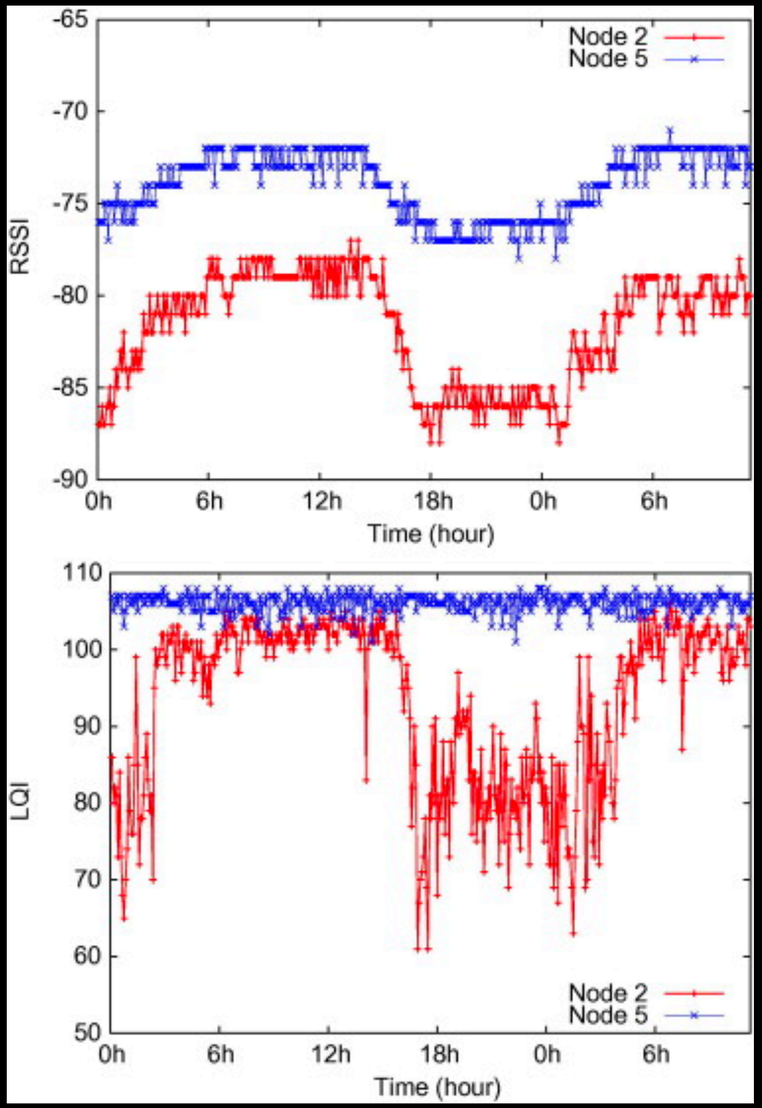
Utilizing Machine Learning to estimate Link Quality

Network Coding WS22/23

Philipp Zagar & Nikolai Madlener

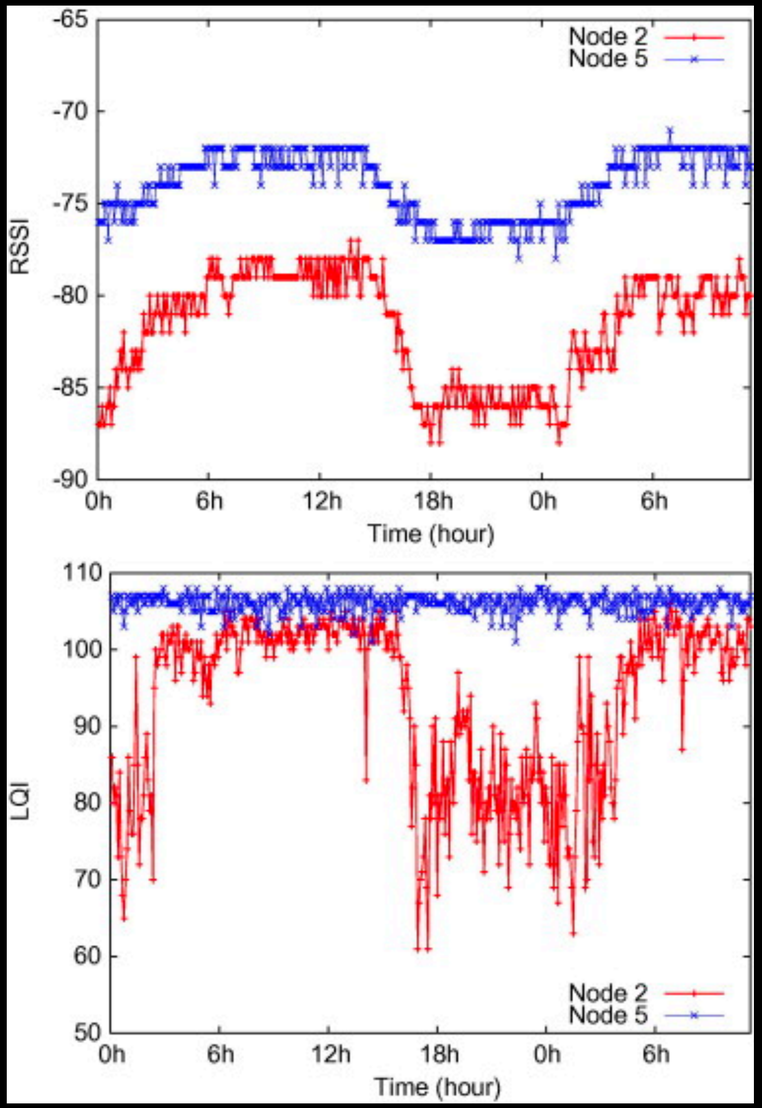
April 14, 2023

- Wireless networks face **varying propagation channel conditions**, affecting link quality
  - Influenced by wireless channel, physical layer technology, and link layer design decisions



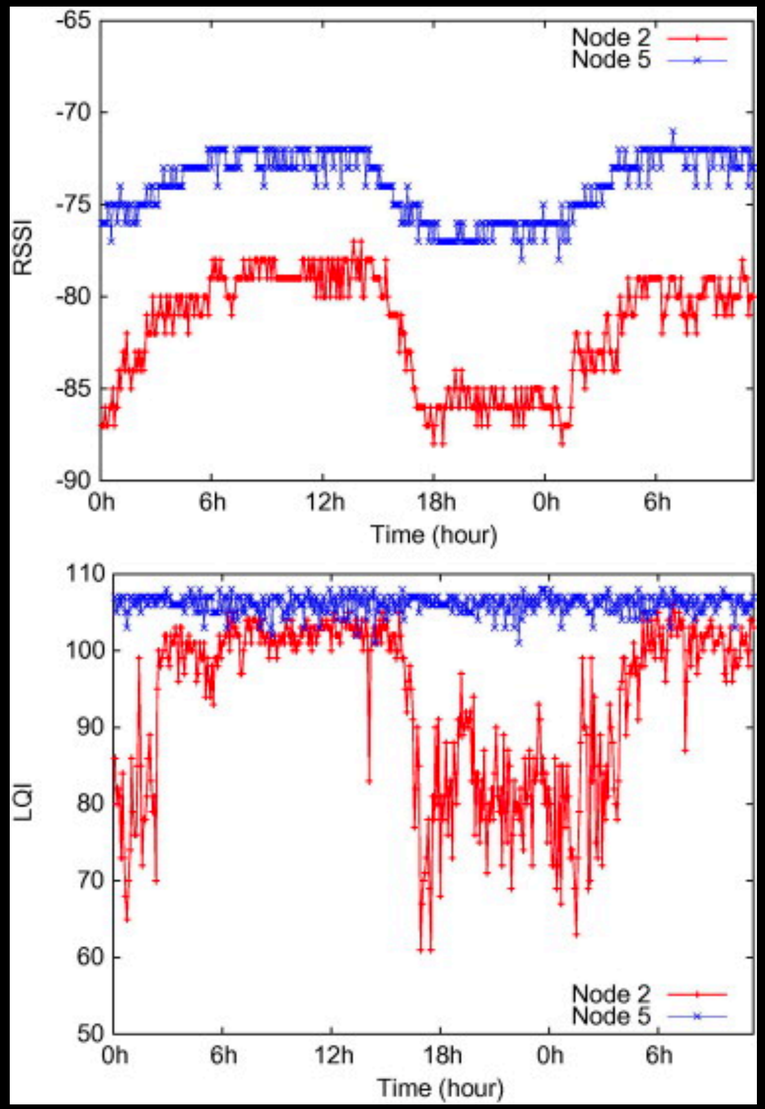
▲ ~/introduction

- Wireless networks face **varying propagation channel conditions**, affecting link quality
  - Influenced by wireless channel, physical layer technology, and link layer design decisions
- Link quality estimation (**LQE**) helps **adapt parameters** and select reliable channels for data transmission



▲ ~/introduction

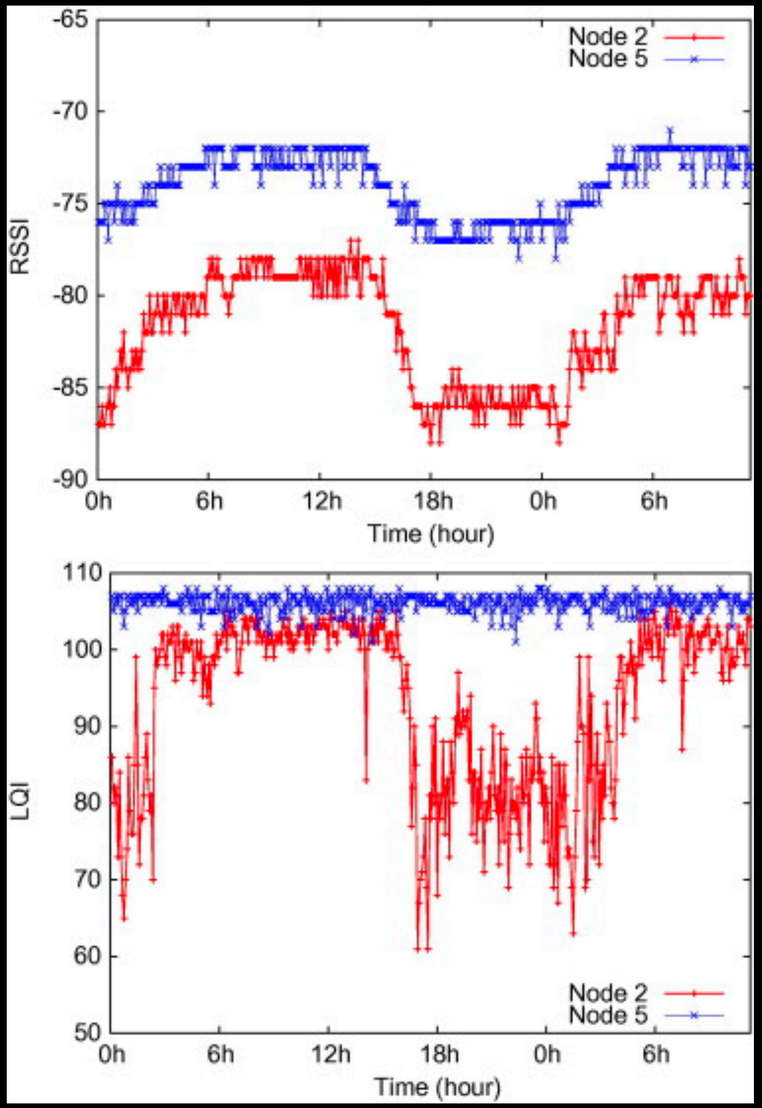
- Wireless networks face **varying propagation channel conditions**, affecting link quality
  - Influenced by wireless channel, physical layer technology, and link layer design decisions
- Link quality estimation (**LQE**) helps **adapt parameters** and select reliable channels for data transmission
- Effective LQE can **improve network** throughput, lifetime, and overall connectivity



▲ ~/introduction

- Wireless networks face **varying propagation channel conditions**, affecting link quality
  - Influenced by wireless channel, physical layer technology, and link layer design decisions
- Link quality estimation (**LQE**) helps **adapt parameters** and select reliable channels for data transmission
- Effective LQE can **improve network** throughput, lifetime, and overall connectivity

➡ **Integration into libmoep-ncm library**



▲ ~/objectives

1. **Capture statistics** of the data link on the NCMs (e.g. RSS)
    i. Either continuously or via a connection test at the beginning

▲ ~/objectives

1. **Capture statistics** of the data link on the NCMs (e.g. RSS)
    i. Either continuously or via a connection test at the beginning
2. Transfer data to an inference pipeline which transforms the data

▲ ~/objectives

1. **Capture statistics** of the data link on the NCMs (e.g. RSS)
    i. Either continuously or via a connection test at the beginning
2. Transfer data to an inference pipeline which transforms the data
3. Based on a ML model, **predict** a **LQE**
    ii. Prerequisite: The model is trained upon a large (available) data set of traces

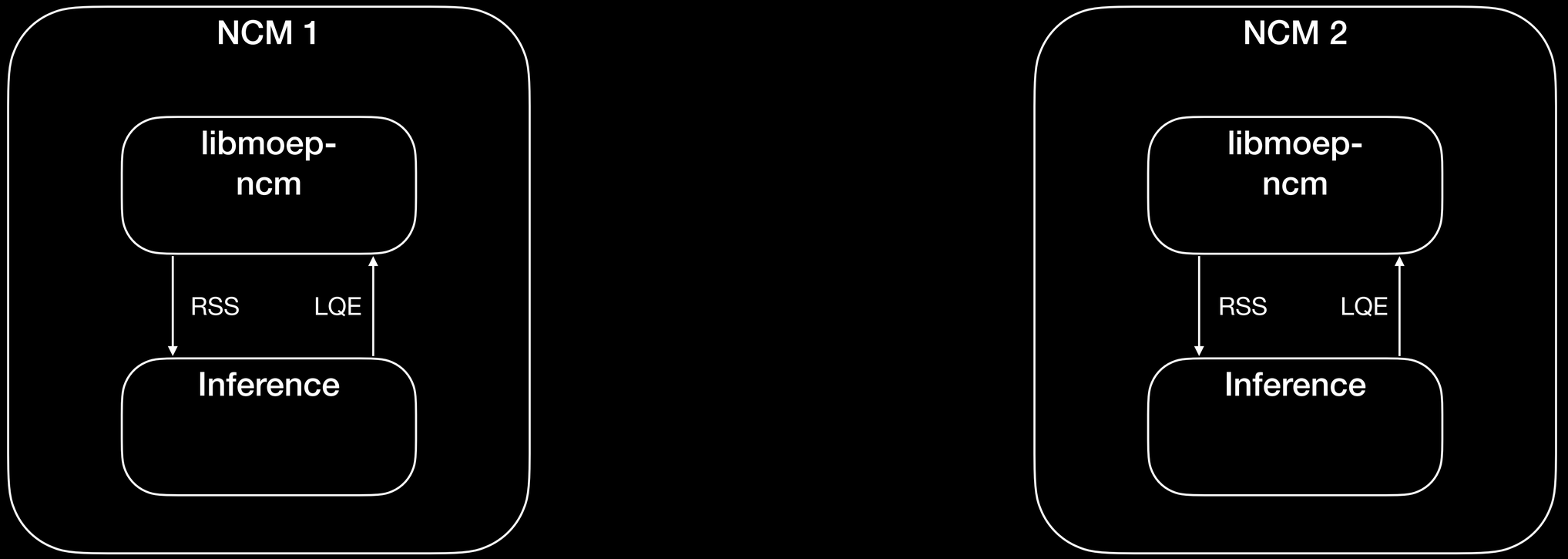
▲ ~/objectives

1. **Capture statistics** of the data link on the NCMs (e.g. RSS)
    i. Either continuously or via a connection test at the beginning
2. Transfer data to an inference pipeline which transforms the data
3. Based on a ML model, **predict** a **LQE**
    ii. Prerequisite: The model is trained upon a large (available) data set of traces
4. The LQE can now be used to adjust wireless parameters within libmoep-ncm

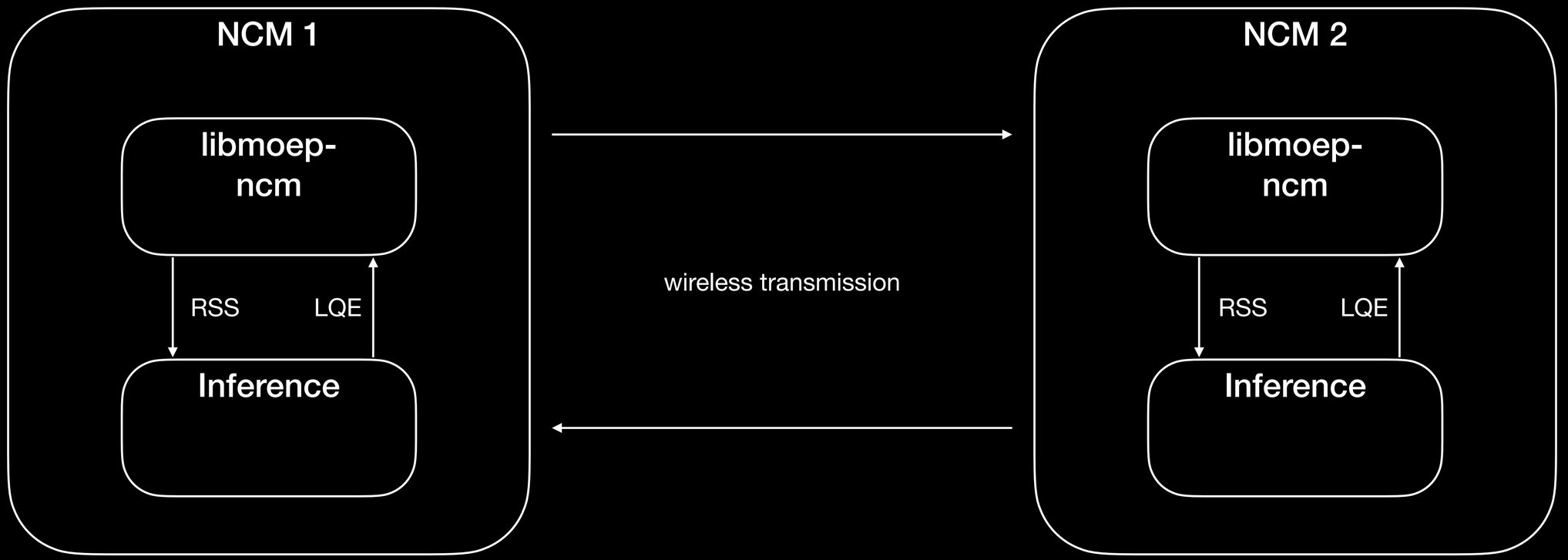
▲ ~/objectives

1. **Capture statistics** of the data link on the NCMs (e.g. RSS)
    i. Either continuously or via a connection test at the beginning
2. Transfer data to an inference pipeline which transforms the data
3. Based on a ML model, **predict** a **LQE**
    ii. Prerequisite: The model is trained upon a large (available) data set of traces
4. The LQE can now be used to adjust wireless parameters within libmoep-ncm
5. (As multiple nodes involved, proper continuous integration / delivery workflow)

▲ ~/architecture

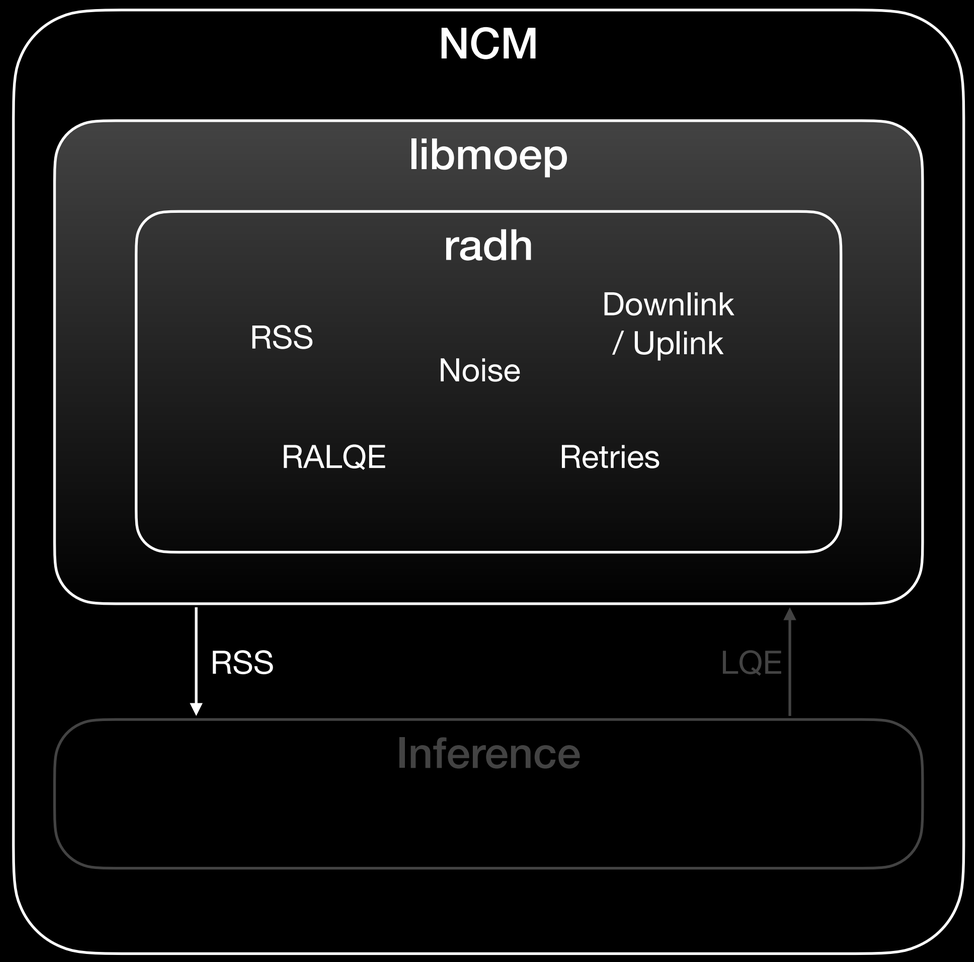


▲ ~/architecture



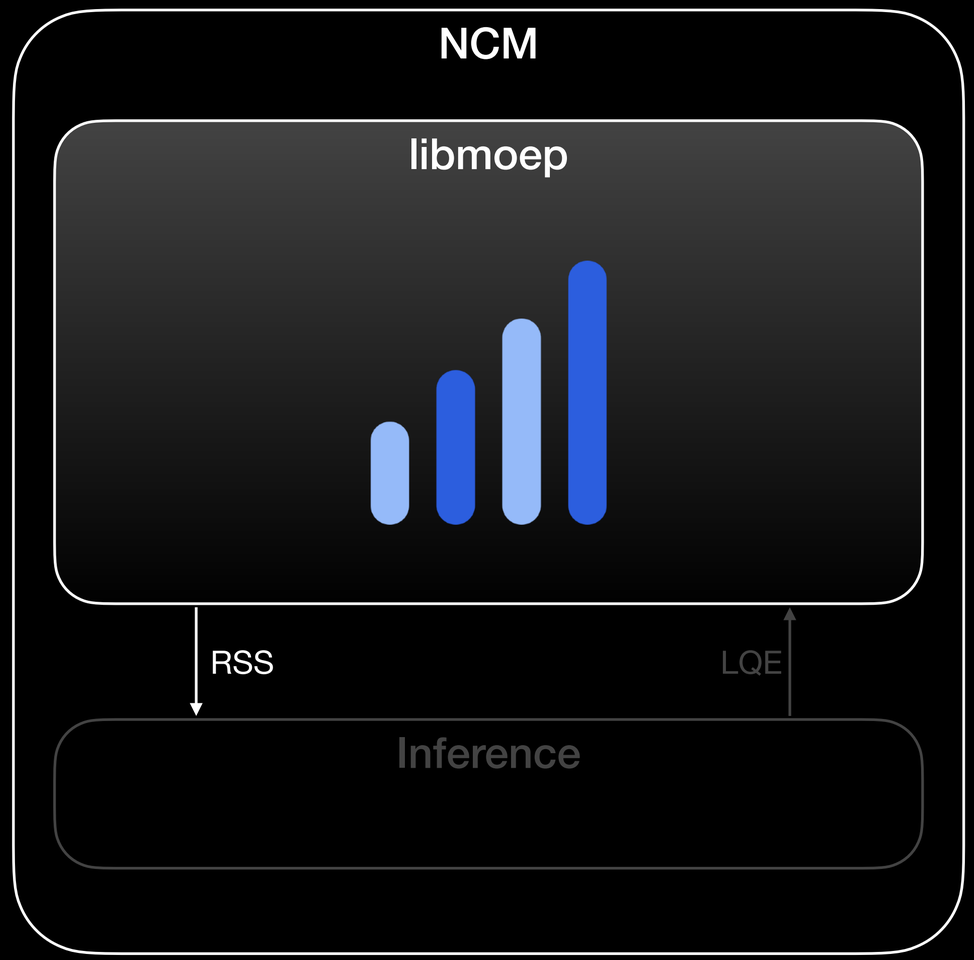
▲ ~/NCM/libmoep/link-statistics

- Capture link statistics within libmoep-ncm in the radh handler (receiving handler)
- Most important data points:
  - **RSS** (Signal strength) [dBm]
  - **Noise** [dBm]
  - RALQE (p/q)
  - Retries
  - Downlink / Uplink
  - …



▲ ~/NCM/libmoep/reproducability

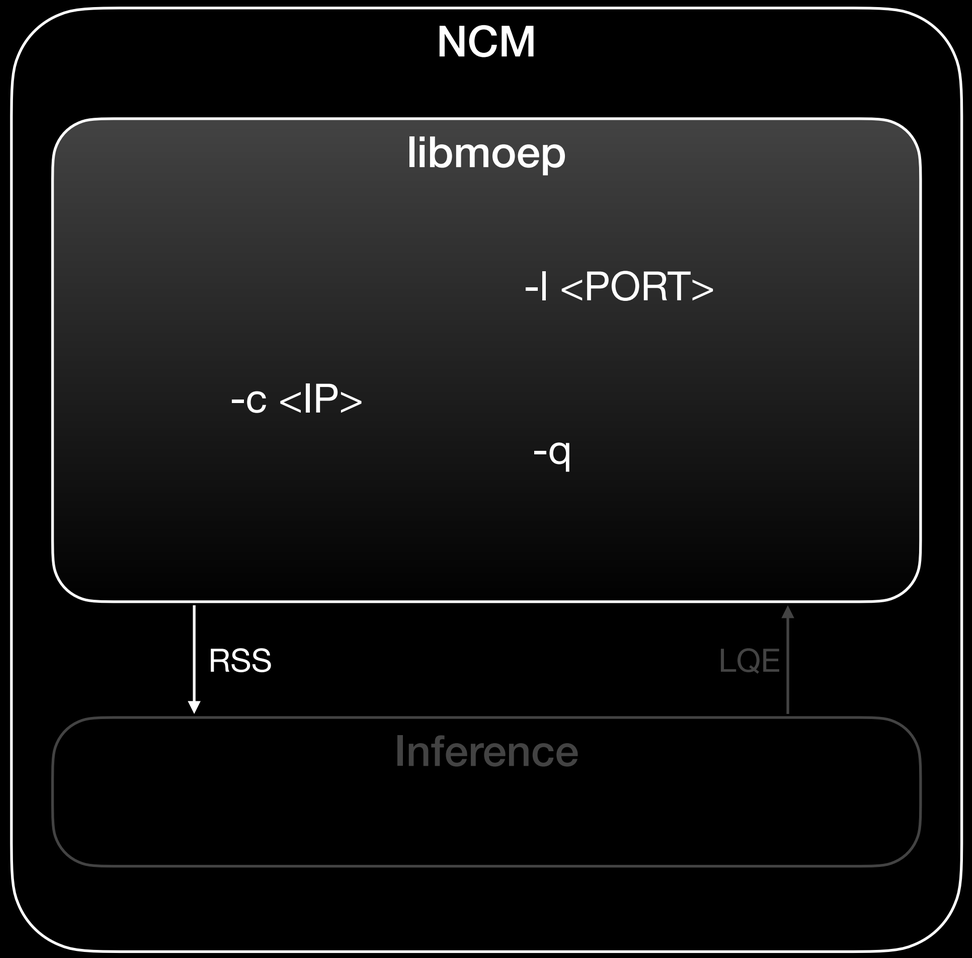
- In order to have reproducible result:
  ➡ Necessary to **fix the transmission power** of the wireless interface
- Set it to -20 dBm
- “iw” is your best friend!



▲ ~/NCM/libmoep/parameters

-l <PORT>

- Starts the collection of link statistics and pushes it via a TCP socket on PORT



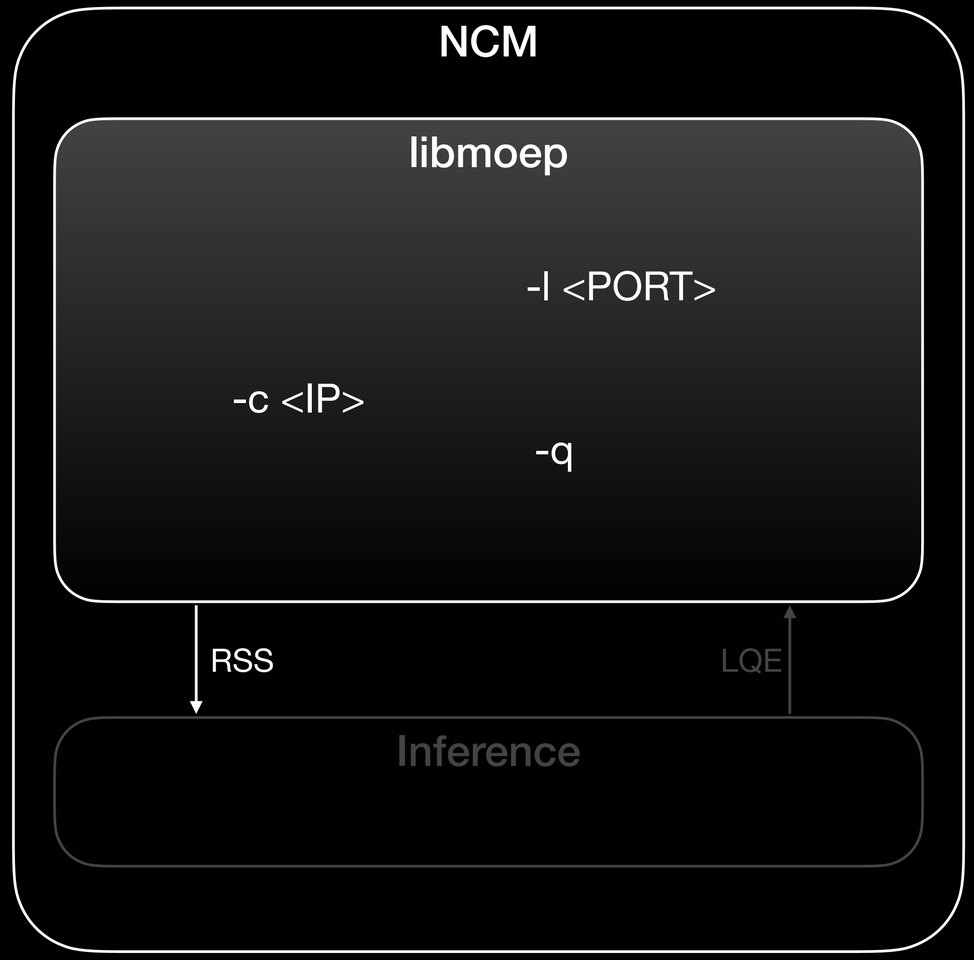
▲ ~/NCM/libmoep/parameters

-l <PORT>

- Starts the collection of link statistics and pushes it via a TCP socket on PORT

-c <IP>

- Performs a connection test upon startup by pinging the IP 5 times with 1kB



▲ ~/NCM/libmoep/parameters

-l <PORT>

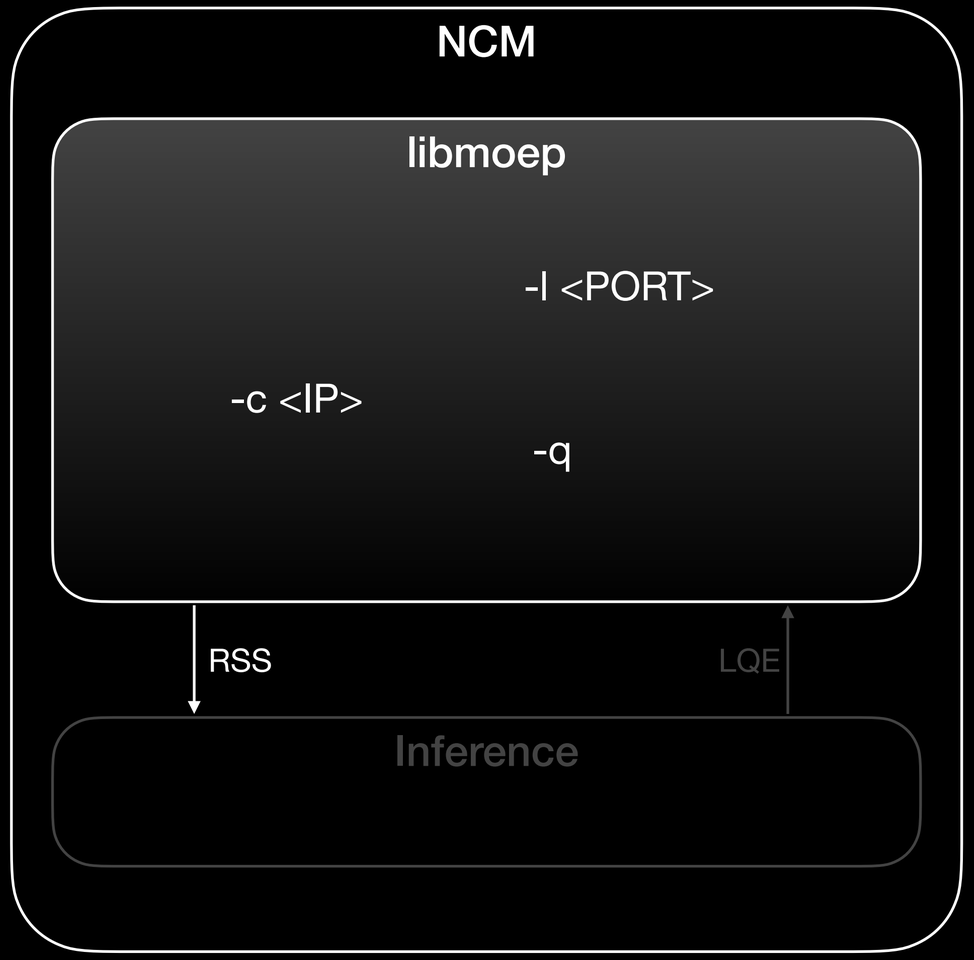
- Starts the collection of link statistics and pushes it via a TCP socket on PORT

-c <IP>

- Performs a connection test upon startup by pinging the IP 5 times with 1kB

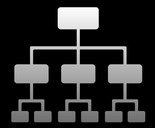
-q

- Receives the LQE from the inference and prints it to stdout



▲ ~/NCM/LQE/model_training

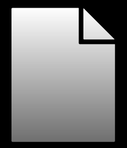
Rutgers trace-set



Decision tree

Training features: RSSI, RSSIavg

3 classes: good, interm., bad

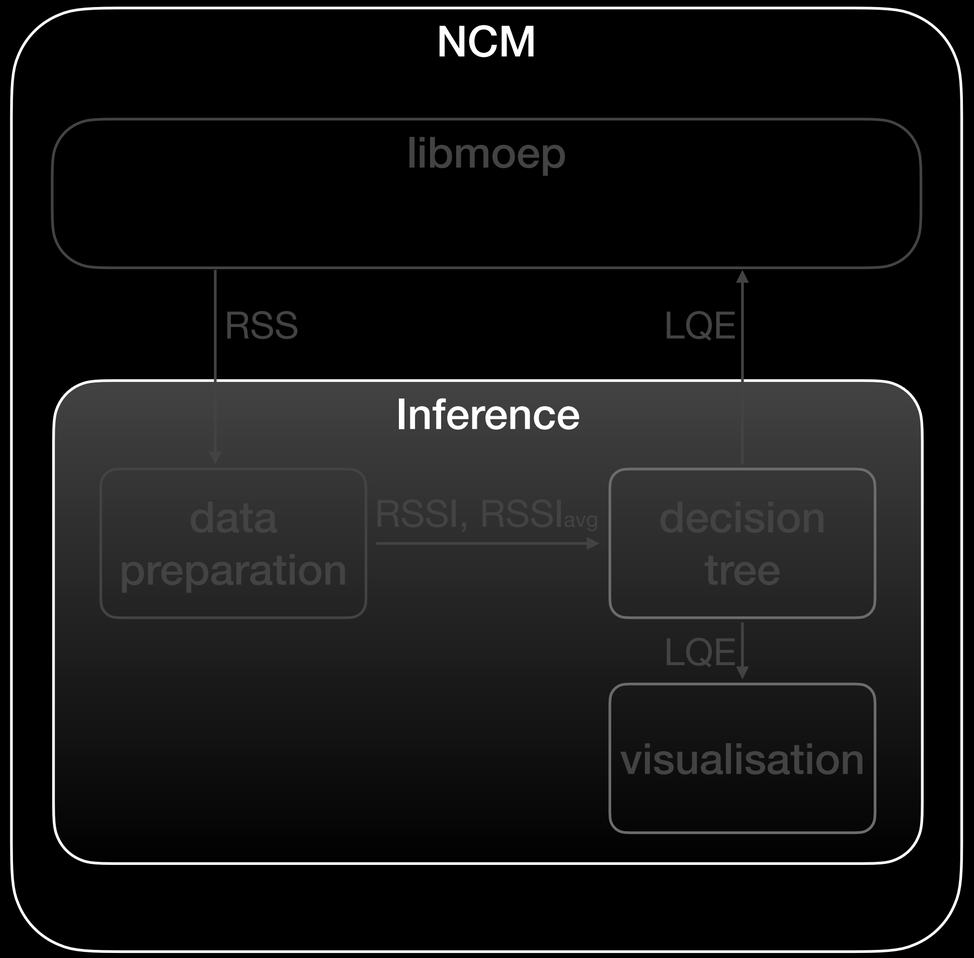


Decision tree saved to file

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

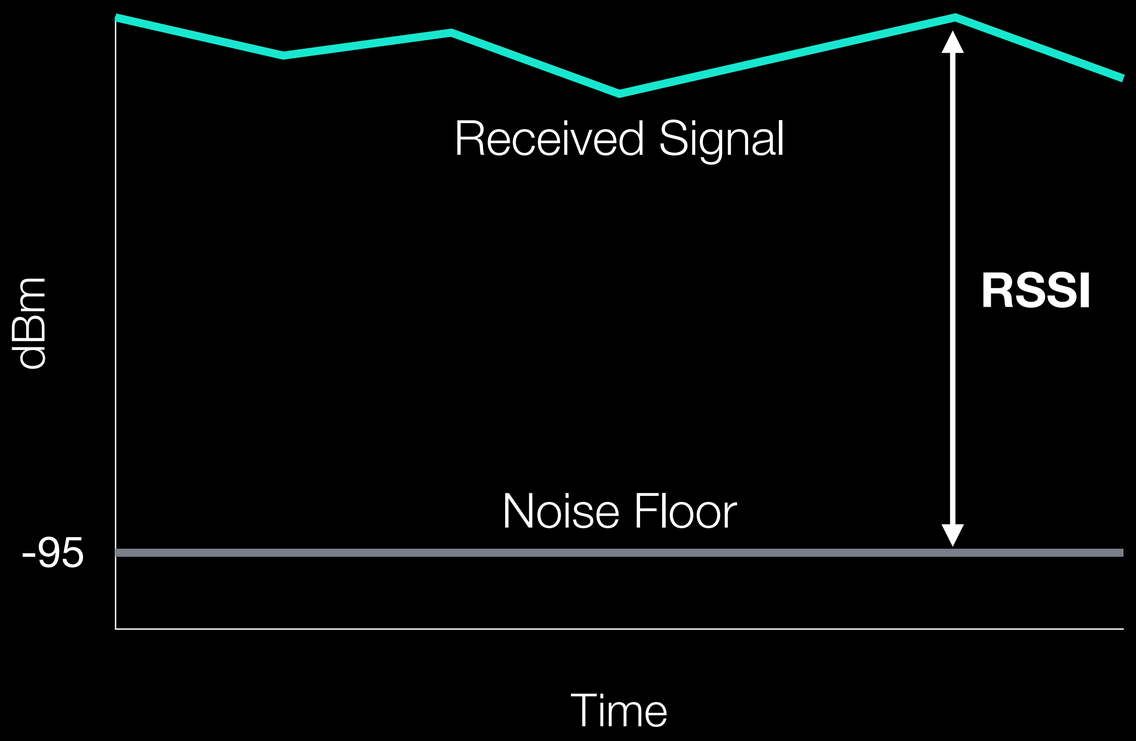
▲ ~/NCM/LQE/inference

Built with Python, FastAPI & ChartJS



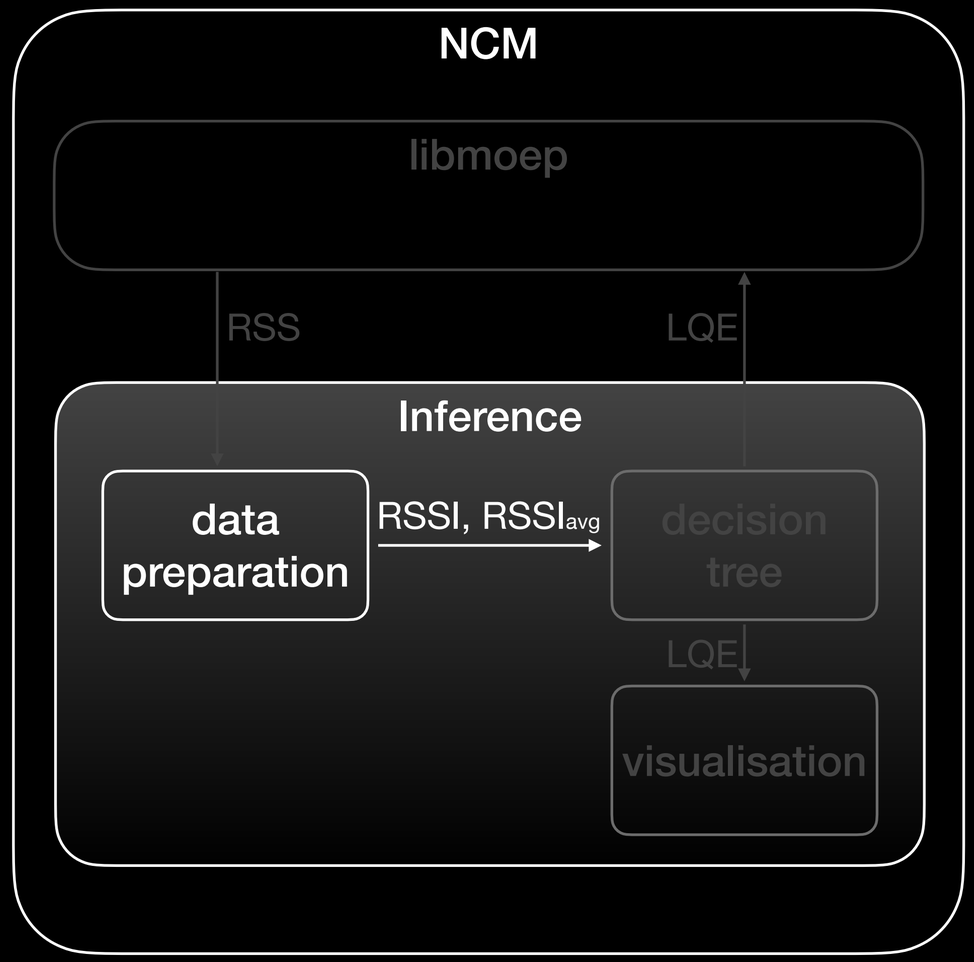
▲ ~/NCM/LQE/inference

Data preparation: RSS -> RSSI, $RSSI_{avg}$

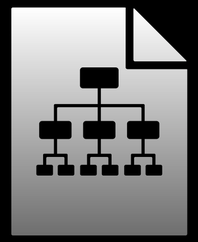


Ex.1: RSSI(-40) = -40 - (-95) = 50

Ex. 2: RSSI(-90) = -90 - (-95) = 5

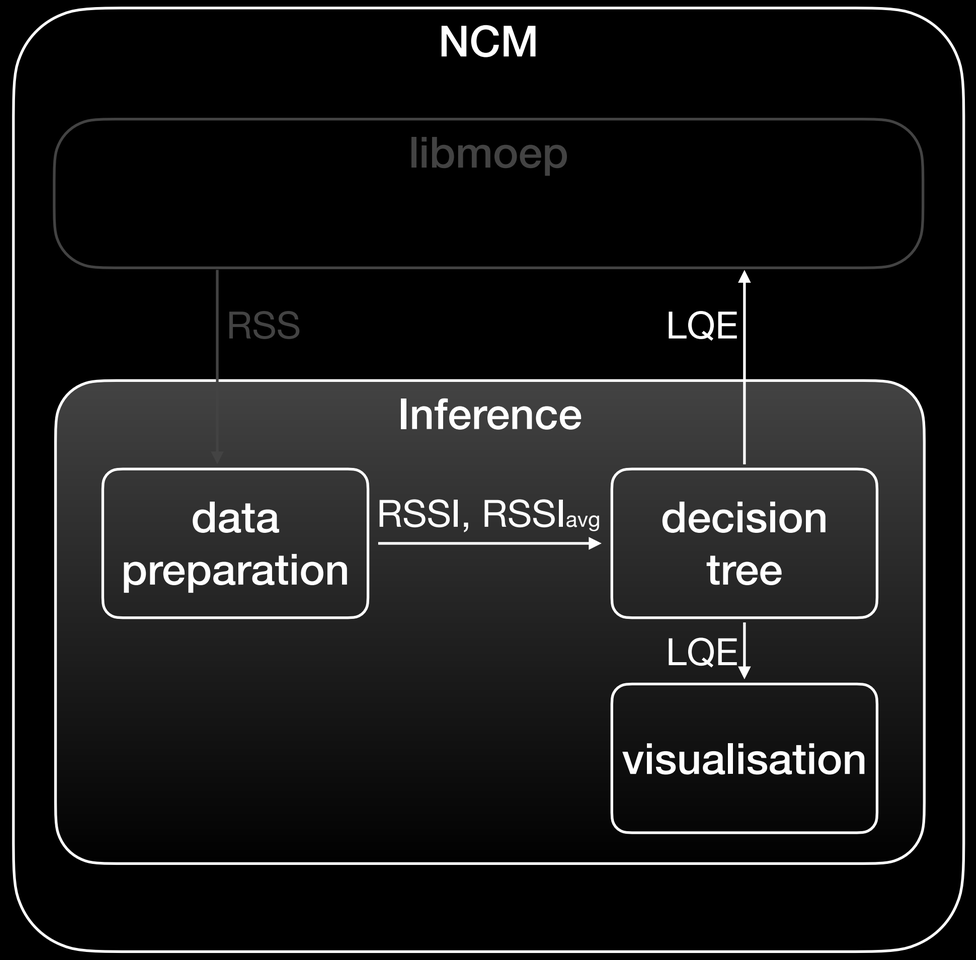


▲ ~/NCM/LQE/inference

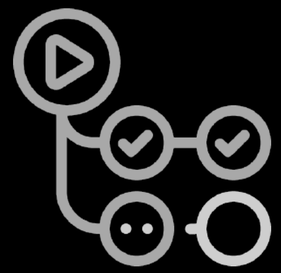


Decision tree loaded from file

LQE sent to libmoep & visualisation

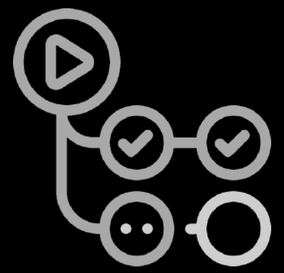


▲ ~/CI_CD



GitHub actions

▲ ~/CI_CD

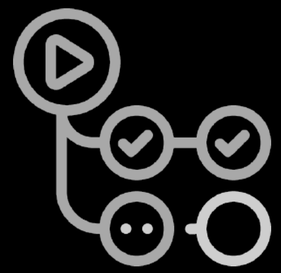


GitHub actions

Containerization

Not trivial for libmoep-ncm

▲ ~/CI_CD



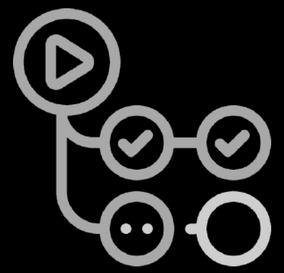
GitHub actions

Containerization

Not trivial for libmoep-ncm

Doesn’t start libmoep upon deployment

▲ ~/CI_CD



GitHub actions

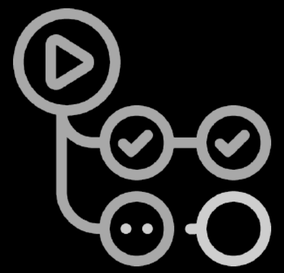
Containerization

Not trivial for libmoep-ncm

Doesn’t start libmoep upon deployment

Starts tmux session libmoep with appropriate parameters

▲ ~/CI_CD



GitHub actions

Containerization

Not trivial for libmoep-ncm

Doesn’t start libmoep upon deployment

Starts tmux session libmoep with appropriate parameters

Starts tmux session with continuous pinging

▲ ~/demo

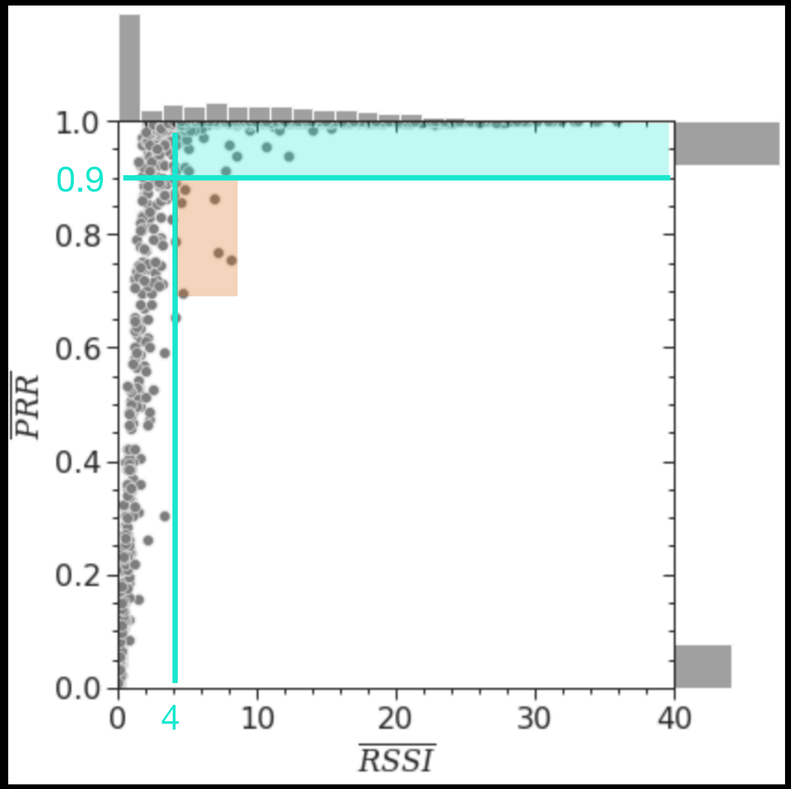
Live demo

▲ ~/challenges

CI/CD

- Proper containerization
- Starting commands via pipeline (tmux sessions)
- Strange behaviours of libmoep-ncm starting on both NCMs but not being able to communicate

RSSI conversion



Ex. 2: RSSI(-90) = 5 => Good

▲ ~/summary

- Capture respective link statistics in libmoep-ncm
- Train our model upon the available Rutgers trace set
- Predict LQE via an inference pipeline and visualize it
- Containerization and CI/CD of the entire software stack
- Laid the groundwork architecture for future wireless parameter adjustment work

▲ ~/future-work

- RSSI conversion
- capture own dataset
- dynamic parameter adjustment (e.g. timeouts)

▲ ~/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.

Questions?