Chair of Network Architectures and Services
School for Computation, Information and Technology
Technical University of Munich

Utilizing Machine Learning to estimate Link Quality and adjust Wireless Connection Parameters accordingly

Abstract

The efficiency of a wireless network is predominantly dictated by the link quality between individual nodes, which, in turn, affects the optimal parameter selections for their corresponding communication devices. However, within the Network Coding Module (NCM), these parameters are frequently assigned static values that fail to align with the current link quality, leading to suboptimal network performance. Capitalizing on recent advancements in the networking domain [2], this study proposes employing machine learning techniques to estimate link quality and subsequently adjust communication device parameters within the NCM.
To facilitate this, we first established a continuous integration and continuous deployment (CI/CD) pipeline to distribute the latest codebase (encompassing libmoep-ncm, link quality estimation backend, and web-based visualization) directly to the NCMs. This pipeline also automates the initiation of libmoep-ncm on the NCMs and initiates a ping command representing data exchange between NCMs. Subsequently, we enhanced the libmoep-ncm library to enable the transmission of current link statistics via a TCP socket to the Python backend responsible for AI-based link quality estimation. The computed LQE score is then returned to the NCM, facilitating the future adjustment of link quality-dependent configurations in libmoep-ncm. In addition, the Python backend provides a client-side web interface for visualizing essential parameters of the established link on the NCM, as well as the predicted link quality estimation. The infrastructure created in this project lays the groundwork for future research focused on dynamically modifying specific parameters in libmoep-ncm, based on the link quality, to enhance overall network performance.

1 Introduction

Wireless communication channels often experience varying conditions, necessitating accurate link quality estimations for the effective functioning of certain wireless protocols and mechanisms. In this paper, we delve into the investigation of diverse approaches to estimating link quality between two nodes in a wireless network using our Network Coding Module (NCM), with a primary focus on collecting link quality statistics of a link as well as machine learning techniques to facilitate the link quality estimation. Our aim is to combine the estimation of link quality with the optimization of wireless connection device parameters, ultimately enhancing overall network performance, particularly in heterogeneous wireless network environments with rapidly changing standards.
To accomplish this, we will employ machine learning techniques within the NCM introduced in the lecture, training a machine learning model using a large volume of traces obtained from real system environments by leveraging publicly available datasets. We will consider the suitability of these datasets for NCM optimizations, as well as the advantages and drawbacks of various metrics, Received Signal Strength Indicator (*RSSI*), as input metrics for link quality estimation models.
Recent years have seen an increasing popularity of machine learning-based link quality estimators. We will focus on supervised learning techniques, such as decision trees. These models are simple but powerful enough for link quality estimators [2]. By employing these techniques, we anticipate the ability to better predict the overall link quality of the established session and by the means of it, improve the overall network performance of the NCM by dynamically adjusting certain parameters.

2 Materials and methods

In this section, we detail the methodological choices and design decisions made throughout the development of the link quality estimator. We elucidate the architectural framework, the data collection process within the NCM and the training and inference pipelines for the model. Additionally, we describe the integration of a web client application for visualizing both the gathered parameters and the predicted link quality estimations.

2.1 Code distribution to NCMs via Continuous Integration and Delivery Pipeline

During the design and implementation of the required code structures for this project, we observed a lack of a suitable software engineering workflow for authoring and deploying the code to the NCMs. Specifically, when concurrently developing the *libmoep-ncm* on two NCMs, there is a necessity for constant synchronization of the code base between both devices. Consequently, in our project, we opted to containerize the entire software stack and employ a CI/CD pipeline, utilizing GitHub Actions[1], for the integration, construction, and deployment of the artifacts to the NCMs.

2.1.1 Containerization

The initial phase of the procedure entailed containerizing all employed software modules utilizing Docker[2]. For most high-level application software, this task is relatively straightforward (e.g., our LQE inference frontend and backend); however, containerizing the *libmoep-ncm* library presents a more complex challenge. This complexity arises because *libmoep-ncm* requires access to low-level interface file descriptors, which are typically unavailable within a container. Consequently, in this project, we employed ptrace[3] to debug the library, which enabled tracing and determination of the library's necessary access capabilities. To utilize ptrace, it is first essential to activate it by adding the *SYS_PTRACE* capability to the corresponding *libmoep-ncm* container.
As a result of our investigation, we assigned the following settings to the *libmoep-ncm* container:

- *network_mode = host*: Enables a container to share the host network stack.
- *privileged = true*: Permits a container to operate with extended privileges, effectively providing the same level of access to the host system as a process executing outside the container.

It is crucial to emphasize that these settings are not recommended for production software systems and may pose potential risks. Nevertheless, they are sufficient for the purposes of this research project.
After applying the respective configurations, the *libmoep-ncm* library can operate within a Docker container, given a Dockerfile with a base image of *debian:latest* and the necessary package dependencies (e.g., *automake*). Additionally, to streamline the *libmoep-ncm* setup process, we devised a *setup.sh* script that executes the required operations for configuring the *libmoep-ncm* library and, if provided with a suitable parameter, the underlying *libmoep* library. It is important to note that the *libmoep-ncm* Dockerfile doesn't start the container but only let it run for eternity. The reason for this is that a Docker container only lives as long as the underlying process started, which isn't wanted in our case. Therefore, we start the *libmoep-ncm* library in a separate step in the pipeline, as can be seen in the next section.

[1] GitHub Actions is an advanced tool for automating software workflows on the widely-used code hosting platform, GitHub. With GitHub Actions, developers can effortlessly create custom workflows to build, test, and deploy their code in a consistent and reliable manner. For more information, refer to the official GitHub Actions documentation at `https://docs.github.com/en/actions`.

[2] Docker is a widely-used containerization platform for developing, packaging, and deploying applications. For additional details, refer to the official Docker website at `https://www.docker.com/`.

[3] ptrace is a system call utilized in Unix-based operating systems for tracing and debugging processes. It can be used to monitor and modify a process's memory, registers, and system calls. For more information, consult the ptrace manual page or the Linux Programmer's Manual at `https://man7.org/linux/man-pages/man2/ptrace.2.html`.

2.1.2 CI/CD pipeline

The subsequent stage involved establishing a CI/CD pipeline to integrate, build, and deploy our software artifacts to the NCMs. These artifacts primarily consist of the *libmoep-ncm* library and the corresponding Link Quality Estimation frontend and backend.
To configure a pipeline, the initial step is the construction and deployment of the artifacts, which is simplified by the fact that our setup is already containerized. To facilitate development, we also execute specific trigger operations within the pipeline, such as initiating the *libmoep-ncm* library with particular parameters and starting a continuous *ping* command between the different NCMs.
The individual sequential tasks within the pipeline are as follows:

- **build-push libmoep-ncm** / **server**: Constructs and pushes the respective containers to DockerHub.
- **deploy libmoep-ncm-1** / **libmoep-ncm-2**: Deploys the artifacts to the various NCMs.
- **start-libmoep**: Launches the *libmoep-ncm* library inside the deployed Docker container on the NCMs. This is accomplished using a *tmux*[4] session, preventing pipeline blocking or the termination of the *libmoep-ncm* library process. Moreover, this approach enables proper session monitoring. The command to initiate the library can be modified to perform specific actions.
- **start-ping**: Activates a continuous ping command between the NCMs, executed within a *tmux* session.

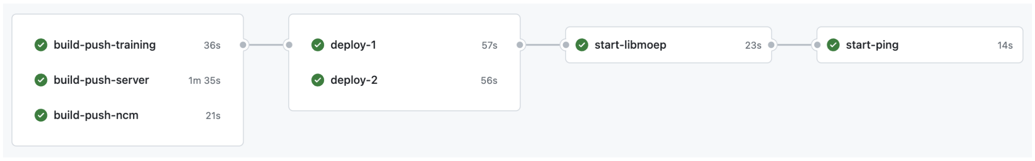

Figure 1: Jobs of the CI/CD pipeline

Upon the pipeline's completion, the various NCMs are equipped with the relevant software, which is started accordingly (particularly the *libmoep-ncm* library), and a continuous network connection between the NCMs is established via the ping command.

2.2 Network Coding Module (NCM)

A critical component of this project involves adapting the *libmoep-ncm* library to gather essential link connection statistics between NCMs, transmit the data to a Link Quality Estimation (LQE) evaluation backend through a socket, and conduct a connection test at startup. To achieve this, we introduced several new arguments to the *libmoep-ncm* library, which enable and configure the specific functionalities:

- **Link Quality Statistics Collection (-l <PORT>):** This argument activates link quality statistics collection and transmits all formatted data through a TCP socket operating on localhost (the LQE backend) on a designated port (specified by the PORT argument). The transmitted link quality statistics consist of the following high-level items:

[4]tmux is a terminal multiplexer that allows users to manage multiple terminal sessions within a single window. It provides features such as session management, window splitting, and session detachment/reattachment, enabling efficient and flexible terminal usage. For more information, consult the official tmux documentation at `https://github.com/tmux/tmux/wiki`.

  - Session Data: Pre-collected by the *libmoep-ncm* library for each session, critical data includes uplink and downlink, RALQE, and transmitted redundancy.
  - Radiotap Header Data: Appended to each frame, the radiotap header is removed or added by the kernel during transmission or reception of the frame. It contains information regarding frame transmission details, valuable for link quality estimation. Notable data points include signal strength, noise, and transmission power.
  - Additional Data: Useful supplementary information such as Master/Slave indication and remote address.

  For an in-depth description of the collected data points, refer to the code, particularly the *lqe.c* file and struct definitions in *lqe.h*.

- **Startup Connection Test (-c <IP>):** When this flag is enabled, the *libmoep-ncm* library conducts a connection test to the specified IP address (provided by the IP argument) during startup. The established connection to the respective IP address is then monitored by the link quality statistics collection, and the gathered data is transmitted to the LQE backend for baselink link quality prediction (assuming stationary stations). The test data consists of a simple *ping* command, transferring 1000B ICMP packets to the specified IP address. The packet size, chosen cautiously below the typical 1500B MTU, prevents the packet from being divided into smaller segments. Although the initial connection test does not currently facilitate dynamic parameter adjustment, this functionality is proposed for future work. (Note: The *-l PORT* argument is required for this flag).

- **Predicted Link Quality Estimation Reception from Backend (-q):** This flag allows for receiving predicted link quality estimations from the LQE backend. As the *-l PORT* parameter is a prerequisite for this flag, the library listens for incoming predictions on the bidirectional TCP socket and outputs the values to *stdout*. In the future, this feature could be employed to obtain LQE predictions for the current connection (e.g., at startup via a connection test using the *-c IP* flag) and adjust connection parameters based on these LQE predictions.

2.3 Dataset and Model

After investigating different datasets listed in [2] we decided upon using the Rutgers trace-set [3]. The trace-set includes 4,060 link traces from 812 unique links with 5 noise levels. Figure 2 shows the setup used for capturing these traces. Features include RSSI, sequence numbers, source and destination node IDs, and artificial noise levels [1]. Based on the findings of [1] we used *RSSI*, which in the case of the Rutgers dataset can be interpreted as signal to noise ratio (SNR), as the training feature and *PRR* as the target value. We used the same mapping as [1] for the target value, namely:

$$f(PRR) = \begin{cases} bad, & \text{if } PRR \leq 0.1 \\ intermediate, & \text{otherwise} \\ good, & \text{if } PRR \geq 0.9 \end{cases} \tag{1}$$

Likewise we have generated synthetic features to improve model performance. Cerar et al. [1] evaluated multiple synthetic features and combinations of those like $RSSI_{avg}$, $RSSI_{std}$ or postive/negative powers of RSSI. Based on their results we have chosen to use $RSSI_{avg}$ with a window size of 10 (using the last 10 received *RSSI* values) in addition to *RSSI* as training features. For the model itself we used a decision tree with a max depth of 4 and random oversampling due to the imbalance of the dataset (underrepresented *intermediate* class). We combined the dataset preparation (e.g. synthetic feature generation) and the model training together in a containerized python executable. One can build and run the container and it will yield a *.joblib* file containing the trained decision tree (dtree) model that can then be further used in inference.

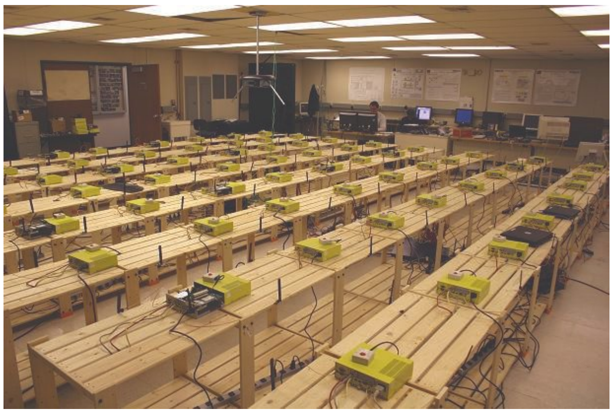

Figure 2: Setup used for capturing the Rutgers trace-set by. The ORBIT indorr testbed contained 64 nodes arranged on an 8 by 8 grid. [3]

2.4 Inference Pipeline

The proposed inference pipeline is designed to run on NCMs with *libmoep* for real-time prediction of Link Quality Estimation (LQE). It has been built using FastAPI [5], a web framework for building APIs with Python.
The pipeline loads the decision tree (dtree) from a file and listens to the specific NCM socket for incoming data every second. Upon receiving the data, the pipeline takes the Received Signal Strength (RSS) value obtained from the NCM and calculates the RSSI. Typically, for Atheros based cards, as found in the NCM, the mapping formula is [4]:

$$RSSI = f(RSS) = RSS + 95 \tag{2}$$

The RSSI value, along with the RSSI average from the last 10 values, make up the inputs for the dtree. The dtree outputs the LQE subsequentially, which is sent back via socket to the NCM. Additionally, the pipeline yields the LQE and RSS values to the dashboard for visualization.

2.4.1 Visualization

In order to monitor the LQE in real time, a dashboard for visualization has been built using Charts.js [6] as part of the inference pipeline (FastAPI server). As the LQE is based on the RSSI value obtained from the NCM, we have chosen to visualize these two parameters as seen in figure 3. The inference pipeline yields the *LQE* together with the *RSSI* and $RSSI_{avg}$ (in dBm) every second.

[5]FastAPI is a Web framework for developing RESTful APIs in Python. Additional information can be found at `https://fastapi.tiangolo.com/`.

[6]Chart.js is a free, open-source JavaScript library for data visualization. For more information refer to `https://www.chartjs.org/`.

3 Results

As mentioned in Section 2, within the scope of this project, we containerized the *libmoep-ncm* library using Docker and established a CI/CD pipeline to automatically integrate, build, deploy, and execute specific commands (such as *ping* or starting the *libmoep-ncm* library) upon the software. Subsequently, we modified the *libmoep-ncm* library to collect current link statistics and transmit them to our LQE backend. This backend then evaluates the received data using a decision tree, a model sufficiently capable for these types of operations, as detailed in [2]. The predicted LQE values are sent back to the NCMs for potential parameter adjustment (not implemented in our project). Additionally, the *libmoep-ncm* library now includes parameters for conducting a connection test at startup and receiving predicted LQEs from the backend. Furhtermore, the LQE prediction backend also offers a frontend visualization to follow the received statistics as well as the predicts LQEs in real time.

Although link statistics collection, socket communication, and the inference pipeline perform as expected for the most part, a persistent issue is the mapping between Received Signal Strength (RSS) values (in dBm) and Received Signal Strength Indicator (RSSI) values. We believe this factor is still limiting the LQE model performance. The challenge lies in the fact that the decision tree was trained using the Rutgers dataset, which mainly features RSSI values ranging from 0 to 40 (worst to best connection, respectively). This is because the Atheros cards employed to capture the trace-set utilize an RSSI that can be interpreted as a signal-to-noise ratio (SNR) rather than signal power (raw RSS) [3]. Conversely, the NCM provides raw RSS values in dBm to the inference server, with values ranging approximately between -30 dBm to -90 dBm (best to worst connection, respectively).
Using the conversion formula in Equation 2 from Section 2.4 for RSSI conversion results in overly optimistic LQEs, as can be observed in Figure 3. Almost none of the LQEs fall within the *bad* class.

4 Discussion

Our choice of a relatively simple machine learning model, specifically a decision tree, was influenced by the findings of Cerar et al. They demonstrated that employing more complex machine learning models, such as Neural Networks, may not necessarily result in significant improvements in model performance. Instead, the dataset's quality and attributes have been shown to play a crucial role in determining the model's effectiveness [1]. Despite the promising nature of the utilized trace-set, as well as its alignment with the work of Cerar et al. and their evaluations, the primary issue remains the conversion of RSSI values, as mentioned in Section 3.
As described in Section 2.2, we collect specific link statistics, primarily from session statistics and radiotap header data, to perform LQE prediction. Future work could explore additional data points that might be beneficial for LQE prediction while ensuring the availability of public datasets reflecting these data points to train our classifier model [2].
Another intriguing aspect is the behavior of the initial connection test at startup, which can be configured through a parameter in the *libmoep-ncm* library. The connection test is carried out in a thread that executes the *ping* command to a specific IP address. However, in some scenarios, the connection established via this *ping* command triggered by the *libmoep-ncm* library fails. This is surprising, as a separate tmux session can successfully establish the connection. Further investigation is required in this area.

5 Conclusion

In conclusion, this project has successfully achieved the containerization of the *libmoep-ncm* library using Docker and the establishment of a CI/CD pipeline for seamless integration, building, deployment, and execution of specific commands (e.g., *ping* or starting the *libmoep-ncm* library). We adapted the *libmoep-ncm* library to collect real-time link statistics and transmit them to our LQE backend, which in turn employs a decision tree model for data evaluation, as supported by [2]. The project has not yet implemented parameter adjustments based on the predicted LQE values sent back to the NCMs, but the *libmoep-ncm* library now features parameters for initiating a connection test at startup and receiving predicted LQEs from the backend. Additionally, the LQE prediction backend provides a frontend visualization for real-time monitoring of the received statistics and predicted LQEs, further enhancing the system's capabilities.
Throughout the project, we present the results of our link quality estimator built using a decision tree approach, drawing on the work of Cerar et al. and outlining key steps in our inference pipeline. Although the proposed system serves as a starting point in this area of research, there are several dimensions for potential improvement. One approach involves generating a new dataset based on captured NCM traces, enabling the use of the same RSSI metric for training and inference without requiring conversion. Alternatively, more sophisticated methods for RSSI conversion could be explored to enhance the system's LQE accuracy.

References

[1] Gregor Cerar, Halil Yetgin, Mihael Mohorcic, and Carolina Fortuna. On designing a machine learning based wireless link quality classifier. 08 2020.

[2] Gregor Cerar, Halil Yetgin, Mihael Mohorcic, and Carolina Fortuna. Machine learning for wireless link quality estimation: A survey. *IEEE Commun. Surv. Tutorials*, 23(2):696–728, 2021.

[3] S.K. Kaul, M. Gruteser, and I. Seskar. Creating wireless multi-hop topologies on space-constrained indoor testbeds through noise injection. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 10 pp.–521, 2006.

[4] Inc. WildPackets. Converting signal strength percentage to dbm values. `https://d2cpnw0u24fjm4.cloudfront.net/wp-content/uploads/Converting_Signal_Strength.pdf`, 2002. Accessed: 2023-04-08.

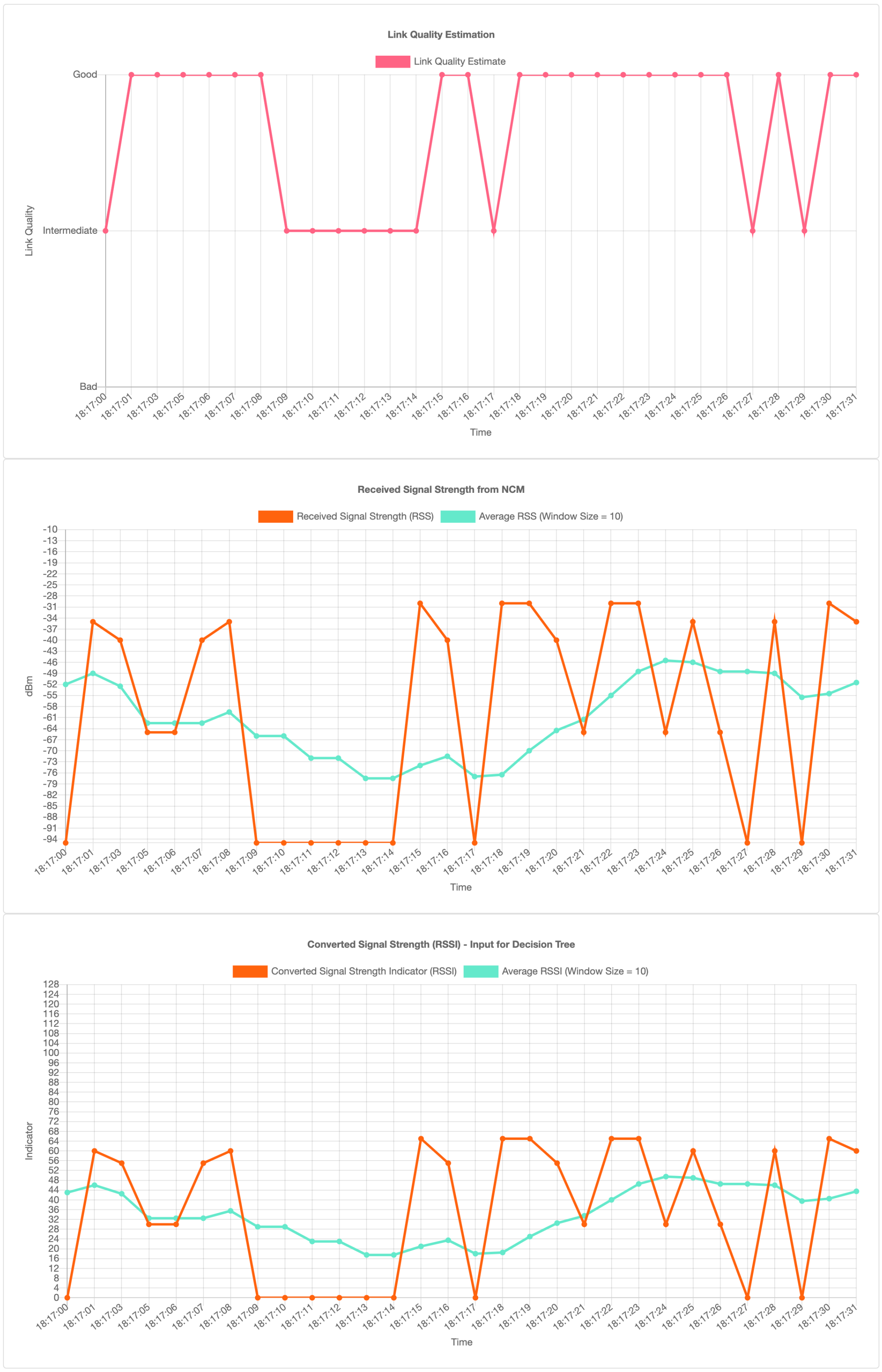

Figure 3: Dashboard for real time LQE monitoring running on the NCM. RSS values randomly generated by our mock-NCM implementation for benchmarking purposes.