D 2019

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

IMPROVING THE PERFORMANCE EVALUATION OF WIRELESS NETWORKS
TOWARDS A SIMULATION-EXPERIMENTATION SYNERGY USING NS-3

HELDER MARTINS FONTES
TESE DE DOUTORAMENTO APRESENTADA
À FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO EM
ENGENHARIA INFORMÁTICA

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Improving the Performance Evaluation of Wireless Networks: Towards a Simulation-Experimentation Synergy using ns-3

Helder Martins Fontes

PH.D. THESIS

Doctoral Program in Informatics Engineering

Supervisor: Manuel Alberto Pereira Ricardo (PhD)

Co-Supervisor: Rui Lopes Campos (PhD)

June 13, 2019

Improving the Performance Evaluation of Wireless Networks: Towards a Simulation-Experimentation Synergy using ns-3

Helder Martins Fontes

Doctoral Program in Informatics Engineering

June 13, 2019

Abstract

The increasing need of wireless communications in emerging scenarios, including aerial and maritime, requires the development of new protocols and the enhancement of existing protocols. To properly validate these developments, we depend on performance evaluation of wireless networks, traditionally considering Simulation and Experimentation activities. Simulations are flexible but usually produce optimistic results, assuming many simplifications to represent complex scenarios such as those related to emerging aerial networks. Testbeds, on the other hand, are increasingly costly to maintain and are frequently unavailable; still, they provide realistic results and are an essential step to achieve proper protocol validation and fine tuning. The problem is that real wireless testbed experiments in emerging networking scenarios are hardly repeatable. Given the same input, they can produce very different output results, since wireless communications are influenced by external random phenomena such as noise, interference, and multipath. Real experiments are also difficult to reproduce. Either the original community testbed can be unavailable – offline or running other experiments – or the custom testbed becomes inaccessible. Without repeatability and reproducibility, the validity of the protocol developed is questionable. Also, there is duplication of effort to develop simulation and real implementation of the protocols, which can be error-prone and lead to non-comparable results.

Based on the identified problems and on our experience in different research projects, in this thesis we propose the use of ns-3 as a common platform to enhance and promote the interaction between Simulation and Experimentation. The objective is to foster the synergy between Simulation and Experimentation, improving the process of performance evaluation of wireless networks. This synergy is addressed in both directions, resulting in two related main original contributions: a) from simulation to experimentation – the Fast Prototyping development process; b) from experimentation to simulation – the Trace-based Simulation (TS) approach. The Fast Prototyping development process allows to reuse the ns-3 protocol model implementation in the real prototype for obtaining comparable results, reducing code duplication and potential implementation differences. In addition, it improves the ns-3 emulation functionality and performance, with a 23-fold increase in throughput and a 15-fold decrease in Round-Trip Time (RTT), when compared to the current ns-3 emulation performance in a scenario of interest. The TS approach captures the physical conditions of real experiments (e.g., radio link quality, the positions of the nodes) and relies on ns-3 TCP/IP and MAC simulation capabilities to enable the reproduction of real experiments in simulation. The TS approach was extensively evaluated, producing more accurate results than the pure simulation alternatives; it achieved average gains above 53% and 90th percentile gains above 57% in scenarios of interest.

With comparable results using the Fast Prototyping development process, and repeatable and reproducible past real experiments using the TS approach, the interaction between Simulation and Experimentation activities results in an enhanced Simulation-Experimentation synergy that improves the process of performance evaluation of wireless networks.

Resumo

A necessidade crescente de comunicações sem fios em cenários emergentes, incluindo aéreo e marítimo, requer o desenvolvimento de novos protocolos e o melhoramento de protocolos existentes. A validação destes desenvolvimentos depende da avaliação de desempenho das redes sem fios, que tradicionalmente contempla atividades de Simulação e Experimentação. As simulações são flexíveis mas tipicamente produzem resultados optimistas, assumindo varias simplificações ao representar cenários complexos como os das redes aéreas emergentes. As *testbeds*, por outro lado, são cada vez mais caras de manter e, por vezes, encontram-se indisponíveis; contudo, elas fornecem resultados realistas e são um passo essencial para a avaliação de desempenho completa. O problema é que as experiências reais realizadas em *testbeds* sem fios que operam em cenários emergentes são dificilmente repetíveis. Dadas as mesmas entradas elas podem produzir resultados muito diferentes, uma vez que as comunicações sem fios são influenciadas por fenómenos externos aleatórios como o ruído, interferência, e o *multipath*. Experiências reais são também difíceis de reproduzir. Ou a *testbed* comunitária original pode estar indisponível – desligada ou ocupada a correr outras experiências – ou a *custom testbed* original deixa de estar acessível. Sem repetibilidade e reprodutibilidade, a validade do protocolo desenvolvido é questionável. Existe, também, duplicação de esforço ao desenvolver a implementação de simulação e real dos protocolos, o que pode tornar-se numa fonte de erros e impedir a comparação de resultados.

Com base nos problemas identificados e na nossa experiência em diferentes projetos de investigação, nesta tese propomos a utilização do ns-3 como plataforma comum para melhorar e promover a interação entre atividades de Simulação e Experimentação. O objetivo é a sinergia entre as duas atividades, melhorando o processo de avaliação de desempenho de redes sem fios. Esta sinergia é abordada nos dois sentidos, resultando em duas principais contribuições relaccionadas: a) da simulação para a experimentação – processo de desenvolvimento *Fast Prototyping*; b) da experimentação para a simulação – abordagem *Trace-based Simulation* (TS). O processo de desenvolvimento *Fast Prototyping* permite reutilizar o modelo de implementação em ns-3 num protótipo real para obter resultados comparáveis, reduzir a duplicação de código e potenciais diferenças de implementação. Adicionalmente, ele melhora a funcionalidade e o desempenho de emulação do ns-3, com um aumento de 23x no *throughput* e uma redução de 15x no RTT, quando comparados com o desempenho atual da emulação em ns-3 num cenário de interesse. A abordagem TS captura as condições físicas de experiências reais (ex., qualidade da ligação de rádio, posições dos nós) e baseia-se no realismo de simulação do ns-3 ao nível MAC e TCP/IP para permitir a reprodução de experiências reais em simulação. A abordagem TS foi extensivamente avaliada, produzindo resultados mais precisos do que as alternativas de simulação pura; ela conseguiu ganhos médios acima de 53% e ganhos do percentil 90 acima de 57% em cenários de interesse.

Com resultados comparáveis usando o processo de desenvolvimento *Fast Prototyping*, e resultados repetíveis e reprodutíveis de experiências reais passadas usando a abordagem TS, a interação entre atividades de Simulação e Experimentação resulta na sinergia melhorada entre as duas, melhorando assim o processo de avaliação de desempenho de redes sem fios.

Acknowledgments

I would like to express my total gratitude to all the extraordinary people which made possible the development of this work in the investigation environment of INESC TEC.

I feel pleased to have integrated the WiN (Wireless Networks) group of the Centre for Telecommunications and Multimedia (CTM) at INESC TEC, whose elements unconditionally helped and supported the various steps of this work. They would be too many to reference separately in this short part, but their contribution is present all over this work.

I would especially like to express my large gratefulness towards my supervisors Prof. Manuel Ricardo and Prof. Rui Campos for their friendliness and absolutely outstanding support. Thanks to them, and the other elements, work had always been an object of personal interest, reflection, and constructive discussion. I owe them the vast wireless networks knowledge they transmitted during these years and the patience to answer all my questions. I would also like to express my deepest gratitude to Thomas Henderson, for his invaluable input to this work.

I would like to acknowledge all my family, friends and especially my wife and parents who, in some way, made it possible for me to be here through their unconditional support in all situations of my life.

I would also like to thank the support from the Portuguese Foundation for Science and Technology (FCT) under the fellowship SFRH/BD/69051/2010. This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundacao para a Ciencia e a Tecnologia within project POCI-01-0145-FEDER-016744. This work was also part of the FP7 SUNNY project (id: 313243).

Helder Martins Fontes

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
AGC	Auto-Gain Control
ARP	Address Resolution Protocol
AUV	Autonomous Underwater Vehicle
BLUECOM+	Connecting Humans and Systems at Remote Ocean Areas using Cost-effective Broadband Communications
BS	Base Station
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
CSI	Channel State Information
DCE	Direct Code Execution
DHCP	Dynamic Host Configuration Protocol
DPK	Data Plane in Kernel Space
DPU	Data Plane in User Space
EMI	Electromagnetic Interference
EU	European Union
FBMN	Flying Backhaul Mesh Network
FER	Frame Error Ratio
FEUP	Faculdade de Engenharia da Universidade do Porto
FMAP	Flying Mesh Access Point
INESC TEC	Instituto de Engenharia e Sistemas de Computadores, Tecnologia e Ciência
IP	Internet Protocol
IPC	Inter-Process Communication
LAN	Local Area Network
LoS	Line-of-Sight
ML	Machine Learning
MareCom	Maritime Community Networks and Services
MAP	Mesh Access Point
MCS	Modulation Coding Scheme
MIMO	Multiple-Input Multiple-Output
NSC	Network Simulation Cradle
ns-2	Network Simulator 2
ns-3	Network Simulator 3
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimized Link State Routing
OS	Operating System
PAN	Personal Area Network
PDR	Packet Delivery Ratio
PDU	Protocol Data Unit
PLR	Packet Loss Ratio

QREN	Quadro de Referência Estratégico Nacional
R&D	Research and Development
RFI	Radio-Frequency Interference
RSSI	Received Signal Strength Indication
RTT	Round Trip Time
SDN	Software Defined Networking
SIMBED	Offline Real-World Wireless Networking Experimentation using ns-3
SISO	Single-Input Single-Output
SITMe	Serviços Integrados para Transportes Metropolitanos
SNR	Signal-to-Noise Ratio
STCP	Sociedade de Transportes Colectivos do Porto
SUNNY	Smart UNattended airborne sensor Network for detection of vessels used for cross border crime and irregular entrY
TTL	Time to Live
TVWS	Television White Spaces
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
WAN	Wide Area Network
WISE	traffic-aWare flyIng backhaul meSh nEtworks
WMRP	Wireless Metropolitan Routing Protocol

Chapter 1

Introduction

1.1 Context and Motivation

In the last decade, we have assisted to the proliferation of Wireless Networks in different application domains, such as Wide Area Network (WAN), Local Area Network (LAN), and Personal Area Network (PAN). In these application domains, increasingly complex scenarios are being considered which may have unique characteristics and requirements that are not yet fully addressed by State-of-the-Art solutions. Examples of such scenarios may include emerging vehicular networks that operate at the extreme conditions experienced by Unmanned Aerial Vehicles (UAVs), Unmanned Surface Vehicles (USVs), and Autonomous Underwater Vehicles (AUVs). The more demanding scenario characteristics, such as very dynamic wireless link quality and network topology, as well as application requirements – e.g., higher bandwidth, quicker link establishment, and faster handover – push further the development of new protocols and the enhancement of existing protocols.

To properly quantify the enhancements introduced by such developments we highly depend on the performance evaluation of wireless networks. Wireless networking systems are too complex to be evaluated analytically. Therefore, we typically evaluate wireless networking systems using two processes: 1) network **simulation** based on mathematical models of the real system; 2) network **experimentation** performed in real testbeds. Simulation and experimentation evaluation processes are both essential and complementary for evaluating the performance of the protocols being developed and for achieving their proper validation and fine-tuning.

In the Wireless Networks (WiN) group [2] of INESC TEC [3], we typically use Network Simulator 3 (ns-3) [4][5] in the performance evaluation process. ns-3 is an open-source, discrete-event and packet level network simulator for Internet systems, targeted primarily for research and educational use. ns-3 provides an architecture that eases the implementation of new simulation modules. Also, it allows to create complex, though fully controlled, repeatable, reproducible, easily observable, and inexpensive to run simulation scenarios. Being a simulated environment, these scenarios may involve new technologies that are not yet available to use in a real system (e.g., a new IEEE 802.11 variant or a new routing protocol), and scale to the number of network nodes, or scenario variations, needed to perform the necessary evaluations. Simulation, however, may be unable to accurately represent the very complex characteristics present in emerging networking scenarios, as the simulation models are abstractions of the real systems or the phenomena they are trying to represent. In such cases it is especially important to complement the performance evaluation process with network experimentation.

In the WiN research group we typically use custom real testbeds in the performance evaluation process. In real testbeds we can experiment and evaluate the real performance of the protocol, running it in a real hardware prototype and collecting experimental results that are typically more accurate than the simulation results. Although this appears to be the better performance evaluation process, especially considering the decreasing size and cost of computer hardware, in real testbeds we typically have problems such as scarce resources – e.g., money, time, space, hardware, and testbed availability –, which directly limits the testbed scale and the number and duration of experiments we are able to run. On top of this, the real scenario may be highly unstable (e.g., emerging aerial wireless networks), which greatly limits the repeatability and reproducibility of the experiments, thus affecting the comparison of the results obtained for different runs.

Most of the research projects developed in the WiN research group, further detailed in Section 1.7, involve the design and development of custom-tailored protocols to address very different communications systems requirements, typically operating in emerging networking scenarios. Usually, for each project, a combination of performance evaluation based on simulation and experimentation is used. This combination helps to achieve better validation and fine-tuning of the protocols before deploying them in the real environment. Nevertheless, our hands-on experience allowed the identification of problems regarding this combined performance evaluation process. These problems are detailed in Section 1.2.

1.2 Problem Definition

The typical development process of new protocols for communications systems generally involves the four phases represented in Figure 1.1: design of the protocol concept, evaluation in a network simulator (Simulation), evaluation in a real testbed (Experimentation), and final deployment in a production system.

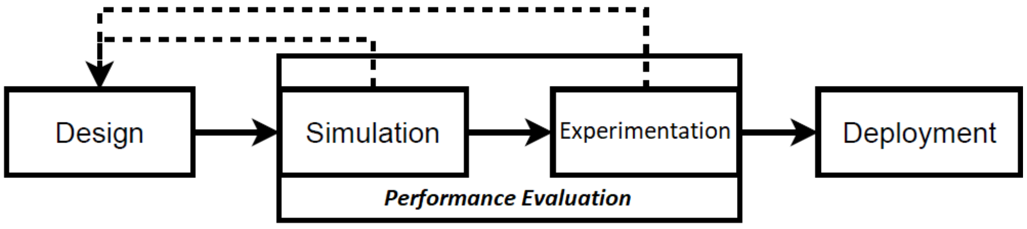


Figure 1.1: Overview of the typical development process of protocols for communications systems highlighting the importance of the Performance Evaluation.

Creating new protocols for wireless communications systems is highly dependent on the performance evaluation phases that rely on simulation and experimentation. Traditionally, a protocol simulation model is created from the design of the protocol. Multiple simulations are run and the results are analyzed and used to improve the protocol. When the simulation results are acceptable, development can continue to the next phase. A prototype of the protocol is implemented in a real system and run in a testbed to be validated. This development process is usually iterative. If the results from simulation or experimentation do not meet the requirements, the protocol design needs to be changed and new simulations and experiments are carried out. At this point, two implementations of the protocol are being maintained: 1) the **simulation model**; 2) the **implementation prototype**. This leads to duplication of effort when a change needs to be made in the protocol, which, in turn, increases both the development time and the chance of unwilling introduction of human errors in the implementations, making their results non-comparable.

Emerging vehicular network testbeds, such as those composed of UAVs, present very unique and extreme physical characteristics, namely highly unstable Signal-to-Noise Ratio (SNR) at receivers due to antenna misalignment and obstruction, asymmetric radio link SNR due to different noise exposures, and unstable nodes' mobility patterns. These characteristics are not accurately reproduced in simulation, thus making the results from simulation and experimentation non-comparable. Also, the performance evaluation in the real testbeds cannot be done extensively due to costly logistics operations, small testbed scale, and limited duration of the experiments. These limitations greatly affect the reproducibility and comparison of the performance evaluation results.

In summary, without comparable results between simulation and experimentation evaluation phases, and without repeatable and reproducible results between different runs of the same real experiment, the performance evaluation of a wireless communications system under study cannot be properly carried out. Ultimately, the validity of the protocol developed – along with its possible gains compared to State-of-the-Art alternatives – becomes questionable.

1.3 Objectives

The main goal of this thesis is to address the problems identified in Section 1.2, in order to improve the process of performance evaluation of wireless networks. By introducing new forms of interaction between Simulation and Experimentation evaluation phases, this work aims at using ns-3 for improving not only both phases individually, but mainly the performance evaluation process as a whole. To achieve the Simulation-Experimentation synergy using ns-3, our objective is to explore how each phase can benefit from the output of the other without affecting each other's expected functionality and performance.

An overview of the proposed interaction between Simulation and Experimentation evaluation phases is depicted in Figure 1.2. From simulation to experimentation (A), our objective is to use a shared protocol model implementation between simulation and experimentation to avoid potential implementation differences and non-comparable results. More specifically, we aim to explore

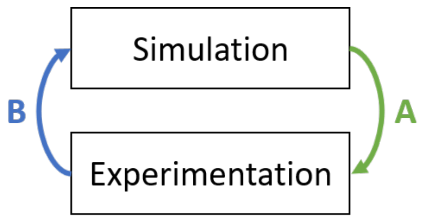


Figure 1.2: Overview of the proposed interactions between Simulation and Experimentation which promote the Simulation-Experimentation Synergy: A) reuse ns-3 protocol model implementations (as emulated resources) in real experiments; B) improve simulation accuracy based on physical conditions and results from past real experiments.

and improve the ns-3 emulation functionality, which has the potential to combine emulated (simulations using ns-3 emulation) and real resources in the same test run. From experimentation to simulation (B), our objective is to improve the simulation accuracy based on the physical conditions captured from real experiments and enable repeatability and reproducibility, namely in emerging networking scenarios.

1.4 Challenges

This section presents the main challenges faced during this PhD work in order to achieve the two main objectives defined in Section 1.3. The challenges related to each objective are defined in the following.

Experimentation benefiting from simulation:

- Assessment of the correct operation and performance overhead of the ns-3 emulation mode and how it affects the network performance of real testbeds.
- Improvement of the compatibility of the ns-3 emulation mode to work with different real-world network configurations and interfaces.
- Reduction of the ns-3 packet processing overhead in order to keep real-time operation and performance while scaling the network capacity or the number of emulated nodes per simulation process.

Simulation benefiting from experimentation:

- Identification of the suitable variables to capture from real experiments, in order to reproduce them in simulation while dealing with the hardware limitations on reporting up-to-date and accurate data.

1.5 Thesis Hypothesis

It is possible to use ns-3 as the platform to improve the process of performance evaluation of wireless networks enabling: 1) the combination of emulated and real testbed resources in the same experiment, while maintaining the designed system behavior and performance; 2) the perpetuation of real experiments via trace-based ns-3 simulations that are more accurate than pure simulation.

1.6 Original Contributions

The two main original contributions provided by this thesis are the following:

1. **Fast Prototyping Development Process.** This is a new shared protocol model implementation process to be used during the performance evaluation phases of protocol development. By reusing the already implemented ns-3 protocol model over the real hardware prototype, then installed in a testbed, we eliminate the duplicate effort to develop simulation and real implementations. This reduces the coding effort and also the chance for error introduction, which could render the results non-comparable. This process relies on ns-3 emulation functionality which allows to run simulated resources in real time, interacting with the real network interfaces. This contribution includes the following specific contributions: 1) improvements to the compatibility of ns-3 emulation to a larger set of real network interface types and operational restrictions encountered in the real-world networks used by the testbeds; 2) different approaches to improve the ns-3 emulation performance over the real testbed hardware. The novelty of this process, when compared to the state-of-the-art alternatives, relies on maintaining the benefits of ns-3 regarding the easiness and flexibility of implementation, the important log functionality it provides, and the ability to combine, in the same run, simulated and emulated resources, which can improve the testbed in scale and functionality.

2. **Trace-based Simulation Approach.** This is a novel simulation approach allowing to perpetuate past real-world experiments and rerun them independently of the testbed availability and the external phenomena influencing its physical conditions. This is possible by recording traces of such physical conditions and reproducing them using a trace-based simulation. Trace-based simulation is especially important considering very unpredictable and unstable scenarios such as the emerging wireless vehicular networking scenarios. Using trace-based simulations we can reproduce the same conditions encountered in the real experiment runs. The novelty of our approach, when compared to the state-of-the-art, is that we only reproduce the physical conditions (radio links characteristics and the positions of the nodes) and rely on ns-3 TCP/IP and MAC simulation capabilities for the upper layers.

Five conference papers and two journal papers were produced as a direct result of this thesis:

1. **ns-3 NEXT: Towards a Reference Platform for Offline and Augmented Wireless Networking Experimentation** in Proceedings of the Workshop on ns-3, 2019, Florence, Italy (Conference) [6];

2. **Improving ns-3 Emulation Performance for Fast Prototyping of Routing and SDN Protocols: Moving Data Plane Operations to Outside of ns-3** in Simulation Modelling Practice and Theory, 2019 (Journal) [7];

3. **Improving the ns-3 *TraceBasedPropagationLossModel* to Support Multiple Access Wireless Scenarios** in Proceedings of the Workshop on ns-3, 2018, Surathkal, India (Conference) [8];

4. **A Trace-Based ns-3 Simulation Approach for Perpetuating Real-World Experiments** in Proceedings of the Workshop on ns-3, 2017, Porto, Portugal (Conference) [9];

5. **Improving ns-3 Emulation Performance for Fast Prototyping of Network Protocols** in Proceedings of the Workshop on ns-3, 2016, Seattle, WA, USA (Conference) [10];

6. **Improving ns-3 Emulation Support in Real-World Networking Scenarios** in Proceedings of the 8th International Conference on Simulation Tools and Techniques, 2015, Athens, Greece (Conference) [11];

7. **Fast Prototyping of Network Protocols Through ns-3 Simulation Model Reuse** in Simulation Modeling Practice and Theory, 2011 (Journal) [12];

1.7 Related Research Projects

This section presents some research projects we actively contributed to as researchers at INESC TEC, which inspired and directly influenced this thesis. The acquired hands-on experience in these projects helped to: 1) better define relevant real-world problems to be addressed during this PhD; 2) assess, with first-hand experience, the added value introduced by the original contributions. For each project, a brief description is presented, in order to illustrate the context of this PhD work and motivate with real examples the application of the approach proposed herein.

The SITMe [13] project, whose reference architecture is shown in Figure 1.3, had the goal of developing and supplying information services to passengers traveling in buses, including displaying news and entertainment information in on-board screens and providing Internet access to the passengers using Wi-Fi. Herein, the terms 'Wi-Fi' and 'IEEE 802.11' are used with the same meaning. For this purpose, INESC TEC developed a multi-technology communications system. The communications system had to operate with multiple wireless access networks (HSPA+, IEEE 802.11, and IEEE 802.16e), supporting handovers between them and terminal mobility. On top of this, the network architecture had to be scalable in order to support future network expansion to the entire public transportation fleet of the city of Porto. To validate the fulfillment of the project requirements and goals, INESC TEC greatly depended on the performance evaluation of wireless networks. Firstly, using ns-3 simulation; then, using experimentation over a real testbed instantiated by a pilot of 11 buses that ran for over a year. Simulation allowed to assess the solution scalability and its correct operation in a multitude of scenario variations, while the

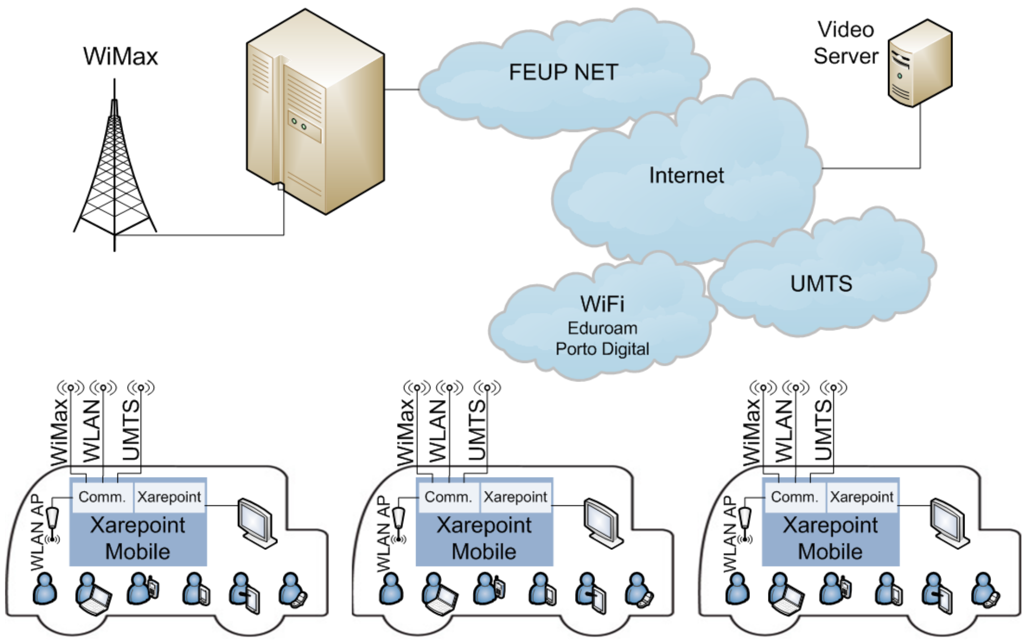


Figure 1.3: SITMe Communications System Overview.

experimentation allowed to further validate the solution using real hardware with real usage by the bus passengers. During this project, the problem of porting the simulation implementation to the real testbed became evident, as this duplication is time consuming and it is difficult to maintain both implementations synchronized, which may lead to non-comparable results. Related to the testbed results, we also found how important it is to have all the relevant operational logs automatically collected, organized and opportunistically uploaded to our experiments database, as we used this data very often to compare with simulation results and validate the system operation. In this project, we tested and validated the use of a shared protocol model implementation between the simulator and the real testbed.

The SUNNY [14] project aimed to develop a solution capable of improving EU border monitoring, compared to the legacy systems, whilst keeping affordability and interoperability as key enabling factors. The project has developed a novel intelligent and heterogeneous aerial sensor network, which provides improved cooperative sensing and real-time data processing and transmission capabilities, as illustrated in Figure 1.4. Each UAV had two redundant WLAN interfaces for Air-Air and Air-Ground data link communications, and a third wireless interface as the control and telemetry link. INESC TEC has been involved in the development of the communications system for the two redundant data-links. A testbed of 4 UAVs and a Ground Radio Base Station has been developed to validate the sensors, the on-board data processing units, and the communications system. In this project, we faced several issues regarding testbed availability due to the complex logistics and high costs involved, which resulted in very few UAV test flights. As such, we had to rely heavily on simulation for the performance evaluation of the wireless communications system. One problem was ns-3 simulation performance results were very optimistic when compared with the real performance results, due to the extreme physical environment char-

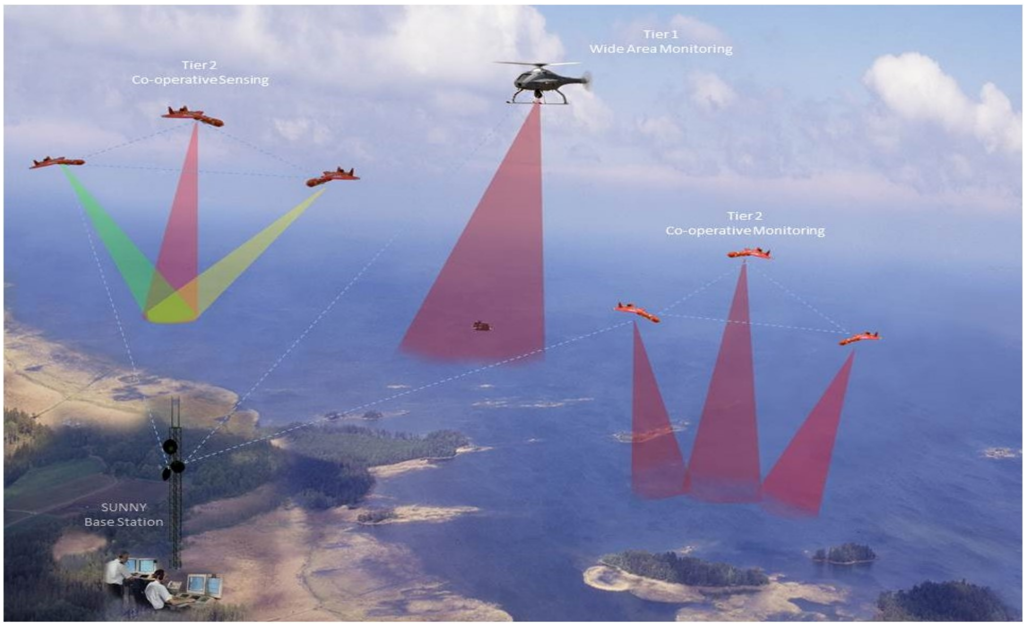


Figure 1.4: SUNNY Communications System Overview.

acteristics involved (e.g., high relative speeds, antenna misalignment, radio Line-of-Sight (LoS) obstruction by the UAV body, higher noise exposure). This influenced the radio links stability and performance. Another problem was that other project partners developing solutions at the application layer, thus acting as end-users of the communications system, for instance, UAV on-board data-processing equipment, highly depended on the realistic operation of the communications system to develop and fine tune their data transport solution: simulation results were, once again, too optimistic. In this project, we tested and validated the process of using past-experiments' traces, in order to reproduce those physical conditions in ns-3. This allowed offline validation and fine-tuning of the communications system and of the applications developed by the end-users.

The MareCom [2] project, whose system overview is shown in Figure 1.5, aimed at developing a highly available, broadband maritime communications solution based on the IP protocol and off-the-shelf hardware, in order to: 1) support the activities of the communities operating in maritime environment, such as fishermen, maritime transportation, recreational vessels and navies, with reliable and broadband communications; 2) enable the convergence with the terrestrial communications scenario, fostering the digital inclusion of these communities. For evaluating the developed wireless communications system INESC TEC has used simulations and a small-scale testbed (e.g., one fishing ship, one ship from the Portuguese Navy, and a land base station). Once again, we experienced physical layer radio link conditions that were unstable, due to the sea undulation, which constantly affected antenna alignment and, consequently, the radio channel. The sea conditions and the path of the vessels were also non repeatable, which affected the reproducibility of the experiments and the validation of new versions of the communications solution. External factors such as weather conditions, fishing quotas, and fishing interdiction periods also had an impact on the availability of the testbed, as the ships were retained at the harbor for long periods.

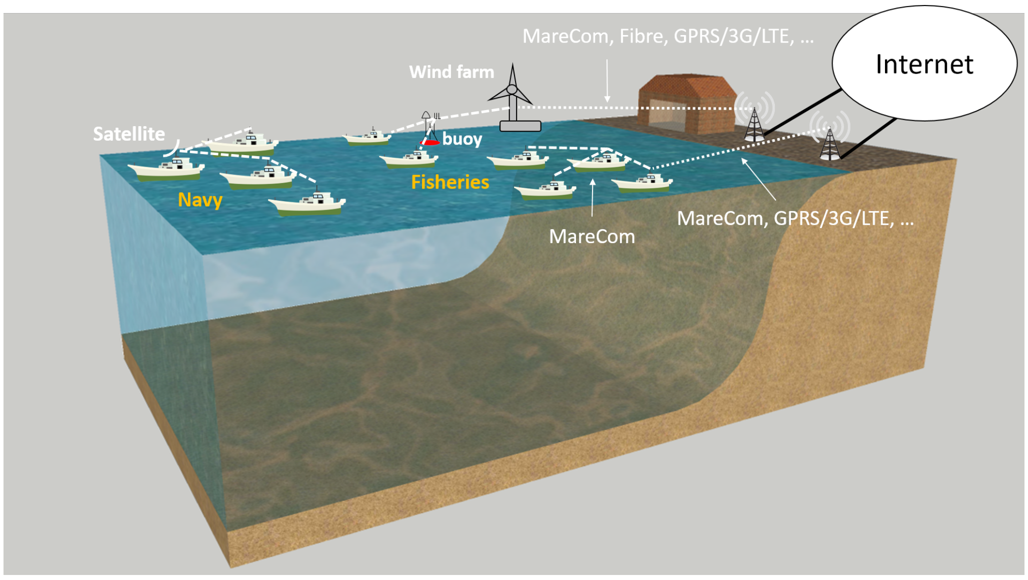


Figure 1.5: MareCom Communications System Overview.

Again, replaying the past experimental variable traces was important to this project, in order to keep fine-tuning and validating the wireless communications system.

The BLUECOM+ [15] project had the goal of developing an innovative communications solution that enables broadband, cost-effective Internet access at remote ocean areas to regular devices, using standard wireless access technologies such as GPRS/UMTS/LTE and IEEE 802.11. The long range radio links maintained by a flying network of balloons was our main focus at INESC TEC, as the rest of the communications solution highly depended on this backhaul network. The concept proposed by the project is illustrated in Figure 1.6. As in other projects, we firstly used simulation as a preliminary validation of the solution. Then, a testbed composed by a land base station and two balloons was deployed, and tests were run considering synthetic traffic and real applications. Similarly to what happened in the SUNNY project, the deployment of the testbed was very costly and involved very complex logistics, which highly limited the number and duration of the experiments performed. After running the experiments in real environment we found that, once again, simulation results were very optimistic. The wind caused oscillations in the balloons, which affected antenna alignment. The altitude of the balloons also played an important role in the real performance of the system, due to the constructive or destructive effect of the ray reflected in the water surface at the receiver. Replaying past experimental variable traces was relevant to keep fine-tuning the solution and also helped improving the simulation models used.

The SIMBED [16] project was defined with the goal of further validating our approach of replaying past experimental conditions in ns-3, as previously used in SUNNY, MareCom, and BLUECOM+. This time, instead of using custom made private testbeds from specific research projects, SIMBED focuses on using Fed4FIRE+ wireless testbeds – the world's largest federation of community testbeds – which operate in controlled indoor and outdoor environments, as illustrated in Figure 1.7. Fed4FIRE+ wireless testbeds such as w-iLab.t and NITOS, although

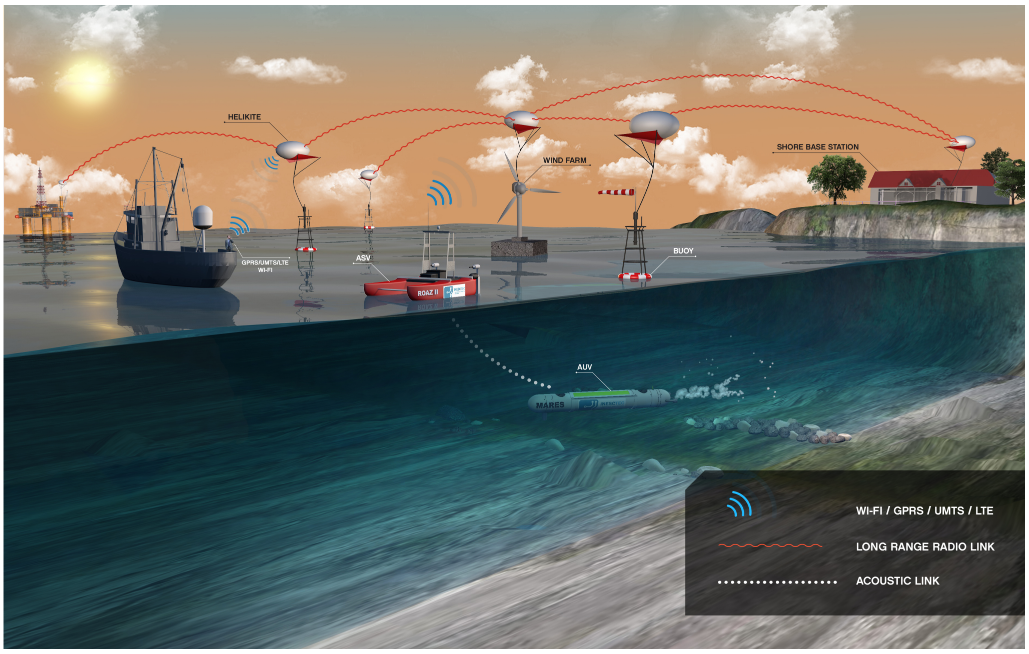


Figure 1.6: BLUECOM+ Communications System Overview.

deployed in controlled environments, are also affected by environmental phenomena (e.g., noise, interference, multipath), giving different results between multiple runs of the same experiment. To assure repeatability and reproducibility of experiments Fed4FIRE+ has tools to exclude results considered as outliers. In SIMBED we argued that these outliers are also representative of the real system operation and should not be discarded. Collecting traces of all runs of the experiment we should be able to replay those conditions in ns-3, producing equivalent results for every experiment execution – even the one that produced the outliers. SIMBED has allowed us to validate our approach in a large scale and scenario diversity that would be very difficult to accomplish using private testbeds. Such results are presented in this thesis.

The WISE [17] project aims to develop a new communications solution based on the concept of Flying Backhaul Mesh Networks (FBMN). The proposed concept is shown in Figure 1.8. At the core of the FBMN concept is the usage of UAVs, more specifically Quadrotors, which will carry on-board Mesh Access Points (MAPs) and will form a mobile and physically re-configurable wireless backhaul mesh network composed of Flying MAPs (FMAPs). This network will be self-organized and the FMAPs will position themselves according to the data traffic generated by the mobile terminals and the need to relay the traffic towards the Internet. INESC TEC is, at the moment, evaluating the communications system in ns-3. Later, a testbed composed of FMAPs installed in UAVs will be implemented. According to current legislation in Portugal, we face the need for a special authorization for flying UAVs over people. As such, we anticipate the need to combine emulated resources with real ones during the performance evaluation process, in order to achieve proper validation of the solutions. The interaction between real and emulated resources for enhanced experimentation is also addressed in this thesis.

Figure 1.7: Example of indoor and outdoor Wi-Fi testbeds used in SIMBED.

1.8 Thesis Structure

Each of the four main challenges identified in Section 1.4 are addressed in separate chapters, in the order they were listed. Chapters 2–4 are related to the first main original contribution – *Fast Prototyping Development Process* –, while Chapter 5 is related to the second main original contribution – *TS Approach*. The thesis is then structured as follows. Chapter 2 presents the Fast Prototyping development process. Chapter 3 presents our proposed solution to improve the Fast Prototyping compatibility with real-world networks by enhancing the functionality of ns-3 emulation. Chapter 4 describes the two proposed approaches to improve even further the Fast Prototyping development process, and reduce ns-3 emulation per-packet processing overhead. Chapter 5 presents the Trace-based Simulation (TS) approach. Finally, Chapter 6 draws the conclusions and presents the future work.

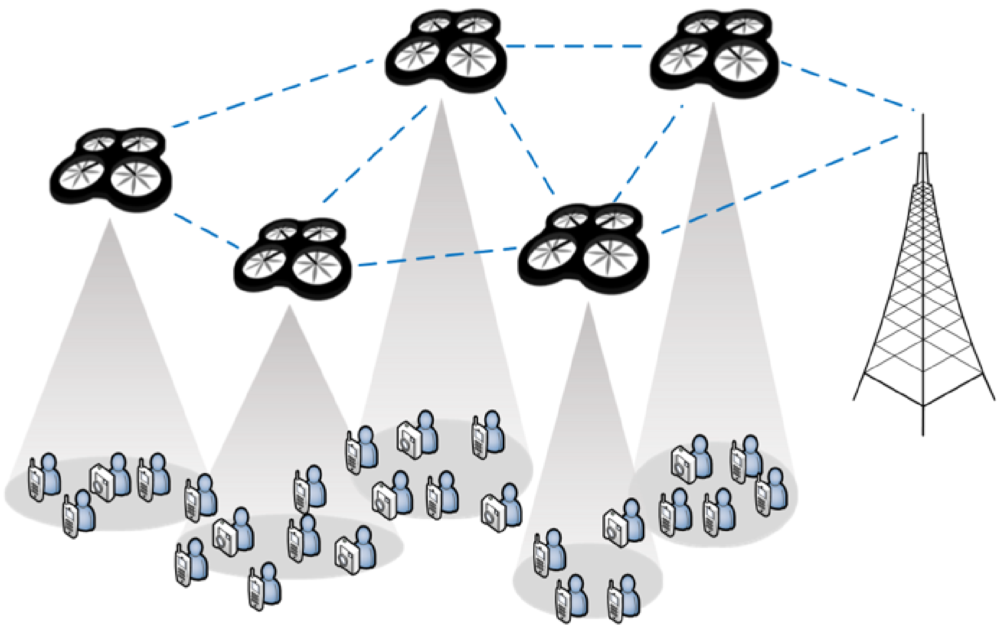


Figure 1.8: WISE Communications System Overview.

Chapter 2

Fast Prototyping of Network Protocols through ns-3 Simulation Model Reuse

Research and development in the area of wireless communications protocols is a fast evolving field of research. Considerable time is spent by researchers and developers from the design of a protocol to the deployment of its implementation in real systems. One of the main contributors for the time spent is the need to properly validate the protocol through performance evaluation, both via simulation and experimentation. For that purpose, a simulation model and an implementation prototype need to be developed for the same protocol design. We have past experience [18] with this time-consuming duplicate effort on evolving the Wireless Metropolitan Routing Protocol (WMRP) [19] version for the SITMe project. The evolved WMRP version needed to be thoroughly simulated and then implemented in a real prototype to be executed in the SITMe real testbed. Motivated by this hands-on experience we started to explore ways to overcome this problem: what if we could develop a single shared protocol model for simulation and experimentation?

In this chapter we present a protocol development process, named Fast Prototyping, that explores that possibility. The Fast Prototyping process uses a shared protocol model implementation that takes advantage of the existing ns-3 emulation capability. In order to assess the feasibility of the Fast Prototyping process, we compare the packet processing performance in terms of achievable throughput, packet loss ratio, and Round Trip Time (RTT), when ns-3 emulation mode and pure kernel space IPv4 forwarding are used. Finally, we point out the open problems to be addressed in order to improve the Fast Prototyping process. These problems are the subject of Chapter 3 and Chapter 4.

2.1 Traditional Protocol Development Process

Traditionally, developing a new network protocol involves the phases depicted in Figure 2.1. In the Design phase, we assess the scenario requirements and problem we want to solve and create the protocol specification. Then, in the Simulation phase, we create the Simulation Model for the protocol and run multiple simulations using a given set of input parameters. When the results are acceptable, we move to the Experimentation phase and create the Implementation Prototype, which runs in the real hardware of the testbed. If the simulation or the experimental results are not satisfactory, the protocol specification is updated and new results from Simulation and Experimentation obtained, reflecting the new protocol specification. Finally, when the protocol specification is stable, after its performance is successfully validated in Simulation and Experimentation, the protocol is ready for final deployment in the production system (Deployment phase).

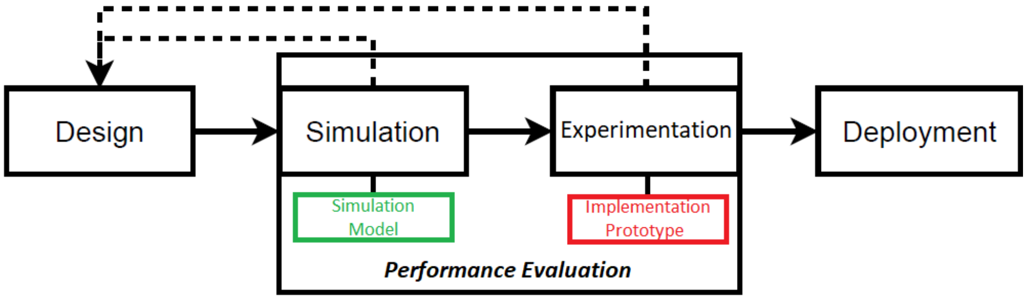


Figure 2.1: Traditional protocol development process resulting in a separate Simulation Model and Implementation Prototype.

This development process has problems that become evident when focusing on the Performance Evaluation phases. First, there is a lot of duplicate effort when developing both a Simulation Model and an Implementation Prototype. Second, when modifications to the protocol are needed, they have to be done in three parts at once: Specification, Simulation Model, and Implementation Prototype. The risk of inconsistencies between them being accidentally introduced is non-negligible.

We may question what is so different between the Simulation Model and the Implementation Prototype that justifies the need for duplicate code. A network protocol can generally be described as a concurrent Timed Automata [20], i.e., a state machine in which state transitions are triggered by input messages and constrained by the passage of real time. Both simulation and implementation of the same protocol include the very same automata. Only the way messages are received and transmitted and the way the passage of time is measured is different between the two. A protocol implementation (e.g., a routing agent) usually has an event loop, which is an infinite loop that waits for data to arrive on one or multiple sockets, decodes the data to extract the Protocol Data Units (PDUs), and processes the PDUs according to the protocol. As a result of the processing, new PDUs may need to be transmitted; at this point the new PDUs are encoded as data and written to one or more sockets. Time-based transitions are typically implemented using a system call to suspend the process for the required time, thus saving CPU cycles. For example, in UNIX systems it is frequent to have protocol implementations that use a select or poll based main loop, allowing them to wait for a certain elapsed time with the process suspended, but at the same time be notified when new data has arrived at a socket. In simulation environment, on the other hand, time is virtual, represented by a virtual clock, which is a simple numeric counter. In a simulator the passage of time is modeled by an event and does not require the process to be suspended during that time;

the virtual clock only needs to be incremented by the amount that simulates the real passage of time. Events are also used to represent the reception of data from a network interface.

In the simulator, the event scheduler is essentially an infinite main loop that processes pending events in order. It is similar to the select-based event loop in the protocol implementation. The differences are that: 1) in the simulator, elapsed time is virtual, in the implementation it is real; 2) in the simulator, a node simulates the reception of data from another node, while in the implementation the data is actually received from a real network interface. The protocol-specific aspects are common to the simulation and implementation, only the "environment interface" aspects are different. The question is whether we can adapt one to the other. This question is answered in Section 2.2.

2.2 Proposed Protocol Development Process

In this section we propose Network Simulator 3 (ns-3) to be used as the basis for a new protocol development process that reuses the code of the Simulation Model for the Implementation Prototype. The new protocol development process is named Fast Prototyping.

ns-3 is a network simulator popular for research and educational use. It officially started around mid 2006, and the first stable version was released in June 2008. Even though ns-2 is still being used, ns-3 has a better core architecture and it is better suited to receive community contributions. Core architecture features such as a COM-like interface aggregation and query model, automatic memory management, callback objects, and realistic packets, make for an easy to use environment in which to develop new complex simulation models. In addition, it is one of the best performing simulation tools available [21, 22].

Since version 3.2, ns-3 has received support for the "real-time simulator". The real-time simulator is an alternative event scheduler that can be selected at run-time or compile time. It synchronizes the virtual clock of the simulator with the real time of the host system where the simulation program is running. Thus, if an event is scheduled to happen in *t* seconds, then the callback function associated with the event will be called after exactly (within a small tolerance) *t* seconds of real time have elapsed. Shortly after, in version 3.3, ns-3 received support for an "emulated NetDevice", also known as EmuNetDevice, currently called EmuFdNetDevice. In ns-3, a NetDevice is a class of objects that simulate a particular link layer type, such as Ethernet, Wi-Fi, and point-to-point. The addition of EmuNetDevice provided on ns-3 the ability 1) to receive packets from a real network interface and convert the packets into simulated packets, and 2) to transmit simulated packets generated by the simulation through a real network interface. Emulation in ns-3 is not as difficult as in ns-2. In ns-2 packets are simulated as objects that do not know how to serialize themselves into a byte stream. Conversely, in ns-3 packets are represented internally as bytes, such as real packets, even in pure simulation mode, and the simulator uses Header classes for conversion between PDU format and byte format. For this reason, emulation in ns-3 works with any protocol, not just a selected few protocols prepared to support emulation, as in ns-2. These are the main ingredients for enabling real protocol implementations to emerge from an ns-3 simulation model of that same protocol. We only need to build a simulation program, with one node only, and a number of EmuNetDevice instances equal to the number of real network interfaces being used. The protocol simulation model runs on that single node, but remains unaware that it is running in emulation mode. Thanks to the ns-3 architecture and features, very little code is needed on top of the protocol simulation model to deploy it as an Implementation Prototype to run in a testbed.

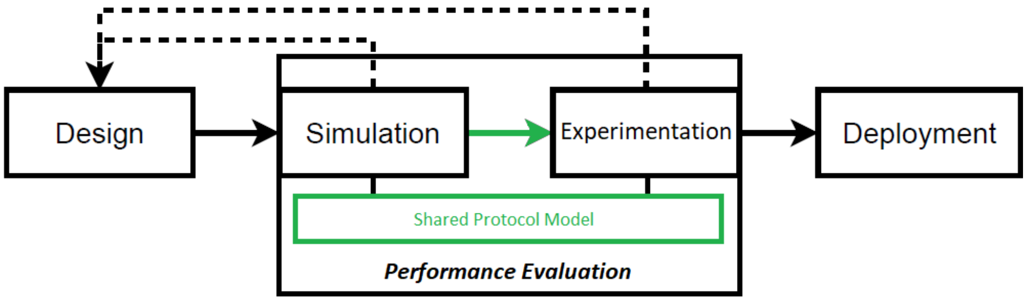


Figure 2.2: Proposed shared ns-3 protocol model development process, named Fast Prototyping, resulting in a Shared Protocol Model between Simulation and Experimentation.

If the Simulation Model and Implementation Prototype of a protocol follow this approach, wherein most of the code is exactly the same for both, the Fast Prototyping protocol development process illustrated in Figure 2.2 is defined. The phases composing the Fast Prototyping development process are the same, although the code is shared between Simulation and Experimentation (real-world testbed). The advantages of the shared protocol model for both evaluation phases are clear. First, development time is saved, since only a small (and very generic) wrapper needs to be written to run the ns-3 protocol model as an Implementation Prototype for the testbed. Second, when the protocol is used in a real-world testbed and needs to be adjusted, the modifications are made in a single software module; this is simple and does not have the risk of inconsistencies between the Simulation Model and the Implementation Prototype.

2.3 Related Work

The idea of using a shared protocol model as the Implementation Prototype is not new. However, there are issues with the previous attempts. In this section we analyze the state-of-the-art solutions using a shared protocol model approach.

The popular **ns-2** simulator [23] does have some support for emulation [24], but only some protocols are prepared for this emulation mode. The packets are simulated as objects and do not know how to serialize themselves into a byte stream. Moreover, ns-2 does not use real IP or MAC addresses, making the emulation more complicated. We need to maintain an addressing scheme so that ns-2 can determine the source and destination nodes from the source and destination address of the real packets. Finally, ns-2 does not have a software architecture as clean as other existing simulators, and does not provide an easy environment to develop new protocols to be used for emulation, for the reasons previously stated.

RapidNet is a toolkit that enables the development of network protocols for simulation and experimentation [25]. The protocols are specified in a declarative paradigm using Network Datalog (NDlog), which extends the recursive query language of Datalog. RapidNet can compile the protocol specification into ns-3 code for either simulation or emulation mode, thereby achieving the goal of using the same protocol model for simulation and experimentation. One of the problems of RapidNet is that it uses a declarative programming paradigm, which is unusual in protocol development and results in a steep learning curve; also, the generated ns-3 code depends on the RapidNet library and it cannot be submitted for inclusion into ns-3 as a new module.

The **Protean Protocol Prototyping Library (Protolib)** is a toolkit that provides a cross-platform C++ Application Programming Interface (API) for the development of network protocols. The developed protocol can run on various operating systems (including Linux, MacOS, Windows, and FreeBSD) and on the ns-2 and OPNET simulators [26]. The main abstractions needed to develop new network protocols are provided by the following Protolib's classes: ProtoSocket, which abstracts the underlying systems TCP and UDP sockets; ProtoTimer, a generic timer class; ProtoRouteMgr, which provides a common interface to the systems routing tables; and ProtoCap, used for raw MAC-layer packet capture. While Protolib appears to be an alternative architecture (abstraction layer) to accomplish similar goals, features found in ns-3, such as logging and trace sources, would not be available in the same way on Protolib-based implementations.

The **Click Modular Router (Click)** is a flexible software architecture for developing configurable routers [27]. A router can be created by combining small elements in a graph-like architecture. Each element has a defined number of input and output ports and implements some simple functionality such as decrementing a packet TTL and looking up an IP route. The combination of these elements can create routers with complex functionality providing a flexible platform for researchers to experiment with new protocols. Click can run in several systems such as Linux and FreeBSD. There are also tools that integrate it with ns-2 (nsclick [28]) and ns-3 (ns-3-click [29]). The ns-3 simulation can even become more efficient in some cases by using Click to perform the layer 3 functionality, although at the cost of increased memory consumption. Click also has a steep learning curve since it uses a flow-based programming paradigm [30], which is different from the more usual discrete event simulation used in ns-3. In addition, researchers have to learn the configuration syntax that is used to describe the connections between the elements of a router.

The **OPNET Modeler** is a simulator that also allows integration of protocol models with real systems, thanks to its "System-in-the-loop" functionality [31]. However, its closed and commercial nature invalidates it being a viable alternative in many contexts. For instance, not having the full source code available precludes porting it to certain router architectures.

The **ns-3 Direct Code Execution (DCE)** module [32] provides an environment that is able to execute network protocols developed for Linux in ns-3 without source code changes [33]. Both user space and kernel space protocols as well as applications can be used, for instance, the ping tool and the real Linux TCP/IP stack. There are a number of problems when using the DCE module as a way to achieve a shared protocol model implementation. First, there is simulation performance degradation resulting from having to virtualize the protocol under evaluation [33].

Second, it is more time consuming to develop protocols in Linux than in ns-3. Third, not all system calls used by native Linux applications are supported by DCE, which implies the need for the application to be open source (to make the necessary changes to the code, e.g., *iperf* modified for DCE [34]) and the maintenance a second version to use in DCE, which can be error prone and produce non-comparable results between both implementations. Finally, not all of the functionality provided by ns-3 can be used, since there is no Linux counterpart. In some cases an alternative is available; for instance, it is possible to add tracing support to a real protocol by using aspect based programming [35], but this is not as simple as using the ns-3 tracing/log facilities.

There are some other tools that work with some form or subset of an operating system kernel code. The **ENTRAPID** [36] project virtualizes just the networking portion of a BSD kernel, thereby enabling hundreds of kernel instances to run on the same system, connected by virtual links. A similar approach is followed by **ALPINE** [37]. The **Network Simulation Cradle (NSC)** [38] also virtualizes a kernel, but instead of allowing real applications to communicate over the network of virtualized kernels, it embeds the virtualized kernel instances into simulated nodes of an existing simulator (ns-2 and ns-3 are supported), and supports multiple kernel stacks, not just BSD; however, it was hard to maintain because it relied on source code modifications. The ns-3 DCE module can replace most of the functionality provided by NSC [39]. **IMUNES** [40] also virtualizes the kernel networking code, but creates virtual nodes and virtual links inside the kernel, instead of user space, to avoid frequent context switching and achieve greater efficiency. All these kernel based approaches have the same basic problem: they require new networking protocols to be developed inside the source code and framework of one of those operating system kernels. That code base, although highly detailed, realistic, and well optimized, is not a very programmer-friendly environment. Moreover, protocols (at least control plane protocols) are almost always developed to run as user space daemons anyway, for security reasons, so the effort to develop using a kernel space API may not be worth it.

Simulator-agnostic ns–3 applications are applications that can run both in simulation and in real systems [41]. They are developed as traditional ns-3 applications but they can communicate with the real environment via the socket API provided by the underlying OS. The actual socket that is used is transparent to the application, so its code does not need to be rewritten. This can be done by implementing the real socket functionality as classes derived from the base Socket class of ns-3 and using the factory pattern to create the actual socket as needed (real or simulated). This approach is very efficient [12] and works well for applications. Yet, for example, it does not work for a routing protocol. A routing protocol needs to access and update routing tables. Also, many protocols do not run on top of TCP or UDP, but instead on top of IP datagrams (e.g., OSPF [42]); others may even work on top of layer 2 protocols (e.g., WMRP [43]). Simulator-agnostic ns-3 applications allow, for example, the control plane of a Layer 3 protocol developed in ns-3 (operating at a specific TCP or UDP port) to communicate using real system sockets. Still, on their own they can not offer a complete shared protocol implementation solution, as they lack the data forwarding capabilities needed for Fast Prototyping.

2.4 ns-3 Emulation

In ns-3, a NetDevice is the class of objects responsible for simulating a layer 2 network interface. In Linux systems, the EmuNetDevice subclass is available, allowing a ns-3 simulation to receive real packets from a real network interface, and to send simulated packets through the same network interface.

At the core of EmuNetDevice is a “packet socket” (PF PACKET, SOCK RAW) socket file descriptor. When the simulator asks the EmuNetDevice to send a packet (ns3 class *Packet*), the method *Packet::CopyData* is called, which extracts the packet contents into a byte buffer. Then, the *sendto()* system call is performed, using the packet socket file descriptor and the packet byte buffer as parameters. The code to receive packets is slightly more complicated. In fact, because we cannot block the main simulation event loop, nor it is efficient to make a poll/select system call between each simulation event iteration, a separate thread is created specifically to receive data from the packet socket. This thread runs an infinite loop that 1) allocates a memory buffer, 2) calls *recvfrom()* to receive the next packet, 3) schedules an EmuNetDevice method to be called from the main simulator thread, passing a pointer to the memory buffer as parameter. The method that is called in the main thread, by request from the receive thread, simply converts the raw memory buffer into an ns-3 *Packet* (using an appropriate *Packet* constructor), releases the memory buffer, and informs the simulated node that it has received a new packet.

The conversion between *ns3::Packet* and raw memory byte array, and back, is trivial in ns-3 because ns-3 *Packets* always store simulated packets in a raw memory format. The only exception being that ns-3 *Packets* support a memory optimization wherein an application can choose to send “dummy” bytes, i.e., a block of bytes whose value is not important for the simulation. In this case, ns-3 avoids allocating memory for those bytes and just records the size and offset of that block of dummy bytes. Nonetheless, the conversion is much simpler and natural than in almost any other simulator.

With this emulation method, we can implement virtually any kind of network operation in ns-3, as long as it works above layer 2. However, a question remains: how does this method behave performance-wise?

2.5 Performance Evaluation of ns-3 Emulation

In order to better assess the computational and network performance footprint induced by using the ns-3 emulation method in a real-world scenario, a set of tests were defined. These tests were focused on aspects such as the impact on the host machine resources utilization and on network performance impact, depending on the offered data rate and packet size used.

The hardware used to run the tests was a mini-itx Intel Atom D510. It had an x86 architecture for ease of use and better compatibility, avoiding cross compiling. The hardware was chosen having in mind a balance between cost, performance, power consumption, and physical size, so that it could be easily deployed in a real-world testbed as a network element performing operations

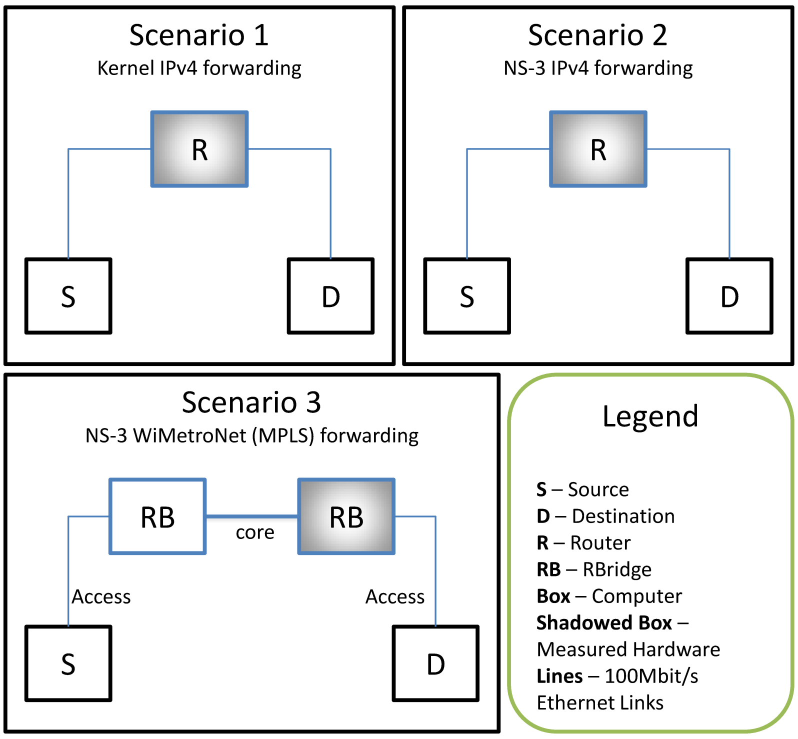


Figure 2.3: Data plane forwarding test scenarios.

such as routing and bridging. The Operating System (OS) used was Ubuntu 10.04 x86. The three scenarios implemented to perform the tests are presented in Figure 2.3. While scenarios 1 and 2 were designed to compare IPv4 packet forwarding performance between kernel and ns-3 emulated implementation, scenario 3 was used to derive the performance of a custom implemented routing protocol, with its own data plane encapsulation operations.

Scenario 1 is composed by three nodes: 1) S runs an *iperf* client, generating an UDP flow to D. S also runs the *ping* application, measuring the RTT while the flow is generated; 2) D runs an *iperf* server, receiving the UDP flow from S and calculating network statistics such as received data rate and packet loss ratio; 3) R represents the router, default gateway for S and D. It is responsible to forward IPv4 packets between the two IP nodes. The IP forwarding operations are performed in kernel space by enabling the IP forwarding option of the Linux kernel. Scenario 2 is similar to Scenario 1. The difference resides on node R. Now, the IP forwarding operations are performed by an ns-3 emulated node, in user space, connected through EmuNetDevices to the real network interfaces. Scenario 3 employs WiMetroNet [43] RBridges elements, instead of an IP stack, which use the standard IPv4/Ethernet stack on the access networks and MPLS encapsulation in the core network. Two RBridge elements are used in order to introduce the need to perform “ingress” and “egress” packet operations, as it happens in a real world scenario.

For each scenario, we ran a set of tests, gradually increasing the generated data rate between 1 and 90 Mbit/s[1], and with two different UDP payload sizes: 160 and 1400 bytes. Packets with

[1]Approximately the maximum throughput attainable using IPv4 UDP packets with payload size of 1400 bytes on 100 Mbit/s Ethernet links

1400 bytes represent the usual application traffic over TCP. While the Ethernet MTU is 1500 bytes, given the UDP/IP payload plus the encapsulation overhead in Scenario 3, a UDP payload size larger than 1400 bytes would risk fragmentation. The 160 bytes packets are representative of typical VoIP traffic. Each test, lasting 30 seconds, was repeated 5 times for confidence interval purposes.

For each test, the received data rate, average round-trip time, packet loss ratio, and CPU load were measured. The received data rate and packet loss ratio were extracted from the *iperf* output statistical data. The average round-trip time was measured with the *ping* utility. The CPU load was measured at the shaded nodes in Figure 2.3, using the "time" built-in command, computed as the sum of "user" and "system" time divided by the real elapsed time. The CPU load can assume values in the range 0–2 because the node has a dual-core processor and ns-3 has a multi-threaded design for implementing Emulated Network Devices. For Scenario 1, the CPU load was not measured because it revealed to be negligible, even when performing full link speed forwarding with small packets.

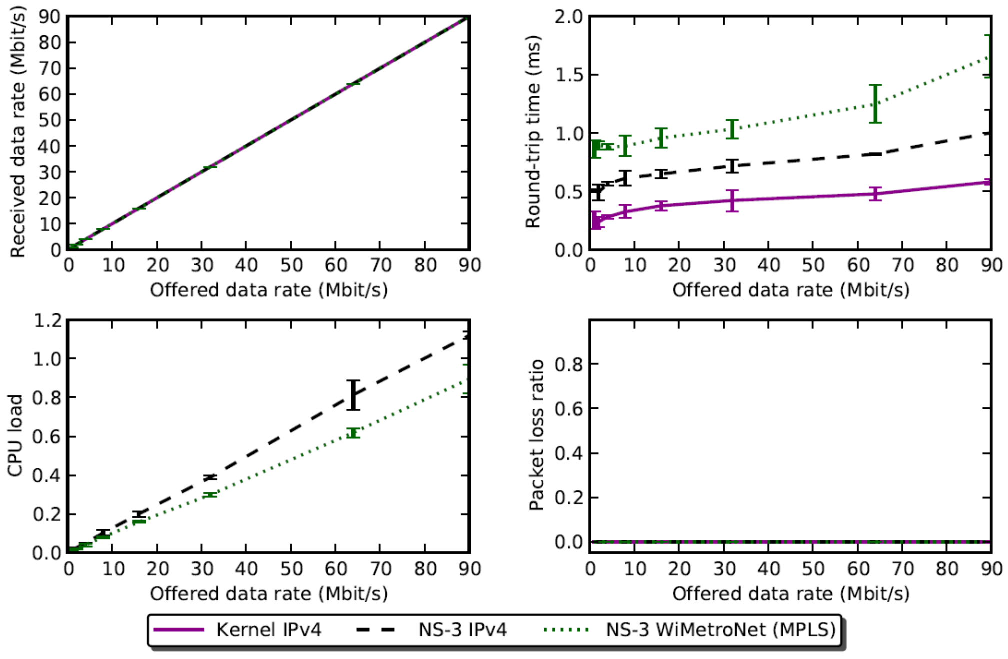


Figure 2.4: Data plane forwarding results for 1400-byte packets.

The obtained results are represented in Figure 2.4 and Figure 2.5 for packets of 1400 and 160 bytes, respectively. For 1400-byte packets, both scenarios were able to attain a data rate of 90 Mbit/s without any packet loss. While in kernel space the CPU load is insignificant, at user space the CPU load increases linearly with the offered data rate, reaching approximately 1.1. The RTT measured in the scenarios running ns-3 increased roughly 0.4 ms, due to the user space processing of each packet. For a 1400-byte packets, despite the high CPU load, the obtained results can be considered very good, since the additional delay introduced has remained below 1 ms.

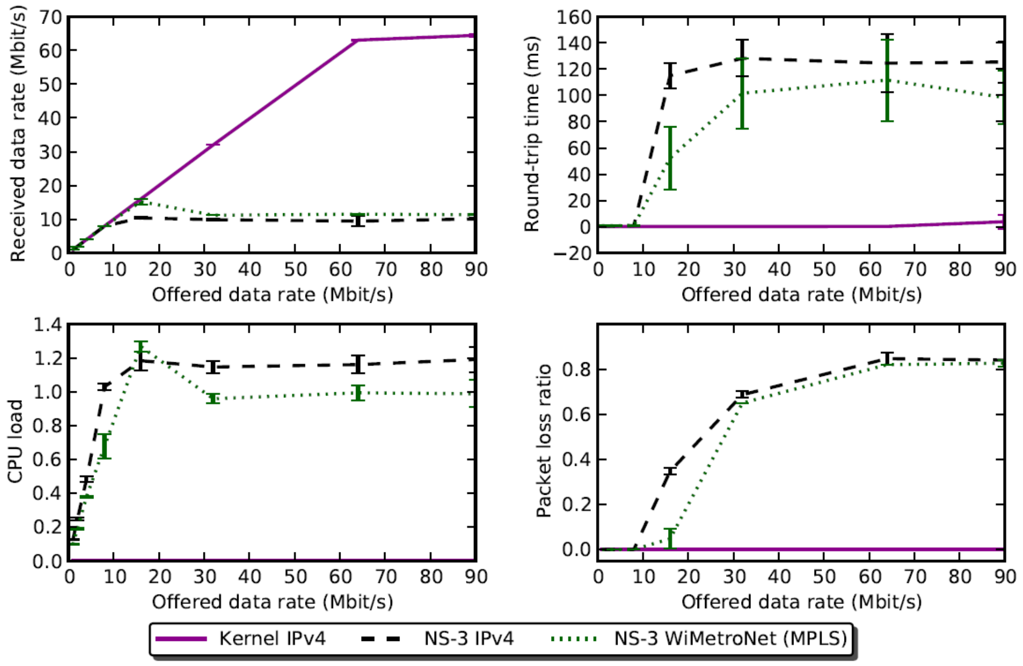


Figure 2.5: Data plane forwarding results for 160-byte packets.

For an MTU of 160 bytes, the plots of Figure 2.5 show clearly a very high performance difference between Kernel and ns-3 IP forwarding solutions. When IP forwarding is done in kernel space it is possible to reach a data-rate of approximately 60 Mbit/s, which is the maximum throughput of 100 Mbit/s Ethernet links for an MTU of 160 bytes. The RTT remains very low and stable, and the CPU load is also negligible. Although the offered data rate was configured up to 90 Mbit/s in iperf, it did not have impact on the packet loss results in this scenario, because iperf client is aware of the link capacity and does not attempt to transmit more than the link capacity.

In contrast, Scenario 2 reaches only a data rate of around 10 Mbit/s, which is approximately six times lower than the kernel implementation. This happens due to the high number of packets that has to be processed in user space. It now becomes evident that in these kind of emulated scenarios the bottleneck is defined by the number of packets to be processed, and not so much by their size. The RTT value stabilizes at around 120 ms, while the packet loss ratio steadily increases because of a limit in ns-3 for the number of packets waiting in memory to be processed. While, at a first glance, these results could seem unsatisfactory, if we consider that these 10 Mbit/s are representative of VoIP traffic, which consume very small bandwidth (less than 64 kbit/s) per-flow, it actually represents a very large number of VoIP flows.

Finally, the results of the third scenario are better than the results obtained for the second scenario, attaining lower CPU usage per packet processed, which resulted in better measured data rates and RTTs. For the packet plots of Figure 2.4, the average RTT of Scenario 3 appears to be worse than the Scenario 2, but it is necessary to point out that each packet was then subjected to two forwarding operations in each direction, instead of only one. The better results obtained for this scenario could be mainly due to the fact that RBridges forwarding operations take place at layer 2.5, not using the ns-3 layer 3 implementation and its associated computational footprint.

It is clear from the previous results that ns-3 has a considerable per-packet performance penalty. The results confirm the hypothesis that the kernel to user space context switch and data transfer are the main bottleneck. Most of the performance issues associated to ns-3 are related to the number of packets to handle, and not so much to the size of the packets. Thus, the proposed shared protocol model can handle reasonably well application flows dominated by large packets, such as any TCP based protocol (HTTP, FTP), but it does not cope with high bitrates and small packets. Fortunately, the combination of high bitrate and small packet size is not very common. It is true that VoIP (Voice over IP) flows are composed mainly of small packets, but each of those flows consumes a small bitrate. Only a high number of simultaneous VoIP flows will be able to saturate a network link with a few tens of Mbit/s. For this type of scenario, implementing a data plane using ns-3 is not recommended.

2.6 Summary

In this chapter we addressed the topic of improving the development process of network protocols. The traditional protocol development process was reviewed, and the main problems associated with it were identified. One recurring problem is the duplication of effort to write the simulation model and, then, implementation prototype code. Another problem is the possible behavior differences that may be accidentally introduced between the two versions, leading to different results in Simulation and Experimentation. We propose an alternative protocol development process, named Fast Prototyping, that takes advantage of the built-in network emulation capability of ns-3. The Fast Prototyping development process allows developers to write a single model for the protocol that can be both simulated and deployed in a real node. The main difference between ns-3 and other similar frameworks is that in ns-3 everything can be done in C++ and using C++ programming best practices for ease of development.

In order to support the proposed Fast Prototyping process, the performance of ns-3 running in emulation mode has been evaluated. The results show that the ns-3 IPv4 stack, in emulation mode, is able to process packets at a rate high enough to exhaust an 100 Mbit/s Ethernet link, when handling large packets, but can have problems forwarding traffic if it is composed mostly of small packets, or if the network interface data rates increase. The Fast Prototyping development process is the subject of further work in this PhD: in Chapter 3 we introduce the support for more real-world network interfaces and configurations, while in Chapter 4 new forms of reducing the performance footprint of ns-3 Emulation mode are proposed.

Chapter 3

Improving ns-3 Emulation Support in Real-World Networking Scenarios

Chapter 2 introduced the Fast Prototyping protocol development process, integrated in the concept of improving the performance evaluation of wireless networks, which defines the use of a shared ns-3 protocol model implementation for Simulation and Experimentation. Although this functionality is partially supported by using ns-3 emulation, there are still limitations regarding the support of real network interfaces and the easy configuration of the network settings, such as IP and MAC addresses.

In this chapter we propose an improved version of the ns-3 emulation component by introducing new functionalities that address these limitations. The new functionalities include the support of new types of real network interfaces and the easier integration of emulation nodes with existing networks by means of a new auto-configuration mechanism for ns-3 nodes. Finally, we evaluate the proposed new functionalities through experimental results obtained in a laboratory testbed and in a real vehicular network testbed, and demonstrate their proper operation and backwards compatibility with previously coded ns-3 scenarios.

3.1 Overview of ns-3 Communication Types

ns-3 is an event-driven packet level network simulator, which is largely adopted by the scientific community to evaluate networking solutions in simulation environment. Being a packet level simulator, ns-3 allows to produce fully detailed simulation environments that accurately represent the real network behavior; for instance, each network packet exchanged in the simulator uses the same exact structure of a real packet. This realism enabled the development of the ns-3 emulation functionality that is essentially the ability to run a simulation scenario in real time with the capability of exchanging network traffic between real and ns-3 nodes. From the real nodes' perspective, the ns-3 emulated network appears as an extension of the real network, with transparent exchange of network traffic between them.

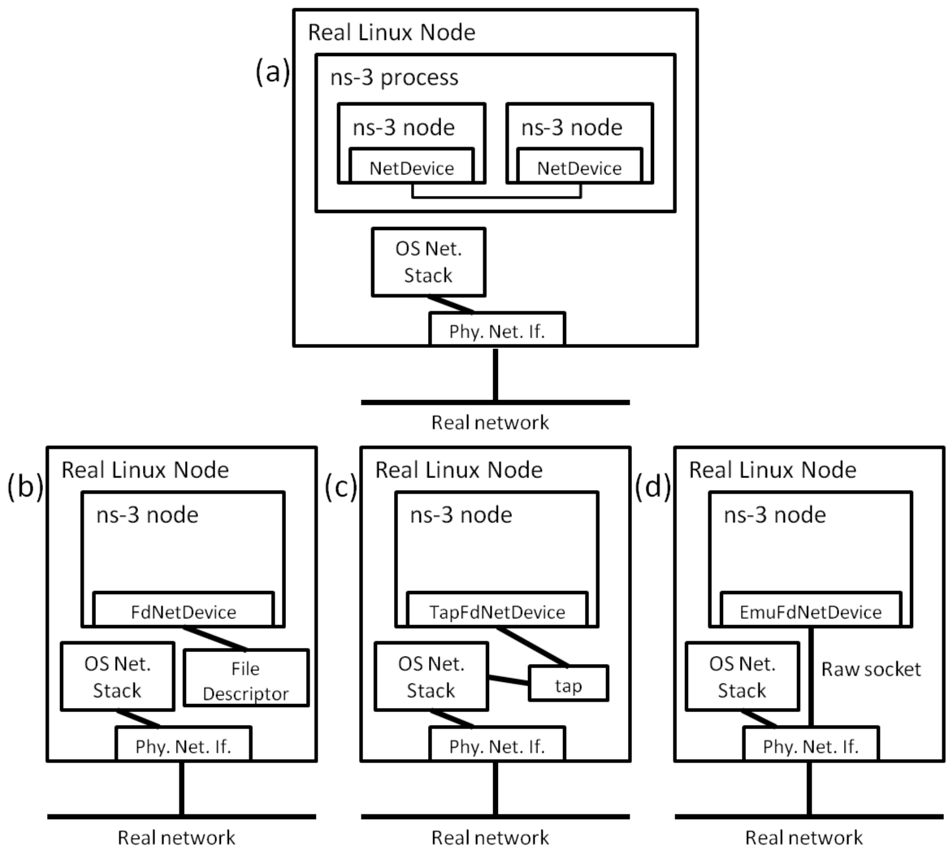


Figure 3.1: Overview of the communications provided by the ns-3 (a)NetDevice, (b)FdNetDevice, (c)TapFdNetDevice and (d)EmuFdNetDevice.

In the real world, network nodes have network interface cards, allowing them to connect to a network and exchange network traffic. Likewise, in ns-3, simulated nodes are connected to a network using NetDevices. Figure 3.1 presents an overview of the two communication types possible when using ns-3: 1) internal – between ns-3 nodes (Figure 3.1a); 2) external – between an ns-3 node and the outside world, either the real Linux node hosting the ns-3 node or a real network (Figures 3.1b–d).

In Figure 3.1a two ns-3 nodes exchange network traffic over a virtual channel using standard NetDevices, hence the nodes can not communicate with the outside of the ns-3 process. In Figure 3.1b an ns-3 node uses a specific NetDevice – the FdNetDevice –, which allows exchanging network traffic with the real Linux node using a file descriptor managed by the Operating System (OS) of the real node. In Figure 3.1c an ns-3 node is using a specialization of the FdNetDevice – the TapFdNetDevice – designed to exchange network traffic with the real Linux node using a tap interface. Every write from the ns-3 node via the TapFdNetDevice appears to the real Linux node as incoming network traffic via the tap interface, and vice versa. Finally, in Figure 3.1d an ns-3 node is using another specialization of the FdNetDevice – the EmuFdNetDevice – designed to allow direct communication to outside of the real Linux node using a raw socket bound to a real network interface (e.g., eth0). This allows exchanging the ns-3 node's traffic with that specific real

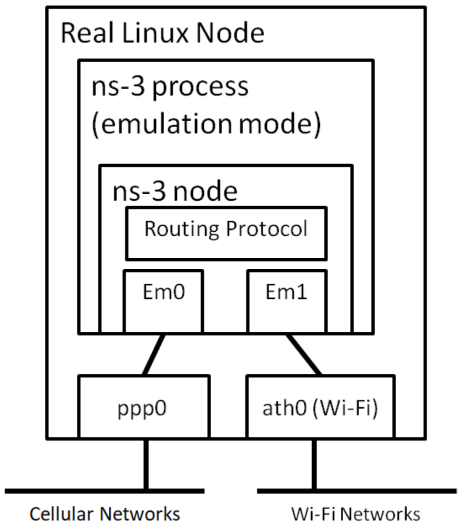


Figure 3.2: Emulation in a vehicular network scenario, considering possible Wi-Fi and Cellular network links.

network interface, thus enabling the emulation functionality.

The Fast Prototyping process proposed in Chapter 2 uses the EmuFdNetDevice module, taking advantage of the built-in network emulation features of ns-3. In Figure 3.2 we have illustrated an example of a real Linux node running a vehicular mobile router prototype developed using the Fast Prototyping process, where a routing protocol implemented in ns-3 is reused. The EmuFdNetDevices Em0 and Em1 shown in Figure 3.2 provide direct communications to the real networks as if the ns-3 node was a real node. From the real networks' perspective, the ns-3 nodes are real nodes running a real network protocol instance.

3.2 Problem and Motivation

When the Fast Prototyping process is employed in a controlled, static testbed, the experimental scenario is usually characterized by Ethernet or Wi-Fi real networks that are administered by the experimenters themselves. The experimenters can then deploy emulated ns-3 nodes accessing those real networks, with emulated network devices, using configurations (e.g., MAC and IP addresses) that are pre-defined for the experiment and remain constant and controllable during the whole experiment timeframe. Yet, when the Fast Prototyping process is used in a more complex and dynamic scenario – e.g., a vehicular network – different network access and usage characteristics take place.

Figure 3.2 depicts the use of ns-3 emulation in a vehicular network scenario, where two interfaces are available to access real networks as the vehicle moves along a given path. The ppp0 interface represents the cellular connectivity, and enables IP over the Cellular Network. This interface is only present in the real Linux node when there is an active cellular connection. In practice, it is common to have this interface intermittently available, due to possible intermittent cellular connectivity and the dynamic IP renewal policy that may be imposed by the telecom operator. The ath0 interface represents the Wi-Fi connectivity to multiple Wi-Fi networks available along the vehicle path, to which the real node connects opportunistically. Each of these networks may have its own administrator and assign specific dynamic IP level settings. Also, as an access control policy, network administrators may use MAC addresses to identify the users and provide IP level configurations.

The ns-3 emulation mode and the related emulated network devices, represented by Em0 and Em1 in Figure 3.2, were tested in the SITMe's [13] multi-technology vehicular network scenario, where the real network interfaces had characteristics similar to those previously mentioned. These characteristics precluded the use of Fast Prototyping in the vehicular network scenario due to the existing ns-3 limitations. A detailed description of these limitations is introduced below.

3.2.1 Cellular PPP Interfaces Support

Point-to-Point Protocol (PPP) [44] interfaces enabling IP over cellular networks are unsupported by ns-3. The EmuFdNetDevice module is designed to read and write Ethernet frames from/to real interfaces, not IP packets as it is the case for the cellular PPP interfaces. As such, the EmuFd-NetDevice must be modified and capable of detecting whether the real interface is operating at L2 or L3 and adapt itself accordingly. Also, when working at IP level, the EmuFdNetDevice does not need Ethernet Address Resolution Protocol (ARP) [45] support. This aspect needs to be addressed as well; otherwise, the installation of the Internet Stack in the node associated to that EmuFdNetDevice will fail due to the asserts that are made when installing the ARP protocol.

3.2.2 Cellular PPP Interfaces Intermittency

Cellular related PPP interfaces are only available in Linux when there is an active cellular connection established. Throughout the experiment duration, the cellular connection can be lost due to 1) the lack of network coverage in some geographic areas, 2) forced re-connection by the operator to allow dynamic IP renewal, and 3) any sort of other communications problems preventing communications between the PPP Client and Server. This leads to the PPP session shutdown, and the interface ppp0 disappears. The EmuFdNetDevice module does not support this intermittency and has two possible undesirable behaviors: 1) if the real interface is not available when the emulation starts, ns-3 aborts the execution; 2) if the real interface is available when the emulation starts but it disappears, the ns-3 process detects that the raw socket was closed, stops the EmuFdNetDevice, but does nothing to restart it.

3.2.3 Manual MAC Address Configuration

In a simulation scenario there is full control of the elements interacting in the simulation. Typically, ns-3 self-generated MAC addresses are used and assigned sequentially to every simulation node to avoid MAC address collisions. However, when Fast Prototyping is used, we may not fully control the scenario and the interactions with other nodes. So, in order to avoid MAC address collisions and network access control problems, the MAC address of the real interface has to be used by the emulated node. ns-3 does not include any MAC cloning functionality, which results in the need for error-prone, additional manual configurations.

3.2.4 Dynamic IP Configuration Settings

In IP networks, the interface auto-configuration provided by the Dynamic Host Configuration Protocol (DHCP) [46] is frequently used. This auto-configuration mechanism allows a node to establish IP connectivity with other nodes using the given network settings, such as the IP address, network mask, and default gateway, leased by the DHCP server. In a vehicular network environment it is usual for a node to connect to different networks with the same interface, or use a mobile network that imposes IP address renewal from time to time. In the current version of the ns-3 EmuFdNetDevice, there is no mechanism to keep the network settings updated in the emulated node whenever the real network interface settings change. This will make the emulated node use wrong network settings and lose connection with other nodes.

3.3 Proposed EmuFdNetDevice

Motivated by the problems described in Section 3.2, we expanded the functionality of ns-3 by proposing an improved backwards compatible EmuFdNetDevice module. The improved EmuFdNetDevice supports new features to further improve the capability of running emulated and hybrid environments – emulated and real nodes interacting – over different real network scenarios. The solutions proposed to address the problems identified in Section 3.2 are detailed in what follows.

3.3.1 Detection of the Operating Layer of Real Network Interfaces

The current EmuFdNetDevice is hardcoded to read and write Ethernet frames; as such, it only supports real network interfaces operating at MAC level. Conversely, the improved EmuFdNetDevice inspects the underlying real interface, checks whether it has a MAC address assigned, and classifies the real interface as an IP or MAC level real interface accordingly. This information is saved in the new flag named *m_isL2NetDevice*, associated with the FdNetDevice, which is setup during the Helper execution. When the real network interface is operating at IP level, the improved EmuFdNetDevice disables the “NeedArp” setting, in order to avoid assertion errors when installing the Internet Stack in the node. The automatic detection of the network stack, associated to the underlying real interface, effectively avoids the need for additional configurations and enables backwards compatibility with previously coded scenarios.

3.3.2 Support for Intermittent Real Interfaces

The EmuFdNetDevice was designed assuming that the real interface to which it is bound has an identifier always present in the Linux’s interfaces list. As explained in Section 3.2, this is not

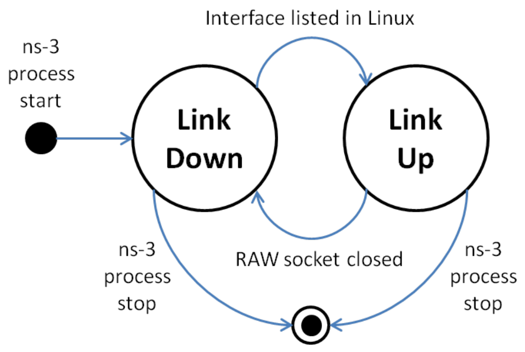


Figure 3.3: State machine representing how the improved EmuFdNetDevice handles communication using intermittent real interfaces.

always true – e.g., the PPP interfaces representing cellular connections. The current EmuFdNetDevice has the two possible undesirable behaviors referred in Section 3.2.2. The improved EmuFdNetDevice expects and handles gracefully this behavior, according to the state machine presented in Figure 3.3. The improved EmuFdNetDevice allows the user to configure the support of intermittent behavior. When the related flag – named *m_isIntermitentInterface* – is set, the improved EmuFdNetDevice assumes the link is down, allowing the emulation process to be executed even if the real interface is not listed. In the ''Link Down" state, the simulator checks periodically whether the real interface ID becomes listed again in the real Linux node. As soon as the interface ID becomes listed, a new raw socket is created and the communication is restarted, assuming the "Link Up" state. When the raw socket is unexpectedly closed, the ns-3 process sees the socket returning "-1". Instead of just stopping the device, the improved EmuFdNetDevice assumes again the "Link Down" state and actively tries to re-establish communication by creating and binding a new raw socket to the new real interface. The ns-3 process can be stopped from either state.

The improved EmuFdNetDevice enables the support for intermittent real interfaces while keeping full compatibility with previously coded scenarios. The user creating the emulation scenario only needs to be sure about the identifier of the real network interface. In the current EmuFdNetDevice, the process is terminated and the user is informed about the non-existing/erroneous interface identifier. With the improved EmuFdNetDevice, if the identifier supplied by the user is wrong and the user configures the interface as being intermittent, the emulation process will run without ever binding to a real interface nor triggering the error.

3.3.3 MAC Address Cloning

Simulated nodes, running inside ns-3, use self-generated MAC addresses by default – e.g., 00:00:00:00:00:01. This is not an issue when running simulations, but can be a problem when using the emulation capability in a real network scenario – e.g., the case of Fast Prototyping of network protocols. In such case there will be a number of ns-3 instances running independently from each other, with emulated ns-3 nodes acting as real nodes. Being different ns-3 instances, means that the ns-3 mechanism for self-generating MAC addresses will assign, by default, coincident MAC addresses (starting in 00:00:00:00:00:01) to the ns-3 nodes running in the different instances. Manually managing the MAC addresses of each emulated node accessing a real network to avoid MAC collisions can be difficult and error-prone. Also, often the MAC address used by the EmuFdNetDevice has to match the MAC address of the real interface of the real node, in order to allow communication in the real network. Using the real interfaces' MAC addresses could then solve the MAC addresses collision problem and ensure compatibility with real networks requiring the use of the MAC address of the real interface. Because ns-3 lacks a MAC cloning feature, a configuration option was added to the EmuFdNetDevice instances, in order to allow automatic cloning in run time of the MAC address of the real interface to which the specific EmuFdNetDevice is bound. This feature is disabled by default, so that the EmuFdNetDevice operation is unchanged when running previously coded scenarios.

3.3.4 IP Address Cloning

In a real network scenario, IP level network configuration is often carried out using the DHCP protocol. This is common when connecting to real networks in a vehicular scenario, for example, where multiple networks can be used and each network has a different IP configuration. Also, usually Internet Service Providers (ISPs) do not provide fixed public IP addresses; as an example, every time a PPP connection is established over a public 3G operator link, a new dynamic IP address is leased. Because the IP address changes every time a new connection is established, even during the same emulation process execution, the emulation scenario rapidly becomes outdated, with an IP address associated to the EmuFdNetDevice that is no longer valid nor corresponds to the real PPP interface anymore. When the emulated node tries to write packets with an outdated IP, they can be discarded or lead to an IP address collision. In order to avoid this problem, we added an IP address cloning feature to the EmuFdNetDevice, which is enabled by configuration in every EmuFdNetDevice instance. When enabled, ns-3 periodically verifies the IP address, network mask, and default gateway of the real interface and applies those settings to the node running the emulated interface. This simplifies the deployment and auto-configuration of emulated nodes in a real environment. IP address cloning is disabled by default, assuring the EmuFdNetDevice's behavior expected by previously coded scenarios.

3.4 Solution Validation

This section describes the tests performed to validate the proper operation of the improved EmuFdNetDevice. Two scenarios were used to perform the tests: 1) a laboratory testbed, used to test the new features and assist the debug process in a controlled environment; 2) a real world testbed using the SITMe's project vehicular network.

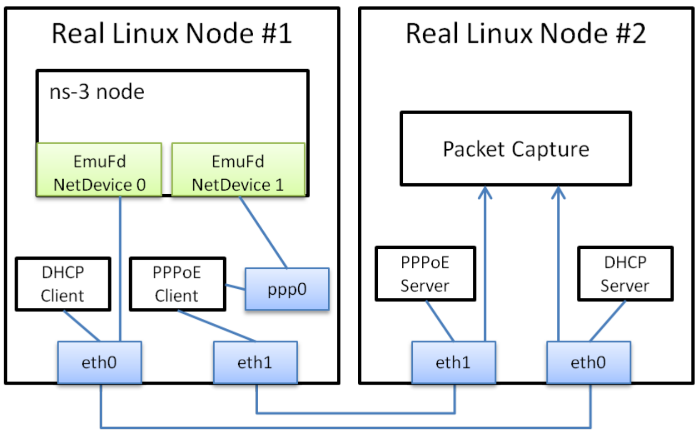


Figure 3.4: Laboratory testbed scenario.

3.4.1 Laboratory Testbed

The first approach to validate the improved EmuFdNetDevice was to conduct the experiments in a small laboratory testbed, in order to easily test the proper operation in a fully controlled scenario. Figure 3.4 presents the components that characterize the laboratory testbed, which was composed by two Real Linux x86 nodes:

Real Node #1 is a multi-interface real network node hosting an ns-3 emulated node. The real node has two physical Ethernet interfaces: 1) eth0, auto-configured at IP level using the DHCP client; 2) eth1, to establish a PPPoE connection from the PPPoE Client to the PPPoE Server. When the PPPoE connection is established, the ppp0 interface is listed in Linux. eth0 represents the use of real network interfaces operating at MAC level and ppp0 represents a real network interface working at IP level. The ns-3 node has two emulated network devices, one bound to the real node's eth0 and the other to the intermittent ppp0.

Real Node #2 is a communication peer used to interact with the emulated node, but it also implements the access control and auto-configuration mechanisms present in most real networks, such as DHCP server and PPPoE server. Real Node #2 has two Ethernet interfaces connected directly to the equivalent interfaces in Real Node #1. Real Node #2 also captures the network traffic from the two Ethernet interfaces, in order to assess the correct operation of the emulated node when communications between Real Node #2 and ns-3 emulated node are attempted.

In order to reproduce the conditions needed to test each functionality of the improved EmuFd-NetDevice, the following procedures were considered:

PPP intermittent behavior. The PPPoE Server was periodically stopped and restarted in order to close and reestablish the PPPoE connection and make the ppp0 interface disappear and reappear in Real Node #1.

Dynamic IP configuration on eth0. Short duration IP leases were used, in order to generate frequent lease renewals. By using manual MAC-IP address associations in the DHCP Server that were periodically changed to different IPs, led to periodic IP address changes in the eth0 interface of Real Node #1.

Dynamic IP on ppp0. Every time the PPPoE Server was stopped, the configuration file was changed to assign a different IP addresses range to the PPPoE client, so that the ppp0 IP address was changed every time the PPPoE connection was re-established.

In order to generate network traffic between Real Node #2 and the ns-3 node running in Real Node #1, two mechanisms were used: 1) ICMP echo requests/replies between the two nodes; 2) an UDP echo server running in Real Node #2, which replied to UDP packets sent by the ns-3 node with exactly the same UDP payload. The traffic generated was captured using Wireshark [47], which was running in Real Node #2.

To validate each new feature of the improved EmuFdNetDevice, the following observations were made:

Detection of the operating layer of real interfaces. The use of two EmuFdNetDevices bound to eth0 and ppp0 interfaces confirmed the correct operation of this feature. In the case of the eth0 interface, the lower level transfer unit used by EmuFdNetDevice0 was the Ethernet frame, while with the ppp0 interface, EmuFdNetDevice1 exchanged IP packets.

Support for intermittent real interfaces. The use of the PPPoE protocol led to the creation of the ppp0 interface in Real Node #1. This interface was only listed intermittently according to the test conditions referred above, which allowed the test of the EmuFdNetDevice ability to recover from the following conditions: 1) ns-3 process started without the ppp0 interface available; 2) ppp0 interface unexpectedly becomes unavailable. In both situations, the ns-3 process detected the anomalous conditions and successfully kept the emulation running.

MAC Address Cloning. The EmuFdNetDevice0 successfully cloned the real eth0 MAC address, and every communication made to Real Node #2 appeared in the Wireshark logs as originating from the MAC address of the real eth0 interface. Also, the ns-3 emulated node replied successfully to every communication directed to it.

IP address cloning. The usage of different IP addresses during the tests, in both ppp0 and eth0 interfaces, allowed to successfully test whether the IP addresses were updated in the ns-3 node. This mechanism worked with success. The time interval between checks of the real interface's IP address is configurable. If the IP address changes very frequently, it is recommended to use low interval checks to keep the ns-3 node settings updated; yet, too small time interval uses more CPU resources.

After validating each new feature, the correct operation of the improved EmuFdNetDevice was confirmed.

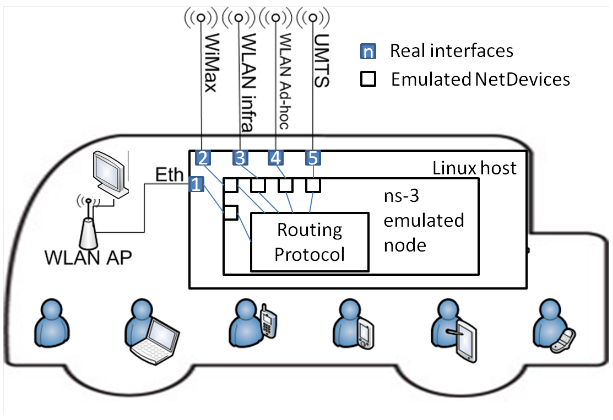


Figure 3.5: Elements present in each bus of the Vehicular network testbed scenario.

3.4.2 Vehicular Network Testbed

With the improved EmuFdNetDevice tested in the laboratory testbed, the next step was to test it in a real world testbed. The selected testbed was the one used in the real pilot of the SITMe's project, composed by 11 buses with an on-board Linux router, supporting multiple access technologies to provide Internet access to bus passengers.

Figure 3.5 depicts the elements composing each of the 11 buses used in the SITMe's real pilot. This testbed used the Wireless Metropolitan Routing Protocol (WMRP) [19], a proactive multi-technology routing protocol based on OLSR, entirely developed in ns-3. Each bus had a Linux computer installed with five network interfaces, one to give network access to the passengers and other bus equipment, and the other four to connect to the outside world. Using the Fast Prototyping process, the routing protocol ran in an ns-3 emulated node, as illustrated in Figure 3.5. This emulated node had as many Emulated NetDevices (EmuFdNetDevice) as the number of real interfaces connected to the real node. Through this mechanism, the ns-3 router had direct access to the real networks and acted as a real router from the real networks' perspective.

All features introduced in the improved EmuFdNetDevice were tested in this scenario: multiple interfaces were configured using dynamic IPs obtained via DHCP; the 3G operator provided IP level PPP interfaces, which were intermittent; MAC address cloning was used in all the interfaces with MAC address, to avoid MAC address collisions and obey to the ISP policy. WiMAX traffic was only allowed by the operator for specific real interfaces' MAC addresses.

This experiment ran successfully for more than one year with good results and very good feedback from the bus passengers. The real world usage of the improved EmuFdNetDevice proved its correct operation and usefulness. This was especially relevant in a heterogeneous real world scenario such as the SITMe's real pilot, where multiple emulated instances needed to be deployed and auto-configured to allow communications in a demanding real network environment.

3.5 Summary

The Fast Prototyping process has several possible applications and was already proven useful, although it fully depends on the ns-3's emulation capabilities. Currently, ns-3 emulation capabilities are enabled by running simulation code in real time and establishing network communications with real networks using the EmuFdNetDevice, which is bound to the real network interfaces of the real node running the ns-3 process. However, if used in complex and dynamic real network scenarios – e.g., a vehicular network – the current EmuFdNetDevice has limitations, thus invalidating the use of the Fast Prototyping process in such scenarios. This is due to the incompatibilities with some real interfaces operation and the overhead and error-prone methods needed to configure each emulated instance.

Motivated by the need to overcome these problems, we proposed an improved, backwards compatible EmuFdNetDevice. The improved EmuFdNetDevice includes a set of features that addresses interface compatibility problems and introduces self-configuration mechanisms to enable the Fast Prototyping process in demanding real network conditions, such as those associated to vehicular network scenarios. Using a laboratory testbed and a real-world vehicular network testbed, it was possible to confirm the proper operation of the improved EmuFdNetDevice. Also, the usefulness of such features was demonstrated in a complex real network scenario, such as the SITMe's real pilot. With this contribution in place, we can now use the improved EmuFdNetDevice over layer 3 real PPP interfaces, such as the ones provided by cellular networks and layer 3 VPNs.

On top of this, the improved EmuFdNetDevice also supports IP layer and MAC cloning autoconfiguration settings, which are helpful when using ns-3 emulation in large scale and complex mobile experiments. These experiments have often dynamic network configurations, connecting to different networks not managed nor owned by the experimenter. We plan to integrate this functionality into the ns-3 release, bringing the aforementioned benefits to the ns-3 simulation community. This can also have a positive impact on the adoption of the Fast Prototyping development process, especially for those complex scenarios in which the current EmuFdNetDevice cannot operate.

Chapter 4

Improving ns-3 Emulation Performance for Fast Prototyping of Network Protocols

Chapter 2 introduced the Fast Prototyping protocol development process for improving the performance evaluation of wireless networks. Chapter 3 addressed the limitations of ns-3 emulation and presented our improved version of the ns-3 EmuFdNetDevice to overcome them. Although the Fast Prototyping process is improved by the proposed EmuFdNetDevice, the additional packet processing involved when the shared protocol model is run in emulation mode brings up overhead that results in a network performance bottleneck.

In this chapter we propose a solution to improve the performance associated with Fast Prototyping that consists in moving the data plane processing operations to outside of the ns-3 process, running such operations natively in the host Operating System (OS). In a well-designed network most of the traffic should be data. By moving the data plane operations to outside of ns-3 the overhead associated with data traffic is greatly reduced, while control plane protocols may still be reused. In order to validate our proposed solution, we extended the Wireless Metropolitan Routing Protocol (WMRP) and the Optimized Link State Routing (OLSR) protocol accordingly, evaluated their performance in real environment when using the proposed solution, and verified the amount of code reuse between the Simulation Model and the Implementation Prototype.

4.1 Problem and Motivation

The Fast Prototyping process was already presented in Chapter 2 and its compatibility with real-world networks was studied and improved in Chapter 3. The main problem of Fast Prototyping is related to running the ns-3 implementation in a real testbed. The impact on performance due to the processing of the real traffic inside ns-3, using ns-3 emulation, can be substantial.

With improved performance, Fast Prototyping can represent a viable complementary approach to existing shared protocol implementation solutions such as Direct Code Execution (DCE) [33], but from the ns-3 starting point rather than from the existing implementation starting point. This is especially relevant when creating new routing protocols, where developers may start by an easier to code ns-3 implementation, taking also advantage of the ns-3 tracing capabilities. On the other hand, with improved performance, Fast Prototyping can enable hybrid scenarios (combining emulated and real resources) where we may need to run multiple emulated nodes in the same host machine. In hybrid scenarios a real testbed composed by fast prototyped nodes can be interacting with a node emulation server. The emulation server may be responsible for augmenting the complexity and scale of the real testbed by introducing a number of fully emulated nodes, which are able to interact, in real time, with other fast prototyped and real nodes. Being able to maintain the necessary real-time network performance, while running these complex networking scenarios is very relevant for the evaluation and validation of networking solutions in large scale; this is usually very difficult or impossible to achieve with real testbeds only, due to the resource limitations to build, manage, and operate large scale testbeds.

4.2 Migrating the Data Plane to Outside of ns-3

In a network node there are two planes: the control plane and the data plane. The control plane is responsible for control functions such as discovering and maintaining network routes and ensuring connectivity. The data plane uses, for example, the routing information generated by the control plane to forward network packets. In a well designed network, most of the traffic corresponds to data packets. Processing large amounts of packets in ns-3 is less efficient than doing it in user space or in kernel space. Thus, by moving the data plane operations to outside of ns-3 the overhead associated with the data traffic can be greatly reduced.

We propose two alternative approaches for doing this: 1) executing the Data Plane in User Space (DPU); 2) executing the Data Plane in Kernel Space (DPK). The two approaches are described in what follows.

4.2.1 Data Plane in User Space (DPU)

The first approach consists in running the data plane outside of ns-3 in user space. The data plane is executed in a user space process to avoid the overhead of processing the large amount of data traffic in the simulator. The control plane is executed in ns-3 and communicates with the real environment through emulation, as illustrated in Figure 4.1.

The two planes are executing in different processes; Inter-Process Communication (IPC) is used to exchange information. For example, the control plane needs to update the routing table information in the data plane, so that data packets can be forwarded correctly. The data plane may have to send feedback to the control plane or request a route in the case of reactive protocols.

The classification of the packets that should be delivered to the control and data planes is done through filters applied to the raw sockets using the Linux Socket Filter (LSF) mechanism [48]. Listing 4.1 shows an example filter that accepts only IPv4 packets. At line 0 we load the halfword (16 bits) at position 12 in the frame (the EtherType field). Line 1 compares the previously loaded

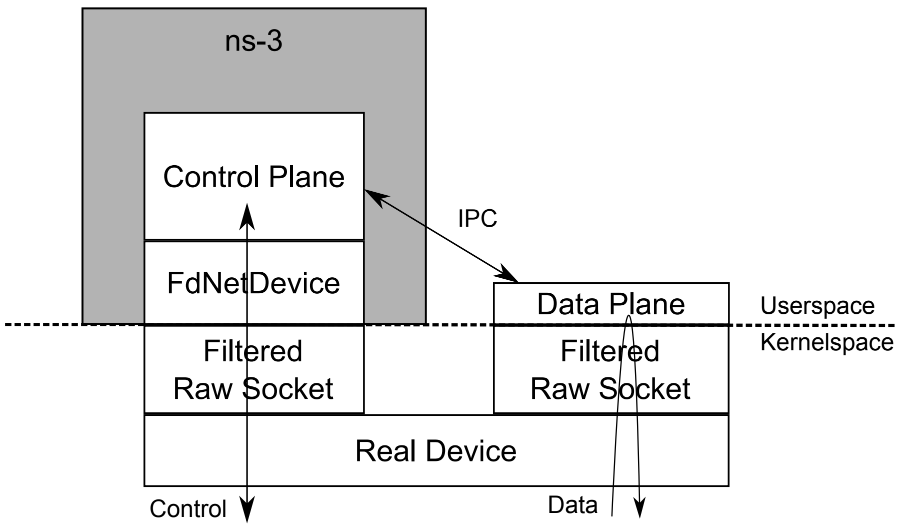


Figure 4.1: Data plane in user space (DPU) approach.

halfword to 0x800 (the IPv4 EtherType). If both are equal we jump to line 2, which returns 65535. This will result in the first 65535 bytes of the frame being accepted and delivered to the user space process. If the comparison of line 2 fails, we jump to line 3 which returns 0. This means that no bytes of the frame are accepted and the frame is dropped.

A simple way to obtain the code of socket filters is to run `tcpdump` with the `-d` option. For example, running `tcpdump -i eth0 -d ip` will return the code in Listing 4.1.

Listing 4.1: Linux Socket Filter code that only accepts IP packets.

```
ldh      [12]
jeq      #0x800         jt 2  jf 3
ret      #65535
ret      #0
```

Socket filters are configured in user space but they are executed in kernel space on received frames. As such, they are very efficient because packets not accepted by the filter are not delivered to user space. In this way, the control and data planes only receive their respective traffic, avoiding the overhead caused by processing all packets received from the real interface.

Using the DPU approach, the control plane code can be reused between the Simulation Model and Implementation Prototype although the data plane code needs to be rewritten for the Implementation Prototype. However, because the data plane is usually much simpler than the control plane, only a relatively small amount of code needs to be ported, for instance, to maintain a cached routing table, correctly interpret and manipulate the frame/packet headers, and forward the traffic to the right network interfaces.

4.2.2 Data Plane in Kernel Space (DPK)

For L3 proactive routing protocols it is possible to specialize the DPU approach, depicted in Figure 4.1. We name this the Data Plane in Kernel Space (DPK) approach. For protocols of this type, whose objective is to forward IP packets, we can update the kernel routing tables with the information gathered by the control plane. In this way, the kernel will forward the traffic received from the real device (Figure 4.2), which is more efficient than doing it in user space.

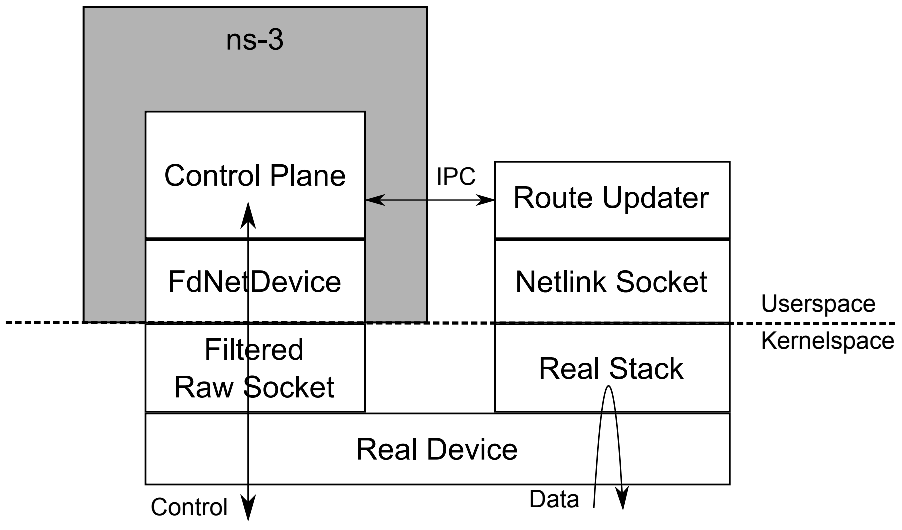


Figure 4.2: Data plane in kernel space (DPK) approach.

In the DPK approach, a route updater process receives the routing information from the control plane and updates the kernel routing tables accordingly. To do this, the route updater process uses a netlink socket [49]. Netlink is a communication channel between kernel space and user space and, among other functionalities, allows user processes to update and retrieve routing information. *Root* access is needed to use this functionality. In order to avoid running the whole simulation as *root*, the route updater is executed in its own process as *root*. The route updater communicates with ns-3 through IPC to receive the routes generated by the control plane. As in the DPU approach, the raw socket used by the control plane needs to be filtered to avoid the overhead of processing all network traffic.

The DPK approach is more efficient than the DPU approach since the forwarding is done in kernel space. In addition, there is even less code that needs to be rewritten because the route updater code can be reused by all routing protocols that use the DPK approach. The downside is that the DPK approach is only applicable to proactive L3 protocols, where the control plane is typically independent from the data plane. For example, reactive protocols such as AODV need to know if a route was not found for a given packet in order to initiate the route request procedure [50]. To support this type of protocols the proposed architecture can be extended with a kernel module that listens on the desired events and reports them back to the control plane.

Real Routing **Module**

Most of the DPK approach code can stay the same across different protocols, namely the route updater and the IPC. The only part that is specific to each protocol is the socket filter used. This is a clear advantage over the DPU approach, which requires the data plane to be ported to the real system. To enable researchers to easily use the DPK approach, an implementation was developed for ns-3, the *real routing* module [51].

The *real routing* module implements a route updater (*RtNetlinkRouteUpdater*) that uses Netlink sockets to update the kernel routing tables and runs in a privileged user space process to avoid having to execute the whole simulation with *root* privileges. Inside ns-3, the class *RealRouteUpdater* was created to spawn and configure the *RtNetlinkRouteUpdater* process, receive the routes from the routing protocols, and send them to the created process, which in turn updates the kernel. The IPC, used to send the routes from the simulator to the route updater, is implemented with Unix domain sockets.

In order for ns-3 routing protocols to be able to update the routes of the system without having to be coupled to the new *real routing* module, three new optional callbacks were added to *Ipv4RoutingProtocol* – the base class of all IPv4 protocols. Support for IPv6 was not implemented but it can be easily added when needed. The new callbacks are: *RouteAddedCallback*, called when the protocol creates a new route; *RouteRemovedCallback*, called when the protocol deletes a route; and *RoutingTableUpdatedCallback*, called when the protocol updates the whole routing table at once. Protocols that use these callbacks will work with the *real routing* module. The *RealRouteUpdater* will register its route updating functions with each of the protocol's callbacks. Thus, every time one of the callbacks is called, the corresponding function in the *RealRouteUpdater* is also called and the received information is sent to the *RtNetlinkRouteUpdater* process, which, in turn, updates the kernel routing tables, as it is illustrated in Figure 4.3.

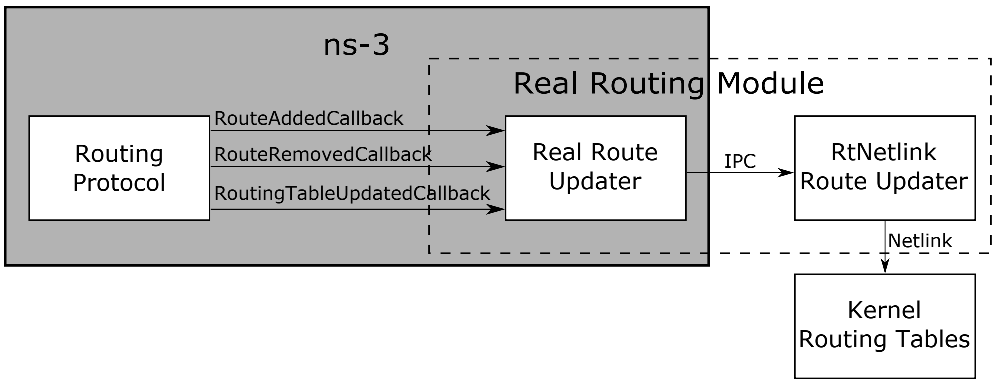


Figure 4.3: Real Routing module architecture.

Finally, two other new features were added to the *EmuFdNetDeviceHelper* in the *fd-net-device* module to support 1) the DPK approach and 2) the emulation of multiple nodes using network namespaces by setting a socket filter in the raw socket and selecting a network namespace for the socket.

With the *real routing* module implemented, a researcher must take the following steps to use the DPK approach for a new routing protocol. When implementing the routing protocol in ns-3, the new callbacks must be called when a route is added, removed, or the whole routing table is updated at once. Then, when using the protocol in real environments, the simulation script needs to consider the following: 1) set the socket filters in the *EmuFdNetDeviceHelper* and, if emulating multiple nodes, set the network namespace as well; 2) for each emulated routing node, create an instance of the *RealRouteUpdater* and configure it with the routing protocol being used, the network namespace of the node, and the mapping between the ns-3 interfaces and the real interfaces of the host. With this configurations done, the created scenario will then be executed using the DPK approach. All the patch files needed to add the *real routing* module to ns-3.21 source code are available in [51]. The patch files have to be applied by the order they are numbered.

4.2.3 Emulating Multiple Nodes

The DPU and DPK approaches assume that a single node is being emulated by each machine of the testbed. However, it may be desirable to emulate multiple nodes in a single machine, as discussed in Section 4.1. Figure 4.4 shows an example of a single host emulating three nodes, where the network traffic coming from one real device is forwarded to other real device by traversing the data plane of the three nodes.

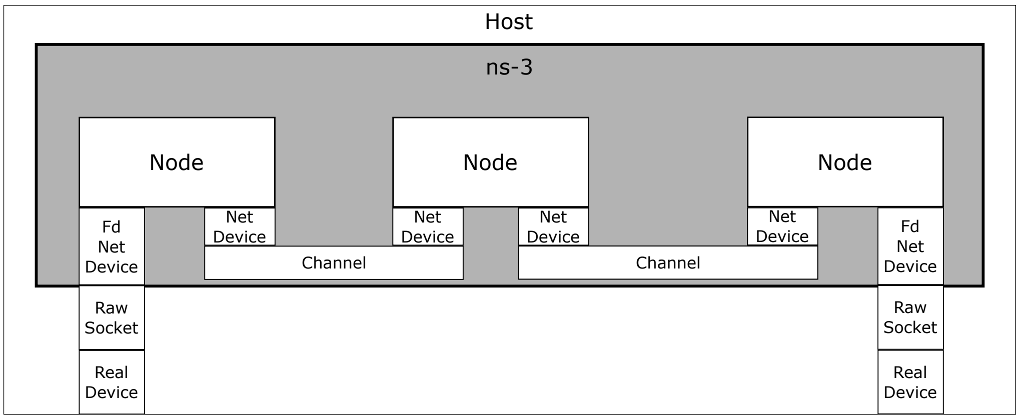


Figure 4.4: Emulation of multiple nodes in a single host machine.

To apply the DPU and DPK approaches for emulating multiple nodes, we need to have execution environments in the host system where the data plane of each node will run. These environments must have the following properties:

1. **Be independent of each other.** Any modification to a given environment must not affect the others (e.g., adding routes to the routing table of one environment must not affect the routing tables of the other environments).

2. **Be able to forward data traffic.** They must enable data traffic forwarding between each other and with the nodes outside the system.

3. **Be efficient.** They must not introduce more overhead than what is saved by moving the data plane to outside of ns-3.

The Linux OS provides environments with these properties named network namespaces [52]. Namespaces allow the network resources of the system to be isolated into different independent environments. Each network namespace has its own set of components, including devices, addresses, ports, routing tables, and firewall rules. For an efficient emulation architecture, network namespaces are more appropriate than virtual machines or Linux containers, because they isolate what is needed for the DPU and DPK approaches – the network resources.

The topology of the simulated nodes in ns-3 must be replicated outside by using the network namespaces. This is done by creating a network namespace for each node and connecting them in the same way they are connected in the simulator. The network namespaces also need to be connected to each other in a way that replicates the connections between the simulation nodes. To do that, virtual network devices (`veth`) are used. These virtual devices are created in pairs and assigned to two namespaces that need to be connected. What is written in one `veth` can be read in the other, and vice-versa. In this way, the network namespaces can forward the data traffic between each other. The virtual devices can also be configured to match the characteristics of the link connecting the simulation nodes (e.g., delay, bitrate, and packet loss ratio). This is possible by using the *netem* kernel module [53]. For example, to set the delay of virtual device `veth1` to 100 ms, we can use the command `tc qdisc add dev veth1 root netem delay 100ms`.

Figure 4.5 shows how the DPK approach can be applied to the example in Figure 4.4 by using network namespaces and virtual devices. The same approach can also be used by the DPU approach. The only difference is the data plane would be processed in a user space process for each network namespace needed, instead of being processed in the Linux kernel as in the DPK approach.

For each emulated node, a network namespace is created and a route updater for that node is executed in that namespace. In this way, each node has its own set of routing tables in the kernel that are independent of the other nodes. To connect the nodes, virtual network device pairs are created. Each `veth` is assigned to a network namespace and configured to match the corresponding *net device* of the simulation (e.g., IP address, subnet mask, and emulation characteristics that can be set with *netem*, such as delay and packet loss ratio). The real devices (cf. Figure 4.5) of the host are also moved to the network namespace of the nodes that should be connected to the real network. The filters of the raw sockets are applied in the same way for the emulation of a single node.

The control traffic is thus processed completely within ns-3, while the data traffic never leaves kernel space, as illustrated by the bidirectional arrows in Figure 4.5. Data packets that arrive in a real device are processed by the real stack of a network namespace and forwarded to the next namespace through a virtual device. This process continues until the data packets leave the host through a real device.

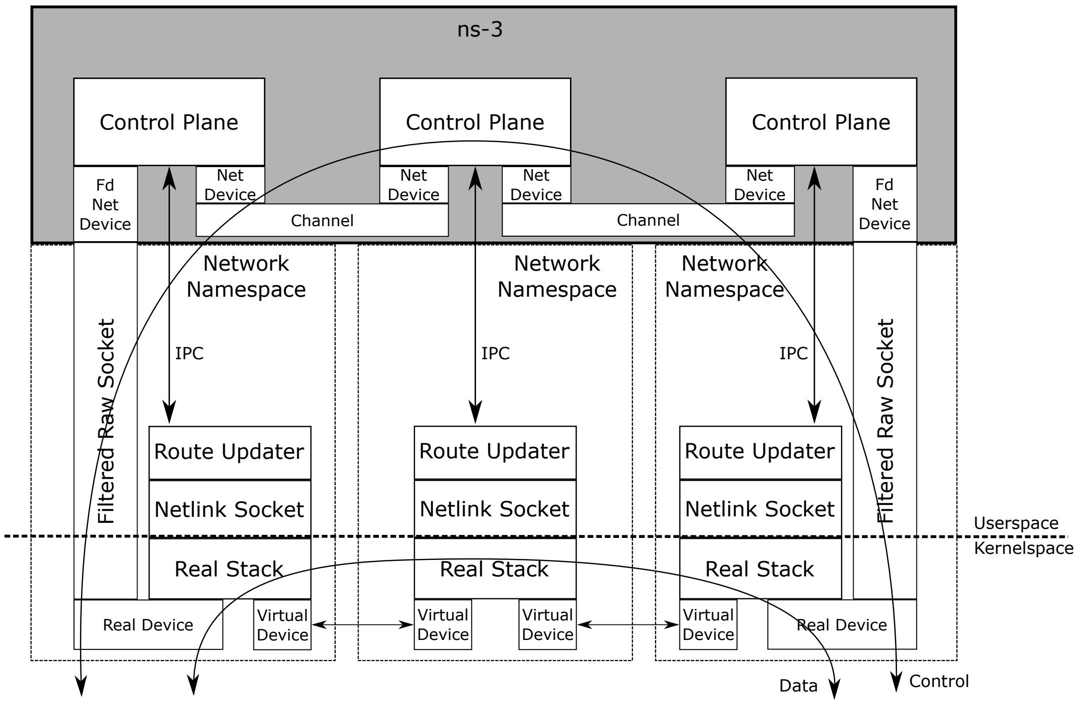


Figure 4.5: DPK approach for emulation with multiple nodes.

4.3 Validation

In this section we validate the DPU and DPK approaches focusing on two main metrics: 1) performance – assessing the gains introduced by running the data-plane outside of ns-3, resulting from the lower expected per packet processing overhead; 2) code reuse – assessing how much extra code in necessary by each approach when compared to the original Fast Prototyping process. The performance was validated by selecting routing protocols already implemented in ns-3 and extending them to use the new DPU and DPK approaches. The performance of the DPU and DPK approaches was then compared to the performance of traditional emulation in real experiments to calculate the respective gains. The code reuse was validated by analyzing the number of lines of code that were added to the Protocol Model for supporting the DPU and DPK approaches.

4.3.1 Data Plane in User Space (DPU)

4.3.1.1 Performance Validation

To validate the DPU approach we used the Wireless Metropolitan Routing Protocol (WMRP), an ad-hoc, proactive routing protocol that operates over Layer 2 [18][43][54]. The nodes running WMRP are called Routing Bridges (RBs). RBs encapsulate the L2 traffic from a source legacy terminal, forward it between RBs using MPLS tags, and decapsulate the original L2 traffic when it reaches the RB containing the destination legacy terminal attached.

The performance gain obtained in real environments was validated by comparing the performance of WMRP running in ns-3 emulation mode with the performance of WMRP when the

DPU approach is used. We then analyze the performance gains obtained with the DPU approach. Without loss of generality, the network topology illustrated in Figure 4.6 was considered. It was composed of four nodes connected by 100 Mbit/s Ethernet links: the source **S** used the `iperf` tool to generate UDP traffic destined to **D**; at the same time it pinged **D** to measure the round-trip time (RTT). The destination **D** ran an `iperf` server. The Routing Bridges **RB1** and **RB2** forwarded the frames between **S** and **D**.

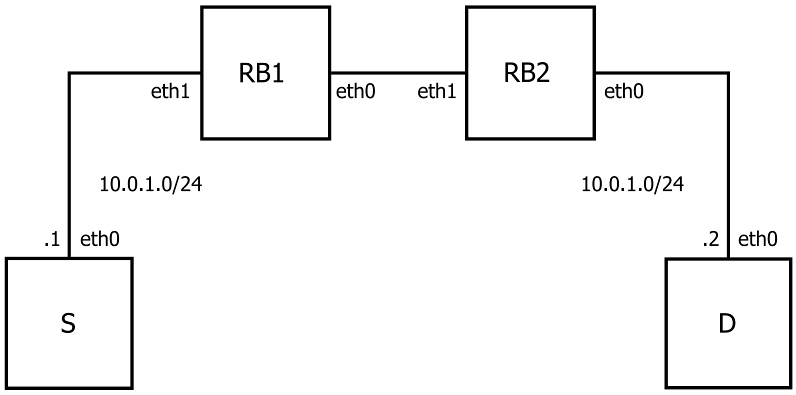


Figure 4.6: Network topology of the test scenarios related to the DPU approach.

Two scenarios were tested. In the first scenario, the two RBs were implemented using the traditional ns-3 emulation mode. In the second scenario, the DPU approach was implemented in both RBs. For each scenario a series of tests was executed, starting with **S** generating 1 Mbit/s of UDP traffic to **D** and then increasing that value to 2, 4, 8, 16, 32, 64, and 70.8 Mbit/s. Each test had a duration of 30 seconds and was repeated 5 times. The payload of the UDP packets was 160 bytes, representing typical VoIP traffic. The small packets induce a network load representing a worst case scenario where the network had to process a large number of packets per second and maxed out the CPU processing capabilities. This was especially important due to the absence of Gigabit Ethernet cards in the test host machine. The maximum offered data rate was of 70.8 Mbit/s because that is the maximum throughput that can be achieved in 100 Mbit/s Ethernet links for UDP packets with 160 bytes of data. For each of these tests, four performance metrics were measured: the received data rate in **D**, the packet loss ratio, the average round round-trip time, and the average CPU load in **RB2**. The host machine running **RB2** has an Intel Atom N270, which is a single core processor. Yet, it uses the Hyper-Threading technology, so it behaves as having two logical cores. The Linux OS used was Ubuntu 14.04 LTS x86.

The results are shown in Figure 4.7, where the lines represent the average value and the error bars represent the standard deviation. The traditional ns-3 emulation implementation is able to forward packets until the offered rate is around 8 Mbit/s. After that, performance degrades and when the offered rate hits 70.8 Mbit/s, ns-3 emulation has 87% of packet loss ratio, can only forward at 9 Mbit/s, and the round trip time (RTT) is around 85 ms. The reason why ns-3 emulation goes above 100% load is that the CPU of **RB2** has two logical cores (due to Hyper-Threading) and ns-3 uses a dedicated thread for reading operations.

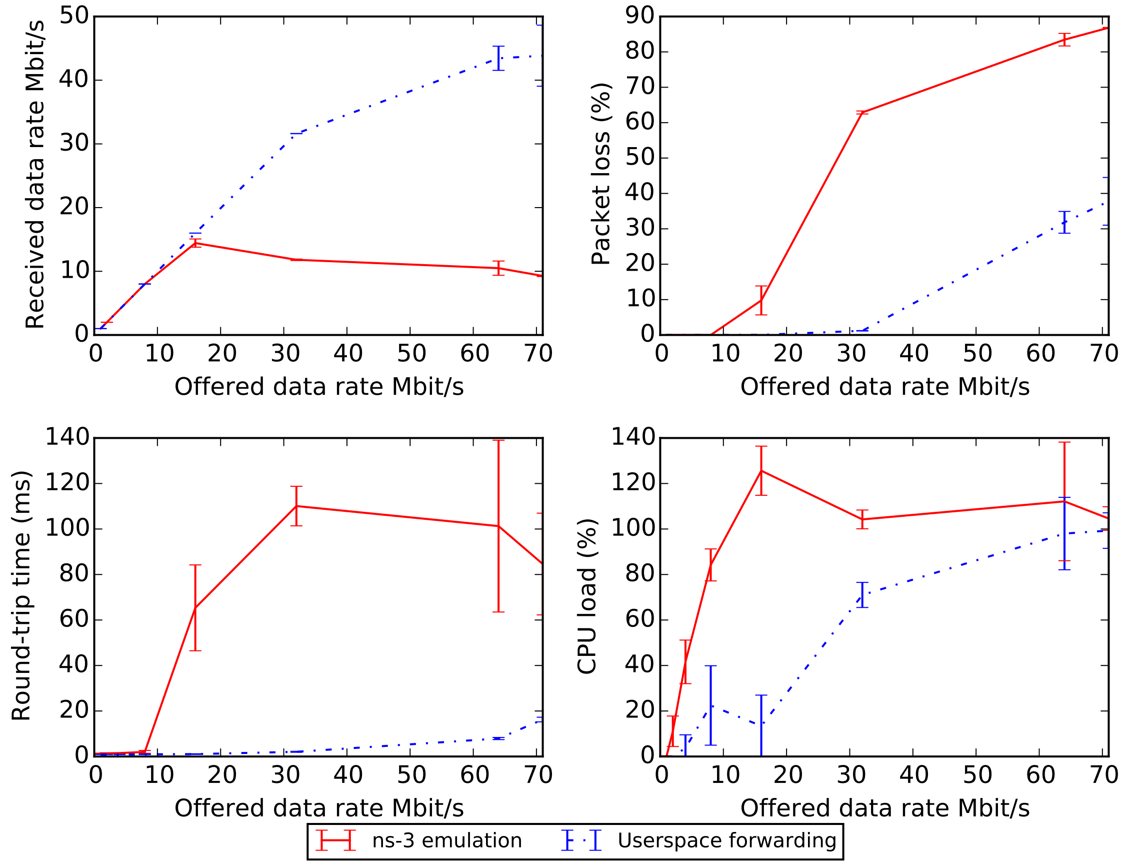


Figure 4.7: DPU performance validation results for UDP traffic with payload of 160 bytes.

The DPU approach is able to forward packets until the offered rate is 32 Mbit/s while keeping RTT low (1 ms until 16 Mbit/s and 2 ms at 32 Mbit/s). For higher data rates the performance starts degrading, but at 70.8 Mbit/s it can forward packets at around 44 Mbit/s (4.9 times more than the traditional emulation mode) while keeping RTT at 16 ms (5.3 times lower than the traditional emulation mode). Additionally, while the offered data rate is lower than 64 Mbit/s the CPU load of the DPU approach is lower than that of ns-3 emulation. For higher data rates both implementations present similar loads, but the user space implementation is processing more traffic for the same load.

From the tests presented in Chapter 2, we know that the processing bottleneck is mostly related to the number of packets processed rather than to the size of the packets forwarded by the node. This is easily understandable as "bulk" memory copy operations are usually much faster than the multiple function calls and processing involved with the handling of each packet object, such as the interpretation of the header and routing/forwarding tables lookups. Based on this information, it is expected a ten-fold performance improvement in the forwarding data rates if **RB2** was handling 1500 bytes Ethernet frames in Gigabit Ethernet links.

4.3.1.2 Code Reuse Validation

To validate that using the DPU approach does not require writing a large amount of new code, we counted the lines of code that were developed to implement the DPU approach on top of the existing ns-3 implementation of WMRP. In this process we discarded the blank and comment lines.

The original implementation of WMRP has 5057 lines of code, while the same protocol implementation using the DPU approach has 5608 lines of code. This means that only around 11% more lines of code were written to implement the data plane of WMRP in user space. This is in part because the implemented data plane was simplified and assumes that the topology of the network does not change. Implementing the full operations of the data plane would increase the number of lines of code. Yet, in general, the data plane is much simpler than the control plane. So, only a relatively small amount of code needs to be ported.

Ultimately, the researchers developing the protocol have to decide whether the extra amount of code that has to be written is worth the performance gains that can be obtained.

4.3.2 Data Plane in Kernel Space (DPK)

4.3.2.1 Performance Validation

The DPK approach was validated using the Optimized Link State Routing Protocol (OLSR), a link state, proactive, L3 protocol [55]. The performance gain obtained in real environments was validated by comparing the performance of three different implementations: OLSR using traditional ns-3 emulation mode; OLSR using our proposed *real routing* module; and olsrd, a Linux implementation of OLSR [56].

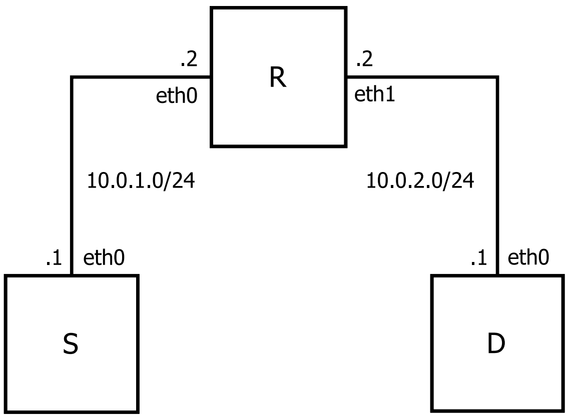


Figure 4.8: Network topology of the test scenarios related to the DPK approach.

Without loss of generality, the network topology illustrated in Figure 4.8 was considered. It was composed of three nodes connected by 100 Mbit/s Ethernet links: the source **S** ran OLSR and used the `iperf` tool to generate UDP traffic destined to **D**; while at the same time it pinged **D** to measure the RTT. The destination **D** ran OLSR and an `iperf` server to receive the UDP traffic from **S**; **R** ran OLSR and forwarded the packets between **S** and **D**. Three scenarios were tested. In the first, **R** ran a traditional ns-3 emulation of OLSR. In the second scenario, **R** ran a ns-3 emulation of OLSR using the *real routing* module developed to implement the DPK approach. In the last scenario, **R** ran *olsrd*. In all scenarios, **S** and **D** ran *olsrd*. The router **R** ran on an Intel Atom N270 with Ubuntu 14.04 LTS x86. For each scenario, a series of tests was executed, following the same procedure and parameters used to validate the performance of the DPU approach. For each of these tests, four performance metrics were measured: the received data rate in **D**, the packet loss ratio, the average RTT, and the average CPU load in **R**.

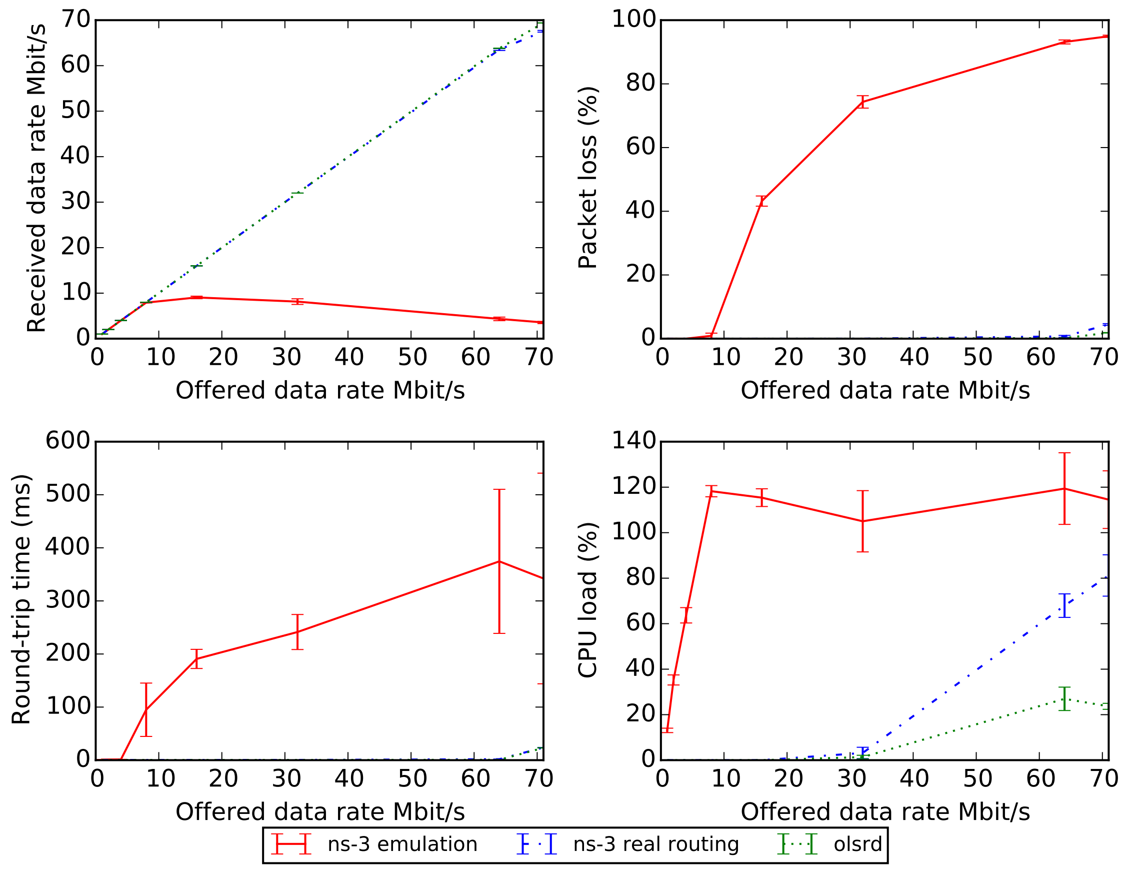


Figure 4.9: DPK approach performance validation results for UDP traffic with payload of 160 bytes.

The results are shown in Figure 4.9, where the lines represent the average and the error bars represent the standard deviation. The traditional ns-3 emulation implementation is able to forward packets until the offered rate is 8 Mbit/s. After that, performance degrades and when the offered rate hits 70.8 Mbit/s, ns-3 emulation has 95% packet loss and can only forward at 3.5 Mbit/s. The round trip time (RTT) also increases to 342 ms, while until 8 Mbit/s it was around 1 ms. The reason why ns-3 emulation goes above 100% load is the same as explained for the DPU approach performance validation.

The *real routing* implementation and *olsrd* are able to forward packets at all offered rates except at 70.8 Mbit/s. At this highest rate there is packet loss of 4% and 2% for *real routing* and *olsrd*, respectively, and they forward traffic at 67.5 Mbit/s and 69 Mbit/s, respectively. Both implementations provide very low round trip times, less than 0.5 ms until 32 Mbit/s, around 1 ms at 64 Mbit/s, and around 23 ms at 70.8 Mbit/s. This means that, at the highest rate, real routing provides an increase in throughput of around 19 times when compared to traditional emulation, as well as a decrease in RTT of around 14 times.

The reason that real routing performs slightly worse than *olsrd* is because it incurs a higher CPU load (around 3 times more at 70.8 Mbit/s). This can be attributed mainly to the socket filters, which must be executed for every packet that is received in the system. One way to reduce the overhead incurred by the filters is to enable them to be JIT (just-in-time) compiled, which means that instead of the kernel interpreting them for each packet, it will compile them to the underlying processor architecture just once and then run the compiled code for the received packets. However, we could not enable this optimization in the system we were using because it is a x86 machine and Ubuntu 14.04 only configures the kernel to allow JIT compilation of filters in x64 systems.

As explained in the performance results for the user space implementation using WMRP, in this OLSR scenario it is also expected a ten-fold performance improvement in the forwarding data rates if the router was handling 1500 bytes Ethernet frames in Gigabit Ethernet links.

4.3.2.2 Code Reuse Validation

To validate that using the DPK approach only requires writing a small amount of new code we counted the lines of code that were developed to use the *real routing* module with the existing ns-3 implementation of OLSR. In this process we discarded the blank and comment lines.

The existing ns-3 implementation of OLSR has 5067 lines of code, while the same protocol implementation including the extension to work with the real routing module has 5140 lines of code. The lines of code of the *real routing* module itself are not counted because it is a generic module that can be used with any proactive L3 protocol without having to be re-implemented. This means that only around 1.4% more lines of code were written to use the *real routing* module callbacks to update the routing table; the rest of the lines correspond to code that sets the socket filters and initializes the real routing module. We can thus conclude that the increase in code is negligible, especially when compared to the performance gains that are obtained when using the DPK approach.

4.3.3 Emulating Multiple Nodes

4.3.3.1 Performance Validation

Because the DPK approach presented the best performance and code reuse results for a single emulation node, and also considering the fact that the DPU approach was already maxing out the CPU load for the emulation of a single node, we opted to use the DPK approach for validating the emulation of multiple nodes. We compared the performance of OLSR running on traditional ns-3 emulation of multiple nodes and OLSR using the *real routing* module and network namespaces.

Without loss of generality, the network topology illustrated in Figure 4.10 was considered. It was composed of three physical nodes connected by 100 Mbit/s Ethernet links: the source **S** ran OLSR and used the `iperf` tool to generate UDP traffic destined to **D** while at the same time it also pinged **D** to measure the RTT; the destination **D** ran OLSR and an `iperf` server to receive the UDP traffic from **S**; the machine **R** sat between **S** and **D** and emulated a network of OLSR nodes that were connected in a line topology.

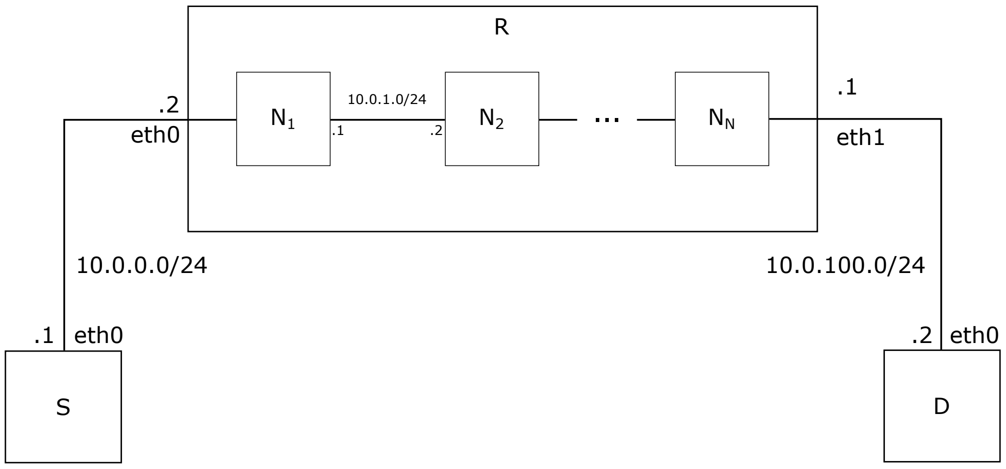


Figure 4.10: Network topology of the test scenarios regarding the DPK approach for emulating multiple nodes.

The line topology was used to determine how performance varies with the number of hops in a real scenario. Two scenarios were tested. In the first scenario, **R** ran a traditional ns-3 emulation of multiple OLSR routers. In the second scenario, **R** ran an ns-3 emulation of multiple OLSR routers using the *real routing* module and the network namespaces architecture. In both scenarios, the links between the emulated nodes (simulated channels in ns-3 and virtual devices in network namespaces) were configured to have zero latency and 1 Gbit/s of maximum throughput.

For each scenario, two series of tests were executed to assess the UDP (first) and TCP (second) throughput. Both series started with **R** emulating a single node, and then increasing the number of emulated nodes in **R** to 2, 4, 8, 16, 24, and 32. Each test had a duration of 30 seconds and was repeated 5 times. In the first series of tests **S** generated 16 Mbit/s of UDP traffic to **D**. This value was chosen because it is the maximum value that allows for some traffic to be received when using traditional emulation of 32 nodes. The payload of the UDP packets was 160 bytes. For each of the tests in the first series, four performance metrics were measured: the received data rate in **D**, the packet loss ratio, the average RTT, and the average CPU load in **R**. In the second series of tests, **S** generated TCP traffic to **D**. For each of the tests in this series, three performance metrics were measured: the received data rate in **D**, the average round round-trip time, and the average CPU load in **R**. The machine **R** ran on an Intel Atom N270, which is a single core processor that uses the Hyper-Threading technology; so it behaves as having two logical cores. The OS used was Ubuntu 14.04 LTS x86.

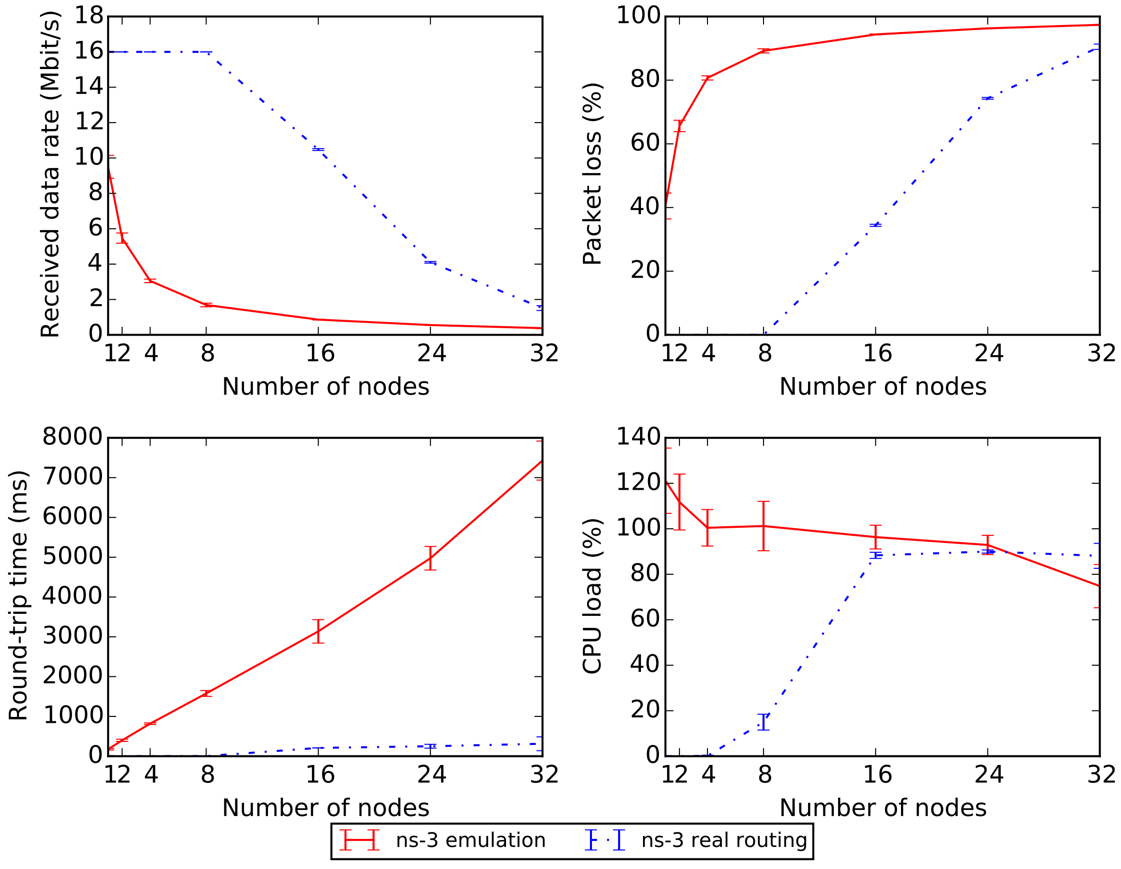


Figure 4.11: Results of emulating multiple nodes for UDP traffic with payload of 160 bytes.

The results for UDP traffic are shown in Figure 4.11, where the lines represent the average and the error bars represent the standard deviation. Even with only one node, the traditional ns-3 emulation can only forward packets at 9.5 Mbit/s. With 8 emulated nodes there is already an 89% packet loss, a forwarding rate of less than 2 Mbit/s, and a RTT of more than 1.5 seconds. When the number of emulated nodes reaches 32, the packet loss is 97% and the throughput is only 0.38 Mbit/s. RTT also peaks at 7.4 seconds.

The real routing implementation with network namespaces is able to achieve significantly better performance. Until 8 emulated nodes it forwards traffic at 16 Mbit/s (8 times more than traditional emulation). The RTT is low as well, around 0.5 ms until 4 nodes and 2 ms for 8 nodes (750 times less than traditional emulation). With more than 8 nodes the performance starts decreasing and with 32 nodes, real routing only forwards at 1.5 Mbit/s (4 times more than traditional emulation) with a packet loss of 88%. RTT also increases to around 312 ms (24 times less than traditional emulation). The CPU load for real routing is lower up to 8 nodes. After that, both implementations have equivalent CPU load, but the real routing implementation can process more traffic for the same load. The results for TCP traffic are shown in Figure 4.12, where the lines represent the average and the error bars represent the standard deviation. With only one node, traditional emulation can forward 55 Mbit/s of traffic. At 8 nodes, throughput decreases to 10 Mbit/s

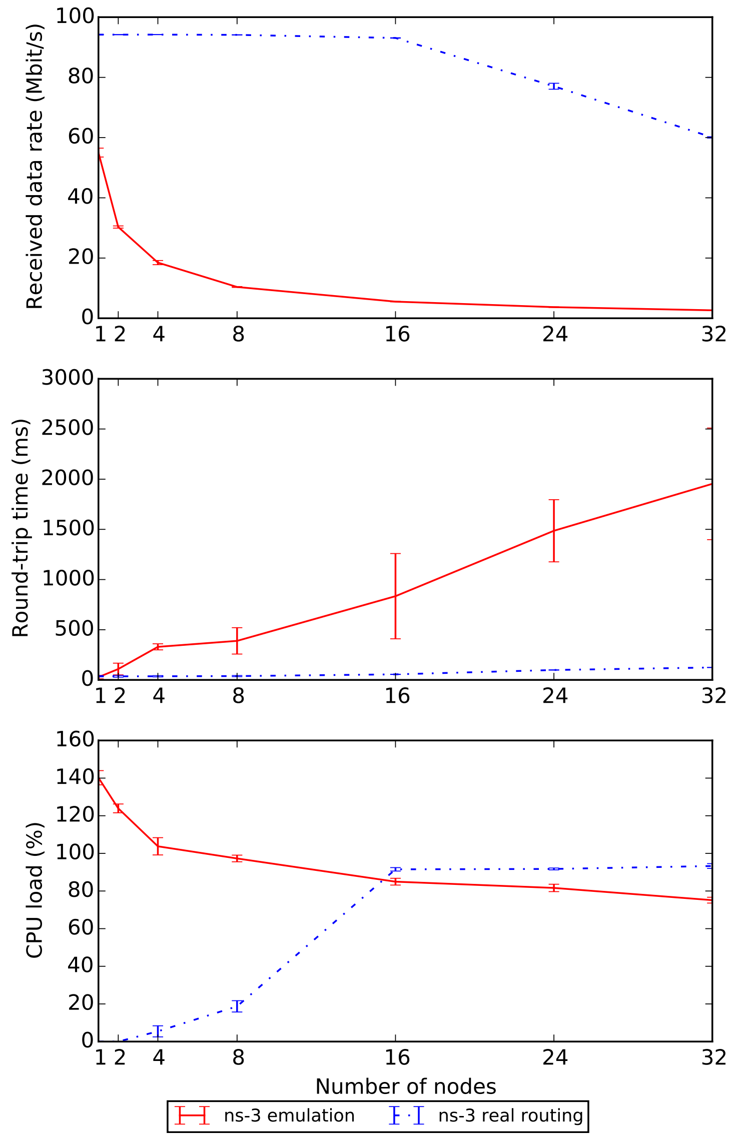


Figure 4.12: Results of emulating multiple nodes for TCP traffic.

and RTT is 389 ms. With 32 nodes, traditional emulation can only achieve 2.6 Mbit/s and an RTT of 1.9 seconds.

Real routing can achieve throughputs of 94 Mbit/s with up to 8 nodes (9.4 times higher than traditional emulation) and 93 Mbit/s with 16 nodes. The RTT at 8 nodes is 38 ms (10 times lower than traditional emulation). When the number of nodes reaches 32, real routing can forward at a rate of 60 Mbit/s (23 times more than traditional emulation) and has a RTT of 124 ms (15 times less than traditional emulation). The CPU load of real routing is also lower until 8 nodes but is slightly higher with 16 or more nodes.

We can conclude that by applying the DPK approach with network namespaces to support multiple emulated nodes increases the performance significantly when compared to traditional ns-3 emulation. Consequently, the DPK approach allows increasing the number of emulated nodes per real host while attaining the same or better performance than traditional emulation.

4.4 Comparing the DPU Approach with the Traditional ns-3 Emulation using *Oprofile*

As presented in Section 4.3.2.1, the DPU approach was able to forward network traffic at a higher throughput than using traditional ns-3 emulation. Because both approaches process the network traffic in user space, in this section we focus on studying the cause for their performance differences by recurring to software profiling techniques using the *Oprofile* Linux profiling tool.

To measure the overhead introduced by ns-3 when handling simple packet forwarding operations, we emulated a simple Ethernet bridge in ns-3 and in a user space process. The computer emulating the bridge was composed by two Fast Ethernet network interfaces, an Intel Atom D625 CPU, and 1 GB of RAM. This computer ran Ubuntu 14.04 LTS x86 OS. The emulated bridge was responsible to perform L2 traffic forwarding operations between two real Linux nodes – Source (S) and Destination (D) – connected to the Fast Ethernet ports. The S node used *iperf* to send UDP traffic at 5 Mbit/s to node D, using packets of 160 bytes.

Oprofile was then used to profile 120 s of execution for each approach (the ns-3 emulation process and the user space process) while forwarding the UDP traffic, allowing to collect the number of CPU "CLK_UNHALTED" events associated with each binary image. The results for ns-3 emulation are presented in Table 4.1; the results for the user space process (DPU approach) are presented in Table 4.2.

Table 4.1: Number of CPU "CLK_UNHALTED" events associated to each binary image, when using the traditional ns-3 emulation.

CPU "CLK_UNHALTED" Events	%	Binary Image
1032489	100.00	TOTAL
572912	55.49	vmlinux-4.4.0-148-generic (kernel)
113370	10.98	libns3.21-core-optimized.so
105014	10.17	libns3.21-network-optimized.so
96978	9.39	libc-2.19.so
39045	3.78	libpthread-2.19.so
31276	3.03	libns3.21-fd-net-device-optimized.so
18252	1.77	libstdc++.so.6.0.19
18147	1.76	libns3.21-bridge-optimized.so
9946	0.96	libns3.21-lr-wpan-optimized.so
5461	0.53	usbnet (eth1 driver)
5053	0.49	libgcc_s.so.1
2144	0.21	ld-2.19.so
2127	0.21	r8169 (eth0 driver)

Complementary to the previous results, *oprofile* allowed an in-depth analysis for calculating the amount of CPU cycles spent per each function offered by the binary images listed in Table 4.1

Table 4.2: Number of CPU “CLK_UNHALTED” events associated to each binary image, when using the DPU approach.

CPU "CLK_UNHALTED" Events	%	Binary Image
296421	100.00	TOTAL
280442	94.61	vmlinux-4.4.0-148-generic (kernel)
4378	1.48	libc-2.19.so
4028	1.36	usbnet (eth1 driver)
2794	0.94	DPU user space program
2070	0.70	r8169 (eth0 driver)

and Table 4.2. Based on this information, the number of CPU “CLK_UNHALTED” events where then thoroughly compared between the two approaches, resulting in Table 4.3.

Based on the results presented in Table 4.3 we can conclude that the traditional ns-3 emulation spends over 3 times more CPU cycles to forward the same network traffic than the DPU approach, which is in-line with our previous DPU performance validation (1032k vs. 296k CPU cycles). This difference is expected to increase when adding the complexity of the IP stack processing and forwarding operations. Table 4.3 also presents, in detail, the components in which the CPU cycles are being spent. The CPU cycles spent by the Ethernet drivers are equivalent for the two approaches, which is expected as the network traffic was equivalent for both test scenarios. The Linux Kernel operations more than doubled from 573k to 280k CPU cycles. The libc-2.19 operations represented a difference of 93k CPU cycles, from which the vast majority are related to memory allocation, copying, and freeing operations. The ns-3 also introduced a significant overhead by itself (275k CPU cycles), with its core needing 113k CPU cycles and the packet handling operations (headers, buffers, tagging, etc) needing 105k additional CPU cycles. Finally, the traditional ns-3 emulation approach also used other methods available via three additional libraries – libpthread, libstdc++ and libgcc – which also accounted for 59k additional CPU cycles.

4.5 Comparison with the new *NetmapNetDevice* and its Impact on Fast Prototyping

Recent related work by P. Imputato *et al.* [57][58], published after our work on the DPU and DPK approaches, added new capabilities to ns-3 emulation. The authors propose an alternative method for ns-3 to read and write Ethernet frames from a real network interface. Traditional ns-3 emulation relies on the *EmuFdNetDevice*, which reads and writes Ethernet frames to a real network interface using a *raw socket* from the OS of the host machine that is bound to a real interface. P. Imputato *et al.* propose a new *NetmapNetDevice* that uses the *netmap* fast packet processing framework to bypass the host networking stack and have direct access to the real network interface. This bypass reduces the per packet processing overhead, which results in a gain of 20% more packets per second when compared to the original *raw socket* approach; it also increases the realism of the packet queuing operations which affect the queue’s backlog, the packets in-flight, and the

Table 4.3: Detailed comparison of CPU "CLK_UNHALTED" events count between using traditional ns-3 emulation and DPU approaches.

	L2 Forwarding - "CLK_UNHALTED" events (thousands)		
Component	**ns-3 Emulation**	**Userspace (DPU)**	**Overhead difference**
TOTAL	1032	296	736
Ethernet drivers	8	6	2
Linux Kernel	573	280	293
libc-2.19	97	4	93
SyscallTemplate (gettimeofday)	*3*		
Malloc+Free	*65*		
Memcopy	*17*		
Bridge functionality	278	3	275
ns-3 core	*113*		
UnixSysCondition+UnixSysMutex	*22*		
WallClockSynchronizer	*20*		
Synchronizer	*15*		
RealtimeSimulatorImpl	*13*		
UnixFdReader	*5*		
MapScheduler+StlMap+StlTree	*5*		
Simulator	*5*		
EventImpl	*2*		
ns-3 network	*105*		
(Ethernet)Header+MacAddress	*27*		
Buffer operations	*19*		
Packet+PacketMetadata	*18*		
ByteTagList	*9*		
Node (ReceiveFrom)	*8*		
Address+AddressUnits	*5*		
ns-3 fdNetDevice	*32*		
ns-3 bridge	*18*		
ns-3 lr-wpan	*10*		
New methods	59	0	59
libpthread	*39*		
libstdc++	*18*		
libgcc	*5*		

packet delay. Although the processing overhead is reduced, the authors acknowledge that the ns-3 emulation performance using *netmap* is lower and has higher overhead than processing the traffic in kernel space.

Netmap is compatible only with real network interfaces operating in *netmap mode*. Real network interfaces operating in this mode are disconnected from the host OS stack, while the original *raw socket* approach allows for both the emulated nodes and the OS stack to use the same real network interface. This ability to share the same real network interface between ns-3 emulation and the real stack is what enables the DPU and DPK approaches, as they rely on migrating the data plane operations to outside of ns-3 but keeping the control plane in ns-3. Thus, the DPU and DPK approaches are incompatible with the new *NetmapNetDevice*. Nonetheless, the Fast Prototyping process will benefit from the use of the *NetmapNetDevice* if neither the DPU nor the DPK approaches are employed.

Based on the results presented by P. Imputato *et al.*, we can infer that by using the Fast Prototyping process with this new interface we would obtain 1.2 times higher throughput when the CPU is fully loaded. In comparison using the Fast Prototyping with DPU and DPK approaches will enable us to obtain respectively 4.9 and 19 times higher throughputs.

In conclusion, although *NetmapNetDevice* is a relevant contribution which reduces the overhead of ns-3 emulation and improves its network performance realism, our DPU and DPK approaches are characterized by a lower per packet processing overhead. When using the Fast Prototyping process without DPU and DPK, the new *NetmapNetDevice* should be used instead, as long as the real host does not need to communicate over the same real network interface.

4.6 Discussion

The DPU and DPK performance results show a significant reduction of the per packet processing overhead while using ns-3 emulation, overcoming the performance problem associated with the Fast Prototyping development process. With the DPU and DPK approaches the emulation performance is greatly improved, producing performance results much closer to a real implementation than before. For protocols compatible with the real Linux stack, the DPK approach should be used to obtain the best emulation performance as it is the most efficient and only requires around 1.4% of additional code to be used. The excellent performance results obtained with the DPK approach enables the use of Fast Prototyping in more traffic demanding scenarios or in real nodes with limited processing power. If the protocol is not compatible with the DPK approach, we provide the more generic DPU approach, with around 11% of additional code.

With the improved performance presented by DPU and DPK approaches, the Fast Prototyping process can also be used to emulate multiple nodes in the same host while maintaining realistic performance. Using network namespaces each emulated node becomes isolated from its peers, having its own network configurations and routing tables. This enables hybrid scenarios (combining emulated and real resources in the same experiment) where we can run multiple emulated nodes in the same host machine. In hybrid scenarios a real testbed experiment composed by fast prototyped and/or real nodes can be interacting with a node emulation server. The emulation server may be used for augmenting the complexity and scale of a real testbed by introducing a number of fully emulated nodes able to interact, in real time, with other fast prototyped and real nodes. Being able to maintain the necessary real-time network performance, while running these complex networking scenarios is relevant for the evaluation and validation of networking solutions in large scale; this is usually difficult, if not impossible, to achieve with real testbeds only, due to the resource limitations to build, manage, and operate large scale testbeds.

The use of the Fast Prototyping protocol development process has the following limitations: 1) if we are improving a protocol already having a real implementation, it may be difficult and error-prone to re-implement its data plane and control plane from scratch in ns-3. In that case, a better alternative would be to use the ns-3 DCE which allows to run a real implementation in ns-3, if the source code is available and it is compatible with DCE; 2) the DPK approach only supports protocols having data planes compatible with the Linux kernel. Otherwise, the DPU approach is a valid alternative and its performance, although worse than the DPK, is better than the traditional ns-3 emulation approach running the data plane inside ns-3; 3) in its current version the *Real Routing* module, developed for the DPK approach, only supports proactive L3 protocols.

The integration of the *real routing* module in the main ns-3 distribution will be considered in order to reach more protocol developers and researchers. In addition, we will develop a module to automatically generate the network namespaces and their connections based on the topology of the nodes created in the simulator; at the moment, users have to manually create the namespaces and configure the emulated nodes with their corresponding namespace. An extension of the *Real Routing* module to support reactive L3 protocols is also for future work.

4.7 Summary

In this chapter we focused on improving the performance of ns-3 emulation supporting the Fast Prototyping protocol development process. Since in a well-designed network most of the traffic corresponds to data packets, we focused on moving the data plane operation to outside of ns-3. This solution improves the network nodes efficiency at the cost of having to port the data plane code. We argue this is not a significant problem because the data plane is much simpler than the control plane, resulting in a relatively small amount of code to be ported. We proposed two approaches to implement this solution: the DPU and DPK. The former is more generic and works for any protocol; the latter is more efficient and allows more code reuse but it is only applicable to proactive L3 protocols. In the case of the DPK approach we created a new ns-3 module named *real routing* which implements the generic parts of the proposed approach. This allows researchers to easily extend their ns-3 protocols to use the DPK approach. Additionally, we presented an approach that enables the proposed optimizations to be used even when emulating multiple nodes in the same machine. This is done by using network namespaces, which are environments where network resources are isolated from each other.

In order to validate the proposed solution, we compared the performance of the new approaches against traditional ns-3 emulation, both for single and multiple emulated nodes in the same physical node, running multiple tests with different configurations. The evaluation results showed that when emulating a single ns-3 node, the maximum throughput can be improved by as much as 4.9 times in user space and 19 times in kernel space, while having the RTT lowered by respectively 5.3 and 14 times; when emulating multiple ns-3 nodes, the maximum throughput can be improved, in kernel space, by as much as 23 times while having 15 times lower RTT. These results show that the proposed DPU and DPK approaches allow much better data plane performance when compared against traditional ns-3 emulation.

The amount of code reuse obtained was also characterized. The DPU approach only requires the development of around 11% additional lines of code on top of traditional emulation implementations; for the DPK approach that value can be reduced to 1.4% by using the *real routing* ns-3 module.

As a result of this work, ns-3 users will be able to reuse their simulation code in real networks, now with reduced packet processing overhead, enabling them to benefit from the Fast Prototyping process without the previous performance problems.

Chapter 5

Trace-based ns-3 Simulation Approach for Perpetuating Real-World Experiments

In the previous chapters we focused on improving Experimentation using Simulation. In Chapter 2 we proposed the Fast Prototyping development process, which uses ns-3 emulation as a basis for a shared protocol model implementation between Simulation and Experimentation. In Chapter 3 and Chapter 4 we proposed solutions to improve the functionality and performance of ns-3 emulation.

Pursuing the thesis goal of promoting a Simulation-Experimentation synergy, in this chapter we close the cycle of cooperation between Simulation and Experimentation, now improving Simulation using Experimentation. We present the Trace-based Simulation (TS) approach, the related approaches found in the state of the art to achieve repeatable and reproducible experiments, and the new ns-3 *TraceBasedPropagationLossModel*. Finally, we describe the evaluation of the TS approach carried out using a set of experiments run over three different experiments, considering a comparison with pure simulation and experimentation.

5.1 Problem and Motivation

Although both simulation and experimentation phases are used for the performance evaluation of networking solutions, as depicted in Figure 5.1, in wireless networking research and development we typically depend on experimentation to further evaluate a solution, as simulation is inherently a simplification of the real-world. However, experimentation is limited in aspects where simulation excels, such as repeatability and reproducibility. Real experiments can be difficult to repeat when external phenomena such as noise, interference and multipath create time time-varying channels. Due to this variability, given the same input real experiments can produce very different output results. Real experiments can be reproducible if the methods are documented clearly, although very often they are difficult to reproduce due to testbed operational constrains and availability.

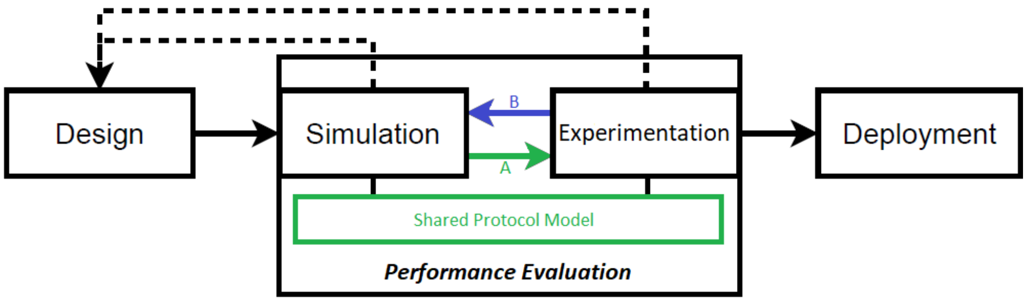


Figure 5.1: Enhanced Simulation-Testbed Synergy via Shared ns-3 Protocol Model and Perpetuation of Experiments.

Our experience in past and current research projects led us to realize that the aforementioned limitations regarding the repeatability and reproducibility of past experiments are clear and more significant in emerging networking scenarios, such as aerial networks (e.g., SUNNY project [14]) and maritime networks (e.g., BLUECOM+ project [15][59]). The related testbeds are becoming increasingly complex and costly to maintain due to the involved difficult logistics operations and reduced duration of the experiments. This has to do with the characteristics of the communications nodes, such as limited battery and fuel autonomy, and the high costs related to renting ships and reserving airbase time slots. On top of this, the radio link quality – herein also referred to as the Signal to Noise Ratio (SNR) at the receiver – is highly unstable and the mobility of the nodes in these scenarios is difficult to reproduce. Furthermore, it is important to realize that in these complex scenarios 1) simulations usually provide very optimistic results, forcing Experimentation to more accurately evaluate a networking solution, and 2) experiments are more difficult to repeat and reproduce, given the temporary and costly nature of many of these testbeds.

A more detailed example of such scenarios was the SUNNY testbed [14]. In the SUNNY testbed there is a static node that acts as a gateway to the ground control operations – the Base Station (BS) – and UAVs that fly in trajectories such as the one presented in Figure 5.2. These UAVs use TV White Spaces (TVWS) wireless links and can fly at altitudes ranging from 100 to 600 m and reach speeds up to 400 km/h. The resulting radio link quality and UAV mobility are very hard to repeat, reproduce, and model due to multiple factors including: 1) the constant movement of the UAV that affects the relative position of the antenna to the BS, due to changes of atmospheric and sea conditions that affect both the signal and UAV mobility; 2) the high noise floor that is often present on these sub-GHz frequencies, due to Electromagnetic Interference (EMI) and/or Radio-Frequency Interference (RFI) caused by close-by hardware and digital TV emissions in the order of kW in neighboring frequency channels.

Contrary to more stable and controlled scenarios, these emerging scenarios may lead to the lack of repeatability and reproducibility of experimental results, and the lack of simulation accuracy, which may impair the fine-tuning and evaluation of a networking solution. This greatly impacts the confidence on the solution's performance results and its scientific relevance. What if we could make any wireless experiment repeatable and reproducible under the same exact condi-

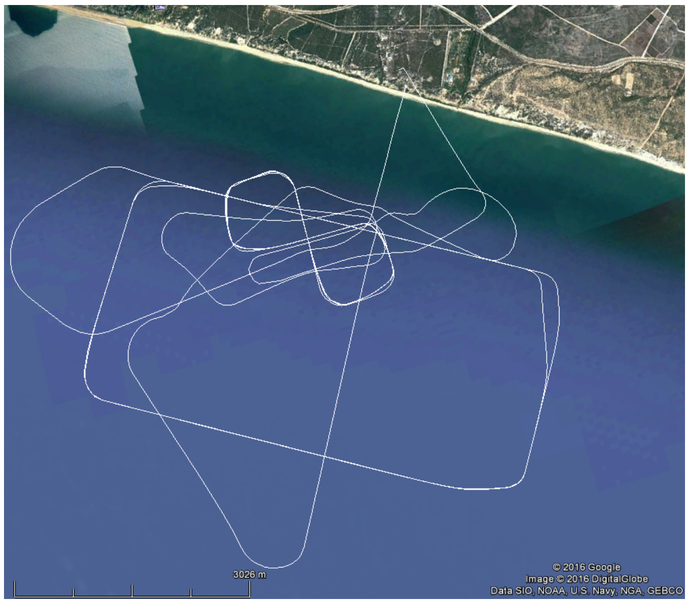

3026 m
© 2016 Google
Image © 2016 DigitalGlobe
Data SIO, NOAA, U.S. Navy, NGA, GEBCO

Figure 5.2: UAV path recorded in real-world experiment performed in the SUNNY project.

tions? What if we could use past execution conditions to test different or fine-tuned networking solutions? What if we could share the same testbed execution conditions among an "infinite" number of users (even for peer reviewing of scientific publications)? What if we could run wireless experiments in faster than real time? These are the questions we address by using the TS approach.

5.2 Related Work

To achieve repeatability and reproducibility of realistic performance evaluation results, different experimentation, simulation, and emulation approaches have been proposed in the state of the art, exploring different alternatives to achieve the same objective. Some of them focus on improving the experimentation, while others focus on improving simulation and emulation. Pure simulation, although repeatable and reproducible, produces results which accuracy highly depends on how well the simulation model (e.g., channel model) represents the real scenario. In emerging networking scenarios, very often the channel model in real-world experiments may not be described by classical stochastic models. The channel model has a high impact on the network performance, so the use of an improper channel model in simulation can make simulations less accurate. This lack of accuracy (by inadaptation of the simulation models) related to pure simulation is a common conclusion from different authors [60][61][62] and may limit the validation of new networking solutions.

The **CONCRETE** [63] tool used in federated testbeds such as Fed4FIRE+ [64] allows to achieve repeatable experiments by analyzing the correlation of the experimental results obtained for different executions and selecting the ones representing the system stable operation (excluding the “outliers”). This needs to happen both for the initial validation, and then also for the reproduction of the experiments; only then the results become comparable. In practice, the “outliers” are equally important and representative of the system operation, as they test the system operation boundaries and often reveal unpredicted phenomena that shall be evaluated too. This solution also depends on the availability of the testbed and requires a mainly stable and repeatable environment. This renders it incompatible with emerging wireless testbeds, where the trajectory of the nodes and the complex multipath phenomena are highly dynamic.

Papadopoulos *et al.* [60] support the use of experimentation in testbeds in order to obtain more realistic results than pure simulation. An important conclusion is that for all the papers with experimental results that they have analyzed, only 16.5% were reproducible. This was mainly due to missing or wrong setup information provided (from possible human error), which made it impossible to replicate the experiments. For emerging networking scenarios, even having the same setup conditions, due to the dynamic nature of these testbeds, the results may be hardly reproducible.

Subsequently, **Papadopoulos *et al.*** [61] improve upon their previous study by presenting an approach to evaluate the repeatability and reproducibility of solutions running in a testbed, and its impact on a proper validation. This approach runs the same solution multiple times along the time. If the results remain reproducible, the solution is validated; otherwise, the results are considered a proof-of-concept needing further validation. Due to the complex logistics operations of emerging network testbeds and their dynamic nature, this approach is not adequate for the emerging networking scenarios.

Bun-Laguna *et al.* [62] explore how reproducible the experimental results are between different testbeds, more specifically, between laboratory testbeds and real-world IoT testbeds. One important conclusion is that laboratory testbeds are suffering from some of the problems found in simulation: due to its stable environment or unrealistic node density, the results often become too optimistic. In this work the authors present a methodology to test if the laboratory results are realistic enough to be considered valid and to allow their reproduction using the same or other testbed. For that purpose, they capture the Packet Delivery Ratio (PDR) and RSSI of the radio links along the time by running a specific firmware. They realize that some scenarios provide highly unstable link connectivity along a day, providing results which may be inconsistent or difficult to reproduce. This is the case with the emerging vehicular networks. They suggest as future work to feed simulators with such captured radio link quality data to reproduce more accurate simulations. This idea is aligned with our goals of improving the simulation accuracy (for reproducing past experiments). It is important to note that our TS approach was originally published in June 2017, one year and a half before [9].

The following related work approaches are focused on the concept of replaying real-world experiments in simulation and emulation.

Mininet-WiFi [65], based on the mininet emulator, is a solution focused on emulation for Software-Defined Wireless Networks, which supports replaying nodes positions and Wi-Fi Received Signal Strength Indication (RSSI). It only supports Emulation Mode and symmetrical Wi-Fi links. According to [66], Mininet-WiFi is lacking the support for Minstrel [67] rate control algorithm (thus not reproducing so accurately the rate adaptation observed in real experiments, which use Minstrel by default), channel contention mechanisms (e.g., CSMA-CA), MAC layer retransmission and interference. Many of these limitations result from the fact that Mininet-WiFi uses *netem* which cannot easily model the lower layers faithfully. Our approach focuses on replaying the physical conditions of a Wi-Fi scenario in ns-3, aims at supporting all the aforementioned missing Mininet-WiFi functionality, while also supporting asymmetric Wi-Fi links and simulation mode.

Regarding simulation, two main approaches can be found: 1) **Packet Based Replay** such as the one proposed in [68], where the authors capture traffic of real networks and try to reproduce the same experimental condition in simulation down to the per packet resolution, including the same throughput and packet rate; 2) **Application Layer Replay**, as the one presented in [69], where the authors try to abstract all low level variables and reproduce the traffic delays and performance bottlenecks experienced in the real network at the application layer. These approaches do not allow to keep improving the solution under evaluation.

In summary, the related work is focused on two approaches 1) improving the experimentation or 2) improving simulation and emulation. The former depends heavily on the testbed being available for multiple runs, and on its stability. The latter is more aligned with our goals. To the best of our knowledge, we did not find a related work solution that allows to replay the conditions of the scenario both in simulation and emulation, thereby enabling to keep improving the solution under evaluation.

5.3 Proposed Trace-based Simulation Approach

Considering the problem described in Section 5.1, the Trace-based Simulation (TS) approach aims to improve the simulation accuracy[1] as a means to achieve repeatability[2] and reproducibility[3] of past real experiments, thus allowing to continue fine-tuning and evaluating a networking solution through more accurate simulations. The TS approach focuses on 1) capturing traces of the execution conditions of an experiment and 2) enable the repetition and reproduction of such conditions using the past traces in ns-3. To attain this goal the TS approach relies on the ns-3 good simulation capabilities from the MAC to the application layer, combining them with the reproduction of traces characterizing relevant physical parameters, such as the variation of the position of the communications nodes and the quality of the radio links over time, to create realistic simulations. Figure 5.3 depicts a high-level comparison between a pure ns-3 simulation and a trace-based ns-3

[1]Simulation accuracy for emerging wireless networking scenarios highly depends on the simulation models being used, especially the channel model. In the context of this dissertation, we use the term "accuracy" to represent how well we can reproduce the performance results of past experiments in simulation using the TS approach. The TS approach can be seen as an intermediate solution to bring realistic channel conditions to simulation, while there is not enough data to create new, or fine-tune current stochastic models.

[2]Ability of the same experimenter/researcher to repeat the experiment in simulation, obtaining comparable results, even though the hardware is, understandably, not the same between a real experiment and a simulation.

[3]Ability of other experimenters/researchers, not the authors of the original experiment, to reproduce accurate performance results close to the original experiment.

simulation (TS approach), showing that the ns-3 simulation remains the same except for the Mobility and the Propagation Loss Models used. By only replaying and reproducing the experimental conditions, the TS approach allows the evaluation of an unlimited number of solutions in the exact same conditions (e.g., an improved or fine-tuned version of a network protocol adapted to the conditions experienced in past experiments).

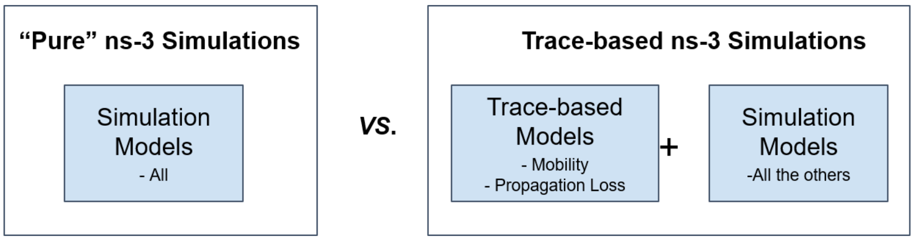


Figure 5.3: High-level comparison between pure ns-3 simulation and a trace-based ns-3 simulation approaches.

The TS approach is illustrated in Figure 5.4. For capturing the execution conditions of an experiment, the TS approach uses every Wi-Fi node participating in a real-world wireless experiment as a probe, automatically capturing traces of radio link quality (RSSI, noise floor), positions of the nodes, and Wi-Fi card configurations such as TX-Power, channel center frequency, channel bandwidth, and the Wi-Fi standard used. Each execution of the same experiment produces unique traces. These traces are gathered and stored by the user in its "Experimental Traces" database or logfiles, where they are identified by the Testbed, Experiment, and specific Execution in which they were produced. A user can now: 1) manage his/her traces by adding, manually, further relevant information to help with the experiment reproduction (e.g., gain of the antennas and cable losses); 2) configure trace-based ns-3 simulation scenarios, according to the real experiments, and use the respective execution traces to reproduce the real experiments in simulations, which can run concurrently using different ns-3 instances; 3) share the traces with the scientific community to foster new research activity and enable repeatability and reproducibility of the experiments.

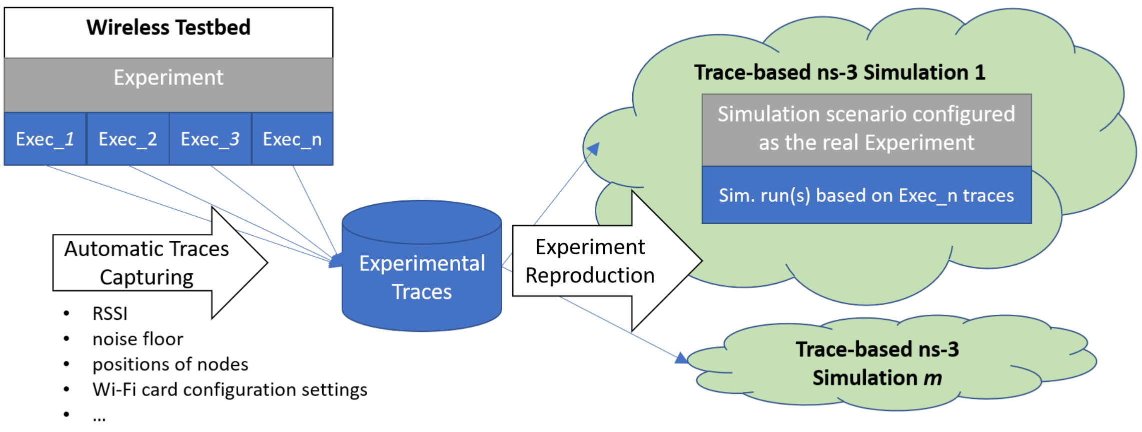

Wireless Testbed
Experiment
Exec_1
Exec_2
Exec_3
Exec_n
Automatic Traces Capturing
RSSI
noise floor
positions of nodes
Wi-Fi card configuration settings
...
Experimental Traces
Experiment Reproduction
Trace-based ns-3 Simulation 1
Simulation scenario configured as the real Experiment
Sim. run(s) based on Exec_n traces
Trace-based ns-3 Simulation m

Figure 5.4: High-level model of the Trace-based Simulation approach.

Our proposed TS approach enables better reproducibility of the experimental channel conditions in the simulator, when compared to the state of the art approaches described in Section 5.2. For experimentation, the TS approach can reproduce each of the real experiments executions, instead of discarding the "outliers" and considering only the stable network operation, and it does this without the need for multiple executions of the same experiment nor access to the real testbed, apart from capturing the traces. For emulation, while the state of the art approaches only support emulation mode, the TS approach, being based on ns-3, supports both emulation and simulation modes, and does this with better realism at modeling the MAC layer operation. For simulation, the state of the art alternatives focus on full reproduction of the real experiment – e.g., same network traffic, delays –, while the TS approach only reproduces the physical characteristics and allows to repeat and reproduce the same or other network solutions in the same conditions. The TS approach can then be used to continue improving a solution under evaluation in simulation environment, which enables: 1) concurrent user access to the same exact experimental setup; 2) running simulations in faster than real time; 3) running multiple simulation instances at the same time, exploring different variants of the solution under evaluation.

In summary, the TS approach relies on recording in real experiments and replaying in ns-3 the physical characteristics that are complex to model in pure ns-3 simulation, due to their highly unstable and unpredictable nature. In the following we identify the physical characteristics to be recorded into variable traces. In addition, we explain how those characteristics can be reproduced in ns-3 and analyze whether ns-3 natively supports the required functionality or new models need to be developed.

5.3.1 Traces to be Collected from Real Experiments

The TS approach assumes the physical conditions experienced by real nodes can be characterized by means of two variables: 1) the node position along the time, e.g., *GPS Coordinates*; 2) the quality of the radio links established with peer nodes, e.g., *RSSI, noise floor*. This information is used to feed the trace-based ns-3 simulation. In what follows, we refer to each of these variables and point out the way they can be collected in a real testbed.

Node GPS Coordinates. In a real-world mobile testbed, the position of the nodes is frequently changing throughout the experiment. As such, its value shall be collected periodically. Using a GPS receiver and *gpsd* in any Linux based Operating System (OS) it is possible to obtain the 3D GPS coordinates of the node (latitude, longitude, altitude) per second, along with the UTC timestamp. This is enough to move the nodes in ns-3, according to the real waypoints.

Radio Link Quality. In a real-world testbed, the radio link quality is constantly changing throughout the experiment, so it shall be collected periodically. The wireless interfaces installed in the real nodes are capable of reporting: 1) the Received Signal Strength Indication (RSSI) (in dBm) for each of the neighboring nodes in radio range; 2) the total Noise Floor power (also in dBm) that includes the white noise, noise figure of the RX circuit, EMI, and RFI. Note that the RSSI results from the combination of multiple variables from the sender, the environment itself, and the receiver. Examples of such variables are the effective TX power, insertion losses, antenna

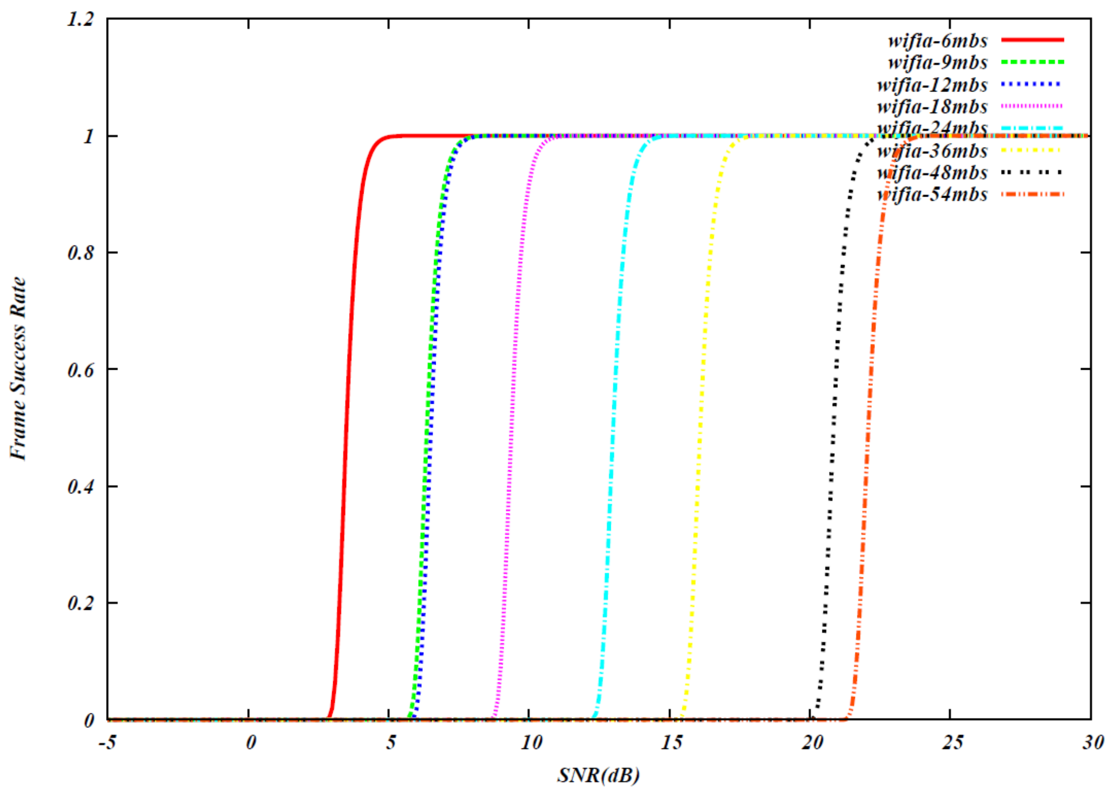


Figure 5.5: Frame success ratio of NIST OFDM model for a frame size of 200 bytes using IEEE 802.11a PHY rates. [1]

cables losses, antenna gains and polarization used (e.g., vertical, horizontal) and path losses that are a function of the propagation medium, obstacles, multipath, and other propagation phenomena. For trace-based simulations, the RSSI can be used as the variable that captures the combination of the multiple variables above mentioned. Based on the Noise Floor and the RSSI, the expected Signal to Noise Ratio (SNR) (in dB) at the receiver can be calculated. It is important to note that this information should be collected at both ends of the radio link, because very often radio links are asymmetric [70]. For instance, UAVs tend to have higher noise floor than Ground Station nodes, due to the proximity of the antennas to other electronic equipment in the UAV and the lack of physical obstacles to block interfering signals originated from distant sources on the ground.

The variable that better represents the radio link quality is the SNR, i.e., how much stronger the signal is being received above the total Noise Floor when considering dBs. There are models, such as the NIST OFDM error model [1], which determine the Frame Error Ratio (FER) – probability of a frame being received with errors and discarded at the receiver – based on the SNR, the frame size, and the PHY rate used. Figure 5.5 shows the frame success ratio (the complement of FER) of NIST OFDM error model for a frame size of 200 bytes using IEEE 802.11a PHY rates. In conclusion, if the real traces of link quality need to be compressed, using the SNR is a good alternative to save both the RSSI and Noise Floor, as it is the metric that will be used by the simulator error models.

Using a *bash* script it is possible to compile the output of the command *"date"* (outputs the time that can be synchronized with *gpsd* using *ntp*), the command *"iwinfo wlan0 info"* (outputs the noise floor) and the command *"iw wlan0 assocli"* (outputs the current RSSI per associated peer) to obtain the radio link quality once per second. If more resolution is needed, a packet capture program such as *horst* can be used to collect the RSSI per frame successfully received. To increase the number of samples captured by *horst*, DATA frames as well as ACK and BEACON control frames can be used. Also, in a multiple access scenario, if a node overhears frames exchanges between its peers (interface in *monitor* mode) those samples can also be used as if they were sent to it.

Other characteristics also need to be collected once per real-world experiment, such as *Channel Center Frequency (MHz)*, *Channel Bandwidth (MHz)*, *TXPower (dBm)*, *Physical Rate (Fixed/Auto)*, and *Wi-Fi standard (a/b/g/n)*, as they remain constant throughout the experiment. These characteristics are essential to make the ns-3 simulation scenario consistent with the real-world experiment.

5.3.2 Reproducing Real Node Positions in ns-3

ns-3 already implements the *MobilityModel*, which defines the node position and enables node movement. The *WaypointMobilityModel* is specifically suitable for reproducing the positions of real nodes in ns-3. The *WaypointMobilityModel* accepts a list of *Waypoints* – each *Waypoint* composed by a *Cartesian Coordinate Vector* – that can be directly derived from GPS coordinates. Between each *Waypoint*, ns-3 moves the node at a constant speed such that the node arrives at the next *Waypoint* in the defined time. We just need to adjust the necessary *offset* between *Simulation Time* (starts at 0 s) and the experiment *WallClockTime*.

5.3.3 Reproducing Radio Link Quality in ns-3

As explained in Section 5.3.1, the metric that represents all the phenomena affecting the radio link quality is the SNR. We argue that the same applies in ns-3 simulation if we use an adequate ns-3 error rate model such as the NistErrorRateModel [1]. For a given combination of SNR at the receiver, frame size, and PHY rate/modulation used the probability of a frame being lost can be calculated using the functions represented in Figure 5.5. Then, ns-3 uses random variables streams to decide whether the frame is dropped due to transmission errors. Data frames or ACKs being dropped result in MAC layer retransmissions, which lowers the throughput and increases link delay. Frames being dropped can also trigger the auto-rate adaptation mechanism (e.g., Minstrel) to lower the PHY rate used, also lowering the throughput and increasing the link delay. Because of the realism of ns-3, we assume that reproducing the same SNR in ns-3 will enable more accurate reproduction of the radio link quality, achieving FER equivalent to the real experiment and closer-to-real link behavior.

To reproduce the same radio link quality in ns-3 we need to look for mechanisms to replay the SNRs observed in the real experiment along the simulation. Instead of changing both the ns-3 RSS and noise floor to recreate the necessary SNR, we found that it is enough to maintaining the ns-3 noise floor constant and only change the ns-3 RSS to achieve the desired SNR. In what follows, we explain how the ns-3 RSS is calculated and how we control the ns-3 RSS during a trace-based simulation.

ns-3 Wi-Fi nodes communicate using a *Channel* abstraction that is a shared medium between them. Two nodes need to be in the same *Channel* to communicate via a wireless link. In order to account for propagation loss, ns-3 supports several *PropagationLossModels* that can be associated to a *Channel*. When a node is sending a frame, ns-3 calculates the Received Signal Strength (RSS) – the equivalent of the RSSI in the real wireless cards – for the destination node, considering the following characteristics: 1) the TX-Power of the source node; 2) the antenna gain of the source node; 3) the path loss calculated by the *PropagationLossModel* associated to that *Channel* taking into account the distance between the nodes, using the function *DoCalcRxPower*; 4) the antenna gain of the destination node. By manipulating these four characteristics in ns-3 we can recreate the desired RSS in simulation. The characteristics 1, 2 and 4 are constant throughout the simulation and the *PropagationLossModel* is used as a way to reproduce the real link quality in simulation, changing the simulation RSS according to the traces of real link quality, which consist of the SNR values collected.

There is an ns-3 *PropagationLossModel* named *FixedRssPropagationLossModel* which allows setting an RSS for a given channel, ignoring the attenuation calculated for the distance between the communicating nodes; the antenna gain of the destination node is considered after the *PropagationLossModel*, so it should be set to 0 dBi when using the *FixedRssPropagationLossModel*, as recommended by the ns-3 manual. Using the *FixedRssPropagationLossModel* allows to change the RSS value during the simulation execution, which is important to reproduce the radio link quality changes along the simulation. The problem is that the RSS is only defined per *Channel*. It cannot be defined per link between each pair of nodes nor per link direction. This means that all the nodes communicating in the same Wi-Fi *Channel* have the same RSS. This is not suitable for trace-based simulations as the link quality between each pair of nodes is usually heterogeneous and asymmetric. In order to properly reproduce the real radio link behavior, the RSS shall depend on 1) the peers involved in the communication and 2) the link direction. For this purpose, a new *PropagationLossModel* was developed for ns-3, called *TraceBasedPropagationLossModel*. The new model is described in Section 5.4.

5.4 *TraceBasedPropagationLossModel*

The new *TraceBasedPropagationLossModel*, characterized by the class diagram shown in Figure 5.6, is a subclass of the existing *PropagationLossModel*. The new class includes the new attributes and operations necessary to implement the functionality required to reproduce asymmetric radio link quality in multiple access scenarios, based on real-world radio link quality traces. Throughout this section we present its implementation details. The ns-3 version 3.28 was used for implementing the model. The source code of the *TraceBasedPropagationLossModel* implementation can be found in [71].

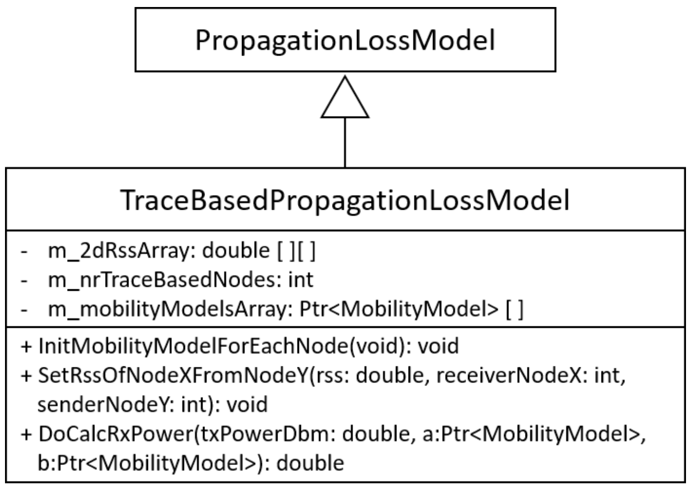

PropagationLossModel
TraceBasedPropagationLossModel
- m_2dRssArray: double [ ][ ]
- m_nrTraceBasedNodes: int
- m_mobilityModelsArray: Ptr<MobilityModel> [ ]
+ InitMobilityModelForEachNode(void): void
+ SetRssOfNodeXFromNodeY(rss: double, receiverNodeX: int, senderNodeY: int): void
+ DoCalcRxPower(txPowerDbm: double, a:Ptr<MobilityModel>, b:Ptr<MobilityModel>): double

Figure 5.6: Class diagram for the proposed *TraceBasedPropagationLossModel*.

The following attributes are included in the proposed *TraceBasedPropagationLossModel*:

- **m_2dRssArray** – a two-dimensional array to store the current RSS (in dBm) of all nodes with respect to all other nodes. Table 5.1 represents an example snapshot of this two-dimensional array for a trace-based ns-3 simulation containing three nodes experiencing asymmetric SNR.
- **m_mobilityModelsArray** – an array to store the pointer of the *Mobility Model Object* belonging to every node. This helps identifying the sender and receiver of each frame to be sent, whenever the *DoCalcRxPower* method is called.
- **m_nrTraceBasedNodes** – an auxiliary variable representing the number of nodes using the *TraceBasedPropagationLossModel*.

The following new methods are featured in the proposed *TraceBasedPropagationLossModel*:

- **InitMobilityModelForEachNode** – method that must be called before the simulation starts, in order to initialize the *m_mobilityModelsArray*.
- **SetRssOfNodeXFromNodeY** – method that updates a given position of the *m_2dRssArray*, allowing the dynamic change of the RSS of the link between Node X and Node Y according to the real experiment traces.

Table 5.1: Example of the 2D RSS (in dBm) array for a trace-based ns-3 simulation containing three nodes, where the lines represent the receiver nodes and the columns represent the sender nodes.

	TxNode0	**TxNode1**	**TxNode2**
RxNode0	-	-60	-70
RxNode1	-65	-	-40
RxNode2	-71	-45	-

- **DoCalcRxPower** – method that calculates RSS of the node receiving a given frame. Each time the method is called, the *Mobility Model* pointers “a” – the sender node – and “b” – the receiver node – are matched with the ones present in *m_mobilityModelsArray* to find the IDs of the corresponding nodes. These IDs are then used to fetch the respective RSS value from the *m_2dRssArray* two-dimensional array.

While in a real node the wireless card reports the RSSI – an indicator that greatly depends on the Auto-Gain Control (AGC) being applied on the RX circuit – ns-3 uses the actual (theoretically) calculated Received Signal Strength (RSS). When the radio signal is too weak to be decoded the AGC increases the gain, while when signal is too strong the AGC reduces the gain (attenuates the signal) to avoid distortion by overwhelming the receiver. This causes a problem on the reported RSSI and noise floor, as they do not account for the current gain, making the reported RSSI lower than expected in scenarios with very strong radio signal, and higher than expected for scenarios when we are about to loose radio link connectivity. This AGC problem that affects the RSSI also affects the Noise Floor reported by the real wireless cards, although the resulting SNR remains realistic. Due to this, when reproducing the radio link quality in ns-3, the proposed *TraceBased-PropagationLossModel* considers the SNR of the real system. We can reproduce such real-world SNR by setting a user defined RSS in ns-3 above the ns-3 noise floor.

The SNR is used in the *ErrorRateModel* to calculate the probability of receiving a frame with an error and dropping it. The *ErrorRateModel* used is the *NistErrorRateModel* which is considered in the ns-3 documentation to be a more realistic model for OFDM modulations than the traditional “YansErrorRateModel”.

5.4.1 Trace-based Simulation Settings

Before running the ns-3 simulation, the user should consider the following settings to better represent the real-world experiment:

- **TX Power End, TX Power Start and TX Gain** – no need to alter as the RSS is set based on traces.
- **RX Gain** – should be kept as *“0”* so that it is not added to the RSS based on the real traces.
- **WiFi Standard, Wi-Fi Mac, Frequency, Channel BW and Remote Station Manager** – should be set with the values assumed in the real world experiment, for instance, *802.11g*, *AdhocWifiMac*, *739 MHz*, *5 MHz* and *ConstantRateWifiManager*.
- **Data Mode and Control Mode** – should be set to the corresponding rates, if constant, for instance, *ErpOfdmRate6Mbps* and *DsssRate1Mbps*.
- **Propagation Delay** – the *ConstantSpeedPropagationDelayModel* should be used.
- **Propagation Loss** – use the new *TraceBasedPropagationLossModel*.

- **Error Rate Model** – the *NistErrorRateModel* should be used.
- **Mobility Model** – use the *ConstantPositionMobilityModel* for fixed nodes, and the *WayPointMobilityModel* for the mobile nodes.
- **2dRssArray** – events should be scheduled throughout the simulation to reflect the correct trace values over the simulation time, and can be accessed by *Config::Set*.

5.5 *TraceBasedPropagationLossModel* Functional Testing

To assess the correct operation of the proposed *TraceBasedPropagationLossModel* two functional tests were performed to cover all fundamental functionalities: 1) **asymmetric point-to-point radio link test**, which tests both the ability to represent asymmetric radio link qualities and the capacity to change the RSS along the simulation time; 2) **asymmetric multiple-access radio link test**, which tests the ability to represent different radio link qualities per each pair of nodes, considering also the direction of the communication.

5.5.1 Asymmetric Point-to-Point Radio Link Test

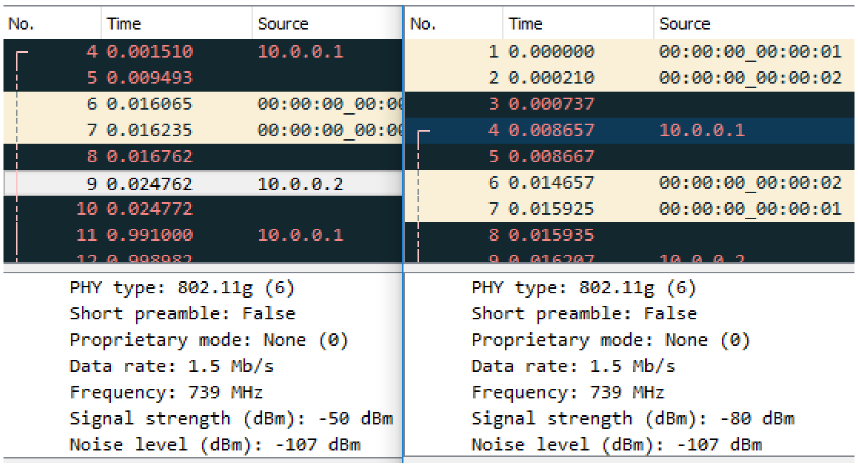
A simple simulation scenario was created containing two nodes that stayed in static positions throughout the simulation. Each node had a Wi-Fi 802.11b/g interface configured in Ad-Hoc mode, and operating at 739 MHz with a channel bandwidth of 5 MHz. Node 0 is set as the GroundNode and Node 1 acts as the MobileNode. The initial simulation RSS values were set to -50 dBm for both nodes. The MobileNode (Node 1) ran an *UdpEchoServerApplication* and the GroundNode (Node 0) ran an *UdpEchoClientApplication*. The GroundNode sends UDP packets with 1400 bytes to the MobileNode at a rate of 1 packet per second. The packet capture option was enabled in both nodes. The RSS values were *Scheduled* to change 3 times for both nodes at different time periods during the same simulation, as follows. This simulation scenario is defined in file “first_scenario.cc”, available for download in [72].

1 Node 0 RSS = -50.0 dBm and Node 1 RSS = -80.0 dBm;

2 Node 0 RSS = -60.0 dBm and Node 1 RSS = -81.0 dBm;

3 Node 0 RSS = -70.0 dBm and Node 1 RSS = -82.0 dBm.

After running the simulation, two output *.pcap* files generated by ns-3 were obtained – one for each node – with *RadioTap* header included. By opening those files in *Wireshark* we got the data represented in Figure 5.7, Figure 5.8, and Figure 5.9. In the three selected time periods, the RSSI values reported for each frame had the same values that were defined by the trace. Therefore, for this test we could conclude that the new *TraceBasedPropagationLossModel* was working as expected.



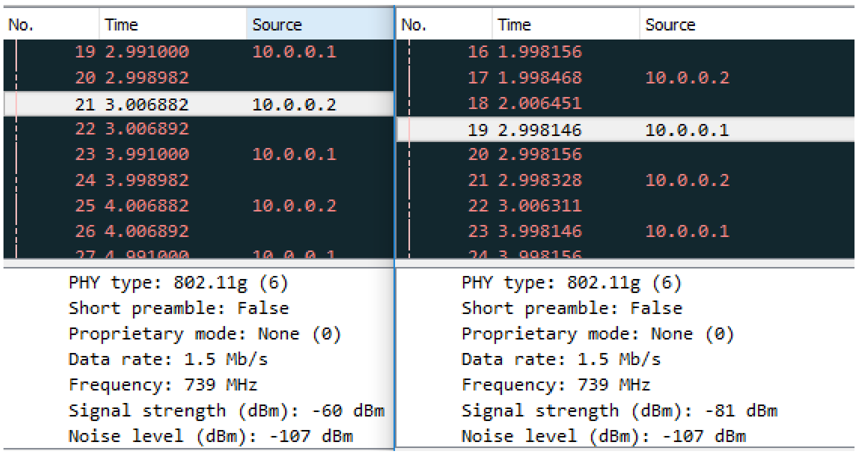
```
No.   Time        Source
  4 0.001510      10.0.0.1
  5 0.009493
  6 0.016065      00:00:00_00:0
  7 0.016235      00:00:00_00:0
  8 0.016762
  9 0.024762      10.0.0.2
 10 0.024772
 11 0.991000      10.0.0.1
 12 0.998982

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -50 dBm
Noise level (dBm): -107 dBm
```

```
No.   Time        Source
  1 0.000000      00:00:00_00:00:01
  2 0.000210      00:00:00_00:00:02
  3 0.000737
  4 0.008657      10.0.0.1
  5 0.008667
  6 0.014657      00:00:00_00:00:02
  7 0.015925      00:00:00_00:00:01
  8 0.015935
  9 0.016207      10.0.0.2

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -80 dBm
Noise level (dBm): -107 dBm
```

Figure 5.7: Wireshark screenshot #1 showing RSSI for Node 0 and Node 1 after the first *Scheduled* update.



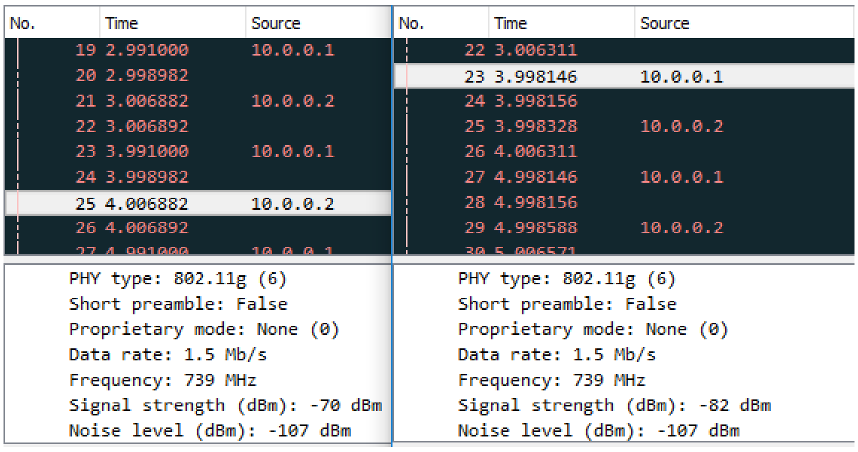
```
No.   Time        Source
 19 2.991000      10.0.0.1
 20 2.998982
 21 3.006882      10.0.0.2
 22 3.006892
 23 3.991000      10.0.0.1
 24 3.998982
 25 4.006882      10.0.0.2
 26 4.006892
 27 4.991000      10.0.0.1

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -60 dBm
Noise level (dBm): -107 dBm
```

```
No.   Time        Source
 16 1.998156
 17 1.998468      10.0.0.2
 18 2.006451
 19 2.998146      10.0.0.1
 20 2.998156
 21 2.998328      10.0.0.2
 22 3.006311
 23 3.998146      10.0.0.1
 24 3.998156

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -81 dBm
Noise level (dBm): -107 dBm
```

Figure 5.8: Wireshark screenshot #2 showing RSSI for Node 0 and Node 1 after the second *Scheduled* update.



```
No.   Time        Source
 19 2.991000      10.0.0.1
 20 2.998982
 21 3.006882      10.0.0.2
 22 3.006892
 23 3.991000      10.0.0.1
 24 3.998982
 25 4.006882      10.0.0.2
 26 4.006892
 27 4.991000      10.0.0.1

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -70 dBm
Noise level (dBm): -107 dBm
```

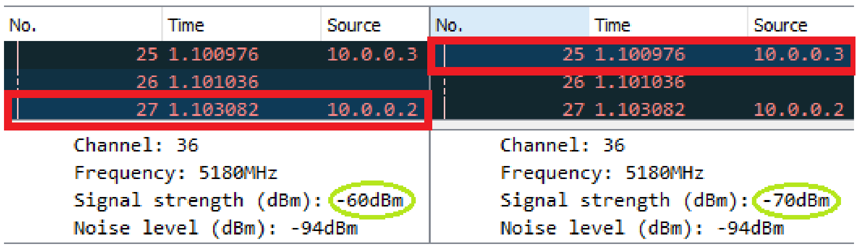
```
No.   Time        Source
 22 3.006311
 23 3.998146      10.0.0.1
 24 3.998156
 25 3.998328      10.0.0.2
 26 4.006311
 27 4.998146      10.0.0.1
 28 4.998156
 29 4.998588      10.0.0.2
 30 5.006571

PHY type: 802.11g (6)
Short preamble: False
Proprietary mode: None (0)
Data rate: 1.5 Mb/s
Frequency: 739 MHz
Signal strength (dBm): -82 dBm
Noise level (dBm): -107 dBm
```

Figure 5.9: Wireshark screenshot #3 showing RSSI for Node 0 and Node 1 after the third *Scheduled* update.

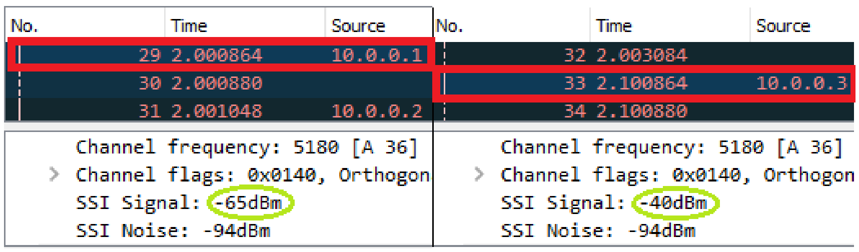
5.5.2 Asymmetric Multiple-Access Radio Link Test

The simulation scenario consisted of three IEEE 802.11a nodes configured in channel 36 (center frequency of 5180 MHz) with 20 MHz of bandwidth. The three nodes were set to fixed throughput in the simulation using the *ConstantPositionMobilityModel*. Node1 ran an *UdpEchoServerApplication*, while Node0 and Node2 ran an *UdpEchoClientApplication*. The clients sent UDP packets with 1400 bytes to Node1 at a rate of 1 packet per second. The packet capture option was enabled in the three nodes. The IP addresses configured for Node0, Node1, and Node2 correspond, sequentially, to the range of 10.0.0.1–10.0.0.3. The RSS for each node was configured based on the example values presented in Table 5.1. The simulation scenario is defined in file "first_scenario_2018.cc" available for download in [71].



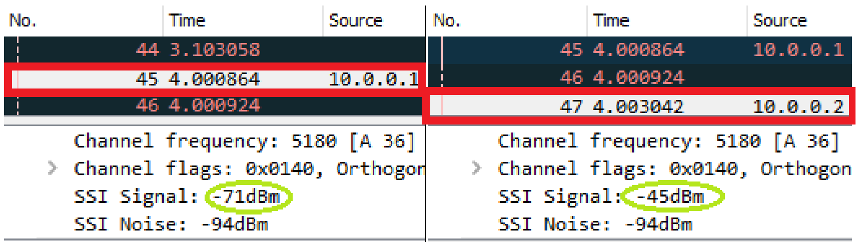
```
No.     Time        Source          No.     Time        Source
25 1.100976         10.0.0.3        25 1.100976         10.0.0.3
26 1.101036                         26 1.101036
27 1.103082         10.0.0.2        27 1.103082         10.0.0.2
  Channel: 36                         Channel: 36
  Frequency: 5180MHz                  Frequency: 5180MHz
  Signal strength (dBm): -60dBm       Signal strength (dBm): -70dBm
  Noise level (dBm): -94dBm           Noise level (dBm): -94dBm
```

Figure 5.10: Wireshark screenshots showing Node0 RSSI perspective of the network.



```
No.     Time        Source          No.     Time        Source
29 2.000864         10.0.0.1        32 2.003084
30 2.000880                         33 2.100864         10.0.0.3
31 2.001048         10.0.0.2        34 2.100880
  Channel frequency: 5180 [A 36]      Channel frequency: 5180 [A 36]
> Channel flags: 0x0140, Orthogon   > Channel flags: 0x0140, Orthogon
  SSI Signal: -65dBm                  SSI Signal: -40dBm
  SSI Noise: -94dBm                   SSI Noise: -94dBm
```

Figure 5.11: Wireshark screenshots showing Node1 RSSI perspective of the network.



```
No.     Time        Source          No.     Time        Source
44 3.103058                         45 4.000864         10.0.0.1
45 4.000864         10.0.0.1        46 4.000924
46 4.000924                         47 4.003042         10.0.0.2
  Channel frequency: 5180 [A 36]      Channel frequency: 5180 [A 36]
> Channel flags: 0x0140, Orthogor   > Channel flags: 0x0140, Orthogon
  SSI Signal: -71dBm                  SSI Signal: -45dBm
  SSI Noise: -94dBm                   SSI Noise: -94dBm
```

Figure 5.12: Wireshark screenshots showing Node2 RSSI perspective of the network.

After running the simulation, we obtained three output *.pcap* files generated by ns-3 with *RadioTap* header included. By opening those files in *Wireshark* we got the data represented in Figure 5.10, Figure 5.11, and Figure 5.12. Each node was able to capture the network traffic, including the unicast flows between its neighbors, and the RSSI values reported by *Wireshark* correspond to the values defined by the trace. This proved the new version of the *TraceBasedPropagationLoss-Model* was working as expected for this scenario, now with support for Multiple Access wireless scenarios.

5.6 Evaluation of the Trace-based ns-3 Simulation Approach

The TS approach was evaluated considering a set of experiments over three testbeds: 1) **SUNNY UAV-Ground Communications Testbed** – over which one 2000 s experiment using a TVWS IEEE 802.11b/g point-to-point link with fixed PHY rate was run; 2) **Isolated Laboratory Testbed** – over which ten 60 s experiments were conducted using a multiple-access Wi-Fi IEEE 802.11a scenario with 3 nodes and auto PHY rate; 3) **Fed4FIRE+ w-iLab.2 Testbed** – over which hundreds of 300 s experiments were conducted considering multiple IEEE 802.11a point-to-point scenarios using both fixed and auto PHY rate. These experiments were replayed via their physical condition traces in trace-based ns-3 simulation. Pure ns-3 simulations were also performed to establish a comparison baseline. In the end, the accuracy of the TS approach was compared against pure ns-3 simulation, considering as reference the experimental results obtained for two network performance metrics: throughput and Round-Trip Time (RTT). The accuracy of the TS and pure simulation approaches was measured using the relative error for the throughput and the absolute error for the RTT network performance metrics. The absolute error is calculated using Equation 5.1 and the relative error is calculated using Equation 5.2, where PM_i is the value of the network performance metric PM obtained for the approaches i (TS and pure simulation) and PM_e is the value of the network performance metric PM obtained in the real experiment. The accuracy gain introduced by the TS approachwith respect to pure simulation approach is calculated using Equation 5.3, where $RelativeError_{TS}$ is the relative error for the TS approach and $RelativeError_{PS}$ is the relative error for the pure simulation approach.

$$AbsoluteError_i = |PM_i - PM_e| \tag{5.1}$$

$$RelativeError_i = \frac{AbsoluteError_i}{PM_e} \times 100(\%) \tag{5.2}$$

$$AccuracyGain = \left(1 - \frac{RelativeError_{TS}}{RelativeError_{PS}}\right) \times 100(\%) \tag{5.3}$$

5.6.1 SUNNY UAV-Ground Communications Testbed

Experimental Setup and Results

In this experiment, which ran along 2000 s, there was a static node on the ground – the SUNNY Base Station (BS) – and one UAV flying. The BS was positioned near shore in a control room 50 m above the sea level. The BS antenna was placed in a 5-meter mast, which assured line-of-sight propagation with clearance of the first Fresnel Zone at the radio frequency used (739 MHz). The

UAV altitudes ranged from near 30 m during launching operations and 100 to 600 m during the flight. Figure 5.2 shows the path of the UAV during the flight, while Figure 5.13 shows the distance between the UAV and the BS along the experiment. Both nodes were transmitting at 25 dBm. In the UAV a 2 dBi dipole antenna was used; the BS had a 6 dBi dipole antenna. The network link was loaded primarily with TCP traffic, but also with some residual UDP traffic generated by UAV on-board equipment communicating with the control systems installed at the BS. During the experiment the UAV position and radio link quality shown in Figure 5.14 for both link directions were collected, in order to reproduce the experiment in ns-3. The throughput was measured along the experiment.

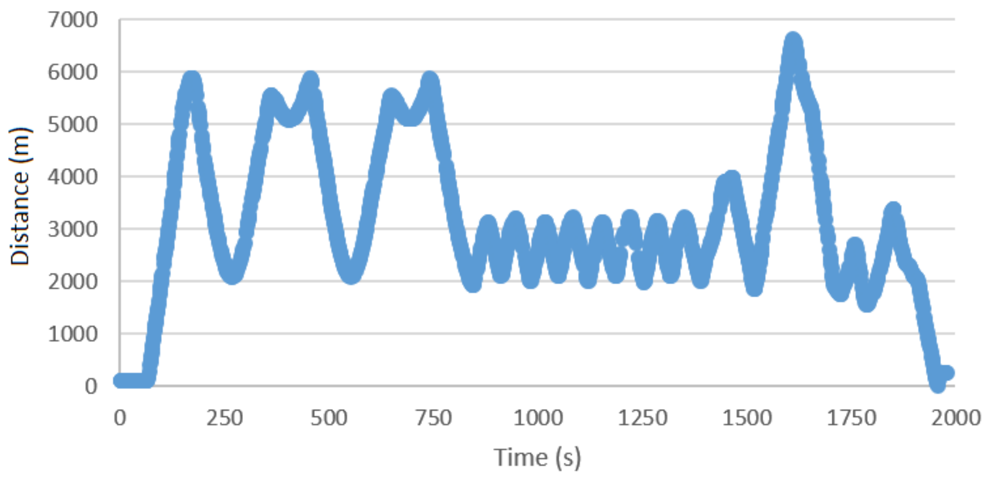


Figure 5.13: Radio link distance between UAV and BS.

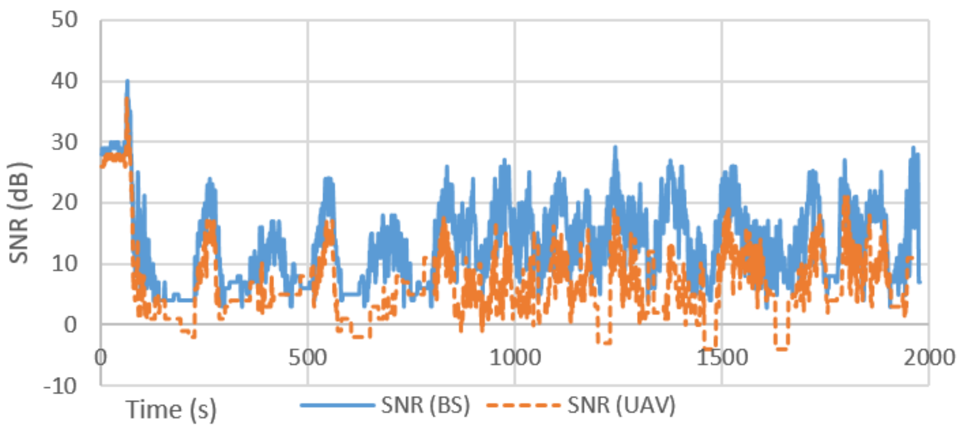


Figure 5.14: SNR recorded in the real-world experiment for the UAV and BS.

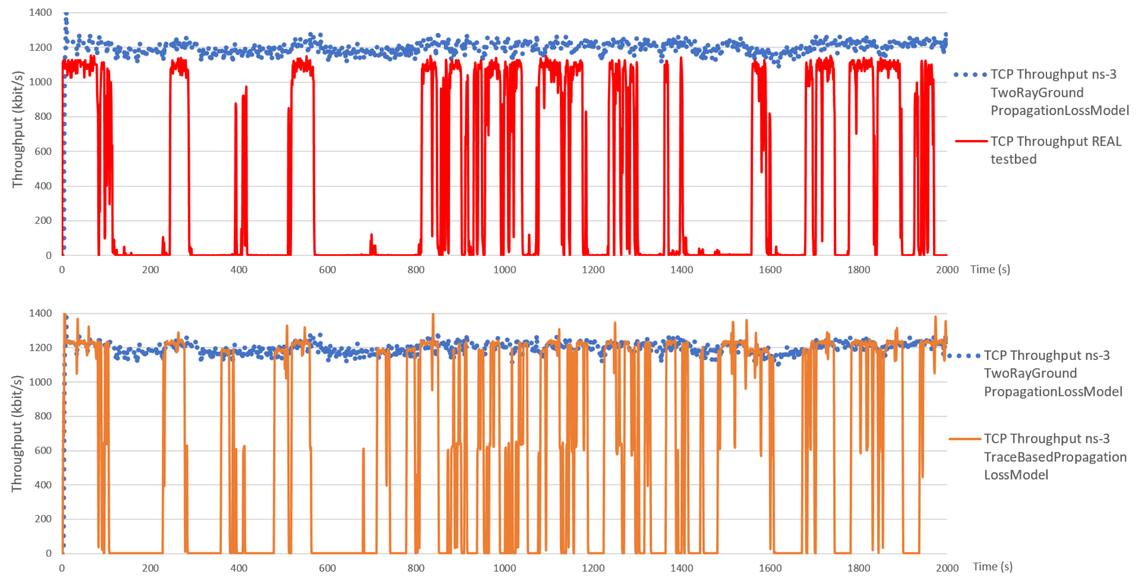


Figure 5.15: Real Throughput vs. 2-Ray Ground Model Simulation vs. Trace-Based Simulation.

Pure and Trace-based Simulation Results

A pure ns-3 simulation was executed considering the node positions, channel frequency, bandwidth, transmission power, physical rate, and Wi-Fi standard used in the real-world experiment. This simulation scenario is described in file "second_scenario.cc" available for download in [72]. The BS node (Node 0) was configured with a *ConstantPositionMobilityModel* using the real coordinate. The UAV node (Node 1) was configured with the *WaypointMobilityModel* for replicating the real positions of the UAV. The *PropagationLossModel* used was the *TwoRayGroundPropagationLossModel* [73]. As in simulation we do not have the real traffic flows, an equivalent TCP traffic flow was generated with the *OnOffApplication* from the UAV to a sink installed in the BS node. The TCP packet Maximum Segment Size (MSS) was set to 1400 bytes – in order to replicate the traffic observed in the real-experiment, where most of the packets were large and only limited by an MTU of 1500 bytes – and the Constant Bit-Rate (CBR) value was set to 2000 kbit/s, which is enough to keep the link fully loaded. For each simulated second, the throughput at the sink application was measured in 5 different simulation runs and the average value was calculated. The results are shown in Figure 5.15. The dotted line corresponds to a moving average using 2 consecutive samples of the simulated results, while the solid line of the upper plot represents the real-world values. As we can see, the results are far from being realistic, as in ns-3 simulation not even once the link was broken between the two nodes. This clearly shows the problem of using pure simulation.

The same ns-3 scenario based on the real UAV positions was executed, but this time using the TS approach with the real-world SNR traces via the proposed *TraceBasedPropagationLossModel*. This simulation scenario is described in file "third_scenario.cc" available for download in [72]. For each simulated second, the throughput at the sink application was measured in 5 different simulation runs and the average value was calculated. The results are shown in Figure 5.15 (bottom plot), using a moving average of 2 consecutive samples. We can see that the TS approach achieves a throughput much closer to the one measured during the real-world experiment. Yet, the maximum real-world throughput does not reach the maximum value measured in simulation. This may be related to the fact that CSMA/CA contention may be triggered by noise, which is abundant in this real-world experiment and reached -68 dBm on the UAV side, lowering the MAC efficiency.

On the other hand, when using the TS approach, in some time-slots the nodes were able to communicate but in the real scenario they were not (e.g., $t = 1500\ s$), which can be related to three other factors: 1) the TS approach only samples the channel with the SNR values of successfully received frames, which acts as an high-pass filter and may falsely represent a more stable channel in simulation than what happened in the real scenario, especially when the radio link connectivity is on the verge of being lost (outage scenario); 2) the TCP Retransmission Timeout (RTO) of the real-world applications could be out of sync with ns-3 RTO at those time-slots. If a TCP ACK repeatedly fails to be delivered to the sender, the amount of time to wait to retransmit it again keeps increasing, and it is difficult to reproduce in ns-3 the same TCP state that occurred in the real-world TCP application; 3) the TCP Data Retries is another configurable parameter that can be different between ns-3 and the real-world TCP applications. In the real experiment, there were time-slots where the TCP connection was closed near the $t = 1500\ s$; it had to be manually restarted later on at $t = 1550\ s$, hence the big difference between the real trace and the trace based simulation plots in that specific time interval. In summary, this is an outage channel, and TCP is very sensitive to this, so it is difficult to exactly reproduce throughput results over a channel in a single run, unless everything is lined up and the implementations at the end points are the same. A lesson learned is that, while continuing to evaluate the TS approach, we should use UDP instead of TCP to avoid a possible out-of-think TCP state machine between the results of real experiments and trace-based simulations.

Even though the TS approach did not attain a perfect match with the real experiment, its realism is much better when compared to the pure ns-3 simulation approach, and it was able to reproduce in simulation the radio link instability problems found in the real experiment.

5.6.2 Isolated Laboratory Testbed

Experimental Setup and Results

The SUNNY testbed was used to evaluate the TS approach in a real-world challenging scenario. However, it was limited to a point-to-point link setup. In order to further evaluate the TS approach for multiple access communications scenarios, we created the laboratory testbed shown in Figure 5.16. The testbed was composed by three static Alix 3D3 based nodes with MikroTik R52 IEEE 802.11a/b/g wireless cards configured to use the IEEE 802.11a standard. These nodes were placed on the floor of a storage room, at a distance of 2 m between each other, as illustrated in Figure 5.16. There were no obstacles between the nodes and the antenna of each node was oriented vertically. The LEDE operating system was running in each node and the clocks were synchronized using the Network Time Protocol (NTP), so that the traces generated by the nodes represented a correct collective snapshot of the radio link qualities at each moment. The IEEE 802.11 cards operated with auto-rate in channel 36 (center frequency of 5180 MHz) with 20 MHz bandwidth, TX Power set to 10 dBm to lower the SNR and trigger auto-rate adaptation, and a 3 dBi dipole antenna (Wi-Fi diversity disabled). There were no other concurrent IEEE 802.11 networks operating in overlapping frequency channels.

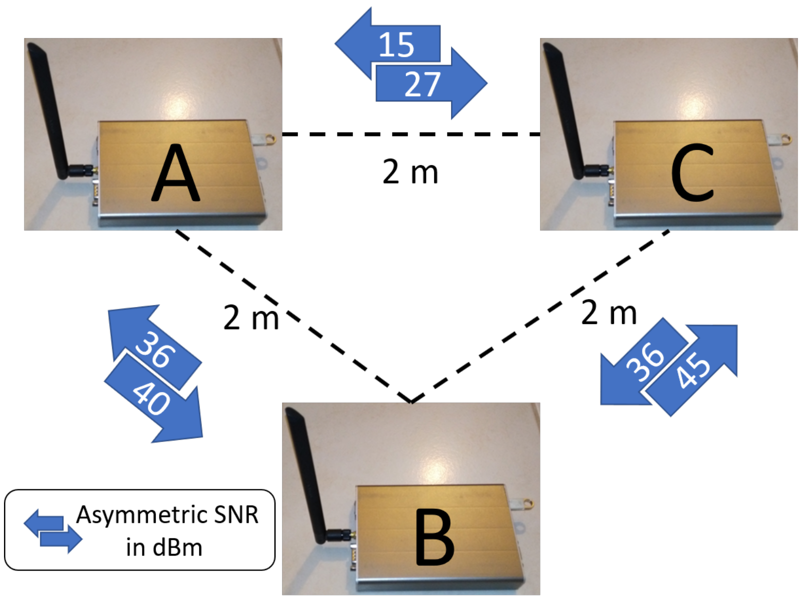


Figure 5.16: Diagram of the real testbed used for the wireless experiments, with the average SNR measured per link direction.

Based on our hands-on experience with testbeds and different IEEE 802.11 cards, we know that there are differences of effective TX Power and RX sensitivity between different wireless cards, even if the same model is used. This becomes even more evident when running experiments with extensively used hardware. Also, the hardware components wear can affect more the TX function than the RX function and vice-versa. This aspect, by itself, represents a source of possible link asymmetry alongside with different noise and interference exposure for each node. In practice, this is one further reason why experimental results are difficult to reproduce in simulation. Recognizing this fact we selected, on purpose, IEEE 802.11 cards from the same model (MikroTik R52) but subjected to different wear levels. Figure 5.16 presents the average SNR values measured throughout all the experiments, clearly showing that: 1) NodeA has less sensitivity on its RX circuit; 2) NodeC has TX Power lower than expected; 3) NodeB is the card with better TX/RX performance. In conclusion, although at first sight this testbed scenario would be creating optimal network conditions, the aforementioned characteristics create a richer scenario to test the TS approach in a multiple access scenario with radio link heterogeneity and asymmetry. Also, knowing these characteristics beforehand, leads us to expect the worst network performance in flows from C to A.

In all experiments we used *iperf2* to generate UDP flows with an application rate of 54 Mbit/s, which exceeds the maximum possible effective rate of an IEEE 802.11a link, usually between 28 and 32 Mbit/s. The objective was to perform all the experiments using the full channel capacity, so that we could latter compare the same limits in simulation. In each experiment, we generated the necessary UDP flows during 60 seconds and recorded the obtained average UDP throughput per second, alongside with traces containing the average values of SNR per second for each possible bidirectional link. The closer the UDP throughput obtained using the TS approach is to the one measured in the real experiment, the better the *TraceBasedPropagationLossModel* and ns-3 are reproducing the real system. Since the three nodes are in wireless range, the first six experiments consisted in each node generating one-hop traffic flows to each of its two neighbors. Table 5.2 presents the details of each flow tested per experiment, together with the obtained average UDP throughput. Because of the lower radio link SNR, the worst performance was measured in Exp.#2 and Exp.#5 – communication between NodeA and NodeB – with Exp.#5 getting particularly low throughput (5.9 Mbit/s), when compared to the 21.7 Mbit/s measured in Exp.#1, for example.

Then, we ran experiments for testing the multiple access scenario, where each node acted as a sink for simultaneous flows originating from its two neighbors. The results obtained for the three additional experiments are presented in Table 5.3. As expected by the lower SNR between NodeA and NodeC, Exp.#7 and Exp.#9 were the ones exhibiting the most asymmetric UDP average throughput per flow, achieving respectively 4.1 Mbit/s and 9 Mbit/s.

Pure and Trace-based Simulation Results

The ns-3 scenarios "expX_scenario_2018_trace.cc" were coded to replicate the experimental setup using the TS approach, with "X" representing the number of the experiment. The average real SNR experimentally collected each second was used as a basis; these ".cc" files, as well as the ".csv" files with SNR traces are available in [71]. In ns-3, we generated the UDP traffic using the ns-3 *OnOffApplication* traffic generator and measured the average UDP throughput for the same exact set of experiments. The same was considered for pure simulation in order to compare what would be the results assuming the *FriisPropagationLossModel* [74] and stable and symmetric link qualities; the pure simulation scenarios "expX_scenario_2018_PURE_SIM.cc" are available in [71]. Table 5.2 and Table 5.3 show the average UDP throughput results obtained as well as the relative error with respect to the experimental results.

The maximum throughput achieved in simulation is higher than in the real experiments. This may be related to limitations on the performance of the real hardware that are not accounted in ns-3, such as the time of packet buffer copy and processing operations that in reality take time and contribute to reduce the throughput. This is more evident for IEEE 802.11a as it does not perform frame aggregation. Any processing inefficiency has impact, especially in high SNR scenarios where any small amount of time not sending packets at the highest rate possible – 54 Mbit/s – increased the gap between simulation and experimental results. Because of this limitation, every experiment for which the flows are transmitted through high SNR links do not bring added value to this comparison. The TS approach is only able to lower the relative error with respect

Table 5.2: Average UDP throughput results obtained for each individual link, one flow direction at each time, in the Real Experiment, the Trace-Based Simulation, and the Pure Simulation, including the relative error with respect to the Real Experiment results.

		Average UDP Throughput (Mbit/s)			***Relative Error***	
Exp.#	**Flow**	**Real Exp.**	**Trace Sim.**	**Pure Sim.**	**Trace Sim.**	**Pure Sim.**
1	A->B	21.7	28.5	28.5	31%	31%
2	A->C	19.2	22.6	28.4	**18%**	**48%**
3	B->A	21.8	28.4	28.4	30%	30%
4	B->C	21.1	28.3	28.3	34%	34%
5	C->A	5.9	9.9	28.3	**68%**	**380%**
6	C->B	22.8	28.2	28.2	24%	24%
					Avg. all flows	
					34%	**91%**

Table 5.3: Average UDP throughput results obtained for two simultaneous flows from different senders to each sink node, in the Real Experiment, the Trace-Based Simulation and the Pure Simulation, including the relative error with respect to the Real Experiment results.

		Average UDP Throughput (Mbit/s)			***Relative Error***	
Exp.#	**Flow**	**Real Exp.**	**Trace Sim.**	**Pure Sim.**	**Trace Sim.**	**Pure Sim.**
7 Sink A	B->A	11.3	12.2	14.3	**8%**	**27%**
	C->A	4.1	5.7	14.1	**39%**	**244%**
	Total	15.4	17.9	28.4	16%	84%
8 Sink B	A->B	11.9	14.3	14.3	20%	20%
	C->B	14.1	14.0	14.0	1%	1%
	Total	26.0	28.3	28.3	9%	9%
9 Sink C	A->C	9.0	4.0	14.2	**56%**	**58%**
	B->C	20.8	19.3	13.9	**7%**	**33%**
	Total	29.8	23.3	28.1	22%	6%
					Avg. all individual flows	
					22%	**64%**

to pure simulation when the real SNR decreases to a point that it triggers the auto-rate adaptation mechanism, caused by frames being lost and retransmitted. Due to this fact, we focus our analysis on the results for the radio links that do not max out the IEEE 802.11a throughput performance.

Considering the radio links with lower quality, the benefit of the TS approach becomes apparent, as demonstrated by the results highlighted in bold in Table 5.2 and Table 5.3. For instance: in Exp.#2 the relative error drops from 48% to 18%, representing an accuracy gain (cf. Equation 5.3) of 63%; in Exp.#5 the relative error drops from 380% to 68%, producing an accuracy gain of 82%. In every case, considering the individual flows of all nine experiments, the TS approach reduced the relative error. On average, from 91% to 34% (gain of 63%), for the single flow scenarios, and from 64% to 22% (gain of 66%), for the multiple flows scenarios, when compared to pure simulation.

Trace-Based Simulation Results using High SNR Sampling Rate

There were cases, such as Exp.#5, where the relative error was still considerably high, although much lower than in pure simulation. That could be related to the fact that we were using real SNR averages per second, maintaining that value constant during each simulated second. We could easily add a random component to make the SNR less stable by using, for example, a Normal, Rayleigh, or Rician distribution. Still, we would be simulating again and not using the actual traces to reproduce reality. As such, we decided to repeat the experiment with the biggest relative error when using the TS approach – Exp.#5 –, and assessed whether the relative error could be reduced by increasing the real SNR sampling rate.

Table 5.4: Average UDP throughput results obtained when rerunning *Exp.#5* and considering the Trace-Based Simulation - High SNR Sampling Rate (HSSR), the Trace-Based Simulation, and the Pure Simulation, including the relative error with respect to the Real Experiment results.

		Average UDP Throughput (Mbit/s)			
Exp.#	**Flow**	**Real Exp.**	**Trace Sim. HSSR**	**Trace Sim.**	**Pure Sim.**
5 (second run)	C->A	5.4	5.3	8.3	28.2
			Relative Error		
			1%	54%	426%

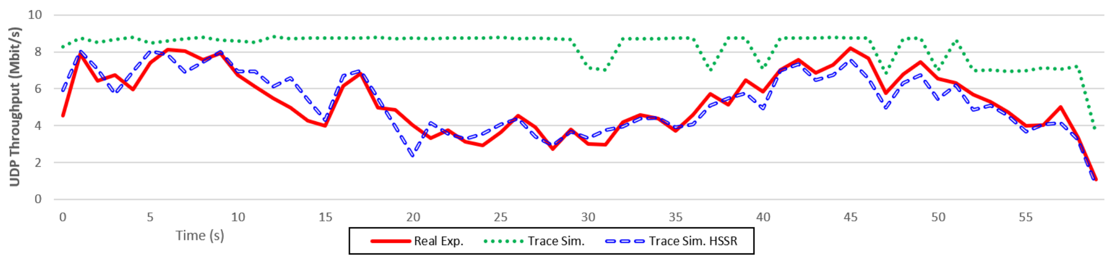


Figure 5.17: Comparison of UDP Throughput per second measured during 1) the repetition of real Exp.#5, 2) the corresponding Trace-Based Simulation based on traces containing the real SNR average per second, and 3) the corresponding Trace-Based Simulation based on traces containing the real SNR with a per packet resolution.

The results after rerunning Exp.#5 with a higher real SNR sampling rate (once per packet received) are presented in Table 5.4. The table shows the average UDP throughput and the relative error for the three simulation-based alternatives to reproduce the real experiment. In this case, we can see the relative error dropped from 426%, for pure simulation, to 54%, for trace-based simulation, resulting in a gain of 87%. But, using the higher number of SNR samples we were able to reduce even further the relative error to 1% only, now with a gain nearing 100%. Figure 5.17 shows a plot comparing the instantaneous throughput per second for the trace-based simulation, the trace-based simulation with high SNR sampling rate, and the real experiment. The pure simulation throughput was not plotted as it remained almost constant at around 28 Mbit/s and it would reduce the throughput resolution needed to better compare the other results. We can see that trace-based simulation results using per packet SNR samples was significantly more accurate, almost overlapping with the real UDP throughput results. This proves the potential of the TS approach. Still, the disadvantages are: 1) more storage space is required to save per packet SNR samples from real experiments; 2) the simulation time may increase due to the higher number of events to update the SNR throughout the simulation. Nevertheless, we argue that it is computationally lighter to just read and update SNR values that are already known than using simulation models to calculate new values. Thus, we believe that even with the higher rate of SNR updates the running time should remain lower than in pure simulation.

5.6.3 Fed4FIRE+ w-iLab.2 Testbed

The SIMBED project was proposed in the context of this thesis. It was approved in the Fed4FIRE+ Open Call 3 for medium and large experiments. The SIMBED project, as explained in Section 1.7, aims to carry out an extensive TS approach evaluation by benefiting from the high quality and large number of resources provided by Fed4FIRE+ community testbeds. For that purpose, SIMBED is running a set of wireless experiments on top of Fed4FIRE+ wireless testbeds in order to demonstrate it is possible to use ns-3 together with the TS approach as a means for repeating and reproducing wireless experiments. By the time of writing this thesis, SIMBED project is not finished. Nonetheless, a large number of SIMBED experiments were already completed using the w-iLab.2 testbed, an indoor and isolated testbed located in Belgium. The relevant evaluation results obtained for the TS approach are presented herein.

Experimental Setup

To extensively validate the TS approach we ran a large number Wi-Fi experiments. The experiments considered Wi-Fi point-to-point links established between a static transmitter and a static receiver. The Wi-Fi links were tested for auto rate mode (using Minstrel) and fixed rate mode. Because the SNR is the metric that better represents the impact of the testbed physical conditions on the radio link quality – and the resulting throughput and RTT – we have run experiments to measure the network performance of Wi-Fi point-to-point links ranging from very low SNR to very high SNR at the receiver. This allowed us to test the TS approach from the lower to the upper performance limits of a Wi-Fi point-to-point link.

For these experiments we reserved and used Wi-Fi nodes from the w-iLab.2 testbed. Figure 5.18 presents a map[4] of the w-iLab.2 testbed showing its available resources and their relative position. For our experiments, we have used a selection of Zotac (light blue colored) nodes. The Zotac nodes are placed in a grid pattern, with a column width of 6 m and row height of 3.6 m. These nodes have an Intel Atom D525 CPU (2 cores, 1.8 GHz), 4 GB of RAM and 2x IEEE 802.11abgn Sparklan Wi-Fi interfaces with AR9280 Atheros chipset. Attached to the AR9280

[4]Available on the following website: https://inventory.wilab2.ilabt.iminds.be/?viewMode=inventory [Accessed: 28th January 2019]

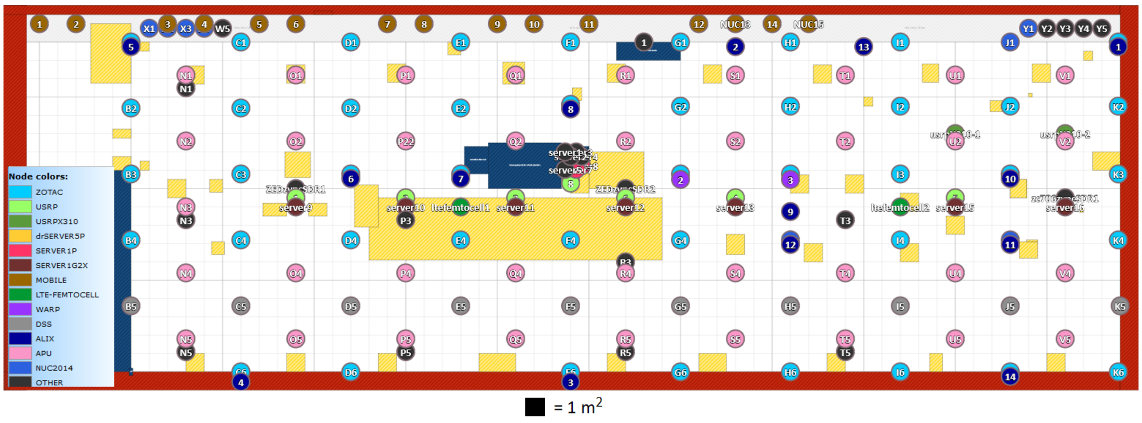


Figure 5.18: Map of the available resources, and their location, in w-iLab.2 testbed.

interfaces the nodes have 3 dBi gain dipole antennas with 10 dB attenuators inline – adding a total of 20 dB to the path loss – to avoid overwhelming the receivers and limit the testbed interference. The TX power of these interfaces can be controlled from 0 to 17 dBm; this was very important to use the same set of nodes to create 18 different experiments when it comes to the SNR levels at the receiver. The OS used was Ubuntu 14.04 LTS x64 with patched ath9k Wi-Fi driver to allow setting up the interfaces in ad-hoc mode using the 5 GHz frequency band.

For each experiment executed we collected the following data per node: 1) traces of real SNR at the receiver for each received frame, organized by peer node and time-referenced with a microsecond resolution – HSSR was used for all experiments due to its improved accuracy, as shown in Section 5.6.2; 2) the position of the nodes, once per experiment as the nodes are static; 3) network performance metrics measured for each link – average throughput and RTT. The traces of real SNR and node positions are used to feed the trace-based ns-3 simulations. The network performance metrics are used to evaluate the TS approach against pure ns-3 simulation.

The TS approach focuses on reproducing the experimental physical conditions. However, the network performance can also be influenced by the number of queues and respective sizes, and the related traffic control and queue management mechanisms used in the communication nodes. If different between the real experiment and the ns-3 simulation we may end up with very different results between simulation and experimentation; this is currently the case if we compare ns-3 simulations with experiments ran over Linux. For this reason, we focused on measuring the performance metrics in the following conditions:

- **Throughput.** This metric is measured at the receiver, considering the sender is generating traffic with offered network load above link capacity. This assures that there are always packets queued waiting to be sent. Only one flow is generated at a time.

- **RTT.** This metric is measured without concurrent network traffic. This assures the queues are always empty when an ICMP request and reply is generated, as each subsequent ICMP request is only generated after a reply or a timeout. This assures we are only considering the delays related to the access and (re)transmissions over the Wi-Fi half-duplex multiple access wireless medium, and discarding any queuing related delays.

In order to carry out the aforementioned experiments over w-iLab.2 we used the following methodology:

1. **Nodes reservation:** We started by placing a reservation for 4 consecutive Zotac nodes in the same grid row. The leftmost node was the *Master*; then we had *ClientA*, *ClientB* and *ClientC* nodes respectively at 6, 12 and 18 m from the *Master* node.

2. **Nodes startup:** We selected our custom Ubuntu 14.04 LTS x64 OS image to boot in all nodes, containing our experimentation scripts and patched ath9k driver.

3. **Nodes configuration:** After all the nodes finished booting, we needed to make the following configurations:

    (a) **NTP client:** We configured the NTP client so that all the nodes became clock synchronized. This was very important so that the collected traces and network performance metrics were correctly synchronized between the nodes participating in the same experiment. Only in this way we can later repeat and reproduce a close-to-real scenario in ns-3 using the TS approach, and then correctly compare the obtained performance metrics.

    (b) **Wi-Fi standard:** We used IEEE 802.11a operating in the 5 GHz frequency band as we are focused on testing the TS approach for OFDM modulation operating in Single-Input Single-Output (SISO) scenarios. Using regular ACK control frames, even if we generate UDP flows only in one direction, we get an ACK control frame in the opposite direction per DATA frame sent, which results in sampling both directions of the radio link with higher resolution.

    (c) **Wi-Fi operation mode:** ad-hoc.

    (d) **Channel bandwidth:** 20 MHz.

    (e) **Channel center frequency:** 5220 MHz, corresponding to channel 44. For each reservation we performed a channel survey and avoided concurrent Wi-Fi networks. Channel 44 remained free during all the experiments performed in w-iLab.2.

    (f) **PHY rate:** Depending if we were testing the auto or fixed PHY rate mode, we configured the nodes to use auto PHY rate or fixed the PHY rate to 6, 9, 12, 18, 24, 36, 48 or 54 Mbit/s, according to the experiment we were running.

    (g) **TX power:** The tx power varied according to the experiment. This setting accepts a range from 0 to 17 dBm in 1 dBm steps. We found that a link distance of 18 m associated to a TX power of 0 dBm was sufficient to create low SNR scenarios where the link was becoming intermittent, hence the use of only 3 *Client* nodes up to 18 m.

4. **Run batch of auto PHY rate experiments:** We started by running a batch of auto PHY rate experiments (up to a total of 216 unique experiments – 3 different clients, 18 TX power levels, 4 different experiments between each *Client* and the *Master* node). We configured all the 4 nodes with the same TX power. For each TX power value we ran the following experiments, each one of them with the duration of 300 s and considering the communication between the *Master* node and a single *Client*, going through the *Client* nodes one at a time:

    (a) **Idle network link between the *Master* and the *Client*:** Without any concurrent traffic flow, ICMP echo requests were issued from the *Master* node to the *Client* node. The *ping* application was used and configured to generate 10 requests per second with a packet size of 1472 bytes (frames of 1500 bytes including protocol headers). The objective was to measure the RTT.

    (b) **Unidirectional UDP flow from *Client* to *Master*:** The *iperf3* application was used to generate a UDP flow from the *Client* node to the *Master* node with offered load (54 Mbit/s) above link capacity (28-30 Mbit/s).

    (c) **Unidirectional UDP flow from *Master* to *Client*:** Similar to the previous experiment, except the UDP flow was generated in the opposite direction. This helped to identify asymmetric radio link qualities and respective network performance results.

    (d) **Bidirectional UDP flows between *Master* and *Client*:** Similar to the previous two experiments, except this time the UDP flows were generated at the same time, in each opposite direction.

5. **Calculate the most common PHY rates:** Analyzing the past auto PHY rate experiments, we calculated the most common PHY rates of the received DATA frames (Relative Frequency > 0.2) for each pair of nodes and for each TX power value.

6. **Repeat the past experiments with fixed PHY rates.** There were usually one or two PHY rates selected as the most common per past auto PHY rate experiment. In this fixed PHY rate scenario, we reran the same past experiments involving the same physical nodes, but now using the most common PHY rates previously calculated (up to 2x 216 unique experiments).

After running the first batches of experiments we realized that we were getting more experiments for higher SNR values than for lower SNR values. To overcome this problem of very different sample sizes along the SNR range, and avoid biasing the TS approach evaluation results, we ran batches of experiments only focusing on testing the network performance between *Master*, *Client2* and *Client3* nodes with configured TX power below 5 dBm.

Trace-based Simulation Results

To assess the gains obtained by using the TS approach, all the real experiments were reproduced via trace-based ns-3 simulations feeding the *TraceBasedPropagationLossModel* with the real traces of SNR. The same network performance metrics were measured in ns-3 (throughput and RTT). To have a baseline of comparison, we reran the equivalent simulations using the pure ns-3 simulation approach, where we tested four different path loss models trying to find the most appropriate for w-iLab.2.

- ***FriisPropagationLossModel* [74]:** As the nodes have radio line-of-sight and the direct radio ray is expected to be dominant due to the low distance and the absence of obstacles in between the nodes. This path loss model is deterministic, so the SNR remains constant throughout the duration of the simulation.
- ***LogDistancePropagationLossModel* [75] (γ = 2.0) plus Rician fast fading [76]:** With γ = 2.0 the *LogDistancePropagationLossModel* has the same output as the *FriisPropagation-LossModel*. The only difference for the previous path loss model is the addition of the Rician fast fading obtained in ns-3 by configuring the *NakagamiPropagationLossModel* [77] with m = 1.25.
- ***LogDistancePropagationLossModel* (γ = 1.7) plus Rician fast fading:** As the w.iLab.2 is an indoor testbed, has radio line-of-sight, and may have a strong multipath component that adds substantially to the direct ray, we decided to test a reduced path loss exponent γ.
- ***LogDistancePropagationLossModel* (γ = 2.5) plus Rician fast fading:** To complement the two previous path loss model options, we considered a higher path loss exponent γ.

Apart from the ns-3 *PropagationLossModel* used, all other simulation parameters are the same for both simulation approaches, except for the “RF gain” which is set to 0 dB in the case of the trace-based simulations and set to -7 dB for the pure simulations. This is to adjust the path loss calculation considering the 3 dBi gain from the antennas and the 10 dB attenuation from each inline attenuator (3 - 10 = -7). Note that the “RF gain” is considered on both ends of the communication, so the resulting “RF gain” becomes -14 dB, which are added to the path loss calculation. In ns-3, we generated the UDP traffic using the ns-3 *OnOffApplication* traffic generator and the RTT measurements were performed using the ns-3 *V4Ping* Internet application.

The most adequate resolution to compare the network performance results between the TS approach and the pure simulation approach depends on the expected coherence time of the radio channel, which is affected by the channel frequency, how dynamic the environment is, and the velocity of the nodes. Because all the w-iLab.2 experiments ran on static nodes operating in a static environment, we used a resolution of 1 second.

For each real second of a given experiment, and their corresponding trace-based and pure simulated counterparts, we compared: 1) the average throughput per second (kbit/s); 2) the median of the RTT samples per second (ms). By comparing these two network performance metrics for the exact same time interval considering the real experiment, trace-based simulation, and pure simulation we calculated the relative error for the trace-based and pure simulation approaches using Equation 5.2. We found this method to be the most adequate to calculate the relative error, as we are trying to reproduce real Wi-Fi experiments, which are influenced by phenomena that cause SNR instability and considerable variations along the time.

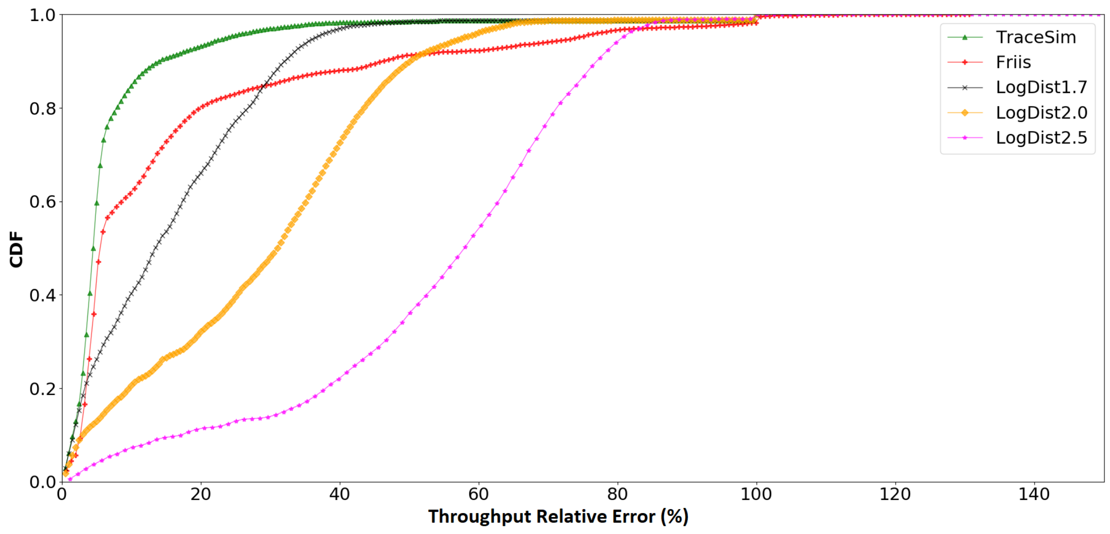


Figure 5.19: CDFs of the throughput relative error when comparing the trace-based (TraceSim) and pure simulations to the corresponding real experiments for auto PHY rate mode.

Figure 5.19 shows the CDF of the throughput relative error for the trace-based ns-3 simulation and pure ns-3 simulation when the Wi-Fi point-to-point link is running in auto PHY rate mode. For computing the CDFs, all the samples with real throughput equal to 0 kbit/s were discarded to filter the initial experiment seconds where *iperf3* did not yet start sending traffic or the cases where *iperf3* experienced some malfunction and stopped sending traffic for the rest of the real experiment. In a total of 31058 samples, the filtered samples represent 1.2%. Table 5.5 summarizes the relevant values extracted from the CDF plot and presents the 90th percentile, 50th percentile (median), and the average throughput relative error. Analyzing the pure simulation results we can observe that between the four pure simulation options the Friis model and the LogDistance model with γ=1.7 plus Rician fast fading (LogDist1.7) are the ones that better approach the real experiment results. This shows that although the Friis path loss model does not consider fast fading it is the one that, on average, more closely matches the real experiment results, which was expected considering the isolated and very stable w-iLab.2 testbed. Analyzing the trace-based simulation results, we can observe that it is the one that more closely reproduces the real experiment: for the 90th percentile,

Table 5.5: Throughput relative error when comparing the trace-based (TraceSim) and pure simulations to the corresponding real experiments for auto PHY rate mode.

	Throughput Relative Error (Auto PHY Rate) (%)		
	90th Perc.	**50th Perc. (Median)**	**Average**
TraceSim	14	5	**7**
Friis	46	6	16
LogDist2.0	50	31	29
LogDist1.7	32	13	16
LogDist2.5	77	58	54

Table 5.6: Throughput relative error when comparing the trace-based (TraceSim) and pure simulations to the corresponding real experiments for fixed PHY rate mode.

	Throughput Relative Error (Fixed PHY Rate) (%)		
	90th Perc.	**50th Perc. (Median)**	**Average**
TraceSim	26	5	12
Friis	69	5	19
LogDist2.0	73	29	34
LogDist1.7	31	11	15
LogDist2.5	99	85	72

the TS approach presents a gain (c.f. Equation 5.3) of 70% over Friis and 56% over LogDist1.7; for the median, the TS approach introduces a gain of 18% over Friis and 66% over LogDist1.7; finally, on average, using the TS approach results in a gain of 53% over Friis and 54% over LogDist1.7. In conclusion, the results show that the TS approach is considerably better at reproducing a closer-to-real throughput than pure simulation approach, showing also that the ns-3 *NistErrorRateModel* and the ns-3 *minstrel* auto-rate algorithm are modeling very well the real system. We found that the gains obtained in the relative error by replicating the real SNR resulted not only from better representing the SNR changes along the time but also from better reproducing radio link quality asymmetry; in many cases we found non-negligible differences between each direction of the same link which affect the resulting throughput. The asymmetry obtained in this testbed may be related to slight differences between the nodes transceivers hardware – even though they are identical –, resulting in different effective RF losses both in TX and RX operations.

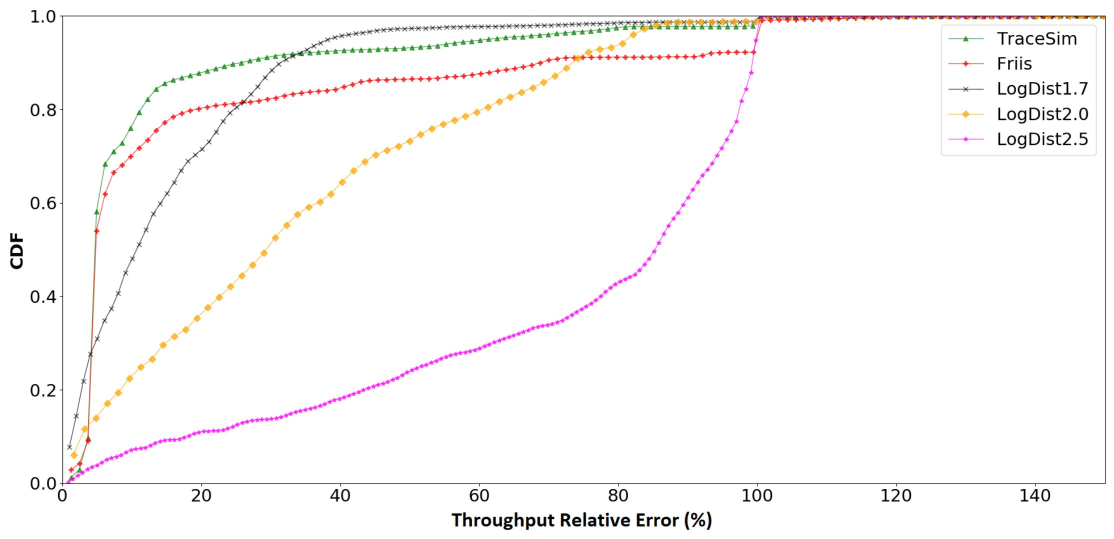


Figure 5.20: CDFs of the throughput relative error when comparing the trace-based (TraceSim) and pure simulations to the corresponding real experiments for fixed PHY rate mode.

Figure 5.20 presents the CDF of the throughput relative error from trace-based ns-3 simulation and pure ns-3 simulation in comparison to the throughput obtained in the real experiments for fixed PHY rate mode. As for the auto PHY rate mode, we filtered all real throughputs equal to 0 kbit/s. The filtered results account for 1.3% of the total of 32299, 1 second samples. Table 5.6 summarizes this CDF plot and presents the 90th percentile, 50th percentile (median), and average throughput relative error values for all the trace-based and pure simulations. Analyzing the pure simulation results we can observe that, as for the auto rate, the Friis and LogDist1.7 are again the best ones at representing the real throughput. Analysing the trace-based simulation results we can observe that, once again, TraceSim is the one that more closely reproduces the real-world experiment: for the 90th percentile, the TS approach presents a gain of 63% over Friis and a gain of 18% over LogDist1.7; for the median, the TS approach is only 2% more accurate than Friis and 56% more accurate than LogDist1.7; finally, on average, using the TS approach results in a gain of 37% over Friis and 21% over LogDist1.7. In conclusion, although the TS approach relative errors are not as low as in the auto PHY rate mode, after analyzing the plot and table we can conclude, once again, the higher accuracy associated to TS approach when compared to pure simulation throughput results. The CDF curves of the throughput relative error for some pure simulation scenarios show a very defined step around 100% of relative error. This is caused by a number of experiments where the ns-3 underestimated the SNR. Because we were forcing a fixed PHY rate based on the real experiment results (based on a higher SNR), that resulted in simulations with a fixed PHY rate that was too high and caused a packet loss ratio of 100%.

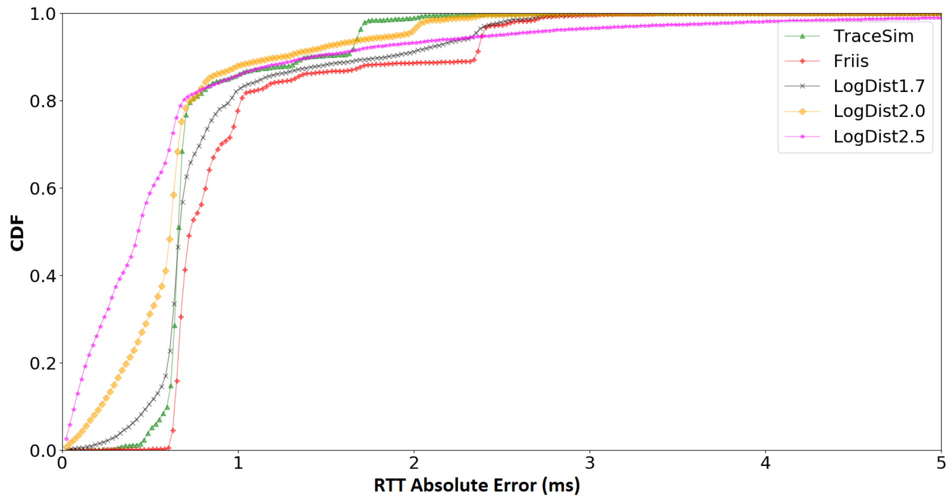


Figure 5.21: CDF of the trace-based ns-3 simulation (TraceSim) and pure ns-3 simulation (PureSim) RTT absolute error in comparison to the RTT obtained in the real experiments for auto PHY rate mode with packet size of 1472 bytes .

Regarding RTT we chose to represent the absolute error instead of the relative error (cf. Equation 5.1). The reasons for that were the following: 1) a very small and insignificant difference of delay for very low values of RTT can result in very high relative errors, e.g., 0.1 ms to 0.3 ms would give a relative error of 200%, but 0.2 ms can be insignificant for most applications; 2) we know in advance that ns-3 does not account for the nodes processing time, so we are expecting real results shifted by a value close to half a millisecond which would cause very high relative errors and make more difficult the results analysis. With absolute RTT errors, we can also easily estimate the processing overhead introduced by the real node in comparison to ns-3. Figure 5.21 shows the CDF of the absolute error of the RTT for trace-based ns-3 simulation and pure ns-3 simulation in comparison to the median value of RTT obtained in each second of the real experiments for auto PHY rate mode with packet size of 1472 bytes. In the case of RTT we filtered the seconds in which we could not get a delay sample in either the real experiment, trace-based simulation, or pure simulation, as we can only compare samples that contain RTT measurements. The filtered results account for roughly 1.5% of all the 1 second samples. In the plot we can observe that the median of the error for all simulated scenarios is around 0.7 ms, even for the TS approach. In this case, the TS approach suffers from the same abstraction problem of the pure simulation: the fact that ns-3 does not account for the nodes processing time. In the real experiments the RTT samples were always above 1.2 ms, while in ns-3 the minimum delay was around 0.5 ms, thus explaining this persistent error between all the simulation approaches and the real experiment. Taking this aspect into consideration, we can conclude that if a CDF curve rises significantly before the 0.7 ms absolute error mark it is a sign that the *PropagationLossModel* used in each simulation approach is not representing well the reality, as the only way to get an error lower than 0.7 ms is to have more frame retransmissions or using a lower PHY rate than the reality. Based on this conclusion, we can see that the Friis and LogDist1.7 are, as in the case of the throughput performance metric, the path loss models that represent more accurately the conditions of the real experiment. Nevertheless, the TS approach is again better. For the 90th percentile the TS approach presents a gain of 35% (0.8 ms) over Friis, lowering the absolute error respectively from 2.3 to 1.5 ms, and a gain of 21% (0.4 ms) over LogDist1.7, lowering the absolute error respectively from 1.9 to 1.5 ms. After analyzing the results we can conclude that the TS approach provides RTT values closer to the real experiments than a pure simulation approach. This gain over the pure simulation approach is due to the fact that it is reproducing a more accurate SNR, which enables more realistic frame retransmissions replication and the usage of closer-to-real PHY rates.

Based on the obtained results, we can conclude that even for the w-iLab.2 scenarios – where we are dealing with static nodes in a isolated testbed – the TS approach brings significant gains over pure ns-3 simulation, successfully reproducing closer-to-real network performance results by perpetuating the SNR observed in the real experiment. A pure simulation approach, independently of the ns-3 *PropagationLossModel* used, is unable to reproduce the asymmetric radio links quality of the real testbeds. By using the TS approach in emerging testbed scenarios we expect to get even higher gains when compared to the pure simulation approach, as was observed from the preliminary evaluation of the TS approach in the SUNNY testbed presented in Section 5.6.1, considering the highly complex and unstable nature of those testbeds physical conditions which are even harder to model and reproduce in simulation.

5.7 Discussion

The evaluation results presented in Section 5.6 demonstrate the added value of the TS approach. For all the three testbeds over which experiments were run, and later reproduced in ns-3 using the TS approach, we found significant gains introduced by the TS approach. Although the TS approach was proposed to address the repeatability and reproducibility of real experiments over emerging testbeds such as aerial testbeds – where its gains become more evident – throughout the TS approach evaluation we concluded its scope of application is broader; for instance, it also has significant gains in controlled environments such as the Fed4FIRE+ w-iLab.2 testbed. Nonetheless, our acquired hands-on experience on developing, testing, and using the TS approach for the last years in different research projects gave us a better insight on its strengths and weaknesses. In this section we discuss each of them. Regarding the weaknesses, we also discuss how they can be overcome.

5.7.1 TS Approach Strengths

The TS approach major strength builds upon identifying 1) the aspects which ns-3 is able to model accurately and 2) the aspects which ns-3 does not model accurately due to its inherent abstractions of the real world phenomena. On the one hand, ns-3 is very realistic at modeling the operation details of the MAC layer and upper layers. Based on a given SNR at the receiver, and using the validated *NistErrorRateModel* [1], ns-3 realistically calculates the FER depending on the frame size and the OFDM PHY rate used. Using a random variable stream, ns-3 randomly decides, per frame, whether the frame will be delivered successfully or dropped. This will cause the simulation of the necessary Wi-Fi MAC retransmissions, which affect directly the throughput and delay of a given link; in addition, the PHY rate will be automatically adjusted – based on the widely used Minstrel algorithm – to improve the frame delivery ratio. On the other hand, ns-3 is limited at modeling the testbed physical characteristics that cause highly unstable SNR on mobile testbeds. A node is represented by a dimensionless point without roll, pitch, yaw, and heading. This model does not account for the antenna misalignment or node body obstruction of radio line-of-sight. Also, the external phenomena such as multipath and different noise exposures that may make the radio links highly asymmetric are not modelled. By reproducing traces of real SNR for each link direction we overcome this ns-3 limitation. Using the real SNR traces we schedule SNR changes synchronized with their original timestamp in the real experiment. Yet, we are unable to enforce that the generation of simulated frames are synchronized with the timestamp of the SNR samples, i.e., the timestamp of the real frames. We are only reproducing the same SNR values, not the same frames. Although we do not reproduce a specific frame drop at a specific timestamp, by reproducing the same FER, on average, we reproduce equivalent frame losses and retransmissions triggering more realistic auto PHY rate adaptations. In turn, this enables the TS approach to achieve closer-to-real network performance.

Due to the accuracy of the TS approach in reproducing past real experiments, it now enables: 1) concurrent user access to the real testbed conditions based on past traces – the past experiment physical conditions become a virtual resource that can be shared by different users, without the need to reserve real resources or wait for their availability; 2) running simulations in faster than real time – past experiments can now be executed in simulation time, only limited by the processing power of the node hosting the simulation, which can help save time to test different iterations of the solution under evaluation; 3) running multiple simulation instances at the same time. Past experiments can now be executed at the same time in the same or multiple hosts, exploring different variants of the solution under evaluation. Using the Fast Prototyping process it is even possible to reproduce the same experiment in real-time, connected to external real nodes, which allows to keep improving and fine-tuning systems that depend on the communications to operate. E.g., in SUNNY project we reproduced UAV flights multiple times in order to adapt the UAV and Ground CS software applications to cope with the real radio link instability and throughput. Because the number of flight experiments was very limited, the TS approach proved to be very useful.

Although implemented and evaluated over ns-3, the TS approach has the potential of being used on other packet-level discrete event wireless network simulators, as long as an equivalent *TraceBasedPropagationLossModel* can be implemented – e.g., a new OMNeT++ [78] *PathLossModel* – and the realism of the Wi-Fi MAC implementation and upper layer in such network simulation is as good as in ns-3.

5.7.2 TS Approach Weaknesses

In the TS approach, SNR sampling resolution depends on the network traffic. To increase the number of samples captured we not only use the DATA frames as samples, but also the BEACONS and ACK control frames. Also, in a multiple-access scenario, if a node overhears frames exchanged between its peers (interface in monitor mode) it can also use those samples as if the sending node was transmitting to it. If we saturate an high SNR IEEE 802.11a link with UDP traffic with packets size of 1500 bytes, we can obtain around 2500 SNR samples per second in each direction of the link (the DATA frames in one direction, the ACK frames in the other direction). Nonetheless, in a real scenario the number of frames per second can be significantly lower. If the nodes are configured in ad-hoc mode, each of them sends BEACONS at a default rate of 10 per second; this provides at least 10 SNR samples per second in each direction, between each node in radio range. If we use infrastructure mode (AP plus STAs) we may need to actively generate some baseline traffic in the network, so that we can record the necessary traces to use the TS approach. The minimum resolution needed may depend on how unstable the radio link is for each specific testbed.

The TS approach is not currently focused on detecting link failures between peers. It only records the SNR of frames that were successfully received. If a real frame is not received due to a sudden drop of SNR, that phenomenon is not recorded in the SNR traces. However, from our experience in the real UAV test flight, we could observe that when the SNR reaches a value so low that the DATA frames are no longer delivered, the BEACONS – with lower FER – are still passing through until the link is lost. When the link is lost, the very low SNR value of the last BEACON remains constant in the trace-based simulation throughout the period of no connectivity. During this period, the ns-3 nodes experience an equivalent loss of connectivity at the network layer, because no DATA frames are successfully transmitted with such low SNR even if the ns-3 MAC layer remains with an established link. When the real link connectivity is established again, new BEACONS and DATA frames start to populate the SNR trace again, which in turn raises the SNR and re-establish the network layer connectivity in ns-3. Although this weakness was not evident in the TS approach evaluation presented in Section 5.7, it may be overcome by adding a timeout for the validity of the last SNR sample. If the timeout is triggered, it means there were no frames received along that time period and we may assume the connectivity is lost and force a link failure in that direction. This can be done, for example, by setting the ns-3 SNR equal to 0 dB until a new SNR sample is collected.

Beamforming is currently supported by our ns-3 *TraceBasedPropagationLossModel* as it allows to set any desired SNR for each specific link between two peers. Whether the real SNR trace was sampled with beamforming gain or not, what matters is the resulting SNR. In beamforming enabled networks, only DATA frames specifically sent to the receiving node taking the SNR samples can be considered as valid samples. In these scenarios there is a need for a mechanism to actively generate baseline traffic for each possible one-hop radio link. This is especially relevant for the more recent Wi-Fi standards.

The current TS approach only supports SISO links, where a given SNR variation over time in the single radio stream directly translates to a FER, the respective auto PHY rate adaptations, and a resulting network performance. In MIMO, a given SNR at the receiver does not directly translate to the number of radio streams being used; for instance, we can have a radio link with very high SNR but multipath radio propagation could not be favorable to spatial multiplexing. The support for MIMO may be added to the TS approach by using two alternative approaches: 1) record traces of real Modulation Codding Scheme (MCS) per frame, along with the SNR values. A Wi-Fi card supporting MIMO uses the Channel State Information (CSI) to assess the radio link quality for each OFDM subcarier between each pair of antennas., This information is used alongside with Minstrel-HT to decide the MCS to be employed. In IEEE 802.11n, for the same modulation used, different MCS indexes are reported according to the number of streams being used. By reproducing the same MCS and SNR in ns-3, we could be able to obtain in ns-3 a closer-to-real network performance for MIMO radio links; 2) record the CSI and evaluate how the radio link quality could be replicated in ns-3 using the *SpectrumPhy*, which allows for controlling each of the OFDM channel sub-carriers.

The TS approach assumes no other concurrent Wi-Fi network or other technology (e.g., IEEE 802.15.4 [79], LTE-U [80]) is sharing the spectrum with our testbed when we run the real experiments, unless its signal is so low that it is reported in the total noise floor. Because a radio channel is a shared medium, the resulting network performance (throughput and RTT) can also be affected by the channel occupation generated by nodes of networks external to the testbed and out of the experimenter control. The TS approach may be improved by recording the channel busy time per second (the busy time caused by transmissions external to the testbed) and also reproducing that characteristic in ns-3.

The TS approach uses a “record and replay” methodology. Therefore, when using the TS approach we are limited to reproducing a specific real experiment, for the same amount of nodes, the sequence of nodes positions observed in the real experiment, related SNR, and duration of the experiment. Despite these limitations, the “record and replay” methodology is very useful in scenarios for which there are no adequate *PropagationLossModels* and *MobilityModels*. Eventually, depending on the amount of traces collected over time, this data may be used to overcome this weakness by, for example: 1) creating new (or adapting existing) stochastic models to the new scenarios; 2) using an hybrid approach with stochastic and Machine Learning (ML) based models, a concept explored (proof-of-concept) by a related MSc dissertation [81]. In this way we could change and scale the simulation scenario while maintaining a closer-to-real performance than pure simulation.

In the TS approach, we are gathering a SNR sample per each frame received. For large scale experiments, the size of the traces becomes non negligible. It would be interesting to find a perfect balance between sampling resolution and accuracy of the TS approach. For instance, by doing an initial benchmarking of the network performance with full sampling resolution; then using an automatic mechanism to run multiple trace-based simulations using the same traces – iteratively lowering their resolution – until we reach a minimum accuracy threshold.

5.8 Summary

Wireless networking research and development is increasingly dependent on experimentation to further evaluate and validate a solution in real environment. Experiments are also increasingly complex and difficult to repeat and reproduce, especially over emerging testbeds but also in controlled testbeds. With the goal of perpetuating past real experiments, we proposed the TS approach. The TS approach replicates real experiments by feeding traces into ns-3, using the new TraceBasedPropagationLossModel, combined with the ns-3 TCP/IP and MAC simulation capabilities.

The TS approach was evaluated using a large set of experiments over three different testbeds: 1) SUNNY UAV-Ground Communications Testbed; 2) Isolated Laboratory Testbed; 3) Fed4FIRE+ w-iLab.2 Testbed. The evaluation results showed the TS approach has significant gains when compared to the use of a pure simulation approach; it achieved average gains above 53% and 90th percentile gains above 57%. The HSSR for SNR traces is highly encouraged, as it produces with more detail the SNR variations along the time. The TS approach enables: 1) concurrent user access to the real testbed conditions based on past traces; 2) running simulations in faster than real time; 3) running multiple simulation instances at the same time, exploring different variants of the solution under evaluation. Using the TS approach, it is also possible to reproduce the same experiment in real-time, connected to external real nodes, which allows to keep improving and fine-tuning client systems that depend on the communications system to operate. After a thorough evaluation of the TS approach we can conclude that it truly enhances the cooperation between Simulation and Experimentation performance evaluation phases, creating a virtuous cycle between them.

Chapter 6

Conclusions

In this final chapter we overview the work developed in this thesis, recall the major contributions of our work, point out the limitations of the Fast Prototyping process and the TS approach, and refer topics that may be the subject of future work.

6.1 Overview of the Work Developed

This thesis focused on improving the performance evaluation of wireless networks. Performance evaluation typically depends on Simulation and Experimentation to properly evaluate and fine-tune the operation of new protocols or enhancements of existing protocols. For this purpose, it is important to have a simulation model and an implementation prototype that are coherent. Also, it is desirable to obtain repeatable and reproducible results. New emerging vehicular networking scenarios, such as aerial and maritime networking scenarios, have characteristics that present special difficulties to the performance evaluation process. Simulation usually produces optimistic results. Experimentation is mainly limited by the testbed availability. This affects the ability to compare simulation and real results, and precludes the repeatability and reproducibility of the experimental conditions, which often makes proper evaluation difficult, if not impossible, to achieve. Based on our hands-on experience in different research projects, in this thesis we proposed new forms of cooperation between Simulation and Experimentation using ns-3, and a more suitable more suitable performance evaluation process.

From Simulation to Experimentation, we reviewed the traditional protocol development process and identified its main problems: 1) the duplication of effort to write the simulation model and the implementation prototype; 2) the difficulty to maintain both implementations synchronized without introducing errors. In order to address these problems, we proposed the Fast Prototyping development process, which explores the concept of a shared ns-3 protocol model between Simulation and Experimentation. Fast Prototyping is based on the ns-3 emulation capability. In its current version, the ns-3 emulation mode has functional and performance problems, which were also addressed in this work. On the one hand, we proposed an improved version of the ns-3 *EmuFdNetDevice* by introducing new functionalities. The new functionalities include the support of new types of real network interfaces and a new auto-configuration mechanism for ns-3 nodes that makes easier the integration of emulation nodes with existing networks. On the other hand, we explored the DPU and DPK approaches, for offloading the data-plane packet processing to outside of ns-3, thus improving the performance of the ns-3 emulation mode. We compared the performance of the new approaches against traditional ns-3 emulation. The evaluation results showed that when emulating a single ns-3 node the maximum throughput can be improved by as much as 4.9 times for the DPU and 19 times for the DPK, while having the RTT lowered by respectively 5.3 and 14 times; when emulating multiple ns-3 nodes using DPK, the maximum throughput can be improved by as much as 23 times while having 15 times lower RTT. The DPK approach has the best performance, obtaining results very close to a real protocol implementation. This enables the use of the Fast Prototyping process in traffic demanding scenarios or in real nodes with limited processing power. The better performance and the ability to use the DPK approach to emulate multiple nodes in the same host machine allows to extend (in scale or functionality) a real testbed by means of emulated resources.

From Experimentation to Simulation, the proposed TS approach is supported by the new *TraceBasedPropagationLossModel* and combines the ns-3 TCP/IP and MAC simulation capabilities with the physical characteristics of the real experiments captured in traces of node positions and radio link quality. The *TraceBasedPropagationLossModel* helped evaluating the TS approach using traces from experiments ran over three different testbeds: 1) SUNNY UAV-Ground Communications Testbed; 2) Isolated Laboratory Testbed; 3) Fed4FIRE+ w-iLab.2 Testbed. The evaluation results showed the TS approach has significant gains when compared to the use of a pure simulation approach; it can achieve gains above 53% on average and above 57% for the 90th percentile. The TS approach also enables: 1) concurrent user access to the real testbed conditions based on past traces; 2) running experiments in faster than real time; 3) running multiple simulation instances at the same time, exploring different variants of the solution under evaluation. Using the ns-3 emulation mode, it is also possible to reproduce the same experiment in real-time, connected to external real nodes, which allows to keep improving and fine-tuning systems that depend on communications to operate. The TS approach truly enhances the cooperation between Simulation and Experimentation performance evaluation phases, creating a virtuous cycle between them.

6.2 Original Contributions

The working hypothesis has been validated. The Fast Prototyping process enables the use of a shared ns-3 protocol model implementation for Simulation and Experimentation. The TS approach enables the repeatability and reproducibility of past real experiments.

The two main original contributions provided by this thesis are the following:

1. **Fast Prototyping Development Process.** This is a new shared protocol model implementation process to be used during the performance evaluation phases of protocol development.

By reusing the already implemented ns-3 protocol model over the real hardware prototype, then installed in a testbed, we eliminate the duplicate effort to develop simulation and real implementations. This reduces the coding effort and also the chance for error introduction, which could render the results non-comparable. This process relies on ns-3 emulation functionality which allows to run simulated resources in real time, interacting with the real network interfaces. This contribution includes the following specific contributions: 1) improvements to the compatibility of ns-3 emulation to a larger set of real network interface types and operational restrictions encountered in the real-world networks used by the testbeds; 2) different approaches to improve the ns-3 emulation performance over the real testbed hardware. The novelty of this process, when compared to the state-of-the-art alternatives, relies on maintaining the benefits of ns-3 regarding the easiness and flexibility of implementation, the important log functionality it provides, and the ability to combine, in the same run, simulated and emulated resources, which can improve the testbed in scale and functionality.

2. **Trace-based Simulation Approach.** This is a novel simulation approach allowing to perpetuate past real-world experiments and rerun them independently of the testbed availability and the external phenomena influencing its physical conditions. This is possible by recording traces of such physical conditions and reproducing them using a trace-based simulation. Trace-based simulation is especially important considering very unpredictable and unstable scenarios such as the emerging wireless vehicular networking scenarios. Using trace-based simulations we can reproduce the same conditions encountered in the real experiment runs. The novelty of our approach, when compared to the state-of-the-art, is that we only reproduce the physical conditions (radio links characteristics and the positions of the nodes) and rely on ns-3 TCP/IP and MAC simulation capabilities for the upper layers.

With both contributions in place, the interactions between Simulation and Experimentation result in a Simulation-Experimentation synergy that improves the Performance Evaluation of Wireless Networks.

Five conference papers and two journal papers were produced as a direct result of this thesis:

1. **ns-3 NEXT: Towards a Reference Platform for Offline and Augmented Wireless Networking Experimentation** in Proceedings of the Workshop on ns-3, 2019, Florence, Italy (Conference) [6];

2. **Improving ns-3 Emulation Performance for Fast Prototyping of Routing and SDN Protocols: Moving Data Plane Operations to Outside of ns-3** in Simulation Modelling Practice and Theory, 2019 (Journal) [7];

3. **Improving the ns-3 *TraceBasedPropagationLossModel* to Support Multiple Access Wireless Scenarios** in Proceedings of the Workshop on ns-3, 2018, Surathkal, India (Conference) [8];

4. **A Trace-Based ns-3 Simulation Approach for Perpetuating Real-World Experiments** in Proceedings of the Workshop on ns-3, 2017, Porto, Portugal (Conference) [9];

5. **Improving ns-3 Emulation Performance for Fast Prototyping of Network Protocols** in Proceedings of the Workshop on ns-3, 2016, Seattle, WA, USA (Conference) [10];

6. **Improving ns-3 Emulation Support in Real-World Networking Scenarios** in Proceedings of the 8th International Conference on Simulation Tools and Techniques, 2015, Athens, Greece (Conference) [11];

7. **Fast Prototyping of Network Protocols Through ns-3 Simulation Model Reuse** in Simulation Modelling Practice and Theory, 2011 (Journal) [12];

6.3 Fast Prototyping Process and Trace-based Approach Limitations

The Fast Prototyping protocol development process has the following major limitations:

- **Applicability.** In order to benefit from the shared protocol implementation between simulation and experimentation, as proposed by Fast Prototyping, we should be developing a new network protocol. If we are just improving a protocol whose real implementation already exists, it may be difficult and error-prone to re-implement its data plane and control plane from scratch in ns-3. In that case, a better alternative would be to use the ns-3 DCE, which allows to run a real implementation in ns-3 if the source code is available and it is compatible with DCE.

- **Compatibility.** The DPK approach only supports protocols whose data planes are compatible with the Linux kernel. Otherwise, the DPU approach is a valid alternative and its performance, although worse than the DPK, is better than the traditional ns-3 emulation approach running the data-plane inside ns-3. In its current version, the Real Routing module, developed for the DPK approach, only supports proactive L3 protocols. The extension of the Real Routing module to support reactive L3 protocols is left for future work.

Our acquired hands-on experience on developing and using the TS approach along the last years gave us a better understanding of its limitations. In what follows, we refer to each of them and present possible ways to overcome the limitations:

- **Accuracy gain depends on the realism of the ns-3 *ErrorRateModel*.** In our scenarios we have always used PHY rates based on OFDM. For that purpose, we used the *NistErrorRateModel* which was previously validated for IEEE 802.11a OFDM PHY rate modulations. If we use different modulations (e.g., IEEE 802.11b DSSS) we should select the *ErrorRate-Model* that best suites the experiment.

- **ns-3 does not account for node processing time.** For some experiments, such as measuring a Wi-Fi link RTT with low network load, substantial differences between the results obtained for trace-based simulations and real experiments can exist.

- **SNR sampling resolution.** We depend on network traffic to sample the asymmetric SNR of each radio link. If we want high SNR sampling rate we may need to generate some background traffic to have the necessary SNR samples, and assure adequate experiment reproduction using the TS approach.

- **Detection of link failure.** Intermittent link failure is difficult to detect as we can only sample SNR from frames that are received successfully. The link SNR only keeps updated if there are new samples. At the moment, we do not have any timeout to define how long we can consider the last SNR sample as valid and keep the link alive in simulation.

- **Beamforming is not supported.** In the current version we use the interface in monitor mode to collect all frames, even the frames belonging to communications between neighboring peers. We use all these frames, including the broadcasted control frames, to sample the SNR for every possible link. For beamforming the nodes can only consider the frames that are destined to themselves, so that the SNR is recorded with the correct gain.

- **MIMO is not supported.** The TS approach only supports SISO links, where a given SNR variation over time in the single radio stream directly translates to a FER, the respective auto PHY rate adaptations, and a resulting network performance. In MIMO, using the SNR makes it insufficient for reproducing a realistic PHY rate in ns-3, as the SNR does not represent the number of spatial streams being used. In experiments using MIMO-enabled Wi-Fi interfaces, recording the number of streams being used and their SNR should be considered in future work.

- **Record and Replay methodology.** The TS approach aims only at reproducing the exact same conditions of the real experiment (e.g., number of nodes, trajectory, and duration). There is no support to improve the pure simulation scenarios based on models derived from the real traces.

- **Amount of data generated.** The TS approach records all the SNR samples. This can become a problem for very large experiments. A good balance between sampling resolution and TS approach accuracy should be considered in future work.

6.4 Future Work

In the following we refer to further developments that may be considered within the scope of the Fast Prototyping process and the TS Approach. We also identify open research topics with high potential to attain new original contributions on top of this thesis work.

6.4.1 Further Developments

When it comes to the Fast Prototyping process, we may consider the integration of the *Real Routing* module in the main ns-3 distribution in order to reach more protocol developers and researchers. In addition, we may consider the development of a module to automatically generate the network namespaces and their connections based on the topology of the nodes created in the simulator; at the moment, users have to manually create the namespaces and configure the emulated nodes with their corresponding namespace. Finally, to increase the usefulness of the *Real Routing* module, we may extend it to support reactive protocols. Concerning the TS approach, we may improve the *TraceBasedPropagationLossModel* for overcoming the set of limitations identified in Section 5.7 and Section 6.3. In addition, we may develop a framework to assist the related processes of traces capturing, managing, reusing, and sharing.

6.4.2 Open Research Topics

The use of the TS approach leads to the collection of a large number of datasets containing traces of radio link quality and mobility of nodes from a multitude of real experiments. By taking advantage of this large amount of data, we argue that Machine Learning techniques can be employed to, for instance, learn new path loss and mobility models. Those models can then be used to better simulate the physical characteristics of emerging communications environments such as aerial and maritime. This will enable the use of the TS approach beyond the mere replication of past real experiments. The benefits may include 1) the ability to scale the scenarios by using a higher number of nodes and 2) the ability to simulate different node trajectories from the trajectories observed in the real experiment using a custom-tailored mobility model.

Adding support for MIMO and beamforming in the TS approach is a relevant topic as it is a core characteristic of more recent IEEE 802.11 standards, such as IEEE 802.11n, IEEE 802.11ac, and the upcoming IEEE 802.11ax. By being able to capture more details from the real-world radio link, such as the number of spatial streams used, or the Channel State Information, which reports the radio link quality for each OFDM sub-carrier, will allow to broaden the application scope of the TS approach. Also, the association of this new captured link quality information per sub carrier to the new *SpectrumPhy* ns-3 module will enable more realistic MIMO simulation in ns-3 simulator.

Emerging testbeds such as aerial and maritime have scarce resources and scale, and are available for short periods of time mainly due to the logistics and high operation costs involved. What if we could create a kind of augmented reality lab to increase the scale of the real testbed? We imagine, for example, to have a room full of ns-3 emulation servers that are linked via the Internet to real testbed or gateway near the real nodes. These servers can be running ns-3 emulated nodes following mobility and path loss models previously obtained through Machine Learning. Imagine two clusters of vehicle network testbeds operating in distant locations in a city. This emulation server could allow to create emulated nodes interacting with the real ones to allow multihop communications between the two clusters of real nodes, evaluating the solution in a larger scale and with realism.

References

[1] Guangyu Pei and Thomas R Henderson. Validation of ofdm error rate model in ns-3. https://www.nsnam.org/pei/80211ofdm.pdf, 2010. Boeing Research Technology.

[2] WiN - Wireless Networks Group home page. http://win.inesctec.pt/, 2018. Accessed January, 2018.

[3] INESC TEC home page. http://www.inesctec.pt/, 2018. Accessed January, 2018.

[4] ns-3. ns-3 home page. http://www.nsnam.org, January 2018. Accessed January, 2018.

[5] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] Helder Fontes, Vitor Lamela, Rui Campos, and Manuel Ricardo. ns-3 next: Towards a reference platform for offline and augmented wireless networking experimentation. In *Proceedings of the 2019 Workshop on Ns-3*, WNS3 2019, pages 65–72, Florence, Italy, 2019. ACM.

[7] Helder Fontes, Tiago Cardoso, Rui Campos, and Manuel Ricardo. Improving ns-3 emulation performance for fast prototyping of routing and sdn protocols: Moving data plane operations to outside of ns-3. *Simulation Modelling Practice and Theory*, 96:101931, 2019.

[8] Helder Fontes, Rui Campos, and Manuel Ricardo. Improving the ns-3 tracebasedpropagationlossmodel to support multiple access wireless scenarios. In *Proceedings of the 10th Workshop on ns-3*, pages 77–83. ACM, 2018.

[9] Helder Fontes, Rui Campos, and Manuel Ricardo. A trace-based ns-3 simulation approach for perpetuating real-world experiments. In *Proceedings of the Workshop on ns-3*, pages 118–124. ACM, 2017.

[10] H. Fontes, T. Cardoso, and M. Ricardo. Improving ns-3 emulation performance for fast prototyping of network protocols. In *Proceedings of the Workshop on ns-3*, WNS3 '16, pages 108–115, Seattle, WA, USA, 2016. ACM.

[11] H. Fontes, R. Campos, and M. Ricardo. Improving ns-3 emulation support in real-world networking scenarios. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 261–266, Athens, Greece, 2015. ICST.

[12] G. Carneiro, H. Fontes, and M. Ricardo. Fast prototyping of network protocols through ns-3 simulation model reuse. *Simulation Modelling Practice and Theory*, 19(9):2063–2075, October 2011.

[13] SITMe's project website. http://www.sitme.org. Accessed January, 2019.

[14] SUNNY. Sunny - smart unattended airborne sensor network for detection of vessels used for cross border crime and irregular entry. http://www.sunnyproject.eu/, January 2018.

[15] BLUECOM+. Bluecom+ – connecting humans and systems at remote ocean areas using cost-effective broadband communications. http://bluecomplus.inesctec.pt/, January 2018.

[16] SIMBED. Simbed – offline real-world wireless networking experimentation using ns-3. http://win.inesctec.pt/Projects, January 2019.

[17] WISE. Wise – traffic-aware flying backhaul mesh networks. http://wise.inesctec.pt/, January 2018.

[18] Helder Martins Fontes. Multi-technology router for mobile networks: layer 2 overlay network over private and public wireless links, 2010. MSc Thesis, MIEIC, FEUP, Universidade do Porto.

[19] M. Ricardo, G. Carneiro, P. Fortuna, F. Abrantes, and J. Dias. Wimetronet a scalable wireless network for metropolitan transports. In *Proceedings of the 2010 Sixth Advanced International Conference on Telecommunications (AICT)*, pages 520–525, May 2010.

[20] Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 8–22, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[21] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. In *IEEE International Conference on Communications, 2009. ICC '09*, pages 1–5, Dresden, Germany, June 2009.

[22] A.R. Khan, S.M. Bilal, and M. Othman. A performance comparison of open source network simulators for wireless networks. In *2012 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 34–38, Penang, Malaysia, November 2012.

[23] ns-2. ns-2 home page. http://nsnam.sourceforge.net/wiki/index.php/Main_Page, January 2018.

[24] ns-2 Emulation. ns-2 emulation. http://nsnam.sourceforge.net/wiki/index.php/Wireless_Network_Emulation_Tutorials, January 2018.

[25] Shivkumar C. Muthukumar, Xiaozhou Li, Changbin Liu, Joseph B. Kopena, Mihai Oprea, and Boon Thau Loo. Declarative toolkit for rapid network protocol simulation and experimentation. In *ACM SIGCOMM Conference on Data Communications (demo)*, Barcelona, Spain, August 2009.

[26] U.S. NRL Networks and Communications Systems Branch. The Protean Protocol Prototyping Library (Protolib). http://www.nrl.navy.mil/itd/ncs/products/protolib. Accessed January 16, 2015.

[27] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[28] Michael Neufeld, Ashish Jain, and Dirk Grunwald. Nsclick: Bridging Network Simulation and Deployment. In *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '02, pages 74–81, New York, NY, USA, 2002. ACM.

[29] P Lalith Suresh and Ruben Merz. ns-3-click: click modular router integration for ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 423–430, Barcelona, Spain, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[30] J. Paul Morrison. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. CreateSpace Independent Publishing Platform, Unionville, Ont., 2 edition edition, May 2010.

[31] Matjaž Fras, Gregor Globačnik, and Jože Mohorko. Advanced method of network simulations with opnet modeler. In *proceedings of the 14th National Conference on High Education TREND, Kopaonik*. Citeseer, 2008.

[32] Direct Code Execution (DCE) Manual. `http://www.nsnam.org/docs/dce/release/1.4/manual/singlehtml/index.html`. Accessed January, 2018.

[33] Hajime Tazaki, Frédéric Uarbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid Dabbous. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 217–228, New York, NY, USA, 2013. ACM.

[34] iperf for dce - iperf tool source code changes to be supported by ns-3 dce. `https://github.com/direct-code-execution/ns-3-dce/blob/master/example/dce-iperf.cc`, 2019. Accessed January, 2019.

[35] Mathieu Lacage. *Experimentation Tools for Networking Research*. Ph.D., Universite de Nice-Sophia Antipolis, 2010.

[36] XW Huang, Rosen Sharma, and Srinivasan Keshav. The entrapid protocol development environment. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1107–1115. IEEE, 1999.

[37] David Ely, Stefan Savage, and David Wetherall. Alpine: A user-level infrastructure for network protocol development. In *USITS*, volume 1, pages 15–15, 2001.

[38] Sam Jansen and Anthony McGregor. Simulation with real world network stacks. In *Simulation Conference, 2005 Proceedings of the Winter*, pages 10–pp. IEEE, 2005.

[39] Hajime Tazaki, Frédéric Urbani, and Thierry Turletti. DCE cradle: Simulate network protocols with real stacks for better realism. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pages 153–158, Cannes, France, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[40] Marko Zec and Miljenko Mikuc. Operating system support for integrated network emulation in imunes. In *Workshop on Operating System and Architectural Support for the on demand IT Infrastructure (1; 2004)*, 2004.

[41] John Abraham and George Riley. Simulator-agnostic ns-3 applications. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 391–396, Sirmione-Desenzano, Italy, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[42] J. Moy. OSPF version 2. RFC 2328, IETF, April 1998.

[43] M. Ricardo, G. Carneiro, P. Fortuna, F. Abrantes, and J. Dias. WiMetroNet a scalable wireless network for metropolitan transports. In *2010 Sixth Advanced International Conference on Telecommunications (AICT)*, pages 520–525, Barcelona, Spain, May 2010.

[44] W. Simpson. The Point-to-Point Protocol (PPP), July 1994. RFC 1661.

[45] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware, November 1982. RFC 826.

[46] Steve Alexander and Ralph Droms. DHCP Options and BOOTP Vendor Extensions, March 1997. RFC 2132.

[47] Wireshark website. http://www.wireshark.org/. Accessed: 2015-02-13.

[48] Gianluca Insolvibile. Kernel korner: Linux socket filter: Sniffing bytes over the network. *Linux J.*, 2001(86):8–, June 2001.

[49] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux netlink as an IP services protocol. RFC 3549, IETF, July 2003.

[50] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF, July 2003.

[51] Source code for real routing module - sequency of patches to ns-3.21. http://telecom.inesctec.pt/~hfontes/real_routing_patches.zip, February 2016.

[52] Jake Edge. Namespaces in operation, part 7: Network namespaces. https://lwn.net/Articles/580893, January 2014. Accessed June 01, 2015.

[53] The Linux Foundation. netem. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, November 2009. Accessed June 01, 2015.

[54] Gustavo João Alves Marques Carneiro. *Transparent metropolitan vehicular network: design and fast prototyping methodology*. PhD thesis, Universidade do Porto, Porto, 2012.

[55] P. Jacquet and T. Clausen. Optimized link state routing protocol (OLSR). RFC 3626, IETF, October 2003.

[56] OLSR.org. OLSRd. http://www.olsr.org. Accessed June 01, 2015.

[57] Pasquale Imputato, Stefano Avallone, and Tommaso Pecorella. Network emulation support in ns-3 through kernel bypass techniques. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS 2017, pages 259–260, New York, NY, USA, 2017. ACM.

[58] P. Imputato. High performance emulation of real devices in ns-3. https://www.nsnam.org/wiki/SOCIS2017, 2017.

[59] R. Campos, T. Oliveira, N. Cruz, A. Matos, and J. M. Almeida. Bluecom+: Cost-effective broadband communications at remote ocean areas. In *OCEANS 2016 - Shanghai*, pages 1–6, Shanghai, China, April 2016.

[60] G. Z. Papadopoulos, K. Kritsis, A. Gallais, P. Chatzimisios, and T. Noel. Performance evaluation methods in ad hoc and wireless sensor networks: a literature study. *IEEE Communications Magazine*, 54(1):122–128, January 2016.

[61] Georgios Z. Papadopoulos, Antoine Gallais, Guillaume Schreiner, and Thomas Noël. Importance of repeatable setups for reproducible experimental results in iot. In *Proceedings of the 13th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, &#38; Ubiquitous Networks*, PE-WASUN '16, pages 51–59, New York, NY, USA, 2016. ACM.

[62] Keoma Brun-Laguna, Pedro Henrique Gomes, Thomas Watteyne, and Pascale Minet. Moving Beyond Testbeds? Lessons (We) Learned about Connectivity. *IEEE Pervasive Computing*, December 2018.

[63] S. Keranidis, W. Liu, M. Mehari, P. Becue, S. Bouckaert, I. Moerman, T. Korakis, I. Koutsopoulos, and L. Tassiulas. Concrete: A benchmarking framework to control and classify repeatable testbed experiments. In *FIRE Engineering Workshop*, Ghent, Belgium, 2012.

[64] Fed4FIRE+. Fed4fire+ – the largest federation of testbeds in europe. `https://www.fed4fire.eu/`, January 2019. Last Accessed: January, 2019.

[65] mininet wifi. mininet-wifi – emulator for software-defined wireless networks. `https://github.com/intrig-unicamp/mininet-wifi`, January 2019. Last Accessed: January, 2019.

[66] R. Fontes, M. Mahfoudi, W. Dabbous, T. Turletti, and C. Rothenberg. How far can we go? towards realistic software-defined wireless networking experiments. *The Computer Journal*, 60(10):1458–1471, 2017.

[67] Description of the minstrel algorithm. `https://sourceforge.net/p/madwifi/svn/HEAD/tree/madwifi/trunk/ath_rate/minstrel/minstrel.txt`, January 2019. Last Accessed: January, 2019.

[68] P. Owezarski and N. Larrieu. A trace based method for realistic simulation. In *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, volume 4, pages 2236–2239, June 2004.

[69] P. Agrawal and M. Vutukuru. Trace based application layer modeling in ns-3. In *2016 Twenty Second National Conference on Communication (NCC)*, pages 1–6, March 2016.

[70] Mathias Kurth, Anatolij Zubow, and Jens-Peter Redlich. Multi-channel link-level measurements in 802.11 mesh networks. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, pages 937–944. ACM, 2006.

[71] H. Fontes. Source code for the tracebasedpropagationmodel – use in ns-3.28. `http://telecom.inesctec.pt/~hfontes/trace_based_propagation_loss_model2018.zip`, February 2018.

[72] H. Fontes. Source code for the tracebasedpropagationmodel – use in ns-3.28. `http://telecom.inesctec.pt/~hfontes/trace_based_propagation_loss_model2017.zip`, February 2017.

[73] Zhigang Rong and Theodore S Rappaport. *Wireless communications: Principles and practice, solutions manual*. Prentice Hall, 1st ed. edition, 1996.

[74] H. T. Friis. A note on a simple transmission formula. *Proceedings of the IRE*, 34(5):254–256, May 1946.

[75] V. Erceg, L. J. Greenstein, S. Tjandra, S. R. Parkoff, A. Gupta, B. Kulic, A. Julius, and R. Jastrzab. An empirically-based path loss model for wireless channels in suburban environments. In *IEEE GLOBECOM 1998 (Cat. NO. 98CH36250)*, volume 2, pages 922–927 vol.2, Nov 1998.

[76] W. Lindsey. Error probabilities for rician fading multichannel reception of binary andn-ary signals. *IEEE Transactions on Information Theory*, 10(4):339–350, October 1964.

[77] MINORU NAKAGAMI. The m-distribution—a general formula of intensity distribution of rapid fading. In W.C. HOFFMAN, editor, *Statistical Methods in Radio Wave Propagation*, pages 3 – 36. Pergamon, 1960.

[78] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[79] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.

[80] Alireza Babaei, Jennifer Andreoli-Fang, Yimin Pang, and Belal Hamzeh. On the impact of lte-u on wi-fi performance. *International Journal of Wireless Information Networks*, 22(4):336–344, Dec 2015.

[81] João Rafael de Figueiredo Cabral. A machine learning approach for path loss estimation in emerging wireless networks, 2018. MSc Thesis, MIEIC, FEUP, Universidade do Porto.