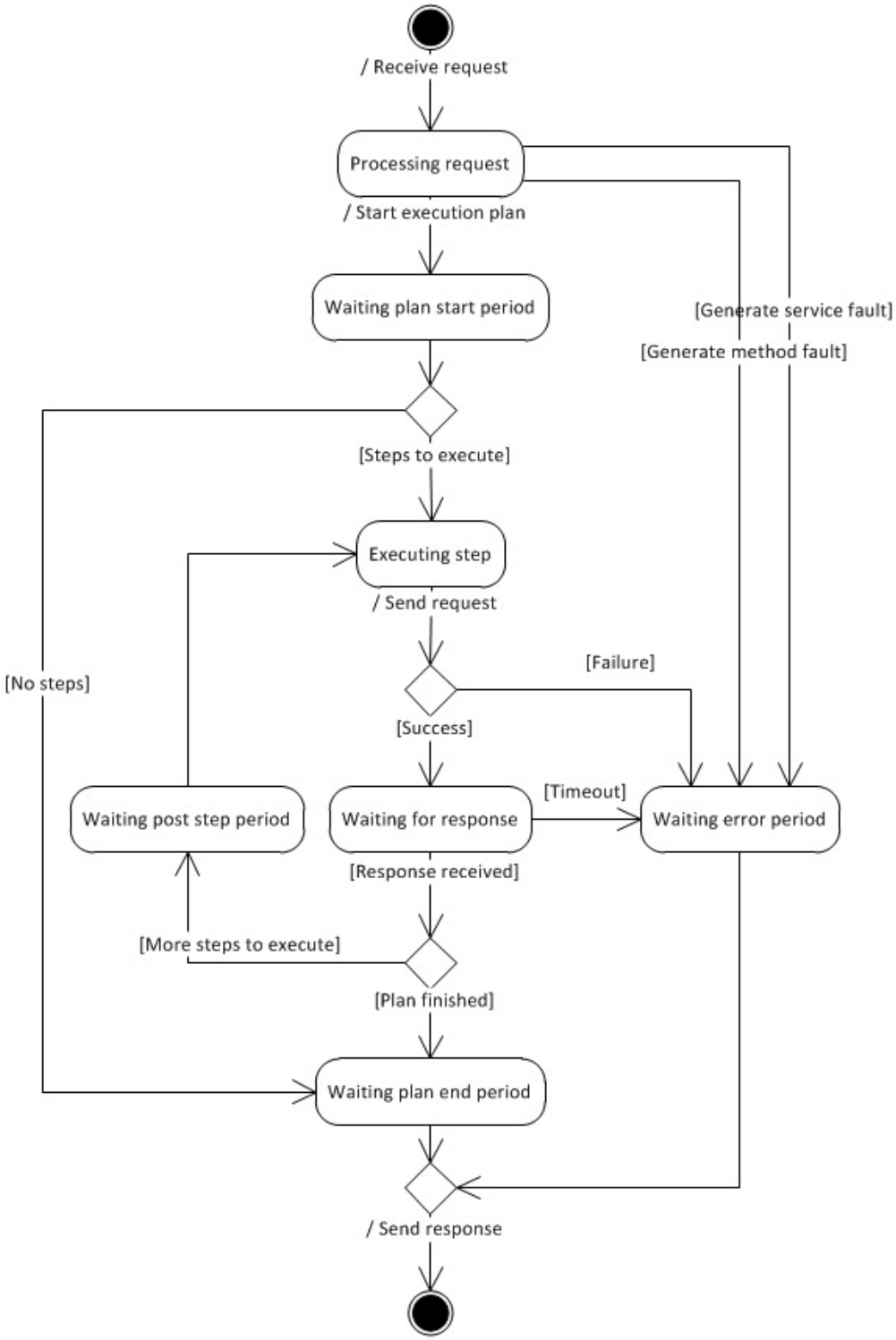


Figure 10: State-machine of the service workload algorithm

| | |
|---|---|
| Entity model | |
| Number of clients | 50 |
| Number of contracts | 30 |
| Number of methods in contract | 2 |
| Number of services per contract | 5 |
| Message model | |
| Request message size | 500-1500 bytes |
| Response message size | 500-1500 bytes |
| Response timeout | 60 000 ms |
| End-point - transport protocol | UDP |
| ACK timeout | 1000 ms |
| Message retransmission limit | 5x |
| Interconnection model | |
| Service topology type | 2-tier |
| Number of front-end contracts | 5 |
| Client to front-end contract method connectivity probability | 0.5 |
| Service method to contract method connectivity probability | 0.01 - 0.1 |
| Service discovery method | hop count |
| Workload model - client | |
| Request rate | 5000-15000 ms |
| Next request after conversation fails | 5000 ms |
| Step selection method | based on step probability |
| Step invocation probability | 0.01 to 1 |
| Workload model - service | |
| Method start delay | 20 |
| Method end delay | 20 |
| Method step delay | 10 |
| Method error delay | 10 |
| Fault model | |
| Service fault model | OnOffRate |
| Deployment model | |
| Client to node | 1 on 1 |
| Service to node | random |
| Network layer | |
| Number of nodes | 50 |
| Spatial bounds | 75m x 75m |
| Mobility speed | 10 m/s |
| Mobility model | RandomDirection2dMobility |
| Propagation delay model | ConstantSpeedPropagationDelay |
| Propagation loss model | LogDistancePropagationLoss/$\alpha$3 |
| WiFi standard | 80211b |
| WiFi rate | 11Mbps |
| Routing protocol | OLSR and Ipv4StaticRouting |
| Protocol stack | UDP/IPv4 |

Table 1: Main simulator configuration parameters of a scenario

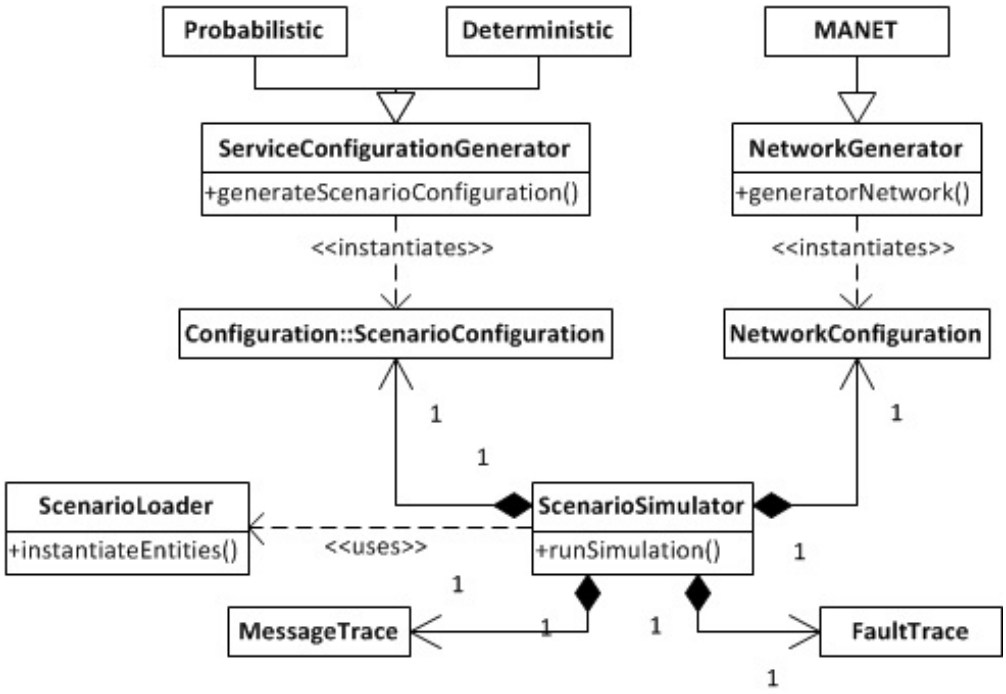


Figure 11: Class diagram of the simulator configuration and runtime components

# 6 Conclusion

In this paper we have introduced our simulator of service-based systems hosted in networks with dynamic topology. The simulator is built as an extension of a standard packet based network simulator NS-3. The simulator thus closely replicates the complex network behavior as well as the service-based system entities and models. We have used the simulator for evaluation of several system management methods such as in method of discovery of software service dependencies in MANETs [4, 2], method of fault localization in service-based systems hosted in MANETs [3, 1], and in analysis of behavior of hybrid wireless networks [6].

To our knowledge, there is currently no other comparable tool providing functionality of network and service layers simulation. Therefore, we believe that the simulator can be valuable tool for many other researchers as well.

# Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

# References


[1] P. Novotny. *Fault localization in service-based systems hosted in mobile ad hoc networks*. PhD thesis, Imperial College London, 2014.

[2] P. Novotny, B. J. Ko, and A. L. Wolf. On-demand discovery of software service dependencies in MANETs. *IEEE Transactions on Network and Service Management*, 12(2):278–292, June 2015.

[3] P. Novotny, A. L. Wolf, and B. J. Ko. Fault localization in MANET-hosted service-based systems. In *31$^{st}$ IEEE International Symposium on Reliable Distributed Systems*, pages 243–248, Oct. 2012.

[4] P. Novotny, A. L. Wolf, and B. J. Ko. Discovering service dependencies in mobile ad hoc networks. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 527–533, May 2013.

[5] W. She, I.-L. Yen, and B. Thuraisingham. WS-Sim: A web service simulation toolset with realistic data support. *Computer Software and Applications Conference Workshops*, 0:109–114, 2010.

[6] S. Tati, P. Novotny, B. J. Ko, A. L. Wolf, A. Swami, and T. La Porta. Diagnosing degradation of services in hybrid wireless tactical networks. In *SPIE Defense, Security, and Sensing*, May 2013.