Emulation of Wireless Multi-Hop Topologies with Online Mobility Simulation

Anders Nickelsen
Department of Electronic Systems
Aalborg University, Denmark
Email: an@es.aau.dk

Hans-Peter Schwefel
Department of Electronic Systems
Aalborg University, Denmark
and Forschungszentrum Telekommunikation Wien - FTW
Vienna, Austria
Email: hps@es.aau.dk

***Abstract*—Communication in wireless networks is affected by uncontrollable disturbances in the channel. Effects of these disturbances are exacerbated in networks with dynamic topologies and multiple hops. Lack of control of the channel complicates testing applications in such networks as test conditions are hard, or impossible, to reproduce. This paper describes a test-bed to create reproducible test conditions for applications by emulating the wireless links. Emulation is performed by a topology emulator to which end-nodes are connected using wired links. In real-time, the emulator drops or delays packets in traffic between end-nodes. These imposed link properties are based on simulations of node mobility, loss and delay models. Two versions for performing the simulations are described; an offline version in which the mobility traces and link properties are calculated beforehand, and an online version where geographic trajectories are depending on the outcome of the communicating applications. Evaluation confirms that both versions of the test-bed are capable of emulating links in real-time and transparently to upper layer protocols. Additional delays from packet processing and bandwidth limitations introduced by using the emulator are shown meet the transparency requirements, also when the emulator is heavily loaded with packet flows.**

Keywords-topology emulation; real-time; scalability; online simulation;

I. Introduction

Wireless technologies are deployed in increasingly many types of mobile devices. The vast popularity of these mobile devices, and thus the increased availability of wireless technologies, makes applications for the mobile domain of great interest. Evaluation is an integral part of developing such applications. Evaluation can be performed in fully virtual simulation models or in experimental lab or field prototypes. Emulation provides a hybrid of simulation and experimental setups by allowing real applications to operate in environments with simulated link properties. The control of the simulation allows for accurate reproductions of the test conditions to repeat test runs. A disadvantage of using emulation is that real applications operate in real-time and thus enforce the link emulation to run in real-time. This paper describes an extension of the work on a topology emulation tool described in [1].

Simulation tools, such as ns-2 [2], provide a high level of detail in the networking layers. The application under test is, however, typically simplified as the real application cannot easily be used in the simulated environment. Hence, the simulation results represent an application model and not the real-life implementation of the application itself.

Experimental setups can be used to test the real application implementation. However, disturbances in the environment may have a huge impact on the test results. Moreover, the characteristics of these disturbance are not always controllable, making it very difficult to repeat even simple test runs [3].

The objective of this work is to develop an emulation test-bed to evaluate applications in wireless multi-hop topologies. In the test-bed the wireless link is replaced by a wired link. The wired link is less exposed to uncontrollable disturbances and thus more controllable than a wireless link. The properties of a wireless link, such as *packet drops* and *delay*, are then imposed on network traffic on the wired link in a controllable and reproducible manner based on simulation models.

Several types of applications benefit from being tested in an emulation test-bed. Performance of implementations of multi-hop routing protocol can be verified under realistic, yet reproducible, conditions. Other types of applications such as platooning [4] or driving assistance [5] to be deployed in car-to-car scenarios can be tested in the risk free environment of the emulator. In the latter case, the network quality experienced by the application is mapped into changes of node movement or transmission parameters. An example of this is a platooning application that reacts to experiencing a high packet loss rate by slowing down the speed of the car to avoid cars colliding. The described applications illustrate two important aspects of the test-bed. In the first case, a routing application optimizes *network* performance based on the emulated link properties, i.e., affects parameters in layers above the emulator. In the other case, a platooning application optimizes *geographic trajectories* of the nodes based on the link properties, i.e., directly affects the input parameters of the test-bed. This paper describes how the test-bed supports both types of applications.

Several additional requirements must be fulfilled by the test-bed in order to emulate properties of wireless links successfully. Emulating *transparently* as seen from the network application is inherent to create results comparable to those of real experimental setups. Transparency is composed of several requirements. The emulator must operate in real-time to resemble access and transmission performance of a real wireless link, as seen from the applications. It must also present link-layer interfaces to the network layer (and upper layers) similar to a real wireless interface. The emulator should be non-intrusive meaning that as little software as possible should be added to the end-nodes where the applications are deployed. In addition, the test-bed must employ *accurate communication models* in the link simulation to resemble influences from a real environment. Lastly, simple and scalable deployment, use and result processing is required to ensure usability of the emulator. Scalable deployment means that it should be easy to connect enough nodes to the test-bed to create a network of realistic size for the application under test.

The contributions of this paper are: 1) description of the design of the topology emulator, a test-bed capable emulating dynamic multi-hop topologies by changing time-varying link properties in real-time; 2) development of both an offline simulation version for complex models to be simulated before emulation and an online simulation version for simple models of which parameters can be affected during emulation; 3) evaluations of both versions of the topology emulator to illustrate that they are capable of emulating dynamic multi-hop topologies while accomplishing the specified requirements.

II. Related work

Many solutions emulate wireless networks by manipulating link properties of connections. In general, the solutions can be put in two categories; with a *central* architecture or with a *distributed* architecture. The central architecture eases deployment, configuration and control of the emulation process. However, the typical problem with a central approach is performance limitation. Tools with distributed architectures address this limitation by distributing the simulation and emulation tasks onto more nodes in the network. On the other hand, these tools are typically challenged by having to coordinate events between the involved nodes and require an additional piece of software to be installed on every involved node. The latter requires that there exists a version of the emulation software for all platforms used in the application, which is difficult to support.

An extension to ns-2, called ns-2e [6], can emulate simulated link properties from a traditional ns-2 simulation onto real traffic. Through so-called *network objects* real-world traffic is redirected through the simulation. Incompatibilities between real and simulated traffic are handled using *tap agents* that effectively tunnel real traffic inside special simulation packets. In its original form, ns-2e is completely transparent toward end-nodes. However, it cannot guarantee timely delivery of real traffic. This limitation has been addressed in [7]. ns-2e has the ability to simulate dynamic topologies through traditional scripting. However, the number of end-nodes connected to the emulator is limited by the number of available network interface cards on the emulating node. This is a limiting scalability factor.

Another widely used emulation tool is NIST Net [8]. NIST Net is described as 'network-in-a-box' capable of emulating an entire network in a single hop. Similar to ns-2e NIST Net employs a central architecture with a single node to handle all end-node connections, simulation and emulation. This means that it suffers from the same deployment limitations as ns-2e (individual network interface per attached node and the need for context switches in the operating system to handle interrupts for all packets on these interfaces). Also, as indicated by the term 'network-in-a-box', NIST Net emulates entire networks and not links. This means that to use NIST Net to evaluate networking protocols they need to be implemented in the emulator, as is the case when using a simulator. Thus, NIST Net is not transparent to network protocols, as required.

The approach of Seawind [9] is similar to the two described above. Seawind has mostly be applied for wide-area wired and wireless networks.

Recently developed tools with centralized architecture include WNINE [10] and Qomet [11]. WNINE employs a *two-stage emulation* process meaning that it completely separates simulation of the link properties and the emulation. In the simulation process it offers detailed models, but for emulation it relies on dummynet [12], which is a tool for traffic shaping on intermediate nodes in a network. Thus, WNINE also suffers from the scalability limitations. Qomet follow the same concept as WNINE, and evaluation of Qomet only been performed using StarBED [13] and not in a general network scenario.

In contrast to the centralized approach, each attached node in a distributed emulation calculates its own view on the link properties and these views must be synchronized between all nodes. This approach is used in NetEM [14], EMWIN [15], JEmu [16]. Testing real applications with these distributed tools has several challenges. The end-node environment is deployed as a virtual machine and not a native environment. This means that the possibility of running the real application in a deployment environment is reduced. Moreover, the communication overhead of synchronization may affect the performance of the protocols under evaluation. Lastly, and perhaps most importantly, this type of emulator is not transparent in the sense that in each case a (potentially small) piece of code must be running on the attached node.

The tool presented here, called the 'topology emulator', employs a central architecture and thus falls in the category of ns-2e and NIST Net. Compared to NIST Net, the tool emulates individual layer 2 links between nodes instead of layer 3 paths. This is done in order to facilitate testing of networking protocols without having to implement them on a software platform in the involved nodes as well as in the NIST Net emulator. Compared to ns-2e, the architecture of the topology emulator uses one central network switch to aggregate node traffic instead of using a number of network interface cards in the central emulating device. This has the advantage that more nodes can be added without changing the hardware set-up of the emulator. When new nodes are added it is only the internal link models that need to be updated accordingly.

III. Topology emulator

The work presented in this paper builds on the scalable centralized emulator that is transparent to real applications on real end-nodes and meets real-time requirements. This version of the emulator is described in detail in [1] and summarized in the following. The version of the emulator described here extends the features of the original emulator to support what we call *online simulation*. To distinguish to the two versions, we refer to the original emulator as the *offline version* and to the extended emulator as the *online version*, which is explained in the following.

In the offline version, all properties of the network are assumed to be known beforehand. By knowing distances between nodes, the specific link properties (packet drop probability and packet delay) for transmitting packets can be calculated. To calculate such link properties, models of the transmission conditions (propagation conditions, transmission power, coding schemes, etc.) must be specified. All simulation models can be dynamic models. This way geographic trajectories of nodes can change during simulation, as well as transmission parameters such as transmission power. Any parameter can be changed in any model, as long as it is known beforehand. Once the emulation is running, the parameters cannot be affected anymore in the offline version.

In the online version, we introduce online simulation which means that the parameters of the transmission models can be affected during emulation. To facilitate changing the parameters online, a control channel from the applications to the emulator is introduced in the architecture.

Next, the basic architecture of the scalable topology emulator is described. This architecture is designed for both offline and online simulation. Then, the details of the core parts of the offline version are described. Finally, the main implementation choices to realize the online version are described.

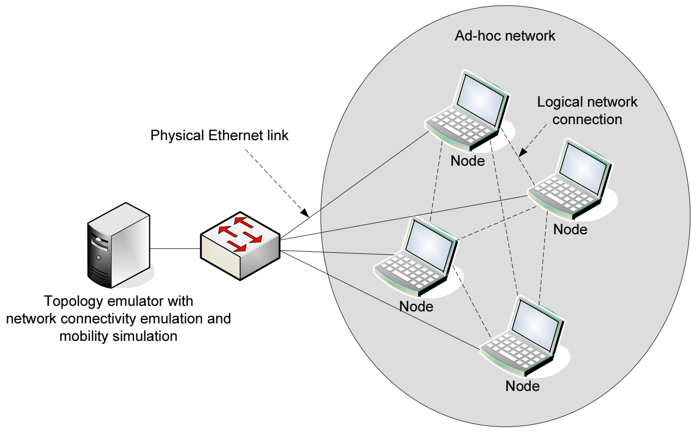
Ad-hoc network
Logical network connection
Physical Ethernet link
Node
Node
Node
Node
Topology emulator with network connectivity emulation and mobility simulation

Figure 1. Architecture of topology emulator.

A. Architecture

The network architecture of the topology emulator is illustrated in Figure 1. It consists of one centralized node called the *emulator node* and a central network switch to which all *end-nodes* are connected. End-nodes contain the applications or networking protocols for evaluation. When applications are evaluated in real setups, the wireless networking interface on the end-nodes is used. When the topology emulator is used for evaluation, the wired networking interface must be used to connect to the central network switch. This change of network interface is the only requirement to use the topology emulator for evaluation. The network switch allows for connecting many nodes to the topology emulator and thereby helps ensure the scalability of the test-bed.

The emulator node receives and forwards all frames transmitted between end-nodes. No end-node receives frames before they have been forwarded by the emulator node. This node separation and traffic concentration is obtained by use of 802.11q virtual LAN (VLAN) tagging [17] on the switch. This concentrates all frames from one end-node in one unique virtual LAN per end-node. The VLAN-tag can thereby be used by the emulator node to identify the sources of the frames.

The emulator node creates virtual links between the end-nodes by controlling bridges between the VLANs. The virtual links allow end-nodes to transmit frames to each other only if enabled by the emulator node. By selectively dropping or delaying frames on the virtual links, the emulator node controls the properties of all virtual links between end-nodes. Ultimately, as the emulator node functions as an enhanced switch and forwards layer 2 frames, its existence is transparent by design to any layer 3 protocols (and higher) used on the end-nodes.

The software architecture of the emulator node consists of three processes; a simulation process, an emulation process and a property updating process. The simulation process simulates link properties (packet drop probabilities and delays) for all virtual links and saves them to a trace. The emulation process emulates the links in real-time by deciding if packets should be dropped or not while end-nodes communicate. If not dropped, the emulator determines a delay for each packet and transmits it once the delay expires. The property updating process binds the simulation process and the emulation process together by periodically feeding link properties into the running emulation process.

In Figure 2 the architecture of the offline version of the topology emulator is depicted. In contrast to the offline version, the online version in Figure 3 also contains the mobility simulation and the link simulation as part of the real-time execution. In the following the different parts of the architectures are described in detail.

B. Offline version

In this section, the simulation process, the emulation process and the property updating process are described for the offline version. The emulation process is the same for the offline and

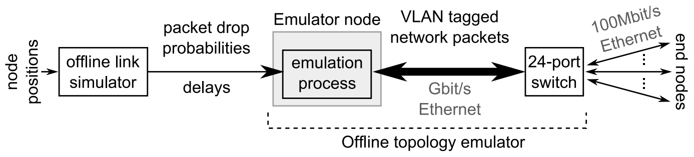

Figure 2. Emulation with offline simulation.

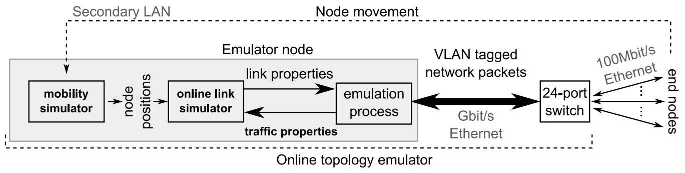

Figure 3. Emulation with online simulation.

online version and therefore it is only described in this section.

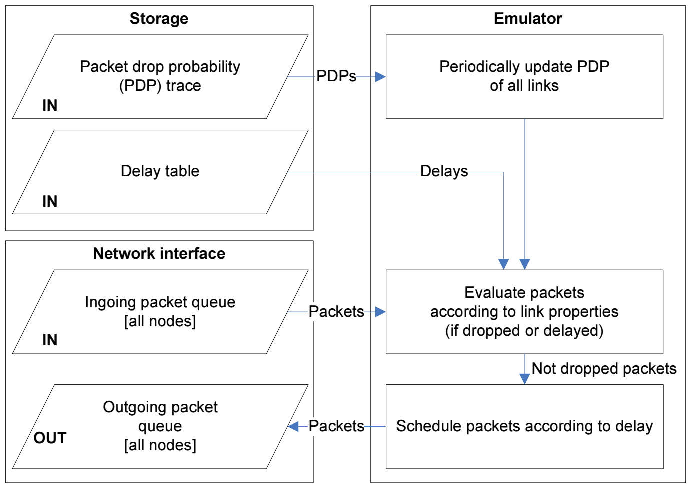
Storage
Packet drop probability (PDP) trace
IN
Delay table
IN
Network interface
Ingoing packet queue [all nodes]
IN
Outgoing packet queue [all nodes]
OUT
PDPs
Delays
Packets
Packets
Emulator
Periodically update PDP of all links
Evaluate packets according to link properties (if dropped or delayed)
Not dropped packets
Schedule packets according to delay

Figure 4. Emulation process

1) Simulation process: In the offline version simulation is carried out before the emulator is started.

The simulation process is a function that outputs packet drop probability and packet transmission delay for all virtual links in time-slices. As input it takes a trace of node positions, a transmission model and input parameters for the transmission model. Two examples of transmission models and parameters are described below. These are also the models used to verify the performance of the two emulator versions. This verification is described further in section IV.

One model is a *simple* unit-disk model that determines packet drop probability of the link between nodes i and j at time t by use of a threshold D on distance $d_{i,j}$ between the nodes:

$$PDP_{i,j}(t) = \begin{cases} 0 & \text{if } d_{i,j}(t) < D \\ 1 & \text{otherwise} \end{cases} \qquad (1)$$

Here the delay is constant on all links, which would represent simply a medium access delay where the channel is assumed error free and without other contenders when a packet is transmitted.

To use this model in the simulation process with a trace of node positions as input, the model parameters needed are the distance threshold and the constant delay.

More realistic (and thus more complex) transmission models than the unit-disk model are supported by the simulation process. In [18], a stochastic model is used which is detailed in the following.

To model packet drop probability, channel properties, physical layer properties and link layer properties are used to relax some of the simplifying assumptions of the simple model. Rappaport's shadowing model [19] as seen in (2) is used to model attenuation of transmitted power to received power (α is the path-loss coefficient, σ is the shadowing parameter which is the standard deviation of the zero-centered Gaussian random variable X_σ).

$$P_r(d) = P_t - 10 \cdot \alpha \cdot log_{10}(d) + X_\sigma \qquad (2)$$

The mapping between received power and bit error rate in the physical layer is modeled by Mangolds OFDM model [20], assuming use of IEEE 802.11a. To calculate a frame error rate from the bit error rate the assumption of independent, identically distributed losses is used, leading the following expression (where L_{frame} is size of frame in bytes):

$$FER = 1 - (1 - BER)^{8 \cdot L_{\text{frame}}} \qquad (3)$$

The frame error rate is then mapped one-to-one into a packet drop probability in this model. To model the delay, we use the Bianchi IEEE 802.11 DCF model [21], assuming that the only causes for frame losses are collisions due to other nodes transmitting and thus the only factor in delay is the time spent backing off from the channel. Propagation delay is considered as a constant and independent of distance. To model the number of nodes contending for a channel, we model how many nodes are likely to transmit packets within a time-slice by using a threshold on the packet drop probability for all virtual links. If a link between two nodes has a packet drop probability below a threshold P, we consider the two nodes as being *neighbors* and consider the transmitted packets from them probable to collide. The more neighbors a node has, the more likely it is for packets transmitted by this node to have a high delay.

In this paper the latter complex model is used for the evaluations of the offline version. For comparison of the offline and online version, the simple link model is used in order to keep processing power needs of the emulator node low (and to avoid having to performance optimize the implementation of the complex model).

When performing offline simulation the distances can be calculated from node positions which are input to the simulation process over time t as (x, y)-coordinates. These positions can be based on realizations of either recorded traces of real movement, deterministic paths that simulated nodes follow or based on simulations of stochastic models such as random walk or random way-point [22].

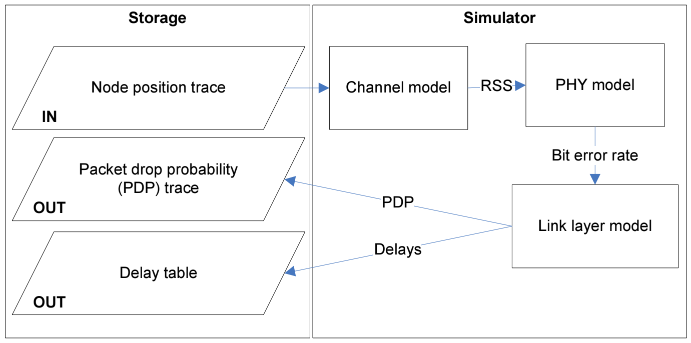

Figure 5. Offline simulation process

Packet drop probability of a link is calculated based on distances between nodes and the transmission models of the channel, the physical layer and link layer as shown in Figure 5.

The output of the calculations is a trace of packet drop probabilities over time $p_{i,j}(t)$ on every link (between nodes i and j) in the topology. Note that the symmetry of the probabilities on a link depends on the simulation model as the topology emulator supports asymmetrical links where upstream and downstream links have unequal properties.

The delays are calculated as a static table of inverse cumulative distribution functions. The table is used primarily to enable fast lookup of delays as it need to be determined for each packet when emulating. As the traffic patterns of the nodes are not known prior to the emulation, a parameterized family of delay distributions for $K = 1, 2, ..., n$ (number of simultaneously transmitting nodes) is used instead of just samples of delays. n is the total number of nodes in a simulation and is scenario-specific. The reason several delay distributions are used is the fact that the distribution of the delay depends on the number of nodes able to transmit in a channel. Therefore we generate a delay distribution for each possible number of next-hop neighbors in a network. When nodes move around or the transmission conditions change, the number of next-hop neighbors will change and the real-time emulator will access the corresponding distribution in the table. This also means that the table containing the delays is generated once during simulation and not continuously updated during emulation.

Using the simple model with constant delay for all numbers of next-hop neighbors the delay table is a column vector of height $n-1$. In the complex model with stochastic delay the table is a $(n-1)\times r$ matrix, where r is the chosen discretization of the inverse CDF in the table.

Based on the probability threshold P, the number of next-hop neighbors $n_i(t)$ of node i at time t is calculated from $p_{i,j}(t)$ for all t. This is used as index to find the appropriate delay distribution during emulation.

All simulation models must compensate for the fact that all calculations are performed before running the emulation. For the delay, this means that the patterns of generated traffic by applications connected to the emulator must match the traffic models used in the simulation models. As an example, the Bianchi-model assumes a saturated channel, which may not always be the case for the applications. Also, in the offline version, it is not possible for the applications to dynamically change communication parameters, such as modulation scheme, when emulation is running. If in need of such changes, they must be implemented in the simulation models when simulating.

2) Property updating process: In the offline version the trace of link properties is saved to a file in the simulation process. To be able to use the trace-file for emulation, the *property updating process* reads the file and updates arrays accordingly which are then read by the emulation process when determining packet drop probabilities and number of next-hop neighbors. The static delay table is only loaded once. The entries in the trace are time stamped relative to the beginning of the simulation and ordered ascending. When the real-time clock reaches the next time-stamp, the next entry in the file is loaded into the emulation process. The property updating process supports update frequencies up to 10Hz.

3) Emulation process: All nodes are physically connected to a Cisco Catalyst 2950T VLAN-aware 24-port switch. Two of the ports have a capacity of 1Gbit/s traffic whereas the remaining ports support 100Mbit/s. One of the gigabit ports is used in trunking mode, such that all traffic received on other ports is concentrated in this port. The emulator node is then connected to the trunk port. This node has an Intel Core 2 Duo 1.86GHz processor and has Linux kernel v2.6.18 installed in our setup.

Packets received on the emulator node are filtered twice before reaching the emulation process; on layer 2 by *ethernet bridging* controlled by *ebtables* and on layer 3 by *netfilter* controlled by *iptables*. The layer 2 bridging is necessary to prevent transmission of link layer datagrams, e.g from ARP, when there is no virtual link. If emulation was only enforced using netfilter, link layer datagrams would not be filtered. Netfilter is used to deliver packets to the emulation process.

The details of the emulation process resemble NIST Net, however, as the packet flow using the VLANs was not easily integrated into the existing solution, a new emulation process was developed.

On layer 2, the packet control uses a binary value $l_{i,j}(t)$ indicating if a link exists or not between source and destination of a packet identified by the VLAN tag. If $l_{i,j}(t) = 1$ then there is a link and the packets are forwarded to layer 3. If $l_{i,j}(t) = 0$ then there is no link and the packet is dropped. $l_{i,j}(t)$ is calculated as

$$l_{i,j}(t) = \begin{cases} 1 & \text{if } p_{i,j}(t) < P \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

On layer 3, the packets are forwarded to the emulation process. For each uni-directional link between two nodes, parameters are stored in tables representing packet drop probability, the number of next-hop neighbors, and packet delay distributions. To determine if a packet is dropped the emulation process draws a uniform random number in [0,1] and compares it to $p_{i,j}(t)$. The process also determines the delay by using the number of next-hop neighbors of the source node i, $n_i(t)$, and the static delay table. To determine a delay, $n_i(t)$ is used to find the appropriate distribution (row in the table). A uniformly distributed random number is drawn and used as index in the tabularized form of the inverse CDF which then indicates the delay of that particular packet. Once the delay has been determined, the packet is scheduled for delayed transmission.

The packet scheduler checks for and transmits packets every $122\mu s$. In order to achieve such a high time resolution, the Linux Real-Time Clock chip (RTC) was used to trigger a scheduler in the emulation process. This clock supports triggering processes at frequencies up to 8192Hz $(1/8921Hz = 122\mu s)$, which is much higher than normal timer resolution of Linux of 10-100Hz. Once triggered, the emulation process advances a ring buffer containing a list of packets to be transmitted in the current time slot and transmits all the packets in the list.

C. Online version

For the online version the simulation is performed in real-time during the emulation. An overview of this process related to emulation is given in Figure 3. The main difference to the offline version is that here parameters of the simulation models can be changed online. This means that instead of having all inputs to the simulation process being known and specified beforehand, these properties can be actively updated during the emulation. This directly enables application to affect mobility parameters, namely to be able to affect the position of a node as a consequence of the experienced network quality. With the online version, this becomes possible as all input parameters to the simulation process can be updated, including the node movement.

As illustrated in Figure 3, the online version requires the simulation process to be deployed on the emulator node. In turn, if applications affect the mobility of the nodes, updated position information must be communicated to the link simulation process. As the example applications in the vehicular scenarios do not manipulate (x,y)-coordinates but rather node physics (such as speeder or brakes), there must exist a function capable of transforming application output to link simulation input. In the car scenario this function would transform braking and speeding information into updated (x,y)-coordinates which would be loaded into the link simulator. To handle this transformation, an additional mobility simulation entity is needed in the network between end-nodes and the link emulator.

Online link simulation during emulation can be performed on several levels; an event-based level (where link properties are calculated per packet) or periodically (where link properties are calculated per time-slice) as in the offline version. The event-based level is very precise as the networking environment is adapted according to each packet. The periodic level may be less precise as it only represents an average of properties over the specific time-interval. On the other hand, simulating complex models per packet can become very processing-intensive when a large amount of packets is sent through the emulator process. Simulating periodically is not influenced by the incoming packet rate and may therefore be more suitable for more complex models as high packet rates can still be supported. In this work, the online simulation is performed periodically as the event-based level is very processing-intensive and not necessary to illustrate support of online simulation. As described in section III-B, a simple simulation model was developed to reduce implementation complexity and processing-intensity. The simple model is sufficient as main purpose of the model is to evaluate the performance of the online simulator compared to the offline simulator.

The connection between the online simulation process and the emulation process is quite similar to the offline version. In the online version the link properties are transmitted from the simulation process to the updating process using a socket. This structure has been chosen as it enables the simulation process to be located on a secondary, resourceful computer. To be able to use the properties for emulation, the *property updating process* for the online version reads the incoming data on the socket and updates the emulator arrays. The arrays are updated in an event-driven manner whenever new data is received by the updating process. As in the offline version, this updating process supports update frequencies up to 10Hz.

IV. Evaluation

The topology emulator has been evaluated on several levels; 1) evaluating that the emulator is capable of emulating dynamic multi-hop topologies and 2) verifying that the performance requirements are met. Results from both evaluations are presented and discussed in the following. As the emulation process is the same for both offline and online simulation, the emulation capability has only been evaluated once by using the offline version. Meeting performance requirements is a different task depending on simulation version and therefore has been evaluated for both offline and online version.

A. Functionality evaluation

To evaluate the functionality of the emulator, a scenario is specified where mobile nodes create a dynamic topology in which end-to-end communication is supported by ad-hoc routing over multiple hops. This scenario illustrates the intended use of the emulator, namely to test prototypes of networking algorithms such as routing or addressing schemes or even transport or service protocols for highly dynamic networks.

The movement of the nodes in the scenario is illustrated in Figure 6. In the scenario, the R2 node moves out of range of the destination node and is replaced in the end-to-end path by R1. Throughout the scenario packets are dropped and delayed on the links according to the simulated parameters.

The objective of the scenario is to illustrate how the ad-hoc routing algorithm is performing in a dynamic multi-hop topology. The link properties model an IEEE 802.11a wireless link incorporating shadowing and fading in the channel, as previously described. The parameters of the physical model are set to $\alpha = 2$ and $\sigma = 4$. For the IEEE 802.11a link the background noise is set to -100 dBm, the modulation scheme to 64-QAM, transmission power to 100 mW, L_{frame} to 1500 bytes and the number of retransmissions in the medium access to 7. The specified movement is used as input to simulate

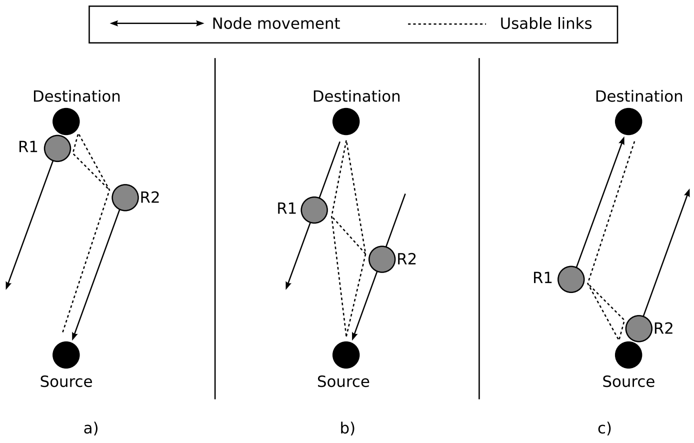

Figure 6. Scenario with relay nodes R1 and R2 move periodically back and forth.

PDP and delay. The PDP of the links is illustrated in Figure 7 as the probability varying over time on each link in the network topology. A 3-hop path is always present (R2-R1-Destination). Also a 2-hop path is always present, however it changes over time. This effectively illustrates the dynamic multi-hop topology. The trace of simulated PDP and the delay table are then used as input for the emulation process to impose the topology information onto real traffic. *ping* is set to continuously ping from source to destination while OLSR [23] is used as ad-hoc routing algorithm between the nodes.

The resulting traffic recorded by the emulator is illustrated in Figure 8. From the figure, we see that the emulator is capable of emulating a dynamically changing multi-hop topology as seen from the end-nodes. This is seen as the flow of packets is redirected to use the available links when the currently used link becomes unavailable. Moreover, the packets sent using the R1-R2 link clearly demonstrate the multi-hop emulation capability.

B. Performance evaluation

As previously described, the emulator must appear transparent to end-nodes meaning that lower network bandwidth and higher link delay (besides the emulated delay) compared to a wireless environment is not tolerated. Evaluations of bandwidth and service time have thus been performed to ensure that the topology emulator meets these requirements.

Bandwidth limitations in a network occur at processing or communication bottlenecks. In the topology emulator, two

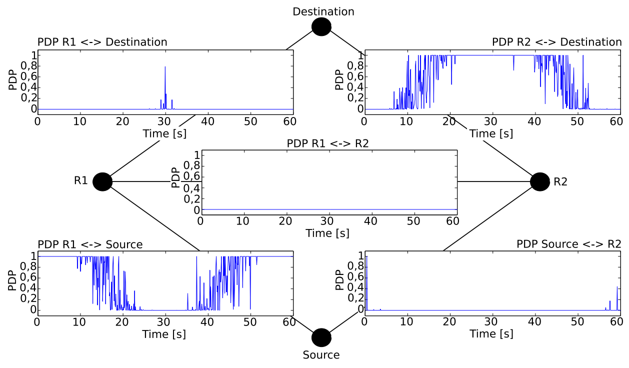

Figure 7. Packet drop probability (PDP) trace on links. The path changes from using *R2-Destination* to *Source-R1*.

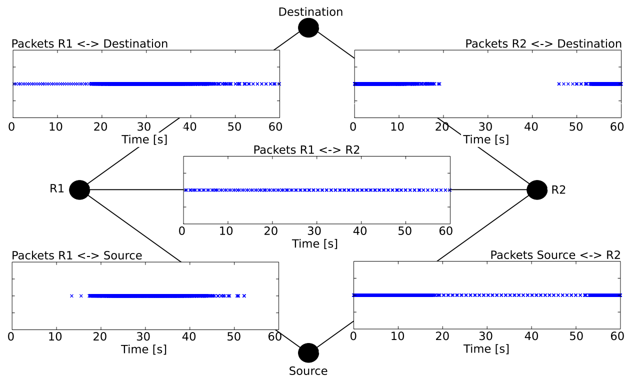

Figure 8. Traffic on links during emulation shows that different available links are used.

such potential bottlenecks exist; in the switch and in the emulator node. As the complexity and processing need in the emulator node is far greater than that of the switch, the emulator node is considered the significant bandwidth bottleneck in the setup. Hence, only the bandwidth capabilities of the emulator node were evaluated.

To evaluate available bandwidth of the emulator node, the traffic generator D-ITG [24] is used. By use of 7 nodes in a fully connected emulated topology, each sending and receiving streams of 100MBit/s asynchronously to other nodes, the emulator node was heavily loaded. D-ITG is also capable of recording the received bandwidth on the nodes which should amount to 100Mbit/s per node deducting a small overhead percentage from transport and network layers. In total this amounts to 700Mbit/s theoretical load on the emulator (experiments showed the real max to be 644Mbit/s).

Considering that the expected maximum throughput of IEEE 802.11a in a real wireless channel is 54Mbit/s, the emulator node is capable of supporting 12 separate channels within the 644Mbit/s limit. This limitation of 12 channels is equal to 24 connected, fully loading nodes in separated networks of only two nodes and is in effect only when the nodes experience a traffic- and noise-free channel.

The measured bandwidth capability of the emulator is considered acceptable due to the following reasons. First, currently it is only possible to connect 20 nodes to the switch. Second, for these 20 nodes to fully load the emulator node on links between them with up to 54Mbit/s, the simulation scenario would need to produce 10 separate channels with only two nodes each and these channels would need to be free of any external noise. This is of course a possible situation when testing the wireless application, although not considered very likely.

Service time of the entire topology emulator is also evaluated. Service time is defined to be the time it takes from a packet is sent from one end-node until it is received at another end-node. This includes transmission time from one end-node to the switch, between switch and emulator node both ways, and finally between switch and the other end-node. To be able to use the emulator to emulate any delay transparently, the service time must not exceed the time it takes to send a packet between two nodes on a real wireless link. The Linux network tool `ping` was used to measure the service time for a link between two nodes by sending out packet probes. `ping` measures the round-trip time when sending and acknowledging a packet. To measure the delay between two nodes, half of the round-trip time was used.

The evaluation is performed on a emulated link (with emulated delay = 0) and compared to both a direct wired link and measurements of an IEEE 802.11a wireless link. This is done to establish if the maximum service time of the topology emulator is less than smallest delays expected on a real wireless link. As the service time includes processing time of a packet in the switch and in the emulator node, and this time is dependent on the packet size, several packet sizes are used. Layer 3 packets of sizes 0-5000 bytes, with Ethernet maximum frame size of 1500 bytes payload, are used. Having a maximum layer 2 frame size smaller than the maximum layer 3 packet size will result in fragmentation, which in turn will result in more frames to traverse the topology emulator, which ultimately will result in longer service times. The measurement of the wireless delays was done using two Linux-equipped laptops with IEEE 802.11a network interface cars in ad-hoc mode standing next to each other. The measurement of the wired link used the same laptops, but connected through a single cross-over Ethernet cable. The delays were measured using `ping` and in the wireless case multiple times during a week. The results of all measurements are illustrated in Figure 9. Here it is illustrated that transmitting a packet through the entire topology emulator uses approx. $250\mu s$ more than when transmitting using a direct wired link. In addition, the figure shows that the excess $250\mu s$ are well below the transmission times of the tested wireless link. This means that the service time of the topology emulator is acceptable as it is below the values of the tested IEEE 802.11a link.

A comparison between the service times of the offline and the online version was also performed. This was done to establish if the transparency requirement is met in the online version as well. As the simulation process consumes

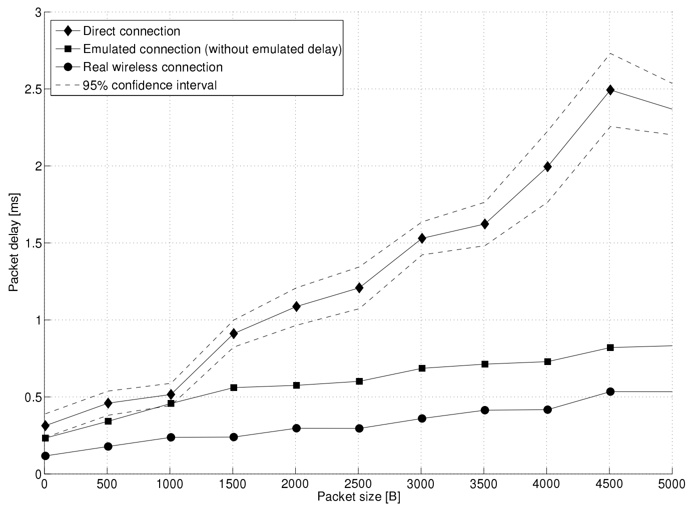

Figure 9. Service time from various packet sizes on: A real wireless IEEE 802.11a link, a direct emulated link (with delay = 0) and a direct wired Ethernet link. The confidence interval is shown for the wireless measurements only, as the variations in the remaining measurements are insignificant.

processing power besides the emulation process, it is important to confirm that the performance of the online version is not different from the offline version.

The evaluation was performed during the bandwidth test described previously using *ping* to probe the delay from one node to another while loading the emulator process heavily. The emulator was loaded traffic ranging from 0 to 700 Mbit/s and *ping* probes of size 1500 bytes measured delay 30 times during the load period. From these measurements the mean one-way delay was calculated. The offline and the online versions were set up using equal simulation models and equal property update rates. The simulated link model was using equal property update frequencies at maximum (10Hz).

The results are shown in Figure 10 and Figure 11. It was possible to measure the delay between load values of 0 to 500 Mbit/s. Above 500 Mbit/s no `ping`-packets were acknowledged within the measuring period. As a consequence load values above 500Mbit/s are not depicted.

As illustrated in Figure 10 the service time of both versions are well below 1 ms, which can be read from Figure 9 to be sufficient to be transparent for packet of 1500 bytes. In Figure 11 it is seen that the service time starts to vary significantly more and increase once the emulation process is loaded with more than 350Mbit/s. This is shown to be the case for both versions. This means that the emulator cannot reliably be considered transparent toward end-nodes above 350Mbit/s, which is then considered the bandwidth performance limit. If the simulated delay is above the service time of 30ms seen in Figure 11, then an algorithm could be applied to take the service time (that changes with the load) into account when delaying packets. As this is out of the scope of this work, the bandwidth performance limit is kept at 350Mbit/s.

The increased service time is most likely due to overloading of the CPU in the emulator node causing buffering on either

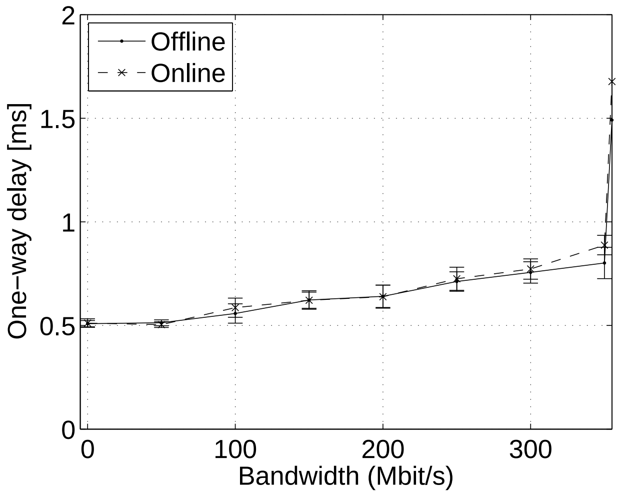

Figure 10. Service delay at different bandwidth (0-350 Mb/s)

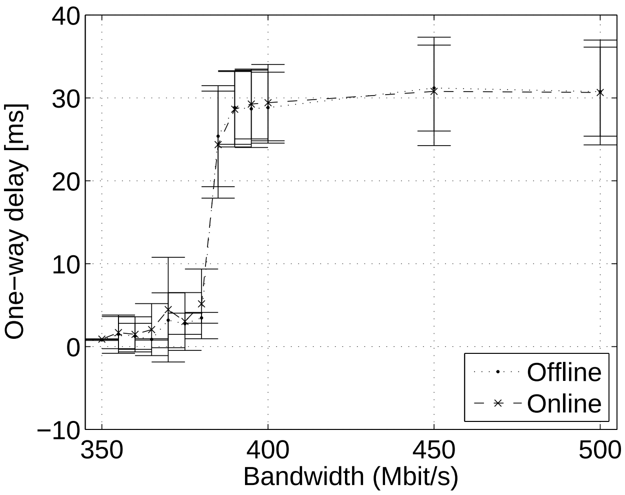

Figure 11. Service delay at different bandwidth (350-500 Mb/s). Continuation of Figure 10 with increased y-scale.

the networking interface card or in the operating system. One immediate solution to this issue would be to distribute the emulation process into two threads working on incoming packets using both cores of the processor.

V. Extensions

Information about node positions or link quality is not communicated to end-nodes. To distribute this information to the end-nodes would require an additional control channel. Such a control channel could be deployed using the existing connections between end-nodes and emulator node. In doing so, the resources consumed by the control traffic on the links should be accounted for in the simulation models. An alternative solution to reusing connections is to deploy a separate network, which would then require additional networking interfaces in all nodes, but would not interfere with the application traffic on the emulated links. Moreover, only packet drop probability and delay are calculated, meaning that the upper layers do not have access to traditional link layer information such as received signal strength indication (RSSI). The setup requires all upper layer technologies to be independent of the link layer and the physical layer, which is a reasonable assumption to test many distributed applications. Approaches to actively deliver such information through an emulated virtual network interface on the end-node are being investigated. Transporting such information from the emulator to the end-nodes requires test-bed specific software to run on the end-nodes during emulation.

VI. Conclusion and Future Work

In this paper, we presented a new network emulation tool capable of emulating dynamic multi-hop topologies. This is especially important when developing wireless applications, as field tests of such applications become cumbersome or even impossible to reproduce. As an advance to existing tools, the topology emulator features real-time emulation of dynamically changing multi-hop topologies that are resulting from node movement in a pre-specified scenario. Moreover, the architecture of the topology emulator is designed to be scalable and modular to facilitate extensions without any modification to the deployed version.

The functionality is divided into two parts; a simulation part and an emulation part. The simulation part simulates a complex wireless network from node movement resulting in packet drop probabilities and delays on each link between nodes. The emulation part imposes these properties in real-time to received packets on a central emulator node. End-nodes, that are usually in the wireless domain, are connected to the emulator node via wired links through a central switch. All frames sent from end-nodes are forwarded to the emulator node and based on the link property traces the emulation part decides if frames should be forwarded further. If so, a delay is determined and the frames are scheduled for transmission to the receiving end-node. This two-part model has been deployed in two different execution environments; offline simulation and online simulation. The offline version simulates the link properties prior to any emulation whereas the online version simulates the properties in real-time during emulation. In the online version it is thereby possible to change the parameters of the simulation model to reflect changes in the environment and allow end-nodes to affect their own positions based on the experienced network properties or on the content of the communicated messages. By designing the connecting point as a switch, the network architecture allows for up to 20 real end-nodes to be connected to the emulator. Evaluations of the functionality and the performance of the emulator shows that it is capable of emulating dynamically changing topology properties toward all connected end-nodes. The results also show that the end-nodes do not experience limitations in bandwidth or longer transmission delays when using the emulator. This has proved valid for both the offline and the online emulator. The maximum capacity of the emulator was determined to be 350 Mbit/s. This limit has shown sufficient to support up to 14 nodes in WLAN scenarios using at maximum 54 Mbit/s per channel.

Future work includes optimization of the performance of the online simulation process. Approaches to deliver simulated link information, such as RSSI-values, layer 2 packet loss rates or even position coordinates, to end-nodes are currently under investigation. An interesting investigation is to establish the maximum possible complexity of the simulation model in the online version. As highly complex model requires many resources during simulation, the processing power remaining from the emulation process can be used to determine the maximum level of complexity for calculating link properties.

Acknowledgments

This work was partially supported by the EU IST FP6 project 'HIghly DEpendable ip-based NETworks and Services – HIDENETS', see www.hidenets.aau.dk and the EU ICT FP7 project 'Open Pervasive Environments for iNteractive migratory services – OPEN', see www.ict-open.eu. The Telecommunications Research Center Vienna (ftw.) is supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

References

[1] A. Nickelsen, M. Jensen, E. Matthiesen, and H. Schwefel, "Scalable emulation of dynamic multi-hop topologies," in *Proceedings of the 4th International Conference on Wireless and Mobile Communications (ICWMC 2008)*, 2008.

[2] S. McCanne and S. Floyd, "Network simulator ns-2." The Vint project, available for download at http://www.isi.edu/nsnam/ns, May 6, 2009.

[3] H. Waeselynck, Z. Micskei, M. Nguyen, and N. Riviere, "Mobile Systems from a Validation Perspective: a Case Study," *Parallel and Distributed Computing, 2007. ISPDC'07. Sixth International Symposium on*, pp. 14–14, 2007.

[4] T. Tank and J. Linnartz, "Vehicle-to-vehicle communications for AVCS platooning," *Vehicular Technology, IEEE Transactions on*, vol. 46, no. 2, pp. 528–536, 1997.

[5] R. Bishop, R. Consulting, and M. Granite, "A survey of intelligent vehicle applications worldwide," in *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pp. 25–30, 2000.

[6] K. Fall, "Network emulation in the VINT/NS simulator," in *Computers and Communications, 1999. Proceedings. IEEE International Symposium on*, pp. 244–250, 1999.

[7] D. Mahrenholz and S. Ivanov, "Real-Time Network Emulation with ns-2," in *Proceedings of the The 8-thIEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2004)*, 2004.

[8] M. Carson and D. Santay, "NIST Net–A Linux-based Network Emulation Tool," *Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[9] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen, "Seawind: a Wireless Network Emulator. In proceeding of 11th GI," *ITG Conference on Meaurement, Modelling and Analysis (MMB 2001)*, pp. 151–166, 2001.

[10] T. Perennou, E. Conchon, L. Dairaine, and M. Diaz, "Two-stage wireless network emulation," in *Proceedings of the Workshop on Challenges of Mobility held in conjunction with 18th IFIP World Computer Congress (WCC)*, pp. 57–66, Springer, 2004.

[11] R. Beuran, L. Nguyen, K. Latt, J. Nakata, and Y. Shinoda, "QOMET: A Versatile WLAN Emulator," in *Proceeding of the 21st International Conference on Advanced Information Networking and Applications*, 2007.

[12] L. Rizzo, "Dummynet FreeBSD network emulator." http://info.iet.unipi.it/~luigi/ip_dummynet, May 6, 2009.

[13] J. Nakata, S. Uda, R. Beuran, K. Masui, T. Miyachi, Y. Tan, K. Chinen, and Y. Shinoda, "StarBED2: Testbed for Networked Sensing Systems," in *Networked Sensing Systems, 2007. INSS'07. Fourth International Conference on*, pp. 142–145, 2007.

[14] S. Hemminger, "Network Emulation with NetEm," in *Proceedings of Linux Conf Au 2005*, 2005.

[15] P. Zheng and L. Ni, "EMWIN:: emulating a mobile wireless network using a wired network," *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pp. 64–71, 2002.

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

[17] D. McPherson and B. Dykes, "VLAN Aggregation for Efficient IP Address Allocation." RFC 3069 (Informational), Feb. 2001.

[18] A. Nickelsen and M. Jensen, "Evaluation of routing dependability in manets using a topology emulator." Master Thesis, 2007.

[19] T. Rappaport, *Wireless communications*. Prentice Hall PTR Upper Saddle River, NJ, 2002.

[20] S. Mangold, S. Choi, and N. Esseling, "An Error Model for Radio Transmissions of Wireless LANs at 5GHz," in *Proc. Aachen Symposium*, pp. 209–214, 2001.

[21] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 3, pp. 535–547, 2000.

[22] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, "Stochastic Properties of the Random Waypoint Mobility Model," *Wireless Networks*, vol. 10, no. 5, pp. 555–567, 2004.

[23] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)." RFC 3626 (Experimental), Oct. 2003.

[24] S. Avallone, A. Pescape, and G. Ventre, "Distributed internet traffic generator (D-ITG): analysis and experimentation over heterogeneous networks," *ICNP 2003 poster Proceedings, International Conference on Network Protocols, Atlanta, Georgia*, 2003.