
gr-RT_Channel_Emulator: Ray-Tracing-Based Channel Emulation for Environment-Aware PHY Wireless Research

Jose A. Sanchez Vioria
George Sklivanitis
Dimitris Pados

JOSESANCHEZ2019@FAU.EDU
GSKLIVANITIS@FAU.EDU
DPADOS@FAU.EDU

Center for Connected Autonomy and AI (CA-AI.fau.edu), Florida Atlantic University, Boca Raton, FL, USA

Abstract

Data-driven physical layer (PHY) research increasingly depends on knowing the propagation channel that produced each received sample, yet the channel models available inside software-defined radio (SDR) frameworks are not tied to any physical environment. We present gr-RT_Channel_Emulator, a GNU Radio out-of-tree (OOT) module for position-aware, ray-tracing-assisted channel emulation. The module loads a three-dimensional scene described in Mitsuba XML, places one transmitter and one receiver at user-defined coordinates, and uses Sionna RT to compute propagation paths for a selected carrier frequency, antenna configuration, polarization, and interaction depth. The resulting paths are converted into a discrete complex channel impulse response and applied to streaming in-phase and quadrature (IQ) samples through a hierarchical wrapper around GNU Radio's native channel model. Unlike an offline tracing workflow, in which taps are exported and later matched to a capture, our block publishes the taps it applied as a direct reference output, so that every impaired IQ stream remains paired with the channel that produced it. Node positions are updated at run time through message ports without rebuilding the flowgraph, and an optional lightweight viewer displays the scene together with the traced rays. We demonstrate the module with a flowgraph that recovers the applied taps by least squares (LS) and estimates the received signal-to-noise ratio (SNR) using only standard GNU Radio blocks, and we report snapshot latency across the bundled FAU, POWDER, and AERPAW scenes together with a preliminary comparison against measured POWDER channel estimates. Results suggest that the proposed

module is a practical basis for controlled PHY experiments and for collecting IQ observations with explicit channel labels. The source code for this project is available at:

[https://github.com/
C2A2-at-Florida-Atlantic-University/
gr-RT_Channel_Emulator](https://github.com/C2A2-at-Florida-Atlantic-University/gr-RT_Channel_Emulator)

1. Introduction

GNU Radio is a convenient environment for developing and testing PHY signal processing. Its channel model accepts finite-impulse-response (FIR) taps and adds common impairments, but it does not derive those taps from buildings, radio materials, or node locations (GNU Radio Contributors, 2026). Sionna RT performs the complementary operation: it calculates propagation paths from a three-dimensional scene and TX/RX positions (Hoydis et al., 2023; Aoudia et al., 2025). Connecting the two lets an experimenter move a node and observe the resulting channel change inside a GNU Radio flowgraph.

Our motivation comes from channel-based wireless learning. In gr-PHYSEC we implemented a GNU Radio pipeline that extracts features from channel probes for physical-layer key generation (Sanchez Vioria et al., 2025). Our subsequent channel-informed learning framework trains a feature extractor using both received probes and channel-response targets, and augments training with ray-traced propagation conditions (Sanchez Vioria et al., 2026). Such applications need more than an arbitrary impaired waveform: each observation must be paired with the channel and the experimental conditions that produced it. Producing those pairs today means running a ray tracer offline, exporting taps, and manually loading them into a channel model, which separates the label from the acquisition that used it.

Early GNU Radio work used precomputed ray-traced channels or combined offline ray tracing with stochastic models (Nitsche, 2012; Schmitz et al., 2013). CaST converts

ray-traced scenarios into FIR configurations for FPGA-based emulators (Villa et al., 2022). More recent work connects Sionna RT to ns-3, to cellular stacks, and to broader digital-twin frameworks (Zubow et al., 2024; Pegurri et al., 2025; OpenAirInterface Software Alliance, 2026; Gurses & Sichitiu, 2026). These systems target network-level or infrastructure-scale emulation. We instead target a small component that fits inside a single GNU Radio flowgraph: the experimenter supplies an IQ stream, an environment, and TX/RX positions, and the block returns the impaired IQ stream together with the channel it applied.

The main contributions of this work are as follows:

- An open-source GNU Radio OOT block that applies Sionna RT channel taps to streaming complex-baseband samples and exposes the applied taps as a direct reference output, together with independent TX and RX position message inputs, external socket-based position sources, and a lightweight scene-and-path viewer.
- We describe a demonstration flowgraph in which LS channel estimation and received-SNR estimation are implemented entirely with standard GNU Radio blocks, so that the recovered taps can be compared against the applied reference.
- We report reproducible snapshot-timing measurements across three bundled scenes and a preliminary comparison of simulated against measured channel estimates for the POWDER testbed.

2. gr-RT_Channel_Emulator

We developed the tool as a Python hierarchical GNU Radio block, `RTChannelEmulation`, which wraps `channels.channel_model` as shown in Fig. 1. Each instance represents one directional link. Every new position update triggers a snapshot, in which the backend runs the ray tracer and sets the resulting taps in the channel model. Tracing is kept outside the sample-processing call, so the IQ stream continues to flow on the current taps while a new snapshot is computed.

For each snapshot the backend loads the scene, places the two nodes, runs the path solver, and requests band-limited channel taps from Sionna using the GNU Radio sample rate F_s as the channel bandwidth. Sionna returns one complex coefficient $\bar{h}[\ell]$ per integer lag ℓ in the interval $[l_{\min}, l_{\max}]$. The block keeps that vector unchanged and maps the signed lag onto the causal FIR index expected by the channel model,

$$h^{\text{RT}}[k] = \bar{h}[l_{\min} + k], \quad 0 \leq k < L, \quad L = l_{\max} - l_{\min} + 1, \quad (1)$$

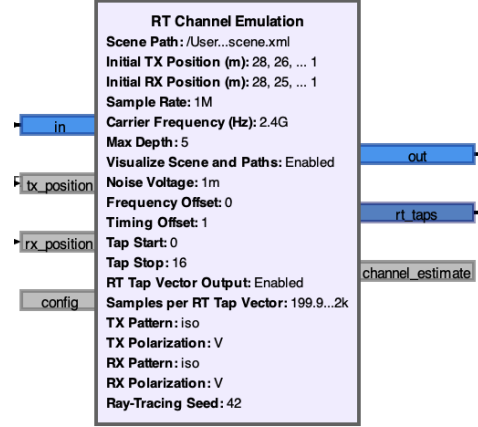


Figure 1. RT Channel Emulation block: one IQ input stream, one impaired IQ output, an optional reference tap stream, and position and configuration message ports.

so that $\mathbf{h}^{\text{RT}} = [h^{\text{RT}}[0], \dots, h^{\text{RT}}[L-1]]^T$ is the vector of channel taps applied to the samples.

The block accepts an input complex signal $x[n]$ from the `in` port and implements the discrete convolution that produces the impaired complex signal $y[n]$ at the `out` port,

$$y[n] = \sum_{k=0}^{L-1} h^{\text{RT}}[k] x[n-k] + w[n], \quad (2)$$

where $w[n]$ is the additive noise term supplied by the channel model (GNU Radio Contributors, 2026). The `noise_voltage` parameter sets σ , the linear complex root-mean-square (RMS) amplitude of $w[n]$, so that $\mathbb{E}[|w[n]|^2] = \sigma^2$.

The environment loader accepts Mitsuba XML scenes, and the solver defaults and material semantics follow Sionna RT 2.0.1 (NVIDIA, 2026b;a). Because a traced channel is only as representative as the assets it is computed from, geometry and material assignments should be verified before a scene is interpreted as a model of a real site.

The `tx_position` and `rx_position` message inputs accept three finite coordinates $[X, Y, Z]$ expressed in scene metres, supplied either as a numeric PMT vector or as a protocol data unit (PDU) whose payload is a UTF-8 JSON array such as $[28.0, 26.0, 1.0]$. Networking is kept outside the emulator so that any position source can be used.

Each node message triggers its own snapshot and updates are serialized and will queue when they arrive faster than the traces complete.

The PyQt5 viewer draws the nodes and the ray segments returned by the same snapshot, as seen in Fig. 2. The dis-

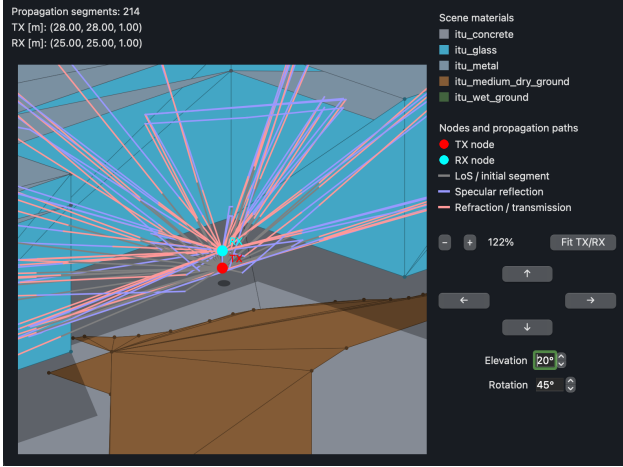


Figure 2. FAU viewer capture at a user-selected node pair.

play supports material, node, and path labels together with zoom, pan, and camera-angle controls.

3. Experimental Demo

We developed a demonstration flowgraph, `channel_estimation_demo.grc`, that estimates the channel taps and the received SNR and compares both against the values the emulator applied. A complex Vector Source repeats a known $N = 255$ bipolar Galois linear-feedback shift-register (GLFSR) sequence $s[n] \in \{-1, +1\}$ through the RT channel emulator block at $F_s = 1$ MS/s. The configuration uses the FAU scene at a carrier frequency of 2.4 GHz, with the transmitter starting at (28, 26, 1) m and the receiver at (28, 25, 1) m, a maximum interaction depth of 5, isotropic vertically polarized antennas, lags 0 through 16 giving $L = 17$ taps, and a starting noise amplitude $\sigma = 0.001$. Amplitude scaling is kept outside the emulator: a transmit multiplier A_T scales the probe before the block and a receive multiplier A_R scales its output, both starting at 1, so that the block input is $x[n] = A_T s[n]$ and the output stream is $v[n] = A_R y[n]$.

The LS estimator operates on $v[n]$, averaged over $M = 8$ probe periods. Defining the probe matrix $X_{n,k} = s[(n - k) \bmod N]$ and the correlation vector $\mathbf{c} = \mathbf{X}^H \mathbf{z}$, where $(\cdot)^H$ denotes the Hermitian transpose, the periodic auto-correlation of the GLFSR sequence makes the LS solution available in closed form,

$$\hat{h}^{\text{RT}}[k] = \frac{1}{N+1} \left(c[k] + \frac{\sum_{j=0}^{L-1} c[j]}{N+1-L} \right), \quad k = 0, \dots, L-1. \quad (3)$$

The resulting estimates are then compared against the ray traced channel taps h^{RT} the emulator applied.

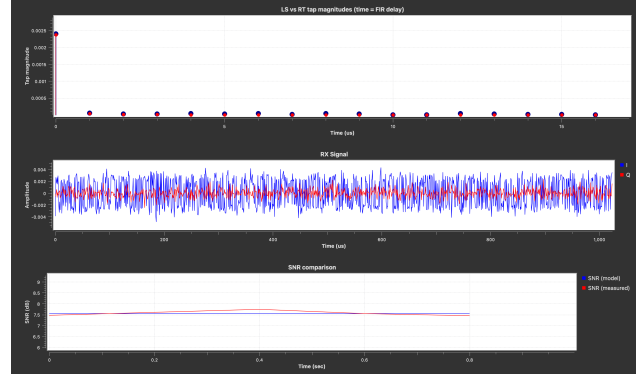


Figure 3. LS tap magnitudes and the direct RT reference for the FAU scene.

For a linear channel the noiseless response repeats every N samples, so the difference $d[n] = v[n] - v[n - N]$ cancels that response and retains twice the noise power. Over matching windows of $W = MN$

$$\hat{P}_n = \frac{1}{2W} \sum_n |d[n]|^2, \quad \hat{P}_t = \frac{1}{W} \sum_n |v[n]|^2, \quad (4)$$

and report

$$\widehat{\text{SNR}}_{\text{dB}} = 10 \log_{10} \frac{\max(\hat{P}_t - \hat{P}_n, \eta)}{\max(\hat{P}_n, \eta)}, \quad (5)$$

where $\eta = 10^{-20}$ is a numerical floor.

The reference signal and noise power are modeled as

$$P_s = A_{\text{TX}}^2 A_{\text{RX}}^2 \sum_{k=0}^{L-1} |h_k|^2, \quad (6)$$

$$P_n = A_{\text{RX}}^2 \sigma^2. \quad (7)$$

For positive signal and noise powers, the reference SNR is

$$\text{SNR}_{\text{dB}} = 10 \log_{10} \left(\frac{P_s}{P_n} \right) \quad (8)$$

Fig. 3 shows the LS tap magnitudes together with the direct RT reference.

4. Experiments and Results

Because every position update triggers a new snapshot, we measured the production snapshot computation call since the time required to compute it bounds how quickly a mobility source may move the nodes. The results cover the FAU, POWDER, and AERPAW scenes at each maximum depth configuration from 3 to 15, with 100 random TX/RX

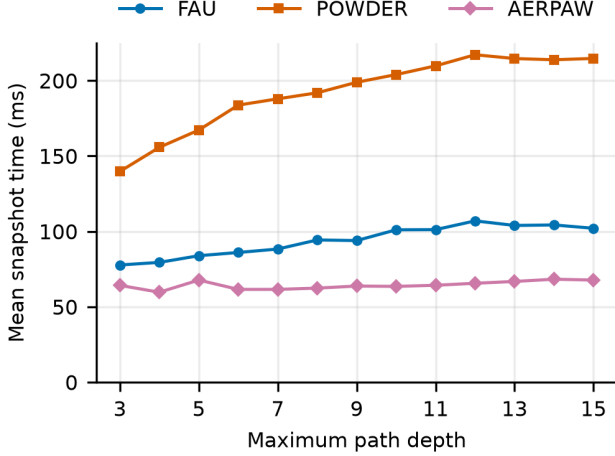


Figure 4. Mean snapshot latency for the three site scenes.

Table 1. FAU, POWDER, and AERPAW scene complexity.

Scene	Shapes	Vertices	Faces	Area (km ²)
FAU	25	596	554	0.1185
POWDER	84	2186	2167	0.7045
AERPAW	5	653	627	29.1600

pairs per scene and depth, and we report the mean snapshot computation time. For each scene the horizontal coordinates are drawn independently and uniformly over the geometry’s XY limits at a fixed height of $z = 2$ m, and the same 100 pairs are reused at every depth configuration.

All profiling configurations use a 2.4 GHz carrier, a 1 MHz sampled bandwidth, one isotropic vertically polarized element per node, and 39 taps ($l_{\min} = -6$, $l_{\max} = 32$), with visualization disabled. Measurements were collected on an Apple M3 Pro on ARM64 with 11 CPU threads, Python 3.13.5, Sionna RT 2.0.1, Dr.Jit 1.3.1, and Mitsuba 3.8.0.

Fig. 4 shows the measured means, while Table 1 summarizes the complexity of each scene. From depth 3 to depth 15, the mean snapshot time increases from 77.74 to 102.07 ms for FAU and from 139.74 to 214.71 ms for POWDER, while AERPAW remains nearly flat, moving only from 64.22 to 67.74 ms. We observe that AERPAW covers by far the largest bounding-box area yet is the fastest scene at every tested depth, while POWDER, which contains the most authored shapes and faces, is consistently the slowest. Neither bounding-box area nor face count alone orders the measured cost, since AERPAW contains more faces than FAU and still traces faster. Scene layout, material assignments, and the sampled links vary together, so these results suggest that the density structures structure near the link, rather than the overall size of the map, domi-

nates snapshot time.

We also compared the ray-traced channel against measured channel estimates by simulating and running over-the-air (OTA) channel estimates on two POWDER nodes, EBC and Guest House (?). We compared each direction against one static RT snapshot computed at the matching coordinates. Both sources use 16 taps at 1 MHz with lags 0 through 15, and the RT configuration uses maximum depth 15, seed 42, isotropic vertically polarized antennas, and no added noise, CFO, or timing mismatch. For each record and for the reference we normalize the tap magnitudes to unit energy, giving power-delay profiles $p_i[k]$ and $q[k]$ that each sum to one over the 16 taps. Fig. 5 plots $\sqrt{\frac{1}{K} \sum_i p_i[k]}$ for the K measured records against the normalized reference magnitude $\sqrt{q[k]}$.

5. Reproducibility

The OOT block, the example flowgraph, the pre-configured scenes, the example position sources, and the profiling tests are available at https://github.com/C2A2-at-Florida-Atlantic-University/gr-RT_Channel_Emulator. The project installs from `pyproject.toml` and requires Python 3.10 or newer, GNU Radio 3.10 with Python bindings and Qt GUI and network blocks, PyQt5, NumPy 1.26 or newer, and Sionna RT 2.x.

6. Conclusion and Future Work

gr-RT_Channel_Emulator delivers a practical, reproducible way to apply a scene-derived wireless channel to a GNU Radio stream while publishing the channel it applied. A demonstration flowgraph recovers those taps by least squares and estimates received SNR. Snapshot latency measured over 3900 traces across three bundled scenes bounds the usable position-update rate, and a preliminary comparison against measured POWDER channel estimates shows a shared dominant arrival together with a heavier measured tail.

Future work will focus on calibrating the scene geometry and antenna assumptions against real testbed channel-sounder data and using the developed block for channel-informed wireless research.

References

- Aoudia, Fayçal Aït, Hoydis, Jakob, Nimier-David, Merlin, Nicolet, B., Cammerer, Sebastian, and Keller, Alexander. Sionna RT: Technical report. Technical Report arXiv:2504.21719, NVIDIA, 2025. URL <https://nvlabs.github.io/sionna/rt/tech-report/>.

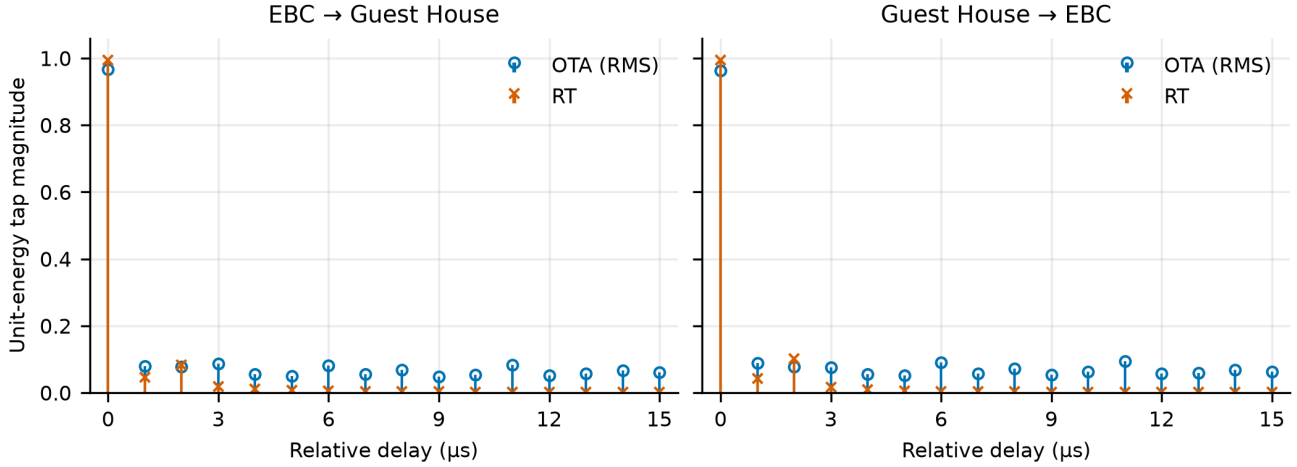


Figure 5. POWDER tap-magnitude comparison for both EBC/Guest House directions.

GNU Radio Contributors. GNU Radio channel model: Api reference, 2026. URL https://www.gnuradio.org/doc/doxygen/classgr_1_1channels_1_1channel_model.html. Accessed September 2026; implementation tested with GNU Radio 3.10.12.

Gurses, Anil and Sichitiu, Mihail L. ACHEM: A real-time digital twin framework with channel and radio emulation. *arXiv preprint arXiv:2604.04742*, 2026. URL <https://arxiv.org/abs/2604.04742>.

Hoydis, Jakob, Aoudia, Fayçal Aït, Cammerer, Sebastian, Nimier-David, Merlin, Binder, Nikolaus, Marcus, Guillermo, and Keller, Alexander. Sionna RT: Differentiable ray tracing for radio propagation modeling. In *2023 IEEE Globecom Workshops*, pp. 317–321, 2023. doi: 10.1109/GCWkshps58843.2023.10465179.

Nitsche, Thomas. A tool for raytracing based radio channel simulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2012. doi: 10.4108/icst.simutools.2011.245555.

NVIDIA. Sionna RT documentation: Radio materials, 2026a. URL https://nvlabs.github.io/sionna/rt/api/radio_materials.html. Accessed September 2026.

NVIDIA. Sionna RT documentation: Path solvers, 2026b. URL https://nvlabs.github.io/sionna/rt/api/paths_solvers.html. Accessed September 2026; this paper reports the settings used with version 2.0.1.

OpenAirInterface Software Alliance. Raytracing channel emulator. <https://openairinterface.org/ran/>, 2026. Accessed 21 August 2026.

Pegurri, Roberto, Linsalata, Francesco, Moro, Eugenio, Hoydis, Jakob, and Spagnolini, Umberto. Toward

digital network twins: Integrating Sionna RT in ns-3 for 6g multi-rat network simulations. *arXiv preprint arXiv:2501.00372*, 2025. URL <https://arxiv.org/abs/2501.00372>.

Sanchez Vilorio, Jose A., Sklivanitis, George, and Pados, Dimitris. gr-PHYSEC: Real-time channel-based key generation for physical layer secure wireless communications. In *GNU Radio Conference*, 2025. URL <https://events.gnuradio.org/event/26/contributions/768/>.

Sanchez Vilorio, Jose Angel, Sklivanitis, George, Pados, Dimitris, and Bentley, Elizabeth Serena. Channel-informed neural network for physical layer key generation. Author-provided manuscript, September 2026; cited as related work, not as results of this tool, 2026.

Schmitz, Johannes, Xu, Xiang, Schröder, Florian, Zivkovic, Milan, and Mathar, Rudolf. SDR-based channel emulator using a semi-stochastic radio propagation model. In *Proceedings of SDR-WinnComm Europe*, pp. 63–66, 2013.

Villa, Davide, Tehrani-Moayyed, Miead, Johari, Pedram, Basagni, Stefano, and Melodia, Tommaso. CaST: A toolchain for creating and characterizing realistic wireless network emulation scenarios. In *Proceedings of the 16th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, pp. 45–52, 2022. doi: 10.1145/3556564.3558243.

Zubow, Anatolij, Pilz, Yannik, Rösler, Sascha, and Dressler, Falko. Ns3 meets Sionna: Using realistic channels in network simulation. *arXiv preprint arXiv:2412.20524*, 2024. URL <https://arxiv.org/abs/2412.20524>.

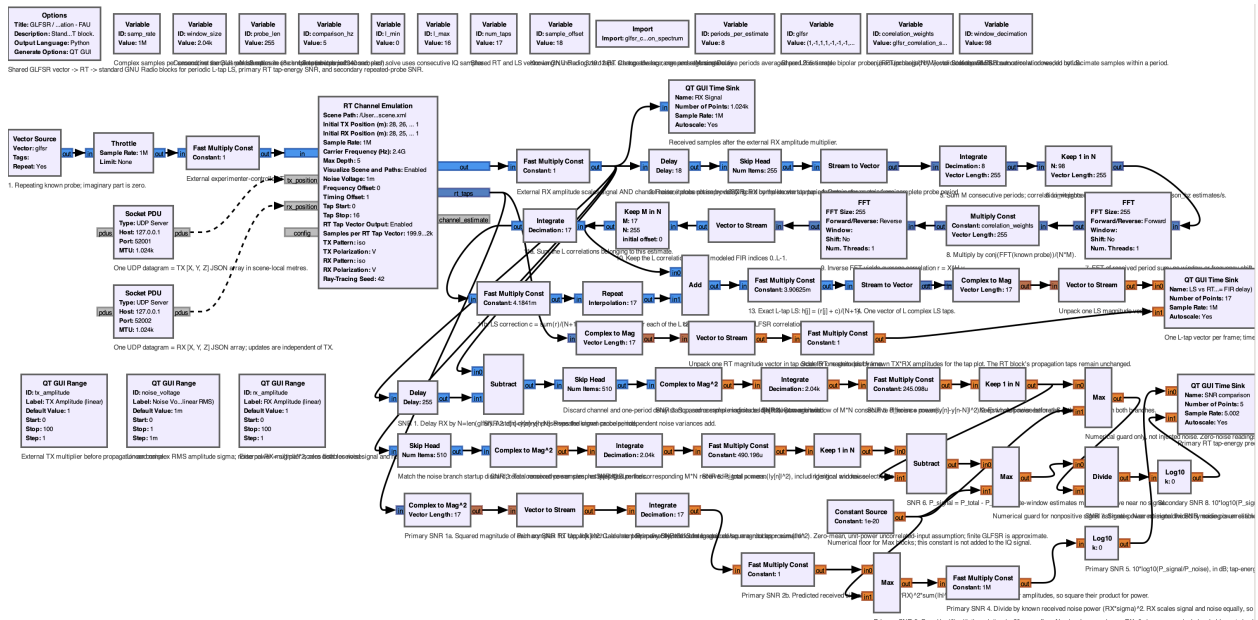


Figure 6. Demo flowgraph using RT_Channel Emulation block and LS and SNR estimation vs RT reference comparison