Emulating Loss and Delay for AD HOC Networks

Alaa Seddik-Ghaleb
Networks and Multimedia
Systems Research Group (LRSM)
Institut d'Informatique d'Entreprise (IIE)
18 alle Jean Rostand, 91025 Evry
CEDEX - France
Email: seddik@iie.cnam.fr

Yacine Ghamri-Doudane
Networks and Multimedia
Systems Research Group (LRSM)
Institut d'Informatique d'Entreprise (IIE)
18 alle Jean Rostand, 91025 Evry
CEDEX - France
Email: ghamri@iie.cnam.fr

Sidi-Mohammed Senouci
France Telecom R&D
2 Av. Pierre Marzin, 22307
Lannion, France
Email: sidimohammed.senouci@orange-ft.com

***Abstract*— Ad hoc networks have gained place in the research area recently. There are many researches regarding different topics related to such networks, such as routing, media access control, security, scalability, and many others. There are two usual methods to test and evaluate the performance of ad hoc networks: simulations and real test-bed. A network emulator is a tradeoff between pure simulation and real test-bed. Here, we propose a multi-hop wireless ad hoc network emulator that uses the advantages of Network Simulator (NS-2) and traffic shaping (DummyNet), that is called SEDLANE. SEDLANE is a network emulator that is based on TCP behavior characteristics. Using SEDLANE, we can emulate a whole multi-hop ad hoc network through the data packet loss and Round Trip Time (RTT) values over the TCP connection. SEDLANE helps testing and evaluating ad hoc network protocols using very simple and inexpensive test-bed configuration. The results confirm that; introducing SEDLANE within a simple configuration network (possibly 2 nodes) gives exactly the same results as having a wireless mobile multi-hop ad hoc network of any size.**

I. Introduction

A wireless ad hoc network consists of independent nodes that communicate through the wireless channel without the need of network infrastructure or a centralized administration. Due to its specific nature, many researches have been developed so that new network protocols and applications are introduced. Thus, it is important to evaluate these new protocols and applications within an ad hoc network environment. Generally, the following two ways can be used in order to achieve that evaluation:

- Building a real test-bed for ad hoc networks with the desired network conditions, protocols and applications. Although that this method is very realistic, it is expensive to setup. Actually, there is no researches has been done in a scale beyond a dozen nodes.
- Using network simulation, such as NS-2 [1]. Simulating a network is always more easy than constructing a real test-bed. On the other hand, the traffic used in the simulator is generated by traffic models that some times do not match real application behavior. Also, some performance parameters can not be evaluated through simulations (mainly, node related performance parameters such as CPU usage and computational energy consumption).

TCP throughput could be calculated using the RTT delay and data packet loss over the connection. Floyd et al. [2] and Ott et al. [3] propose a model that estimates the throughput of a TCP connection under known delay and loss conditions as follows:

$$r_{TCP} = \frac{1.22 * M}{\tau * \sqrt{l}} \qquad (1)$$

Where: r_{TCP} is the TCP connection throughput, M is the maximum packet length, τ is the round trip time of the connection and l is the average loss measured during the lifetime of the connection. From such model, one can say that TCP performance depends strongly on RTT values and loss rates.

In this paper, we propose a multi-hop wireless ad hoc network emulator that uses the TCP trace files of NS-2 [1] in order to introduce the effect (loss rates, and RTT values) of multi-hop wireless ad hoc network into a well-known traffic shaping tool "Dummynet" [4]. If we take into consideration that most network experimentation begin with simulations, it would be useful to extend the simulation results beyond the scope of the simulator capabilities by using an emulator that uses the same network circumstances (through the network simulator trace file). As it is easier to control the network environment using simulation tool, we take this advantages by generating the desired scenarios within an ad hoc network environment using NS-2 (as a simulation tool) and then we inject its trace files into dummynet (as a traffic shaping tool) to have the same effects on a real traffic scenarios as if we had constructed a real multi-hop wireless ad hoc network environment. Hence, we obtain a Simple Emulation of Delay and Loss for Ad Hoc Networks Environment (SEDLANE).

SEDLANE allows to emulate a multi-hop wireless ad hoc network of any scale in a virtual way and with any mobility scenario without the need of moving (physically) the ad hoc nodes. SEDLANE does not need any additional knowledge of a new emulating tool. It uses a well known and free network simulation (NS-2) and traffic shaping (Dummynet) tools. Additionally, it is an inexpensive tool and does not need a special hardware setup. This tool enables us to test the network protocols performance at the end points of any wireless ad hoc network (emulated by SEDLANE). The entire wireless ad hoc network environment is emulated using only one machine (i.e. a particular machine installed in the middle of the two communication end points or even one of the communication end-points can also play this role). SEDLANE represents network nodes' mobility, ad hoc routing protocol, and TCP connection throughput through RTT values (delay) and data packet loss within the network.

Our goal is to propose a simple emulator that helps in evaluating new ad hoc network protocols and applications that are running above TCP. From that point of view, we characterize a useful ad hoc network emulator by the following features:

- Simple and easy to implement.
- Does not require any specific or expensive networking hardware.
- The tests must be controlled and repeatable.
- The ability to emulate different ad hoc network parameters (ad hoc routing protocols, nodes' mobility, and TCP connection throughput) using a small set of emulated variables (delay and loss)
- Emulating multi-hop wireless ad hoc network of any size using a small number of physical machines.

The remainder of this paper is organized as follows: after presenting the motivation behind our work in section II, section III overviews dummynet and its main function, section IV presents SEDLANE. In sections IV-A and IV-B we describe the main operation modes of SEDLANE and the calculation algorithms used. Section V introduces the validation results of SEDLANE. Finally, we summarize main results and give some ideas for future work in section VI.

II. Related Work

According to the above ad hoc emulator features, we will discuss, in this section, the most known emulators in order to find the one that meets our requirements. User Mode Linux (UML) [5] is a system that can be used for emulating wireless networks. It allows using several independant Linux instances each running an adapted version of a complete Linux kernel in user mode, providing a shared network environment through the base Linux system. In [6] the authors stats that, UML emulation performance is severely reduced, when used to create wireless ad hoc network emulations, due to the fact that UML runs in user mode and privileged operating system functions must be emulated by the underlying kernel running on the real hardware. In addition, UML offers only a virtual Ethernet interface and not a virtual WLANN wireless interface (i.e. a 802.11b interface). UML itself emulates only a single wireless node. To emulate an entire network, a central controlling instance is required to monitor and control all UML instances. MobiEmu [7] is one of such central control solutions that emulates a basic wireless network based on UML. The concept that MobiEmu has separate physical or virtual machine for each emulated node, made UML and MobiEmu are not the best choice for our work.

ModelNet [8] was initially developed for testing large-scale distributed services for wired wide-area network environments. ModelNet architecture is composed of Edge Nodes and Core Nodes. Edge Nodes in ModeNet can run arbitrary architecture and operating systems. They run native IP stacks and function as they would in real environments with the exception that they are configured to route IP traffic through ModelNet cores. While, Core Nodes run a modified version of FreeBSD to emulate topology-specific hop-by-hop network characteristics. To decrease the number of edge machines required for large-scale evaluations, ModelNet architecture implies Virtual Edge Nodes (VNs). VNs enable the multiplexing of multiple application instances on a single edge machine, with each instance getting its own unique IP address. ModelNet edge machines use internal IP addresses (10.*), thus the number of interfaces that can be multiplexed onto an edge node is not limited by IP address space limitations, but rather by the amount of computational resources (e.g. threads, memory) that the target application uses. ModelNet configures all VNs to route their traffic through a particular ModelNet core.

Like ModelNet, MobiNet [9] architecture is composed of edge nodes and core nodes. The edge nodes support a variety of platforms and operating systems. While we perform our current experiments on edge nodes running Linux, our edge nodes could be a combination of different devices like laptops, PDAs, etc. running different operating systems. As in ModelNet, edge nodes in MobiNet host multiple virtual nodes (VNs) to allow for large-scale emulations. MobiNet cores emulate wireless network behavior at multiple layers while eventually routing packets to the edge node hosting the destination VN. MobiNet incorporates mobile wireless ad hoc network characteristics. First, MobiNet emulates nodes' mobility behavior, i.e., different movement patterns of nodes in the topology. Second, it implements a routing module that tracks the position of nodes and maintains a list of nodes within transmission range for each node. The routing module is responsible for finding routes to destination nodes as nodes in the topology follow different movement patterns. Third, MobiNet accounts for MAC layer collisions. Effects of packet losses due to collisions in the MAC layer play an important role in wireless networks, thus requiring MobiNet to emulate MAC layer behavior. Although that, the number of physical devices required in MobiNet is reduced, and the platform seems to be well developed for a mobile wireless ad hoc environments emulation, its setup is still complicated, with regards to our requirements.

Mobile Network Emulator (MNE) [10] uses a static network infrastructure to interconnect devices. Each device has two interfaces, where one acts as a mobile emulation control channel while the other is used for the emulated wireless network. The latter can be an actual wireless interface, allowing for some lower layer effects (such as collisions) to be taken into account as well. Information about topology changes is sent through the control channel, causing the nodes to set or remove iptables-rules accordingly, as it is done in MobiEmu. The main problem of this approach is that it still needs a separate device for each emulated wireless host.

EMWIN [11] improves the issue with the number of physical machines by allowing each node to have several network interfaces, each acting as a separate wireless node. EMWIN intends to provide emulation of some MAC layer effects by introducing an additional emulated MAC (eMAC) layer. Again, due to a relatively high number of machines required, this approach is still impractical for our needs.

JEmu [12] represents another emulation system for mobile ad hoc networks. However, It has a limited scalability due to the fact that each emulated node must run on a separate physical machine. That seems impractical for us. In [13] the authors propose an approach for wireless networks and applications experimentations using a real MAC layer. Though that a real MAC layer could be an advantage, scalability stays limited.

NEMAN [14] is designed to emulate a relatively large scale wireless network, up to hundreds of nodes, within a single physical machine. With that respect, NEMAN is closest to MobiNet. On the other hand, the number of emulated nodes is also limited by the physical machines resources.

In addition, none of the above emulations emulates an ad hoc network taking into consideration TCP connections characteristics (data packet loss and RTT delay). As mentioned earlier, emulating TCP connection throughput depends on these values.

III. Overview of Dummynet

Dummynet is a traffic shaping tool that have been originally designed for testing networking protocols [4]. Through dummynet, we can enforce delays, packet losses, queue and bandwidth limitations. Figure 1 illustrates the main function of dummynet. Dummynet is entirely controlled by the system's *ipfw* (IP Fire-Wall) commands and a set of sysctl (System Control) variables. The *ipfw* commands help the user defining the rules to be applied by dummynet on the packets crossing a particular network interface on its input or output. Unlike many other traffic shaping packages, dummynet has a very little overhead. All processing is done within the kernel and no data copying involved in moving packets through pipes. Dummynet implements the concept of pipes, which is defined as a communication channel between the source and the destination. It is able to handle thousands of pipes. Also, any packet could be influenced by several rules. Regarding the rules associated to each pipe (communication channel), the packets will be manipulated. We can use the same command to configure different network parameters, such as bandwidth, delay, queue size, and packet loss. For more details of dummynet configurations and control, refer to [4].

Our choice of using dummynet was based on the fact that, dummynet helps implementing data packet loss and traffic delay constrains easily within the system's kernel. Additionally, with minimum processing overhead.

IV. Simple Emulation of Delays and Losses for Ad hoc Networks Environment [SEDLANE]

The main idea of SEDLANE is to configure the dummynet pipes (defining rules) through NS-2 trace files. To do so, SEDLANE uses the NS-2 TCP trace file to identify the classes of packets by gathering together the packets that have almost the

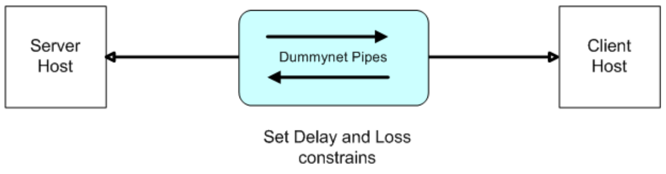


Fig. 1. The principle of Dummynet operation

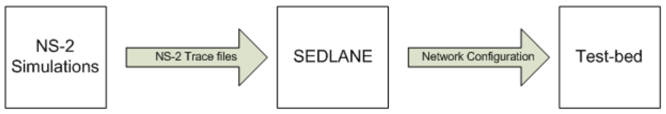


Fig. 2. The principle of SEDLANE operation

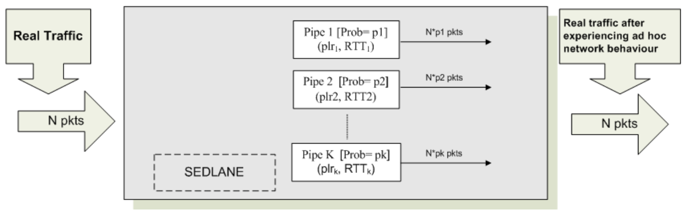
similar values of RTT. Then, SEDLANE dedicates one pipe, or communication channel, for each group of packets; respecting their proportion to the total number of packets within the trace file. Packet loss percentage can be either calculated from NS-2 TCP trace file or NS-2 standard trace file; and then can be applied to dummynet pipes. Hence, according to the identified classes of packets, the proportion of packets per-class, and the loss rates that are distributed among classes, SEDLANE will in the following dynamically generate the dummynet rules to be applied on the packets. Figure 2 illustrates the SEDLANE operations while the different algorithms used by SEDLANE are described below.

A. SEDLANE Operation Modes

We have two operation modes in which we can run SEDLANE; the choice of using them depends on the user's experimentation methodology. We will show in the evaluation study the advantages of each of these modes.

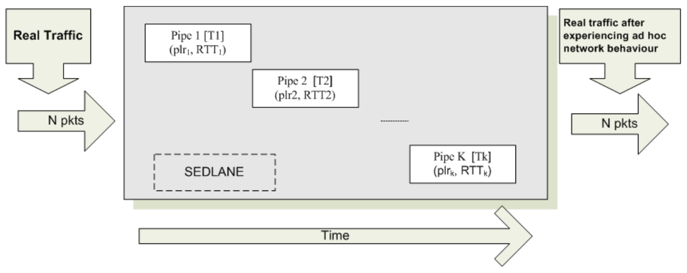
1) Simultaneous operation mode: In this mode of operation, SEDLANE configures all *ipfw* rules (dummynet communication channels) at the same time, assigning each pipe a different probability value that corresponds to the amount of traffic sent with each RTT (as extracted from the NS-2 TCP trace file). SEDLANE operates in simultaneous mode by default. This mode reflects the normal operation mode of the system's *ipfw*. Figure 3 describes this operation mode. as can be shown from the Figure, using simultaneous operation mode implies all the *ipfw* rules without time constrains. Thus, if the user needs to emulate an ad hoc network for a time greater than that of the simulation scenario, SEDLANE Simultaneous mode will be the right choice.

2) Sequential operation mode: In Sequential operation mode, as can be seen from Figure 4, SEDLANE configures only one *ipfw* rule at a time. Each rule will be flushed after a certain "lifetime" (the time during which the simulated connection stayed at an RTT value) before a new rule (with a new RTT and loss value) is configured. The "lifetime" of each rule can be either provided as a command line argument or extracted from the NS-2 TCP trace file. In the first case, the provided lifetime will be applied to all rules, one after another. Whereas in the latter case, each rule will have a different



Real Traffic
N pkts
Pipe 1 [Prob= p1] (plr_1, RTT_1)
N*p1 pkts
Pipe 2 [Prob= p2] (plr2, RTT2)
N*p2 pkts
Pipe K [Prob= pk] (plr_k, RTT_k)
N*pk pkts
SEDLANE
Real traffic after experiencing ad hoc network behaviour
N pkts

Fig. 3. SEDLANE Simultaneous operation mode



Real Traffic
N pkts
Pipe 1 [T1] (plr_1, RTT_1)
Pipe 2 [T2] (plr2, RTT2)
Pipe K [Tk] (plr_k, RTT_k)
SEDLANE
Real traffic after experiencing ad hoc network behaviour
N pkts
Time

Fig. 4. SEDLANE Sequential operation mode

lifetime corresponding to that in the NS-2 TCP trace file. The algorithm used to calculate the sequential delay values from the TCP trace file will be explained later. In this operation mode, SEDLANE emulates exactly the data found in the NS-2 TCP Trace file, respecting the time of the simulation scenario. This time can be controlled, if the user find that it necessary, by specifying it at the command line. In some cases, the life time of a RTT transition is so small that the effect can not be noticed. Then, increasing that time may help making this effect more clear.

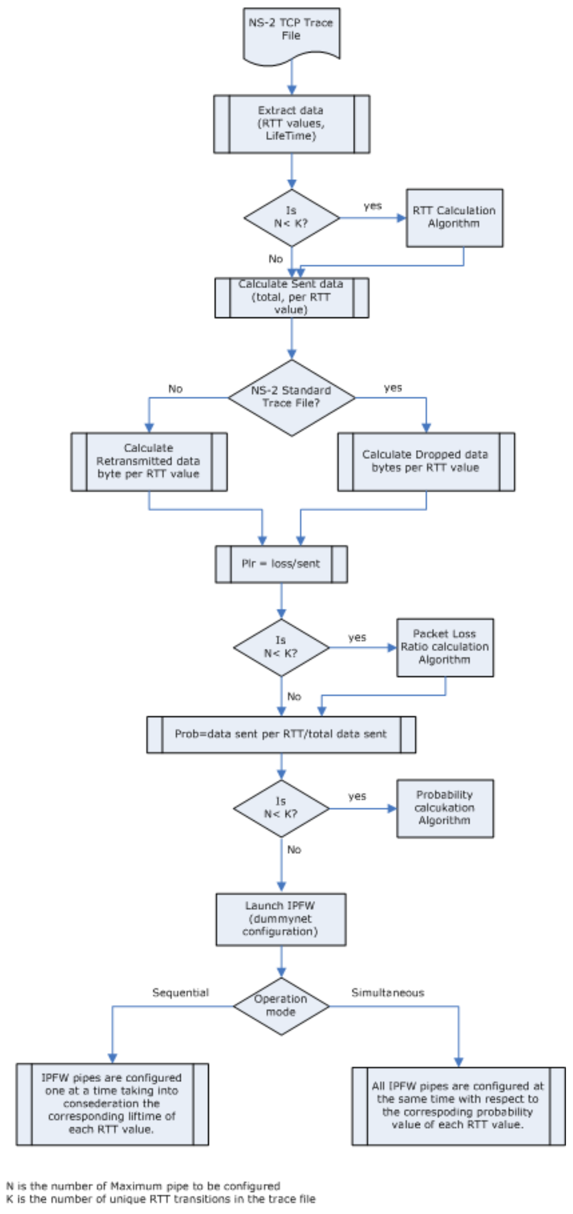
B. SEDLANE Algorithms

SEDLANE starts by reading NS-2 trace files, specified by the user at the command line. According to data contained at these file and command line arguments, SEDLANE decides whether to trigger some calculation algorithms. Figure 5 shows the principal SEDLANE calculation algorithms. These algorithms will be discussed in details by the following.

1) Input Data from trace files: SEDLANE extracts the following data from the NS-2 trace files specified by the user at the command line.

- Number of different RTT samples found in the file.
- Timestamp of each RTT transition
- Total data bytes transmitted and that of each RTT.
- Total data bytes retransmitted and that of each RTT.
- Total connection time.
- Lifetime for each RTT.
- Total data bytes dropped and that of each RTT.

2) Number of Pipes used by SEDLANE: This argument defines the number of rules to be configured according to the desired accuracy. Note that configuring a large number of rules requires more CPU resources might affect the performance of the test-bed. If the "number of different RTT samples found



NS-2 TCP Trace File
Extract data (RTT values, LifeTime)
Is N< K?
yes
RTT Calculation Algorithm
No
Calculate Sent data (total, per RTT value)
NS-2 Standard Trace File?
No
yes
Calculate Retransmitted data byte per RTT value
Calculate Dropped data bytes per RTT value
Plr = loss/sent
Is N< K?
yes
Packet Loss Ratio calculation Algorithm
No
Prob=data sent per RTT/total data sent
Is N< K?
yes
Probability calcukation Algorithm
No
Launch IPFW (dummynet configuration)
Operation mode
Sequential
Simultaneous
IPFW pipes are configured one at a time taking into consederation the corresponding liftime of each RTT value.
All IPFW pipes are configured at the same time with respect to the correspoding probability value of each RTT value.
N is the number of Maximum pipe to be configured
K is the number of unique RTT transitions in the trace file

Fig. 5. SEDLANE Operation Functions

in the TCP trace file" is equal to or less than this argument. SEDLANE dedicates a dummynet pipe for each RTT value. If not so, SEDLANE calculates new RTT values that correspond to the maximum number of pipes to be used given by the user. Calculating the new RTT values is done through RTT calculation algorithm, which will be explained in the following section.

3) RTT Calculation Algorithm: If the "number of different RTT samples found in the TCP trace file" is greater than "maximum number of pipes" defined by the user. SEDLANE starts RTT calculation algorithm. This algorithm is used to calculate new RTT samples from the original ones within the TCP trace file. These new values are calculated as shown in Figure 6. Where; $RTT_{new1,2,..,n-1}$ are the new generated RTT

$$RTT_{new_1} = \frac{RTT_i + RTT_{i+1}}{2}$$
$$RTT_{new_2} = \frac{RTT_{i+1} + RTT_{i+2}}{2}$$
$$\cdots\cdots\cdots$$
$$RTT_{new_{(n-1)}} = \frac{RTT_{(n-1)} + RTT_{(n)}}{2} \qquad (2)$$

Fig. 6. SEDLANE RTT Calculation Function

$$prob_{RTT_i} = \frac{txbytes_{RTT_i}}{txbytes_{total}} \qquad (3)$$
$$prob_{new_i} = \left(\frac{prob_i + prob_{i+1}}{2}\right)\left(\frac{N}{N-1}\right) \qquad (4)$$
$$prob_{adapt_i} = prob_{new_i} \cdot \frac{1}{\sum prob_{new}} \qquad (5)$$

Fig. 7. SEDLANE Probability Calculation Function

samples; RTT_i and RTT_{i+1} are two original consecutive RTT samples.

SEDLANE repeats the calculations until the new number of RTT samples matches the defined number of pipes. It is possible that the number of resulting *ipfw* rules be less than the provided number of pipes. This comes from the fact that SEDLANE does not allow configuring two consecutive pipes with equal RTT values. Additionally, in simultaneous mode, only unique RTT values are allowed, thus there will be one rule per each unique RTT value.

4) Probability Calculation Algorithm: Probability is used by dummynet to represent the probability of getting a match on this rule if all other fields are correct. The probability assigned to each *ipfw* rule depends on the SEDLANE's operation mode: either sequential or simultaneous. In sequential mode, each rule will have a deterministic probability; since there is only one rule present at a time. In simultaneous mode, the probability assigned to each *ipfw* rule is calculated as shown in Figure 7. Where; $prob_{RTT_i}$ is the assigned probability for RTT_i, $txbytes_{RTT_i}$ is the data transmitted during the RTT_i Lifetime, and $txbytes_{total}$ is the total data transmitted over the emulated connection, $prob_{new1}$, is the new generated probability; $prob_i$ and $prob_{i+1}$ are two original consecutive probability values; N is the length of original probability list; $N-1$ is the length of the new generated probability list, $prob_{new}$ is a new generated probability value from Equation (4), $prob_{adapt1}$is the adapted probability value.

The first equation in Figure 7 demonstrates how probability values are calculated for each RTT sample. The probability assigned to each rule is the ratio of the transmitted data bytes during a certain RTT lifetime to the total number of data bytes transmitted over the connection, as extracted from the NS-2 TCP trace file. In case, the number of RTT transitions in the trace file is higher than the provided number of pipes, the probability of each new RTT value is calculated according to equation (4). The sum of the newly generated values is always less than 1. Yet, since these are probability values, the values will be adapted again so that their sum equals 1. This is done by distributing the difference among the different probabilities according to their respective values, as shown in

$$txbytes_{RTT(i)new} = \frac{txbytes_{RTT(i)} + txbytes_{RTT(i+1)}}{2} \qquad (6)$$
$$txbytes_{newTotal} = \sum txbytes_{RTT(i)new} \qquad (7)$$
$$txbytes_{RTT(i)adapted} = \frac{txbytes_{RTT(i)new} . txbytes_{total}}{txbytes_{new\,Total}} \qquad (8)$$

Fig. 8. SEDLANE Data Bytes Transmitted Calculation Function

the third equation of the above Figure.

The above process will continue till we have a number of probability values that is equal to the number of pipes to be configured.

5) Packet Loss Ratio Calculation Algorithm: In order to calculate the packet loss ratio for each *ipfw* rule, SEDLANE performs the following operations:

- Calculate the amount of data transmitted with each RTT.
- Calculate the amount of data dropped or retransmitted during the lifetime of each RTT.
- Calculate the PLR for each RTT as the result of dividing the corresponding amount of loss by the amount of data sent.

These operations are detailed below.

Calculating the amount of transmitted data

SEDLANE calculates the amount of transmitted data for each RTT: $txbytes_{RTT}$, as well as the total amount of data transmitted: $txbytes_{total}$. If the number of RTT values retrieved from the NS-2 TCP trace file file is greater than the provided number of pipes, the following iteration takes place: A new list of $txbytes_{RTT}$ values is generated using the fist Equation shown in Figure 8.

The algorithm then calculates the new sum of transmitted data [as in Equation (7)]. Then, the algorithm maintains the original total amount of transmitted data. This is done by distributing the difference between the original sum $txbytes_{total}$

$$plr_{RTT(i)} = \frac{lostbytes_{RTT(i)}}{txbytes_{RTT(i)}} \qquad (9)$$

Fig. 9. SEDLANE Packet Loss Ratio Calculation Function

$$RTTi_{lifeTime} = TS_{RTT(i+1)} - TS_{RTTi} \qquad (10)$$

Fig. 10. SEDLANE Sequential Delay Calculation Function

and the new sum $txbytes_{newTotal}$ among the elements of the new generated list according to their respective values [Equation (8)]. Thus, after each iteration round, the total amount of transmitted data will always equal $txbytes_{total}$. The iteration above will continue till we have a number of $txbytes_{RTT}$ values that is equal to the number of pipes to be configured.

Calculating the amount of lost data

SEDLANE adopts two methods to calculate lost data. By default, SEDLANE uses NS-2 TCP trace file to calculate the data packet loss ratio. It records the number of retransmitted bytes during each RTT lifetime, and uses this value as the amount of lost data. The retransmitted data bytes include data bytes retransmissions due to retransmission time out (RTO) and fast retransmit (FR). Alternatively, SEDLANE is able to calculate the data packet loss ratio from the standard NS-2 trace file. SEDLANE counts the amount of dropped data bytes during the lifetime of each RTT, and uses this value as the number of lost data. In case the number of RTT values retrieved from the NS-2 TCP trace file is greater than the provided number of pipes, SEDLANE executes the algorithm explained in the previous section (IV-B.5)and maintains the total amount of lost data similarly.

Calculating the Packet Loss Ratio

After determining the amount of transmitted data and lost data for each pipe. Calculation of PLR is simple [Figure 8]. Where; $plr_{RTT(i)}$ is the packet loss ratio of RTT_i; $lostbytes_{RTT(i)}$ and $txbytes_{RTT(i)}$ are the lost data bytes and transmitted data bytes of RTT_i respectively.

6) Sequential Delay Calculation Algorithm: Sequential delay is the Lifetime of each RTT. This argument could be calculated from NS-2 TCP trace file or provided by the user as a command line argument. If entered by the user, this value will be used for all configured *ipfw* pipes. Meaning that all the RTTs will have the same Lifetime duration. If calculated from NS-2 TCP trace file file [Figure 10]. Where; $RTTi_{(lifTime)}$ is the life time of RTT_i; TS_{RTTi} and $TS_{RTT(i+1)}$ are the timestamps of RTT_i and RTT_{i+1} respectively.

V. SEDLANE Validation

In order to validate SEDLANE, we tested it with different network scenarios. The validation test-bed is shown in Figure 11. We use TTCP as a TCP traffic generation tool between the source and destination. The laptop in the middle (SEDLANE) is configured to be the gateway between the two others. Thus, we guarantee that the traffic exchanged between the end point laptops must pass through SEDLANE, where we impose the emulation rules. We send 16Mbytes TCP data using TTCP. We analyze the traffic using Tcpdump [15] and Tcptrace [16]. Tcpdump is an open source powerful tool that allows us to sniff network packets and make some statistical analysis out of those dumps. Tcptrace is a tool for analysis. It can take the files produced by several popular packet-capture programs, including Tcpdump as input. Tcptrace can produce several different types of output containing information on each connection seen, such as elapsed time, bytes and segments sent and received, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis.

We configured our simulation scenarios in NS-2 as follows: each simulation consists of a 20 nodes network confined in a (670m x 670m) area. 14 TCP connections were established (ftp traffic used with a packet size of 1000 bytes). The simulation time is set to 400 seconds. We used OLSR as an ad hoc routing protocol in our simulations. Also, we test both simultaneous and sequential operation modes of SEDLANE. In our simulations, we vary the nodes' mobility rate to get different loss and delay variations.

In order to evaluate SEDLANE performance, we compare the data extracted from NS-2 trace files to be used in *ipfw* pipes' configuration with SEDLANE results. The output results are captured by Tcpdump at the sender end point, and then analyzed by Tcptrace. In sequential operation mode, we examine two parameters; evolution of RTT values and average connection throughput between the end points. While in simultaneous operation mode, we test the probability distribution of RTT values with respect to the total amount of data transmitted.

The results prove that SEDLANE is capable of emulating effectively the delay and data packet loss of an ah doc network since it gives the same results as in the simulations scenarios.

A. SEDLANE Results

In this section, we analyze SEDLANE validation results obtained using the above test-bed configuration.

1) SEDLANE sequential operation mode: In this operation mode, *ipfw* pipes are configured and applied according to NS-2 TCP trace file. Thus, the RTT evolution applied by SEDLANE should follow the RTT transitions sequence in the same file. In addition, the time in which the RTT value will be applied on data packets should correspond to RTT life times within NS-2 TCP trace file. This is confirmed by Figures 13 and 15. When emulating loss and delay within an ad hoc network, it is expected to view the effect of these parameters on the end-to-end average throughput of the emulated connection. In the mean time, this behavior should be the same as in the simulation results. We calculate the average throughput from the NS-2 TCP trace file (simulation result) and compare

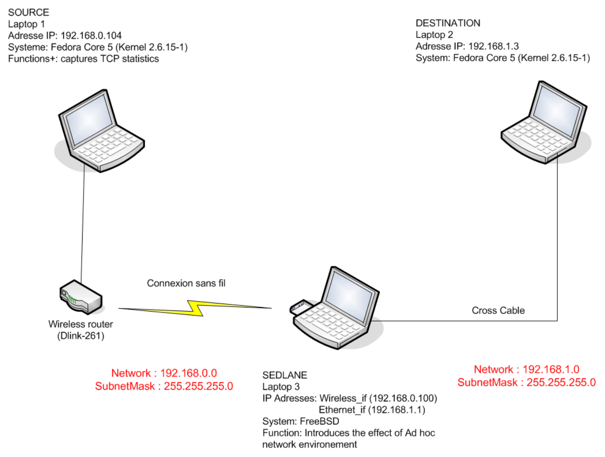
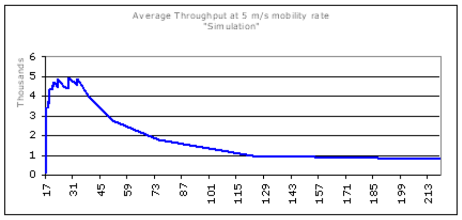

SOURCE
Laptop 1
Adresse IP: 192.168.0.104
Systeme: Fedora Core 5 (Kernel 2.6.15-1)
Functions+: captures TCP statistics
DESTINATION
Laptop 2
Adresse IP: 192.168.1.3
System: Fedora Core 5 (Kernel 2.6.15-1)
Connexion sans fil
Wireless router
(Dlink-261)
Cross Cable
Network : 192.168.0.0
SubnetMask : 255.255.255.0
SEDLANE
Laptop 3
IP Adresses: Wireless_if (192.168.0.100)
Ethernet_if (192.168.1.1)
System: FreeBSD
Function: Introduces the effect of Ad hoc network environement
Network : 192.168.1.0
SubnetMask : 255.255.255.0

Fig. 11. SEDLANE Validation Test-bed



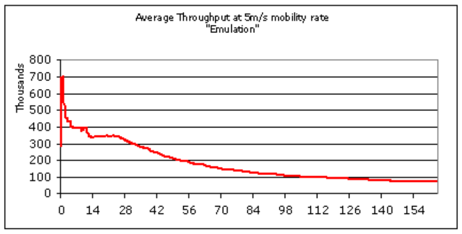
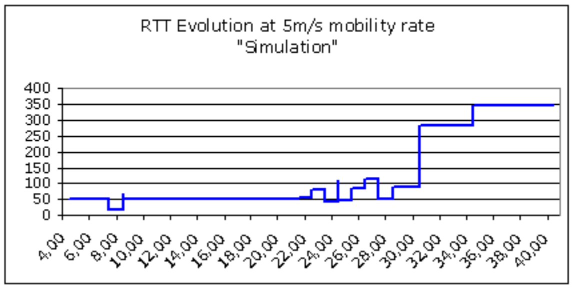


Fig. 12. Average Throughput at 5m/s mobility rate [sequential operation mode]

it with the average throughput of the communicating nodes in our test-bed (emulation result). The result is confirmed by both Figures 12 and 14. It can be shown from these figures that, the effect of loss and delay imposed by SEDLANE gives the degradation behavior as in the simulations regardless of the data transmission rate. We must note that, the total number of data transmitted by NS-2 simulation scenarios is not necessarily the same as in the test-bed scenarios. On the other hand, we get always the same average throughput behavior.

2) SEDLANE simultaneous operation mode: In SEDLANE simultaneous operation mode, the *ipfw* pipes are configured according to NS-2 TCP trace file but applied according to its corresponding probability. This probability reflects the amount of data transmitted for each RTT transition with respect to the total amount of data transmitted over the emulated connection. In this operation mode, SEDLANE follows the normal *ipfw* operation guidelines. Figure 16 proves that SEDLANE respects the RTT probability distribution of the used NS-2 TCP trace file used. Here again, the results are confirmed regardless



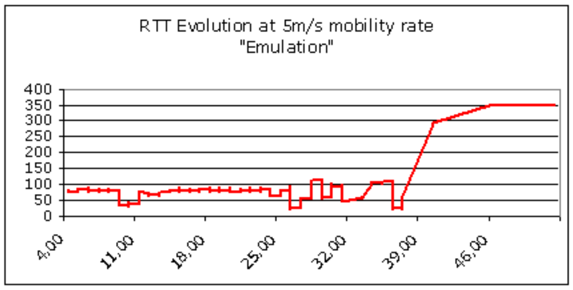


Fig. 13. RTT Evolution at 5m/s mobility rate [sequential operation mode]

of the amount of data used within the simulation scenarios. Note that simultaneous operation mode does not provide a fine grain emulation of the scenario represented by the TCP trace file.

VI. Conclusion

Mobile wireless ad hoc networks have gained more and more interests in the last decade. Many protocols and applications, targeting such networks, have been developed. In order to test and evaluate the performances of these protocols and applications, using simulation could do part of the work. On the other hand, having realistic test-bed configuration is very costly. From which, comes the importance of emulation tools. Emulating communication networks helps evaluating new network protocols and applications using less expensive test-bed configurations and real traffic. In this work, we propose new and simple emulation tool. This emulation takes the advantage of network simulation tools to control the desired network conditions. Then, using the simulations traces to introduce the same effects on real data traffic using simple and inexpensive test-bed configuration. SEDLANE uses both data loss and packets RTT values as network performance parameters from the simulation scenarios and reproduces the same effect of such networks by emulation. The validation results confirms that SEDLANE is capable of emulating the different network parameters and having an accurate network performance. The validation results show that, SEDLANE, as an ad hoc network emulator, can emulate a ad hoc network scenarios and gives same performance in terms of delay and data packet loss as same as in a network simulator. Thus, SEDLANE helps in testing and evaluating many ad hoc network features (such as mobility rates, ad hoc routing protocols, and transmission control protocol).

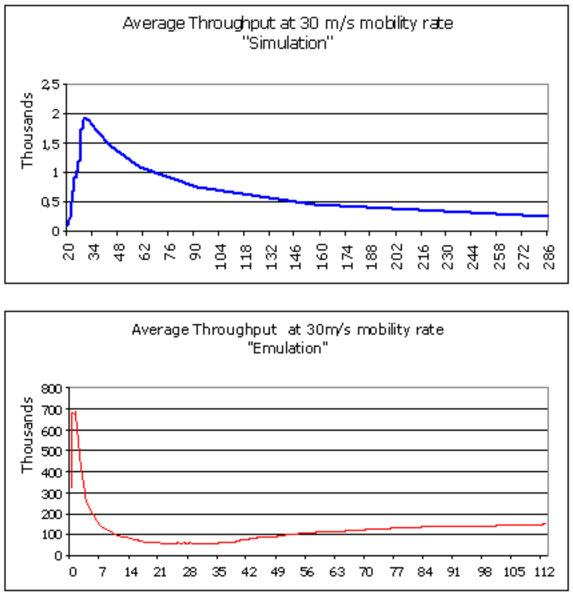


Fig. 14. Average Throughput at 30m/s mobility rate [sequential operation mode]

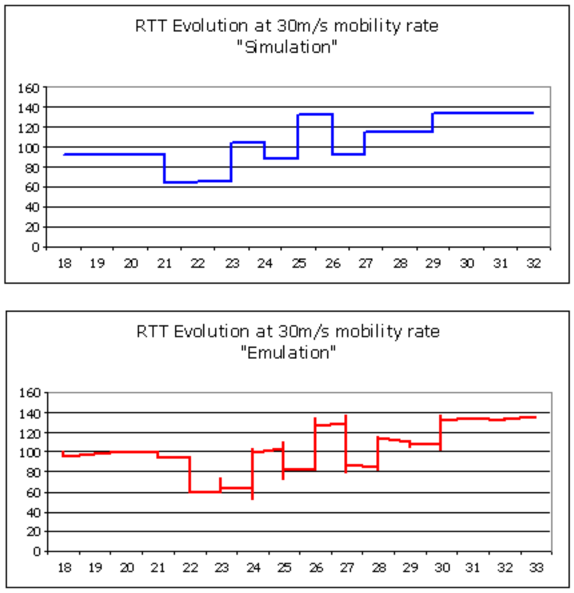


Fig. 15. RTT Evolution at 30m/s mobility rate [sequential operation mode]

In a future work, we think developing GUI (Graphical User Interface) for SEDLANE.

REFERENCES

[1] Network Simulator-NS-2. Available at *www.isi.edu/nsnam/ns/*

[2] S. Floyd and F. Kevin, *Router mechanisms to support end-to-end congestion control*. Technical report, February 1997

[3] T. Ott, J. Kemperman, and M.Mathis, *Window size behavior in TCP/IP with constant loss probability*. In The Fourth IEEEWorkshop on the Architecture and Implementation of Hi gh Performance Communication Systems (HPCS97), Chalkidiki, Greece, June 1997

[4] Dummynet. Available at *http://info.iet.unipi.it/ luigi/ip_dummynet/*

[5] J. Dike, *A User-Mode Port of the Linux Kernel*. 5th Annuel Linux Showcase Conference, Oakland, California, 2001

[6] M. Engel, M. Smith, S. Hanemann and B. Freisleben,*Wireless Ad-Hoc Network Emulation Using Microkernel-Based Virtual Linux Systems*. Proceedings of the 5th EUROSIM Congress on Modeling and Simulation, Marne la Vallee, France, pp. 198-203, EUROSIM Publishers, 2004

[7] Y. Zhang and W. Li, *An Integrated Environment for Testing Mobile Ad Hoc Networks*. Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking Computing, pp. 104-111, 2002

[8] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and David Becker, *Scalability and Accuracy in a Large-Scale Network Emulator*. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), December 2002

[9] P. Mahadevan, A. Rodriguez, D. Becker and A. Vahdat,*MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks*. Mobile Computing and Communications Review, Volume 10, Number 2, 2004

[10] Macker, J. P., Chao, W., Weston, J. W.,*A low-cost, IP-based mobile network emulator (MNE)*. MILCOM 2003 - IEEE Military Communications Conference, 2003, 22, 481-486

[11] P. Zheng and L. Ni, *EMWIN: Emulating a Mobile Wireless Network using a Wired Network*. In Proceedings of WOWMOM, September 2002

[12] J. Flynn, H. Tiwari, and D. O'Mahony, *A Real-Time Emulation System for Ad Hoc Networks*. In Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference, January 2002

[13] G. Judd and P. Steenkiste, *Using Emulation to Understand and Improve Wireless Networks and Applications*. In Proceedings of NSDI, May 2005

[14] M. Puzar and T. Plagemann, *NEMAN: A Network Emulator for Mobile Ad-Hoc Networks*. Technical Report no.321, ISBN 82-7368-274-9, Department of Informatics, University of Oslo, March 2005

[15] TCPDump. Available at *http://www.ethereal.com/docs/man-pages/tcpdump.8.html*

[16] TCPTrace. Available at *http://jarok.cs.ohiou.edu/software/tcptrace/*

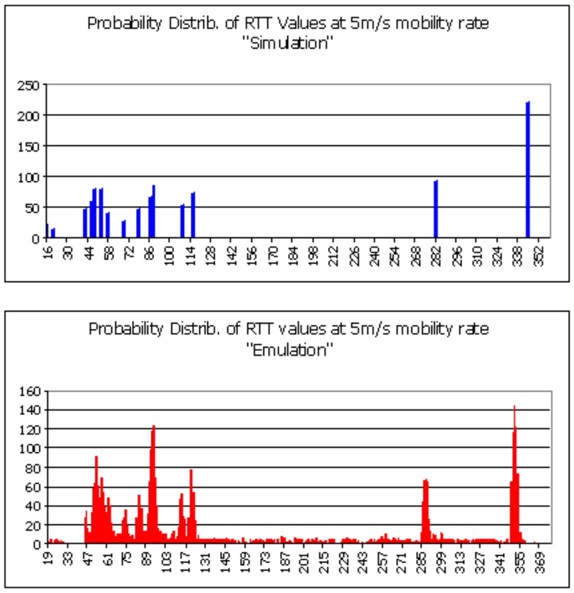


Fig. 16. Probability Distribution of RTT Values at 5m/s [simultaneous operation mode]