# MEASUREMENT-BASED MODELING OF END-TO-END DELAYS IN WLANS WITH NS-2

Christian Resch, Armin Heindl, Kai-Steffen J. Hielscher, Reinhard German
contact email : Armin.Heindl@informatik.uni-erlangen.de
Computer Networks and Communication Systems (Informatik 7)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr.3, 91058 Erlangen, Germany

## Abstract

We enhance the standard ns-2 model for IEEE 802.11 WLANs such that it reflects the delays that are measured in a real ad-hoc network composed of laptops. From timestamps for UDP transmissions, which are recorded at several points on the way from application layer to application layer in a lowly-loaded system, we derive various empirical distributions, which we incorporate into the ns-2 model in order to better capture the real system. For validation, we compare simulated and measured end-to-end delays for UDP transmissions under different load conditions. Our experiments demonstrate the difficulties encountered when tuning a generic and widely used simulation model to a real scenario and suggest that cross-correlation among subsequent delays need to be considered for a satisfactory match of simulation and measurements.

## 1 Introduction

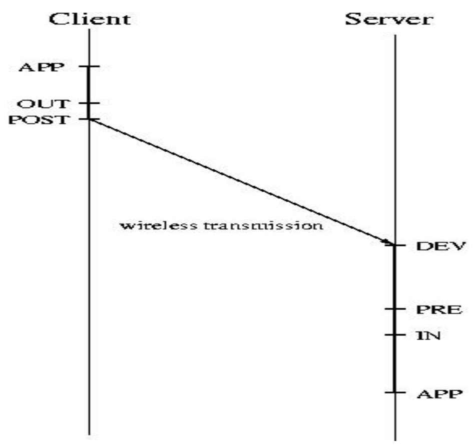
Model building usually implies the adoption of many assumptions the impact of which is hard to assess in complex systems. When a real system exists, measurements taken from this system may help to make assumptions more precise (e.g., by providing a probabilistic distribution inferred from real data) and/or to quantify the overall impact of assumptions (e.g., by comparing sensitive performance measures from the model with measurements from the real system). By measurement-based modeling, we refer to an iterative model building process including input modeling and validation with respect to a real system. Resulting models may be trusted to provide reasonable results in configurations beyond the scope of the actual real system (with respect to traffic load, number of components, etc.).

In the context of performance modeling of wireless local area networks (WLANs), we describe the steps in measurement-based modeling. For data collection, we implemented a high-precision measurement infrastructure for WLANs comprising laptops running Linux. Data analysis for input modeling, especially for various delays in the protocol stack, is prevalently done with the R tool [1]. The models are constructed and evaluated with the network simulator ns-2 [2]. The paper also addresses some modeling details, validation and simulation results (end-to-end delays between stations in a WLAN).

In summary, we provide a framework for measurement-based modeling of WLANs as demonstrated for an ad-hoc network. An infrastructure with access points may of course also be studied.

## 2 Data Collection on the Real System

The real system under study may be an ad-hoc or infrastructure WLAN with laptops running Linux 2.4 (with nano-kernel for timestamp resolution in nanoseconds). Measurements are taken on two (or more) laptops, where timestamps are synchronized offline for one-way delays by means of additionally recorded GPS signals on each machine (as adapted from [3,4]).

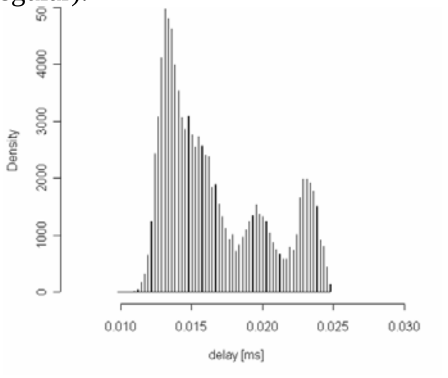


| Delay | Sample mean (in microsecs) | Dependence on payload |
|---|---|---|
| APP-OUT | 15 | Linear |
| OUT-POST | 5 | None |
| POST-DEV | 750 | Linear |
| DEV-PRE | 150 | Linear |
| PRE-IN | 15 | None |
| IN-APP | 130 | Irregular |
| END-To-END | 1065 = 1,065 ms | - |

**Figure and Table 1:** Measurement points on the path of a UDP packet (left) and Measured delays (for payloads of 100 byte; right)

The measurement infrastructure includes packet generators (i.e., C programs) on the application layer, by which clients can send data packets to a server across the wireless link – either via UDP or TCP. Currently, the MAC/PHY layers of the laptops comply with the IEEE 802.11b protocols. For our measurements, the Distributed Coordination Function (DCF) with the Basic Access (BA) handshaking scheme was employed. Apart from timestamps taken in the client/server on the application layer (measurement points APP), from which end-to-end delays are computed (after offline-synchronization), the netfilter framework [5], which is at the base of the measurement infrastructure, allows us timestamp packets at various "hooks" (in the network layer) on their way through the protocol stack. Besides the measurement points APP for both client and server, Figure 1 shows the four netfilter hooks (OUT, POST, PRE, IN) passed by a UDP packet on its way from client to server. Essentially, these four measurement points are before and after the routing decision in the network layer (in either direction). Additionally, timestamps are recorded upon reception of a packet in the network card (measurement point DEV in Figure 1). In the measurement infrastructure, the timestamps (and packet header information) at the latter five measurement points are collected by Linux kernel modules and eventually written to ASCII files. The timestamp files are further processed by custom-designed Java scripts to yield interarrival time or delay distributions at or between measurement points. For one-way delays between two machines (with unsynchronized clocks), the timestamps are synchronized offline involving identical GPS signals received by each station once per second (PPS-API – pulse per second [4,6]).

In total, we can thus measure six contiguous delays (APP-OUT, OUT-POST, POST-DEV, DEV-PRE, PRE-IN, IN-APP) for a single UDP transmission. Between two identical laptops (HP OmniBook XE3, 900 MHz), we roughly obtain the sample means of the respective delays as shown in Table 1. (The delays were measured at very low system load.) On the average, it takes slightly more than 1 ms to send a payload of 100 byte from application layer to application layer. The last column indicates whether the delay is quasi insensitive to the payload (none), grows linearly with the payload (linear) or is affected by more complex (random) effects (irregular).



The histogram of delay PRE-IN to the right (which appears not to depend on the payload, here 100 byte) is a rather representative example. The sample mean is slightly greater than 15 microsecs and the squared coefficient of variation around 0.05. The figure was produced from 100000 pieces of data with the R tool [1]. It is obvious from this trimodal histogram that one cannot easily fit a theoretical distribution to this data. In fact, standard distribution-fitting software does not deliver an acceptable fit for any of the six delay distributions. However, no significant autocorrelation was observed in delay data. Thus, we decide to reflect the measured delays in our simulation model as empirical distributions (according to [8]). In case of linear dependence of the delays on the payload, the empirical distributions are shifted according to an interpolation between three reference measurements (with payloads 100, 500 and 1200 byte). In case of delay IN-APP, the precise approach to capture the irregular effects is outlined in [8].

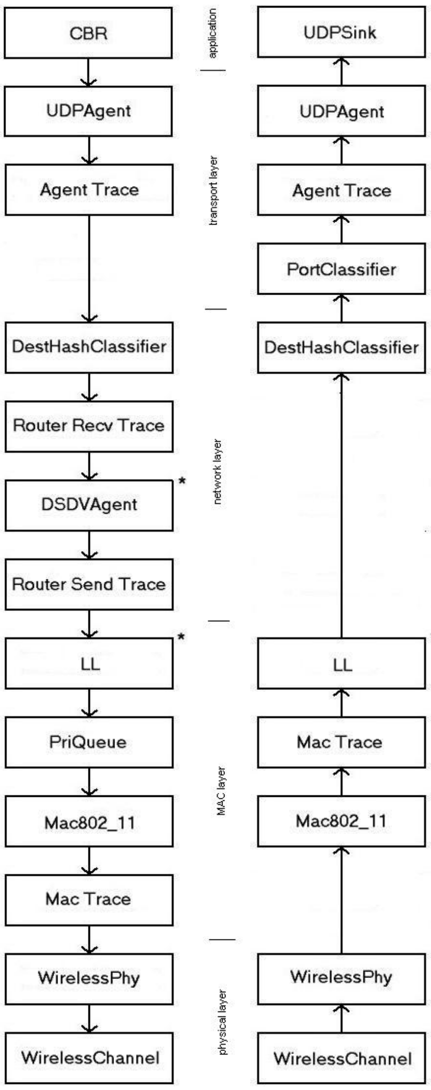
## 3 Simulation Modeling with ns-2

The freely available network simulator ns-2 [2] comes with a number models for network protocols (encoded in the programming language C++), among them TCP/UDP, IP and IEEE 802.11. Complex models are rather conveniently composed in the object-oriented script language OTcl, by which the various submodels can be parameterized. Thus, a model scenario of a WLAN, which corresponds to our measurement setup and which includes the complete protocol stack, is quickly evaluated – theoretically without any knowledge of the C++ code. (In practice, the current ns-2.28 distribution still includes some bugs, which need to be remedied by hand. For instance, as also known from appropriate mailing lists, the default IEEE 802.11 protocol model always goes into backoff irrespective of sensing an idle or busy channel. Also, the size of the 8-byte UDP headers are ignored. ).

When performing experiments with the thus corrected model, one observes that the end-to-end delays for UDP transmissions over a wireless link between two stations exhibit a constant value. For payloads of 100 byte in low load, one obtains around 0.4 ms, which is significantly lower than the 1.065 ms we observed in our real system. In addition, the real end-to-end delays follow some non-deterministic distribution.

Our primary goal now is to enhance the standard ns-2 model such that it reflects the delays that are measured in our real system by means of the measurement infrastructure. We will appropriately integrate empirical distributions in the ns-2 protocol stack (as an input modeling step) and validate the end-to-end delays of the simulation model with measurements.

In ns-2, a packet object is passed through several C++ objects on its way from sender to receiver. To the right, the C++ objects are depicted that a payload sent via UDP traverses in the sender (left-hand side) and in the reveiver (right-hand side). Additionally, the protocol layers, to which the objects may be attributed, are indicated. For instance, we assumed in the figure that payloads are generated by an application according to a CBR source. The blocks labelled *WirelessChannel* indeed denote the same object. Trace objects serve to collect statistics during the simulation. In objects marked with an asterisk, packets may be delayed by the default ns-2 model. In low-load conditions, the DSDVAgent does not contribute to the end-to-end delay.



Logical and technical considerations influence the decision to which objects the empirical distributions (from input modeling) should be linked. Technically, the corresponding C++ object should be accessible from the OTcl script, as this represents the user interface. For example, in the receiving branch, only the objects Mac802_11 and UDPAgent can be addressed from the OTcl script. Logically, the delays must be inserted in such a way that only the intended packets suffer the appropriate delay. For example, the delay PRE-IN cannot be modeled in Mac802_11, because then also packets not directed to this receiving station would encounter this delay. Therefore, incorporating this delay in the UDPAgent appears to be better suited (even though this requires a redundant incorporation for TCPAgents).

Following a thorough discussion in [8], the delays APP-OUT and OUT-POST are associated with the UDPAgent (at the sender), POST-DEV and DEV-PRE with the Mac802_11 object (at the receiver), PRE-IN and IN-APP with the UDPAgent (at the receiver). Special care must be taken for the delay POST-DEV: Only part of this delay, namely the actual transmission time, blocks the wireless channel for other stations. In addition, this transmission time depends on the selected bit rate.

Generally, by implementing a function *int command(…)* in the C++ class, an OTcl script can access a corresponding C++ object in order to perform an arbitrary action. We exploit this mechanism to link objects with available delay data files for a fixed reference payload, from which empirical distributions are constructed before simulation start. Additional parameters (e.g., the slopes for linear interpolation) are provided.

## 4 Validation and Simulation Results

Input modeling and validation require different measurements, under low load in the former case and under moderate/high load in the latter case. We validate our model by comparing the end-to-end delay distributions of generated data packets for different loads in ad-hoc networks with two stations. Again, the real system comprises the two HP OmniBook XE3 laptops, while measurements taken from this system enhance the ns-2 model for IEEE 802.11 WLANs as described in the previous section.

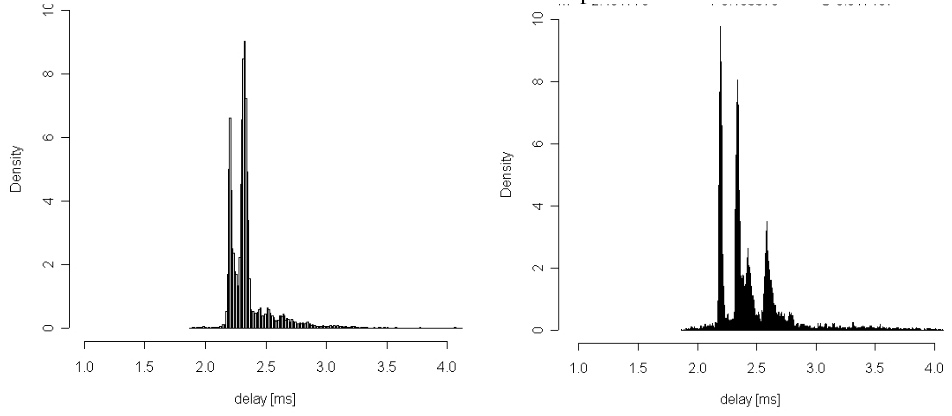


**Figure 4:** Measured (left) and simulated (right) end-to-end delay for a single UDP connection with exponential interarrival times (mean 4ms) and fixed payload of 800 byte

Figure 4 compares the end-to-end delay densities of UDP packets of 800-byte payload, which are sent according to a Poisson process with rate 0.25/ms. This is still a low-load condition, and only the greater payload accounts for the larger mean of around 2.5 ms (as compared to Table 1). In this experiment (with 100000 samples taken for the end-to-end delay), the simulated mean is slightly less than the measured one (by 1.9%). However, the coefficients of variation (CV) of the end-to-end delays differ significantly: For the measurements, we have 0.0251 vs. 0.0172 for the simulation model. Generally, it seems that composing the end-to-end delay of several *independent* random variates in simulation may be suitable for the expected delay, but cannot reproduce the jitter. Since no autocorrelation was observed for the variates, cross-correlations might have to be considered to improve the situation.

In a second experiment, we add another UDP connection to the existing one above – with identical interarrival process and packet sizes, but in opposite direction (i.e., from B to A instead of A to B). Figure 5 now clearly shows the impact of an eventually blocked channel and possible collisions on the end-to-end delays of the original connection. The distributions for both measurement and simulation model become skewed to the right with increased means (now 3,47 ms vs. 3.40 ms, i.e., a 2.1% deviation). The simulation model (right-hand side) produces more values closer to the mean again leading to a significantly lower CV of 0.173 (as compared to 0.242 for the measurements).

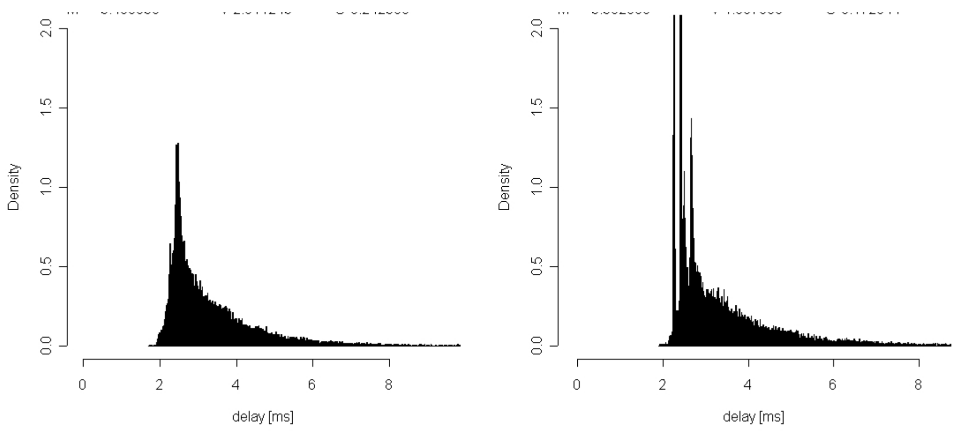


**Figure 5:** Measured (left) and simulated (right) end-to-end delay for two opposite UDP connections with exp. interarrival times (mean 4ms each) and fixed payload of 800 byte

## 5 Conclusions

Our experiments, including many other results in [8], show that mean end-to-end delays of a real system may well be predicted by a simulation model enhanced by empirical distributions as input models for the system-intrinsic delays. However, already for the jitter (i.e., CV), more sophisticated input modeling, which takes into account the cross-correlations between these delays, becomes necessary. Another take-home lesson is that one should not expect the ns-2 WLAN model to produce meaningful results with respect to a specific real system, unless this model is appropriately extended and tuned to this system.

## 6 References

[1] *The R Project.*: R: A language and environment for statistical computing. http://R-project.org

[2] *The VINT Project*: The network simulator ns-2. http://www.isi.edu/nsnam/ns.

[3] *Hielscher, K.-S., German, R.*: A low-cost infrastructure for high-precision high-volume performance measurements of web clusters. Proc. 13th TOOLS 2003, Urbana, IL, USA, pp. 11-28.

[4] *Fischer, F.*: Offline-Synchronisation von Zeitstempeln. Studienarbeit, Informatik 7, Universität Erlangen-Nürnberg, 2004.

[5] *The Netfilter project*: http://www.netfilter.org.

[6] *Windl, U.*: PPSKit. ftp://ftp.kernel.org/pub/linux/daemon/ntp/PPS.

[7] *Law, A, Kelton, W..*: Simulation, Modeling and Analysis. McGraw-Hill, Boston, 2000.

[8] *Resch, C.*: Messungsbasierte Modellierung von drahtlosen lokalen Netzen. Studienarbeit, Informatik 7, Universität Erlangen-Nürnberg, 2005.