FP7-PEOPLE-2008-IAPP : Indoor radio network PLANning and optimization

Contractual date of delivery to EC: before Month 30
Actual date of delivery to EC: 30/05/2013
Lead beneficiary: INSA Lyon
Nature: Restricted
Version: 1.0
Project Name: Indoor radio network PLANning and optimisation
Acronym: iPLAN
Start date of project: 01/06/2009 **Duration:** 48 Months
Project no.: 230745

Leading editor : X. Tu
Contributors : Z. Lai, H. Hu, J. Rowney, JM Gorce

1. Summary

In this task, two radio propagation models will be integrated to the indoor radio network P&O tool. They include a ray-tracing based model which is previously developed by Ranplan and a MR-FDPF method based model which is developed in WP1. In order to let a P&O tool be suitable for different radio propagation models, the input data should be generalized for different models. Basically, there are two parts of the input data.

The main input is the map of the environment and its properties. A material coefficients library as complete as possible must be developed. Some coefficients values can be found in the literature but probably some other measurements in different environments and frequency bands will need to be performed. Automatic import of building map and data is the best approach. But in some case where no data is available the best approach is to create the environment by hand with an easy GUI.

The second important data to take into account is the emitter. If a 3D model is used the full 3D antenna pattern will need to be simulated. But most of the antenna builders only provide 2D patterns (horizontal and vertical cut only). Models must be proposed to build the 3D antenna pattern. Some measurements are also necessary to validate the proposed models.

To have an optimization tool based on realistic signal coverage simulations, a calibration module must be included, to make the simulation fit the measurements. This calibration concerns both the environment data and the antenna data. The number of parameters to optimize can be very high, especially in complex buildings with a lot of different materials, and in the case of heterogeneous network with different kinds of emitters. Because operators want a quick calibration process during the optimization operation, research about complex function minimisation must be performed. Some approaches in the literature propose the use of usual Tabu or Simulated Annealing functions to perform this calibration. We think some new and more efficient methods can be developed, more adapted to our context.

2. Description of the work

RPMLib

Ray-Tracing Method

Ray tracing, origin from 1980's, was first used in the domain of Compute Graphics to generate realistic 3D images. In the 1990's, it was extended to the field of Electromagnetic wave prediction. Both are based on fundamental ray object intersection computation tests. The following 'Ray Tracing' term is referred to the method used in the wireless propagation modelling. Ray tracing, as its name implies, computes the paths between two points in space (usually a pair of transmitter and receiver) in deterministic/non-deterministic manner. The theory of Ray tracing can be rooted to Maxwell equations, where wave-fronts can be modelled as an equation in relation with the time variable. The Ray tracing itself, in general, is limited in accuracy because it does not deal with the shadowing area where rays are not defined. Therefore, the Ray tracing requires GTD (Geometry Theory of Diffraction) or UTD (Uniform Theory of Diffraction, enhanced version of GTD) to model the shadow area, i.e the diffraction rays can be modelled as a Keller Cone, where a bundle of diffracted rays form a 3D cone.

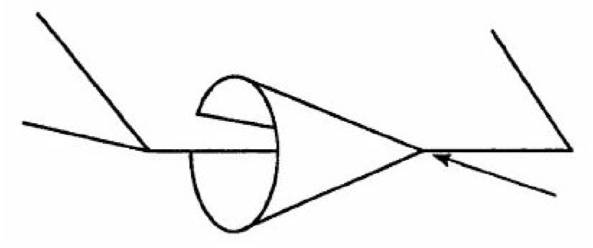


Figure 1 Diffraction rays are modelled as a Keller Cone

The reflections can be constructed using Snell's Law of Reflection. The quantities of P_v and P_h are the reflection coefficients of vertically polarised ray and horizontally polarised ray respectively. They are calculated as:

$$\rho_v = \frac{\sin\theta - \sqrt{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon) - \cos^2\theta}}{\sin\theta + \sqrt{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon) - \cos^2\theta}}$$

$$\rho_h = \frac{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon_0)\sin\theta - \sqrt{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon) - \cos^2\theta}}{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon_0)\sin\theta + \sqrt{(\epsilon/\epsilon_0 - j\sigma/\omega\epsilon) - \cos^2\theta}} \tag{1}$$

Scattering occurs on rough surface, and this leads to reflections in several directions. It is often, in practice, modelled by Ray tracing, as an empirical compensating factor, due to computational efficiency.

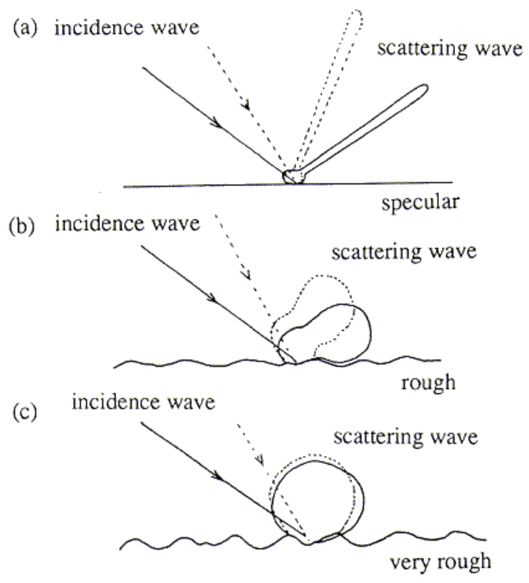


Figure 2 Scattering effects on (a) specular, (b) rough and (c) very rough surface

When radio wave hits on an object, if the size of the object (e.g. edges) is considerably small compared to wavelength, diffractions occur. When radio wave encounters the surface, usually depending on the type of material, partial wave bounces in the backward direction while the others diffract/transmit through the object. The Ray tracing computes the set of rays and derive the field strength. However, since it is impossible compute all possible rays (there are infinite), it is therefore reasonable to configure maximum number of ray iterations (reflections, diffractions and etc.) that rays are abandoned. Apart from this, if rays carry very low signal level (usually a threshold is set to 180 dB, where signal level falls below background noise), they are ignored as well.

Depending on the direction/manner of computing the rays, Ray tracing is often divided into two categories, the SBR (Shoot and Bounce Rays) and the image method. The SBR is also named as ray launching where the rays are gradually traced forward from the emitter. This is also known as the sampling method where rays are sampled according a pre-defined angle. Rays are separated by angles, and no matter how small the angle is, the rays will eventually disperse and some receiver locations will be missed by the rays, especially the distant area. Therefore, many literatures propose methods to avoid this, such as increasing the size of the receiver sphere or the number of rays. More rays will lead to higher accuracy, but also a higher demand for computation power. However, ray launching has a linear computation complexity with the number of ray iterations. It is not so affected by the number of objects in the scenario. Ray launching is often used as a coverage-prediction tool because the running time does now grow with the number of prediction locations, i.e. the prediction can benefit from the ray-memory, and the current pixel is similar to its previous pixels on the same ray direction. In industries, as the Ray launching suffers from inherent problem of ray dispersion, it is often combined with other ray tracing method, such as image method. One typical example is the RRPS tool from Ranplan Wireless.

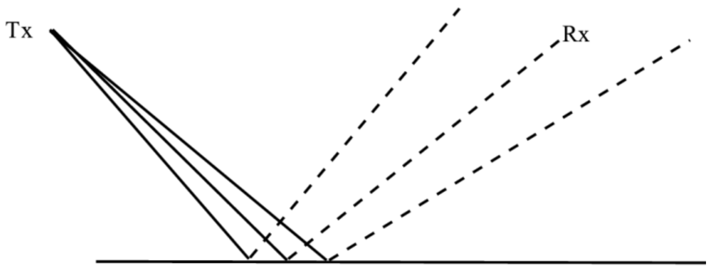


Figure 3 Ray dispersion problem of ray-launching method

The image method, works the other way around. It is the deterministic approach to compute the exact paths between transmitter and receivers. No rays will be missed. For example, the receivers are mirrored to the wall, and by connecting the transmitter to the mirrored receiver, we can conclude a shortest path (reflection ray) between these two. In Ray launching, more than one rays may be captured, but the image method only computes the only one shortest ray. The image method grows its complexity linearly if the ray number increments but it offers a higher accuracy. It is often used in point-to-point prediction where the number of locations of interests is low. Some literature proposes methods to speed up ray-object intersection test (considered 80% of task ray tracing is conducting), e.g. using BSP (Binary Space Partition) to reduce the number of tests. Some others propose methods to speed up computation algorithmically. For example, IRT (Intelligent Ray Tracing) proposed a pre-processing stage where the walls in the scenario are split into tiles and the visibility relations between them are stored. This is only required once, per building, per scenario because the visibility between buildings is not affected no matter where you put the antennas later on. The other acceleration technique is to interpolate neighbouring pixels because more often than not, the locations close to each other have similar multipath components.

Ray tracing computes the multipath components, based on the location of the antenna, the receiver locations and the building environment (walls and their associated material). The accuracy of such models relay heaving on building environment (locations of walls and material values). The multipath components obtained from ray tracing can be easily to derive the channel parameters, such as AoA (Angle of Arrivals), DS (Delay Spread) etc.

In industry, there are many existing excellent implementations of ray tracing tools, such as RRPS from Ranplan Wireless Ltd, and WinProp from AWE communications.

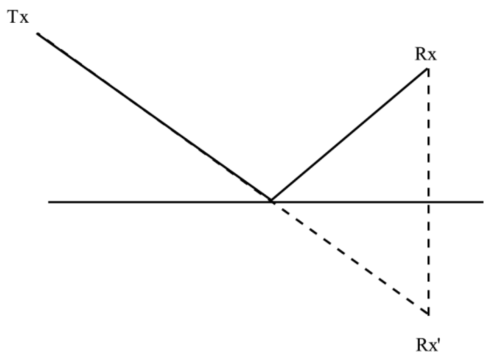


Figure 4 Ray tracing by mirrored Rx

Pluggable Architecture

The RPMLib.dll is the core engine file provided by RRPS. It is a 32-bit COM component which can be easily integrated in many software, and using many programming language. It is implemented in Delphi and Inline Assembly and targeted for Windows platform but with the help of WINE software, it is possible to run it from *nix platforms.

The RPMLib.dll is lightweight, efficient propagation engine that implements several propagation algorithms, such as ray tracing, multiwall. It supports client/server distribute computation framework. The core engine supports around 1500 APIs, with total lines of code up to 300K (excluding comments and blank lines).

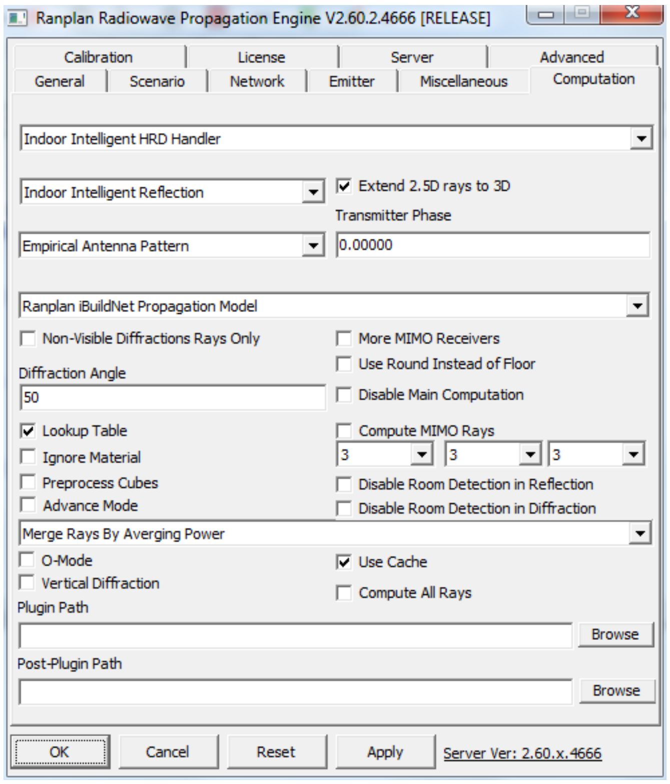
The RPMLib.dll is extensible, which means users can write plugins (e.g. propagation algorithms, calibration, or distribution broker) for RPMLib.dll without re-invent the wheels. The users can benefit from APIs that cover a range of aspects including data files handling, parameters configurations, material management and many others.

The following illustrates the creation of RPMLib.dll propagation engine object.

```
Dim RPM, RPU
Set RPM = CreateObject("RPMLib.RadioTracer")
Set RPU = CreateObject("RPMLib.Utility")
RPM.Reset
```

The declaration of *RPM* allows accesses to its rich set of Library APIs. The syntax are similar for other programing languages.

The *Configure* API will bring a dialog that contains most commonly-used parameters, and in the Tab of 'Computation', users can select a plugin (DLL format).



Indoor Intelligent HRD Handler

Indoor Intelligent Reflection

[✓] Extend 2.5D rays to 3D

Transmitter Phase: 0.00000

Empirical Antenna Pattern

Ranplan iBuildNet Propagation Model

[ ] Non-Visible Diffractions Rays Only

[ ] More MIMO Receivers

[ ] Use Round Instead of Floor

Diffraction Angle: 50

[ ] Disable Main Computation

[✓] Lookup Table

[ ] Compute MIMO Rays: 3 / 3 / 3

[ ] Ignore Material

[ ] Preprocess Cubes

[ ] Disable Room Detection in Reflection

[ ] Advance Mode

[ ] Disable Room Detection in Diffraction

Merge Rays By Averging Power

[ ] O-Mode

[✓] Use Cache

[ ] Vertical Diffraction

[ ] Compute All Rays

Plugin Path:

Post-Plugin Path:

Server Ver: 2.60.x.4666

Figure 5 Configure dialog of RPMLib

The DLL will be invoked instead of the inbuilt computation algorithms once preparation is finished.

The DLL Plugin Path can also be set via API property, PluginPath. Building Data, antenna pattern, transmitters and some other useful information will be passed to the DLL plugin. The DLL entry signature should be defined as 'RPMLib' to be callable. The following shows a quick example of DLL plugin (implements the free space computation).

MR-FDPF

MR-FDPF Method

The MR-FDPF (Multi-Resolution Frequency Domain ParFlow) [1] method has been proven to be a fast and accurate way of predicting the electrical field. It derives from the time domain ParFlow method to be working in the frequency domain by a Fourier transform of the local scattering equation; thus, it is able to solve the wave equation without defining a time boundary. In our implementation, a multi-resolution data structure has been used to reduce the calculation time.

The starting point of the original time domain ParFlow method is the well-known Maxwell's wave equation in pure dielectric media which can be written as

$$\delta_t^2 \Psi(r,t) - \left(\frac{c_0}{n_r}\right)^2 \cdot \nabla^2 \Psi(r,t) = 0 \qquad (2)$$

where $\Psi(\boldsymbol{r},t)$ is the electric field, c_0 is the speed of light and n_r is the refraction index of the media at r. The ParFlow method is the first order approximation of the above wave equation on a 2D grid as demonstrated in Figure 6 leading to

$$\frac{\Psi(r,t-dt)-2\cdot\Psi(r,t)+\Psi(r,t+dt)}{dt^2}=\left(\frac{c_0}{n_r dr}\right)^2\cdot\left[-4\cdot\Psi(r,t)+\sum_{i=1}^{4}\Psi(r+dr_i,t)\right] \quad (3)$$

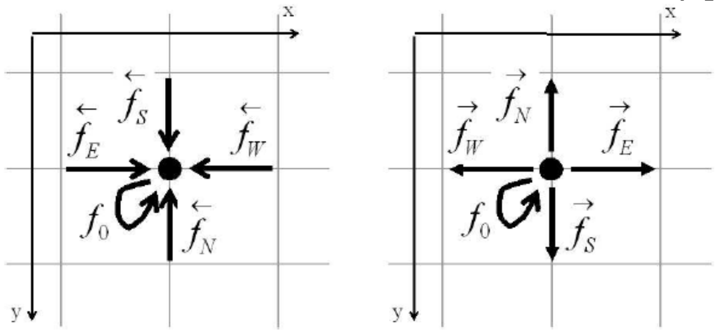


Figure 6 Inward and outward flows of a pixel

By this way, the electric field is divided into 5 components: 4 directive flows and 1 stationary flow. These flows are represented as $\overleftarrow{f_d}$ and $\overrightarrow{f_d}$ respectively for inward and outward flows, and f_0 for the stationary flow.

The ParFlow method uses $\overleftarrow{F}(m,t)$ and $\overrightarrow{F}(m,t)$ to express the inward and outward flow vectors by

$$\overleftarrow{F}(m,t)=\begin{pmatrix}\overleftarrow{f_E}(i,j,t)\\\overleftarrow{f_W}(i,j,t)\\\overleftarrow{f_S}(i,j,t)\\\overleftarrow{f_N}(i,j,t)\\f_0(i,j,t)\end{pmatrix};\ \overrightarrow{F}(m,t)=\begin{pmatrix}\overrightarrow{f_E}(i,j,t)\\\overrightarrow{f_W}(i,j,t)\\\overrightarrow{f_S}(i,j,t)\\\overrightarrow{f_N}(i,j,t)\\f_0(i,j,t)\end{pmatrix} \quad (4)$$

and the electrical field $\Psi(m,t)$ by

$$\Psi(m,t)=\frac{1}{n_m^2}\cdot\left(\overleftarrow{f_E}(m,t)+\overleftarrow{f_W}(m,t)+\overleftarrow{f_S}(m,t)+\overleftarrow{f_N}(m,t)+\Upsilon_m\cdot f_0(m,t)\right) \quad (5)$$

where m refers to the pixel (i,j) with $m=j+i\cdot N_c$, N_c the columns number, and $\Upsilon_m=4n_m^2-4$ is the local admittance.

$\Psi(m,t)$ is the solution of equation (3) if

$$\overrightarrow{F}(m,t)=\Sigma(m)\cdot\overleftarrow{F}(m,t-dt) \quad (6)$$

where $\Sigma(m)$ is the local scattering matrix which is defined by

$$\Sigma(m)=\frac{1}{2n_m^2}\cdot\begin{pmatrix}1&\alpha_m&1&1&Y_m\\\alpha_m&1&1&1&Y_m\\1&1&1&\alpha_m&Y_m\\1&1&\alpha_m&1&Y_m\\1&1&1&1&\beta_m\end{pmatrix} \quad (7)$$

with $\alpha_m=1-2n_m^2;\beta_m=2n_m^2-4$.

The above theory does not consider a source. If we introduce a source into the equations, then the original wave equation becomes

$$\delta_t^2\Psi(r,t)-\left(\frac{c_0}{n_r}\right)^2\cdot\nabla^2\Psi(r,t)=-\frac{1}{\epsilon}\cdot\delta_t i(r,t) \quad (8)$$

where $i(r,t)$ stands for a source located in r. And the local scattering equation (6) becomes

$$\overrightarrow{F}(r,t) = \Sigma(r)\cdot\overleftarrow{F}(r,t-dt)+\overrightarrow{S}(r,t) \tag{9}$$

where $\overrightarrow{S}(r,t)$ contains source flows (null if the source is not located at r).

FDPF is the version of ParFlow method in frequency domain. Conversion to the frequency domain is easy to accomplish by applying a Fourier transform on the equation (9) leading to

$$\overrightarrow{F}(r,\nu) = \Sigma(r)\cdot e^{-j2\pi\nu dt}\overleftarrow{F}(r,\nu)+\overrightarrow{S}(r,\nu) \tag{10}$$

It is not possible to solve it over the whole frequency band; so the study is restricted to a harmonic mode and thus for a specific frequency ν_0, e.g. the carrier frequency leading to an inverse problem given by

$$(I_d-\Sigma_0)\cdot\overleftarrow{F} = \overleftarrow{S}\ where\ \Sigma_0 = \Sigma\cdot e^{-j2\pi\nu dt} \tag{11}$$

Note that the variable ν_0 has been removed for the sake of clarity.

Recall the original ParFlow method, the stationary flow is introduced to model different media in time domain. It can also be called an inner flow because it does not participate to the energy exchange between adjacent pixels. However, in the frequency domain, it can be removed by some formulation derivations leading to

$$\overrightarrow{F_e}(m) = \Sigma_e(m)\cdot\overleftarrow{F_e}(m)+\overrightarrow{S_e}(m) \tag{12}$$

where the index e stands for exchange flows that do not include inner flows compared to the equation (4). $\Sigma_e(m)$ is a function of $\Sigma(r)$ which could be eventually derived as

$$\Sigma_e(m) = \sigma_0\cdot\begin{bmatrix}\sigma_1 & \sigma_2 & \sigma_1 & \sigma_1\\ \sigma_2 & \sigma_1 & \sigma_1 & \sigma_1\\ \sigma_1 & \sigma_1 & \sigma_1 & \sigma_2\\ \sigma_1 & \sigma_1 & \sigma_2 & \sigma_1\end{bmatrix} \tag{13}$$

with

$$\sigma_0 = \frac{e^{-2j2\pi\nu dt}}{2n_m^2} \tag{14}$$

$$\sigma_1 = 1+\Upsilon_m\cdot k_m \tag{15}$$

$$\sigma_2 = \alpha_m+\Upsilon_m\cdot k_m \tag{16}$$

$$k_m = \frac{\sigma_0}{1-\sigma_0\cdot\beta_m} \tag{17}$$

Let the global flow vectors be gathered into a column vector according to

$$\overleftarrow{F_e} = \begin{pmatrix}\overleftarrow{F_e}(0)\\ \overleftarrow{F_e}(1)\\ \vdots\\ \overleftarrow{F_e}(M-1)\end{pmatrix} \tag{18}$$

The system could be modelled by the following equation

$$(I_d-\Sigma_e)\overleftarrow{F_e} = \overleftarrow{S_e} \tag{19}$$

However, it is not applicable to directly calculate the inversion; an iterative method is chose to solve this linear problem by using matrix geometric series according to

$$\overleftarrow{F_e} = \sum_{k=0}^{\infty}(\Sigma_e)^k\cdot\overleftarrow{S_e} = \overleftarrow{S_e}+\Sigma_e\cdot\overleftarrow{S_e}+(\Sigma_e)^2\cdot\overleftarrow{S_e}+\cdots \tag{20}$$

Finally, the electric field can be computed directly by

$$\Psi(\mathrm{m}, \mathrm{v}_0) = \frac{1 + k_m}{n_m^2} \cdot \left[\overleftarrow{f_E}(m) + \overleftarrow{f_W}(m) + \overleftarrow{f_S}(m) + \overleftarrow{f_N}(m)\right] \tag{21}$$

The previous method has converted the time domain ParFlow method to the frequency domain and solved the modelled linear system iteratively by using matrix geometric series. It solves the problem over the entire environment which could lead to a high computational load. Thus, the multi-resolution (MR) approach is proposed to speed up the computation and reuse intermediate result.

A MR-node is defined as a rectangular set of nodes. The whole environment is divided into blocks recursively as demonstrated in Figure 7.

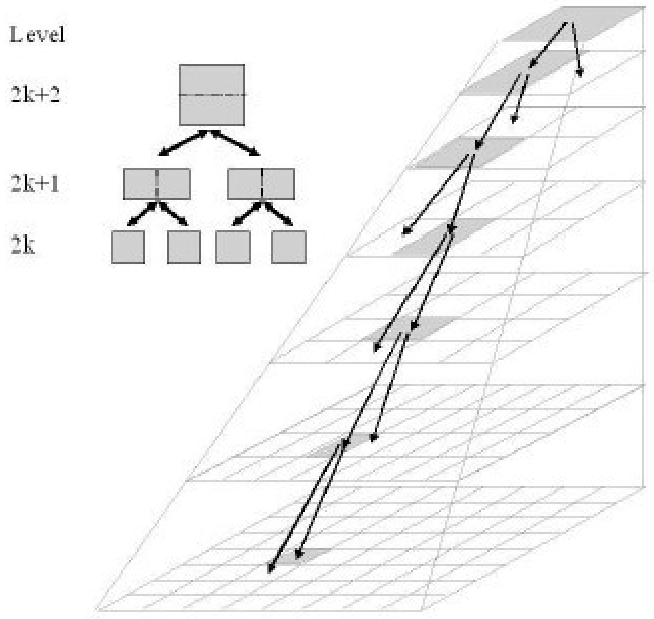


Figure 7 Multi-resolution approach

For each level, the environment is divided into K (l)-level nodes. The exchange flows of each node b_k^l are defined as

$$\overleftarrow{F_e}(b_k^l) = \begin{pmatrix} \overleftarrow{f_E}(b_k^l) \\ \overleftarrow{f_W}(b_k^l) \\ \overleftarrow{f_S}(b_k^l) \\ \overleftarrow{f_N}(b_k^l) \end{pmatrix}; \overrightarrow{F_e}(b_k^l) = \begin{pmatrix} \overrightarrow{f_E}(b_k^l) \\ \overrightarrow{f_W}(b_k^l) \\ \overrightarrow{f_S}(b_k^l) \\ \overrightarrow{f_N}(b_k^l) \end{pmatrix} \tag{22}$$

They are bound by the local scattering equation

$$\overrightarrow{F_e}(b_k^l) = \Sigma_e(b_k^l) \cdot \overleftarrow{F_e}(b_k^l) + \overrightarrow{S_e}(b_k^l) \tag{23}$$

In MR-node based approach, the inner flows are actually exchange flows between pixels of the same MR-node as illustrated in Figure 8(c).

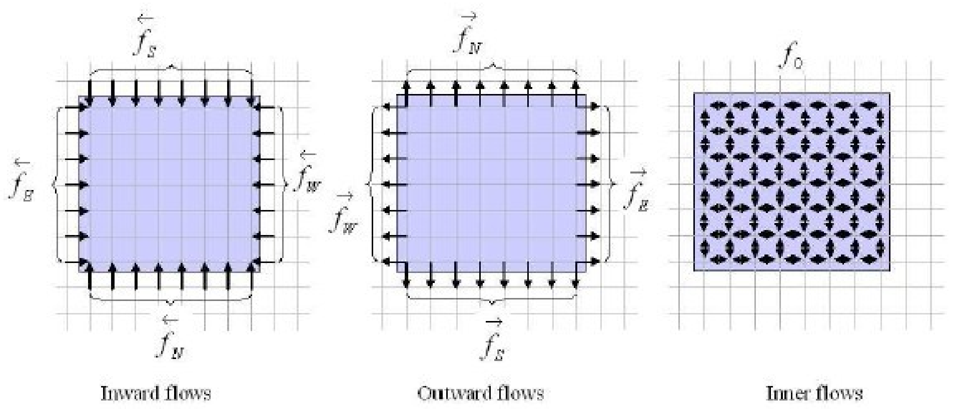


Figure 8 Inward flow (a), outward flow (b) and inner flow (c) of a MR-node

Gathering both exchange flows and inner flows of a MR-node, we have the local scattering matrix

$$\vec{F}(b_k^l) = \Sigma(b_k^l) \cdot \overleftarrow{F}(b_k^l) + \vec{S}(b_k^l) \tag{24}$$

where

$$\Sigma(b_k^l) = \begin{bmatrix} \sigma_{EE}(k) & \sigma_{EW}(k) & \sigma_{ES}(k) & \sigma_{EN}(k) & \sigma_{E0}(k) \\ \sigma_{WE}(k) & \sigma_{WW}(k) & \sigma_{WS}(k) & \sigma_{WN}(k) & \sigma_{W0}(k) \\ \sigma_{SE}(k) & \sigma_{SW}(k) & \sigma_{SS}(k) & \sigma_{SN}(k) & \sigma_{S0}(k) \\ \sigma_{NE}(k) & \sigma_{NW}(k) & \sigma_{NS}(k) & \sigma_{NN}(k) & \sigma_{N0}(k) \\ \sigma_{0E}(k) & \sigma_{0W}(k) & \sigma_{0S}(k) & \sigma_{0N}(k) & \sigma_{00}(k) \end{bmatrix} \tag{25}$$

Note that k herein and below stands for b_k^l for the sake of clarity. This scattering matrix could be divided into 4 blocks such that

$$\Sigma(k) = \begin{pmatrix} \Sigma_{ee}(k) & \Sigma_{ei}(k) \\ \Sigma_{ie}(k) & \Sigma_{ii}(k) \end{pmatrix} \tag{26}$$

with

$$\Sigma_{ee}(k) = \begin{pmatrix} \sigma_{EE}(k) & \sigma_{EW}(k) & \sigma_{ES}(k) & \sigma_{EN}(k) \\ \sigma_{WE}(k) & \sigma_{WW}(k) & \sigma_{WS}(k) & \sigma_{WN}(k) \\ \sigma_{SE}(k) & \sigma_{SW}(k) & \sigma_{SS}(k) & \sigma_{SN}(k) \\ \sigma_{NE}(k) & \sigma_{NW}(k) & \sigma_{NS}(k) & \sigma_{NN}(k) \end{pmatrix} \quad \Sigma_{ei}(k) = \begin{pmatrix} \sigma_{E0}(k) \\ \sigma_{W0}(k) \\ \sigma_{S0}(k) \\ \sigma_{N0}(k) \end{pmatrix} \tag{27}$$
$$\Sigma_{ie}(k) = (\sigma_{0E}(k) \quad \sigma_{0W}(k) \quad \sigma_{0S}(k) \quad \sigma_{0N}(k)) \qquad \Sigma_{ii}(k) = \sigma_{00}(k)$$

Then the local scattering matrix (24) can be expended as

$$\begin{pmatrix} \overrightarrow{F_e}(k) \\ f_0(k) \end{pmatrix} = \begin{pmatrix} \Sigma_{ee}(k) & \Sigma_{ei}(k) \\ \Sigma_{ie}(k) & \Sigma_{ii}(k) \end{pmatrix} \cdot \begin{pmatrix} \overleftarrow{F_e}(k) \\ f_0(k) \end{pmatrix} + \begin{pmatrix} \overrightarrow{S_{0,e}}(k) \\ \overrightarrow{S_{0,\imath}}(k) \end{pmatrix} \tag{28}$$

By setting the source vector to 0 or setting the inward flows to 0, the above equation could be solved to get the scattering matrix $\Sigma_e(k)$ for exchange flows and an equivalent source flow $\overrightarrow{S_e}(k)$ respectively:

$$\Sigma_e(k) = \Sigma_{ee}(k) + \Sigma_{ei}(k) \cdot \left(Id - \Sigma_{ii}(k)\right)^{-1} \cdot \Sigma_{ie}(k) \tag{29}$$
$$\overrightarrow{S_e}(k) = \overrightarrow{S_{o,e}}(k) + \Sigma_{ei}(k) \cdot \left(Id - \Sigma_{ii}(k)\right)^{-1} \cdot \overrightarrow{S_{0,\imath}}(k) \tag{30}$$

The calculation respects to

$$f_0(k) = \left(Id - \Sigma_{ii}(k)\right)^{-1} \cdot \left(\overrightarrow{S_{0,\imath n}}(k) + \Sigma_{ie}(k) \cdot \overleftarrow{F_e}(k)\right) \tag{31}$$

The propagation calculation over the entire environment contains four steps as illustrated in Figure 9:

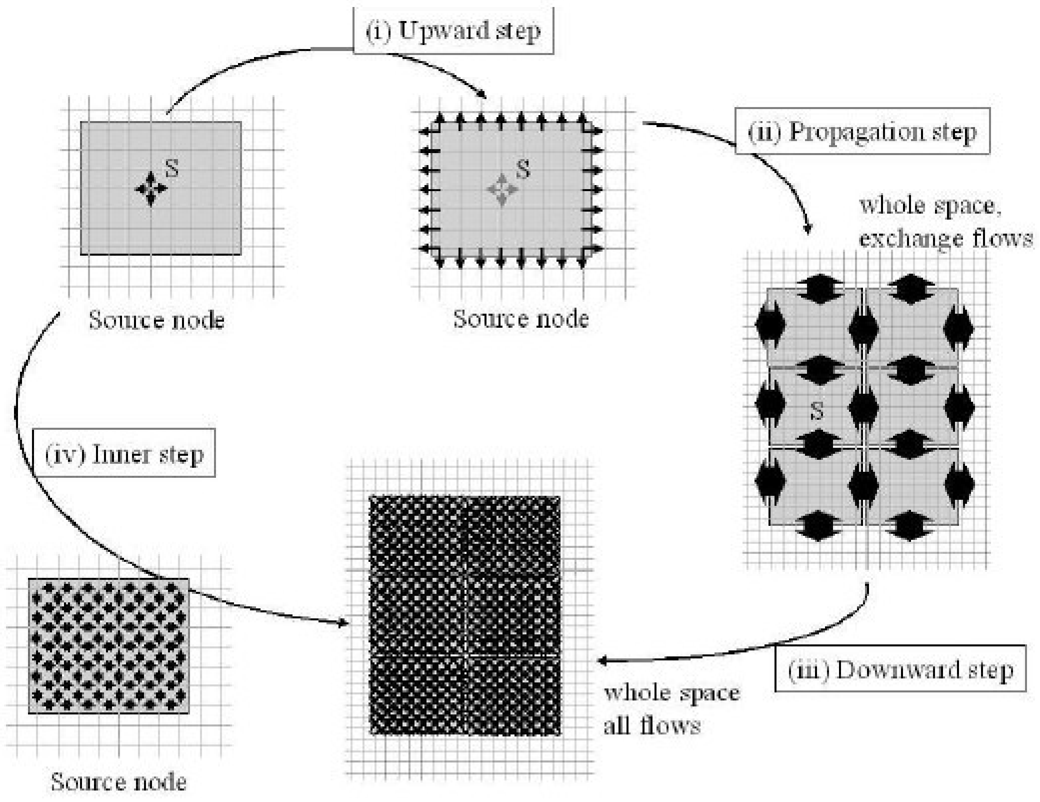


Figure 9 Four calculation steps for the whole environment

- **The inner step**. It corresponds to the propagation of the elementary source inside its MR-node. Let the global inner propagation matrix be defined as

$$I^l = \left(Id - \Sigma^l_{0,ii}\right)^{-1} \tag{32}$$

The local solution of each isolated MR-node is

$$\overleftarrow{F^l_i} = I^l \cdot R^l_0 \cdot S^0_e \tag{33}$$

where R^l_0 is the projection of the (0)-level flows to the (l)-level flows. This is also the first term of equation (31).

- **The upward step**. It will transform the parent node into a source node by computing outward flows of each source MR-node. Let the upward matrix be defined by

$$U^l = \Sigma^l_{0,ei} \tag{34}$$

The equivalent parent source flow is given by

$$\overleftarrow{S^l_e} = \left(R^l_0 + U^l I^l \cdot R^l_0\right) \cdot \overleftarrow{S^0_e} \tag{35}$$

This computation ends when the head-node is reached, thus inner flows associated with the source node are computed.

- **The Iterative scattering step.** This step involves exchange flows only. They are computed by equation (20) for each level leading to

$$\overleftarrow{F_e^l} = \sum_{k=0}^{\infty} \left(\Sigma_e^l\right)^k \cdot \overleftarrow{S_e^l} \tag{36}$$

- **The downward step.** Inner flows with exchange flows of each MR-node will be computed in the step. Let the downward matrix be defined by

  $$D^l = \Sigma_{0,ie}^l \tag{37}$$

  The incoming flows are propagated toward inner flows in each MR-node independently. Including the inner step directly into the equation (31), the downward equation becomes

  $$\overleftarrow{F_\iota^l} = I^l \cdot \left(\overleftarrow{S_\iota^l} + D^l \cdot \overleftarrow{F_e^l}\right) \tag{38}$$

  Then, the (0)-level exchange flows can be expressed as a function of (l)-level exchange flows and (0)-level source only if possible as

  $$\overleftarrow{F_e^0} = \left(R_0^l\right)^t \cdot I^l \cdot R_0^l \cdot \overleftarrow{S_e^0} + \left(\left(R_0^l\right)^t + \left(R_0^l\right)^t \cdot D^l\right) \cdot \overleftarrow{F_e^l} \tag{39}$$

Finally, the electric field of each pixel is calculated as the sum of inward flows from four directions.

Calibration of MR-FDPF

A calibration module is implemented to make the simulation result fits measurement data. The target of the calibration module is to optimize the parameters of each material so that the error between prediction result and measurement is the minimized.

The solution k of this optimisation algorithm is defined as

$$S_k = \{\varepsilon_1, \alpha_1, \varepsilon_2, \alpha_2, \dots, \varepsilon_i, \alpha_i\}, i \in \{0, \dots, N\} \tag{40}$$

where ε_i and α_i are the refraction index and attenuation coefficient of the ith material respectively. N is the number of materials.

The target is to minimize the fitness function of each solution which is defined as

$$F(S) = RMS\left(M_j - P_j\right), j \in \{0, \dots, M\} \tag{41}$$

where M_j is the measured path loss on the point j and P_j is the predicted path loss result on the same point. RMS indicates the root mean square of a variable. M is the number of measurement point.

In our implementation, the genetic algorithm (GA) [2] is chosen to be used for searching the optimal solution. A 'Chromosome' is defined as the same as the solution in equation (40). A 'Gene' that makes up a chromosome is either the refraction index or the attenuation coefficient of a material. The algorithm generates a population with a collection of chromosomes at the first. Then a crossover and mutation operation is performed to generate the next generation.

The crossover operation is built up with two steps: selection and mating. In the selection step, K chromosomes will be selected to the selection pool at first. Each of them is selected based on a tournament selection method which is by randomly selecting T $(T < K)$ chromosome from current population and choosing the one with the best fitness. Then, moving to the mating step, two chromosomes will be chosen randomly from the selection pool and mate to produce an offspring with single point crossover as demonstrated in Figure 10. This crossover operation is performed with respect to a crossover probability. If the crossover is not performed, the selected two chromosomes will be put to the next generation directly.

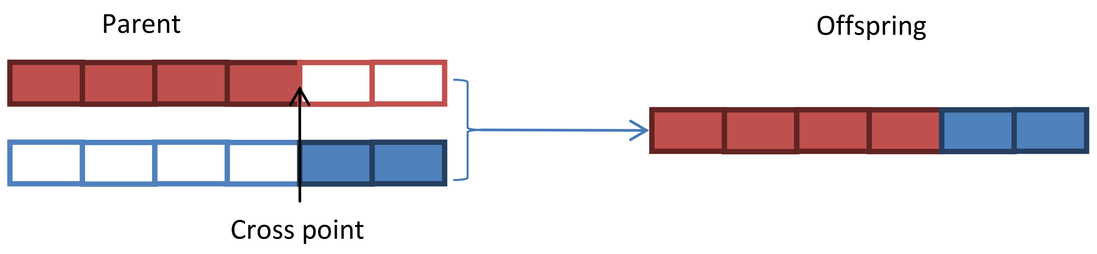


Figure 10 Cross point mating

The second operation is called mutation. It leads the algorithm towards local optima rather than global optima. This operation is also performed with respect to a mutation probability. When the condition satisfies, a gene of a chromosome will be modified.

The algorithm ends when a specific number of iterations have finished. The chromosome with the best fitness will be chosen as the solution and returned back.

Plugin Component

The MR-FDPF algorithm was originally implemented in JAVA. With the 'Write once, run anywhere' (WORA) design of JAVA, it is able to be ran on different operation systems i.e. Windows and *nix. However, JAVA programs are running in virtual machines whose performance is much lower than the native code, and it is not easy to cooperate with other programs that are written in different languages. Thus, in this task, we have rewritten it in C++.

The matrix computation plays a very important role in the MR-FDPF algorithm. In the original implementation, a Java interface (JNI) has been developed to invoke BLAS (Basic Linear Algebra Subprogram) library. This middleware also reduces the performance because it increases the function calls. In C++ version, an open source BLAS library named Eigen has been chosen to do the BLAS operations. Eigen is a fast, reliable and well-documented C++ BLAS library which provides excellent performance of BLAS operations. With the help of SSE2 (Streaming SIMD Extension 2) instruction sets, the performance could be boosted several times. The SSE2 instruction sets have been supported by almost all modern CPUs, so we can benefit from it even on a laptop.

The wiplan.dll file is the main output of this task. It consists of three components as illustrated in Figure 11.

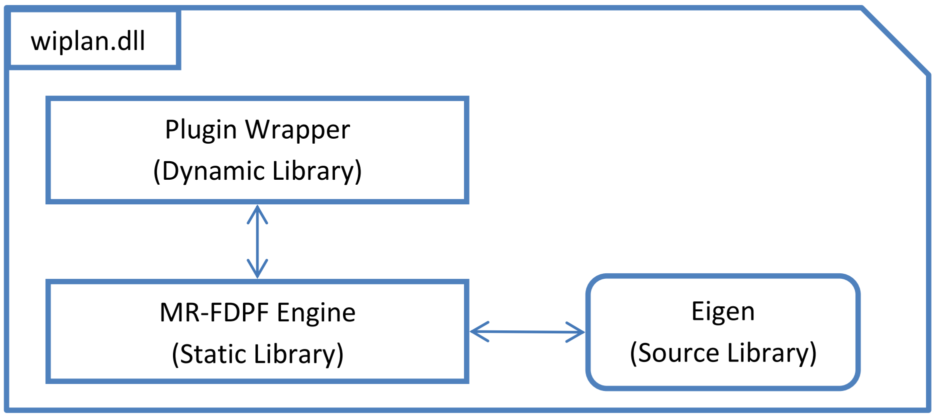
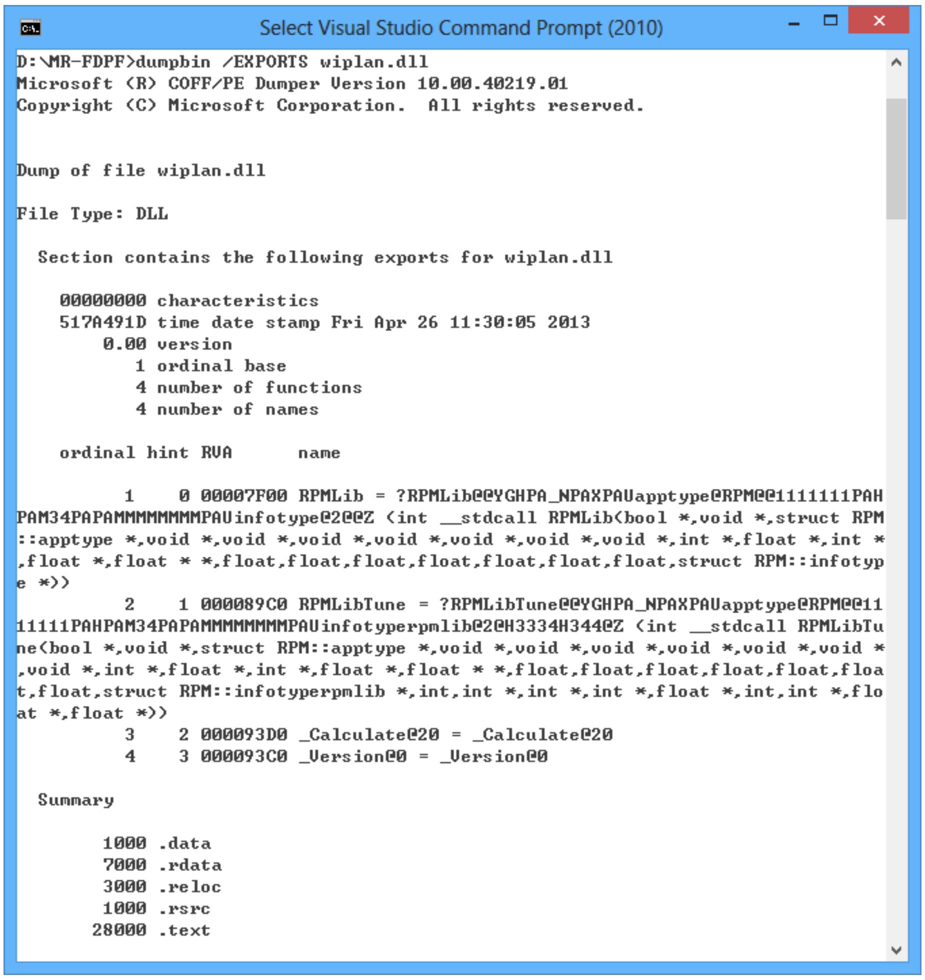


Figure 11 wiplan.dll structure

- MR-FDPF Engine. It corresponds to the pure C++ implementation of the MR-FDPF algorithm. With the help of C++ OOP (Object-Orient Programming) feature, the algorithm has been encapsulated in several classes providing simple APIs (Application Programming Interface) to other programs for the propagation calculations.
- Plugin Wrapper. This component implements the RPMLib interface including both the propagation and calibration calculations. It is responsible for transferring inputs from RPMLib which is the host of this module to the data structure that MR-FDPF engine requires. Once the calculation finishes, it then returns the path loss result to the RPMLib.
- Eigen. This is the BLAS library. It is integrated in this module with source code directly.

The RPMLib requires that the plugin DLL file export functions named RPMLib and RPMLibTune for propagation and calibration calculations respectively. The signature must be the same as documented in the previous section. Figure 12 shows the exported functions of wiplan.dll file. The first and the second one are that required by RPMLib.



```
D:\MR-FDPF>dumpbin /EXPORTS wiplan.dll
Microsoft (R) COFF/PE Dumper Version 10.00.40219.01
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file wiplan.dll

File Type: DLL

  Section contains the following exports for wiplan.dll

    00000000 characteristics
    517A491D time date stamp Fri Apr 26 11:30:05 2013
        0.00 version
           1 ordinal base
           4 number of functions
           4 number of names

    ordinal hint RVA      name

          1    0 00007F00 RPMLib = ?RPMLib@@YGHPA_NPAXPAUapptype@RPM@@1111111PAH
PAM34PAPAMMMMMMMMPAUinfotype@2@@Z (int __stdcall RPMLib(bool *,void *,struct RPM
::apptype *,void *,void *,void *,void *,void *,void *,void *,int *,float *,int *
,float *,float * *,float,float,float,float,float,float,float,struct RPM::infotyp
e *))
          2    1 000089C0 RPMLibTune = ?RPMLibTune@@YGHPA_NPAXPAUapptype@RPM@@11
11111PAHPAM34PAPAMMMMMMMMPAUinfotyperpmlib@2@H3334H344@Z (int __stdcall RPMLibTu
ne(bool *,void *,struct RPM::apptype *,void *,void *,void *,void *,void *,void *
,void *,int *,float *,int *,float *,float * *,float,float,float,float,float,floa
t,float,struct RPM::infotyperpmlib *,int,int *,int *,int *,float *,int,int *,flo
at *,float *))
          3    2 000093D0 _Calculate@20 = _Calculate@20
          4    3 000093C0 _Version@0 = _Version@0

  Summary

        1000 .data
        7000 .rdata
        3000 .reloc
        1000 .rsrc
       28000 .text
```

Figure 12 Export functions of wiplan.dll

Material Library

We have developed the material management module for the radio P&O tool iBuildNet to maintain the reading, manipulation and storage of the material dataset. A typical multitier architecture is used to construct the module, i.e. presentation tier, logic tier and data tier. Such structure would improve the modularization of the software so that it can benefit from the ease of maintenance, update and platform migration. Thus when we integrate the MR-FDPF engine, the task can be done more efficiently.

Among the 3 tiers structure, the presentation tier provides user with forms to read, edit, add and delete materials and their properties. Then the logic tier contains the business logic layer which handles the substantial processes from/to the UI forms, and the data access layer which handles the interaction with material database. In data tier there is the database which is based on SQLite.

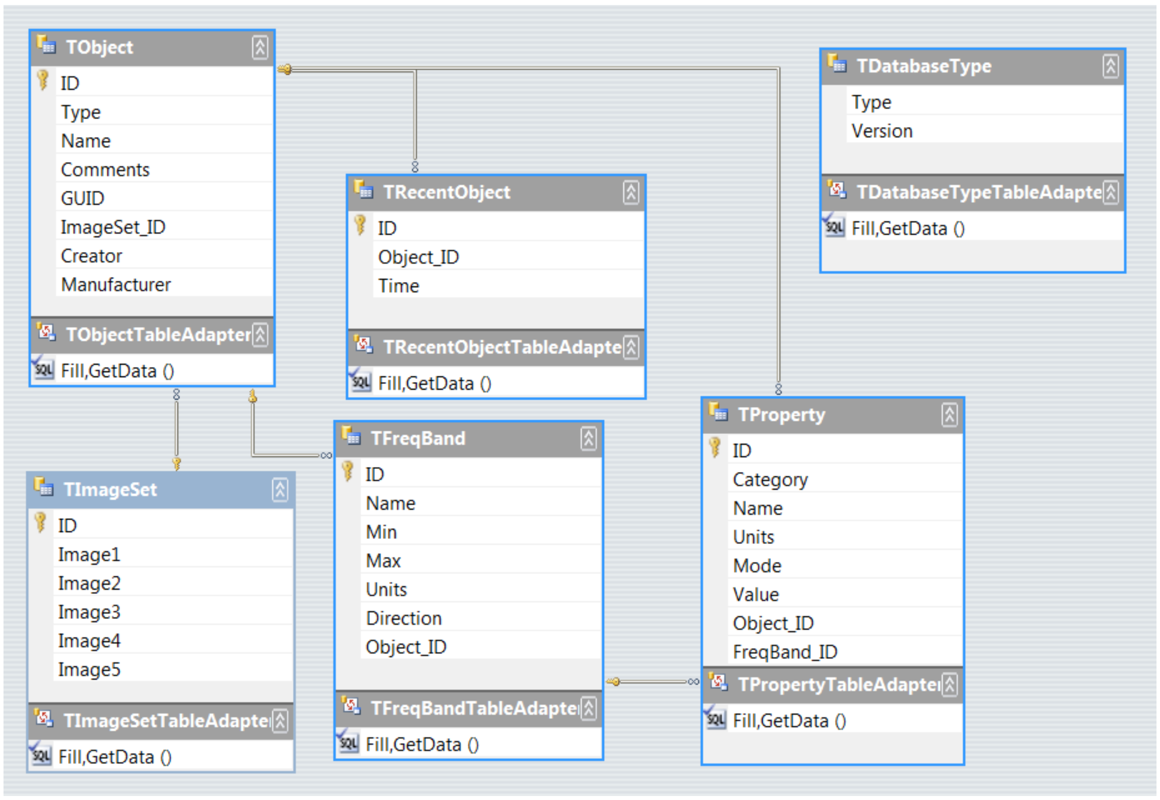
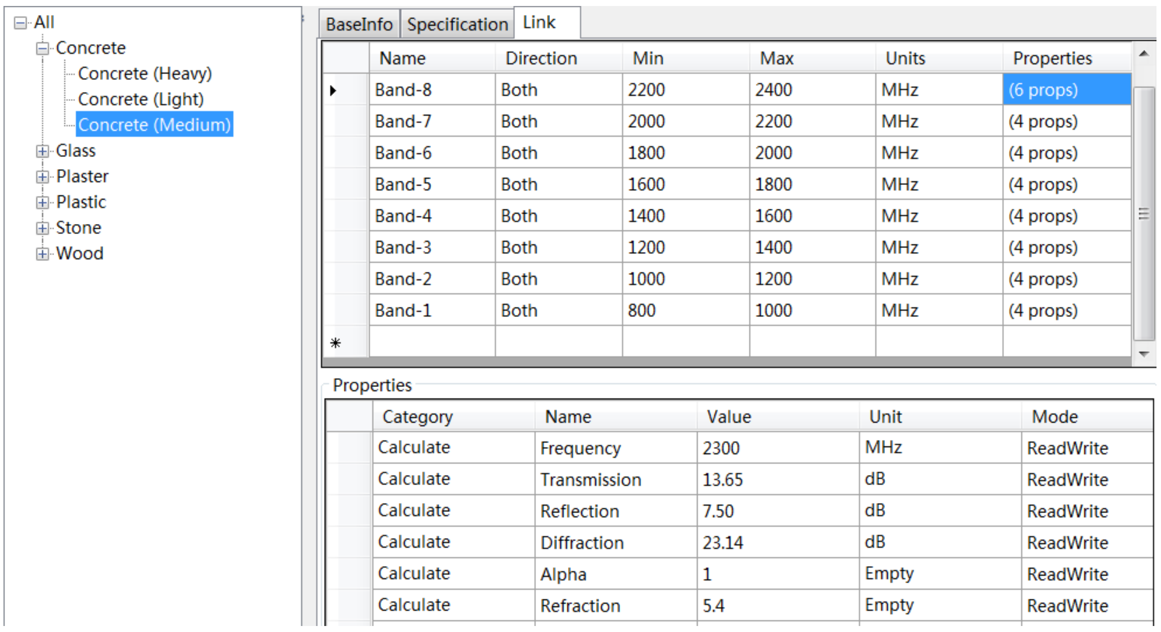


Figure 13 Material dataset structure

Figure 13 demonstrates the dataset structure of the material in iBuildNet. The object typically represents one type of material. Each material has an extendable range of properties, and these properties are classified depending on whether they are frequency dependent. The material is presented in a set of frequency bands, while in each band the frequency dependent properties, like transmission loss, are given. There are also some properties which is not related to frequency, like thickness, are linked to the material directly. Above all, the dataset structure is extendable, and this would help when introduce the new properties, attenuation coefficient and refraction index, for the use of MR-FDPF engine.

While the old MR-FDPF engine use an xml format file to store its material data, we have merged it into the database of iBuildNet, as described above, to support both engines in iBuildNet simultaneously.



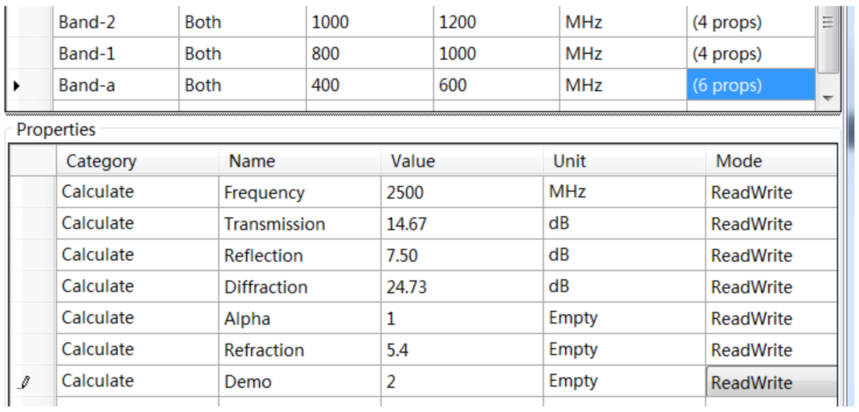
Name	Direction	Min	Max	Units	Properties
Band-8	Both	2200	2400	MHz	(6 props)
Band-7	Both	2000	2200	MHz	(4 props)
Band-6	Both	1800	2000	MHz	(4 props)
Band-5	Both	1600	1800	MHz	(4 props)
Band-4	Both	1400	1600	MHz	(4 props)
Band-3	Both	1200	1400	MHz	(4 props)
Band-2	Both	1000	1200	MHz	(4 props)
Band-1	Both	800	1000	MHz	(4 props)

Properties

Category	Name	Value	Unit	Mode
Calculate	Frequency	2300	MHz	ReadWrite
Calculate	Transmission	13.65	dB	ReadWrite
Calculate	Reflection	7.50	dB	ReadWrite
Calculate	Diffraction	23.14	dB	ReadWrite
Calculate	Alpha	1	Empty	ReadWrite
Calculate	Refraction	5.4	Empty	ReadWrite

Figure 14 Material DB management form

Figure 14 shows the properties for the concrete material in the material management form. Six frequency dependent properties are created at the band-8, which includes the 2.4GHz for 802.11b/g. Among these properties, two are the alpha (attenuation coefficient) and refraction index, which are used by the MR-FDPF engine, while the rest four are for the RPM engine. A parameter criterion is enforced to make sure that the alpha should be between 0 and 1 and the refraction index should be no less than 1, according to [3]. Also it may be noticed that for the rest bands of the concrete material only four frequencies dependent properties are provided. This is due to the nature of alpha and refraction index that they are calculated by calibration in different environments, see [3]. Pre-defined values for them have little sense. Yet, they are default by 1. And the returned value from calibration process will update the database to include accurate alpha and refraction index at specific frequency, so that they can be reused in the same environment.



Band-2	Both	1000	1200	MHz	(4 props)
Band-1	Both	800	1000	MHz	(4 props)
Band-a	Both	400	600	MHz	(6 props)

Properties

Category	Name	Value	Unit	Mode
Calculate	Frequency	2500	MHz	ReadWrite
Calculate	Transmission	14.67	dB	ReadWrite
Calculate	Reflection	7.50	dB	ReadWrite
Calculate	Diffraction	24.73	dB	ReadWrite
Calculate	Alpha	1	Empty	ReadWrite
Calculate	Refraction	5.4	Empty	ReadWrite
Calculate	Demo	2	Empty	ReadWrite

Figure 15 Add new band and property

As shown in Figure 15, if needed, new frequency band and material can be added by user, which can be done by simply typing the new fields into the data table. As MR-FDPF engine may use lower frequency than actual, 480MHz for example, to reduce the work load while maintain accurate prediction. In all, the developed material library module supports both engines very well, and it is capable of potential future expansion benefitting from its modularized structure.

Integration to the P&O tool

In order to give a clear idea on how these components work together, an example of a typical networking planning task is demonstrated using the sequence diagram, see Figure 16. This task is separated into three steps while the first and third step involves the components that this report covers, i.e. RPMLib.dll, wiplan.dll and Material Library.

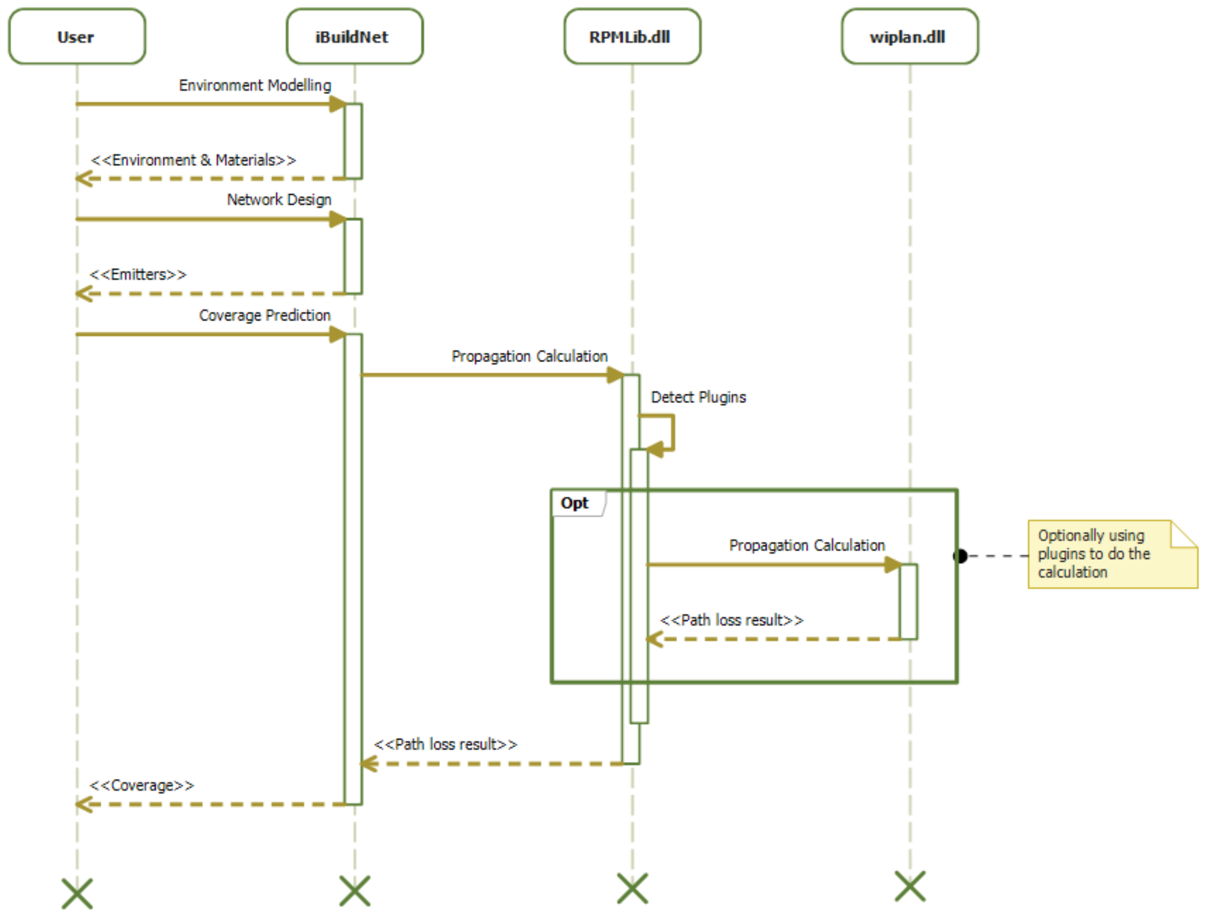


Figure 16 Coverage prediction demostration

- Environment Modelling. In this step, users create building models with easy-to-use drawing tools in iBuildNet. A default material library will be created which contains common materials. Each material consists of parameters for radio propagation calculations for different frequencies as stated in the previous section. Users may change these parameters for specific scenarios or different propagation models. This step produces the map of the environment with materials defined.
- Network Design. It corresponds to deploying the wired-devices of a radio network. Once the design finishes, all emitters will be determined.

- Coverage Prediction. This is a task which is often carried out in the early stage of a network planning project. When the users request a coverage prediction, the radio propagation calculation will be invoked. The environment and emitter data will be parsed to RPMLib.dll. It will then check if there is a plugin being configured. If, for example, the wiplan.dll file path has been set to PluginPath property of RPMLib.dll, the RPMLib.dll will hand over the calculation to the plugin. The wiplan.dll will analyse the environment data and calculation the electrical field over the entire environment with respect to the emitter settings, and finally produces path loss result.

For an automatic network P&O tool, the propagation model is considered as a fundamental module. Almost all the network planning and optimisation algorithms are based on the path loss prediction result which is the main output of the propagation model. However, researches on the radio propagation model never stops because all the models have their advantages and disadvantages. So, it is important for a network P&O tool to support not only one propagation model.

iBuildNet has been designed to support different propagation models. The RPMLib.dll is the primary interface between iBuildNet and different propagation models. Most popular propagation models e.g. COST231 Multi-Wall and Free Space Model are built in the RPMLib.dll. It also supports third-party plugins that could provide other propagation models as explained in the previous sections.

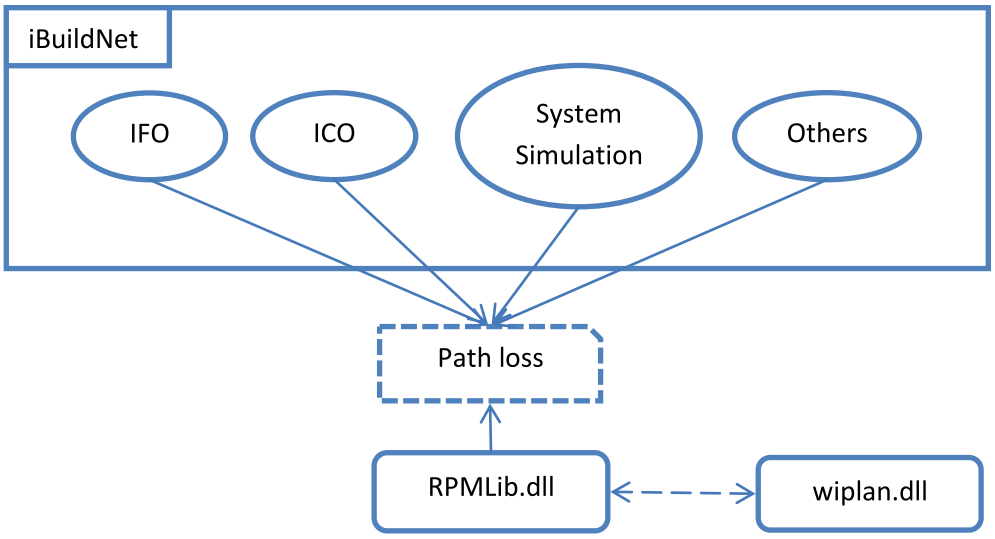


Figure 17 Interaction of iBuildNet and propagation models

As demonstrated in Figure 17, the dependency between iBuildNet and propagation models is the path loss data which is in the snip single corner rectangle. In our implementation, the RPMLib.dll is responsible for creating the path loss result file. The path loss data can be generated either by built-in propagation models or the third-party plugins e.g. wiplan.dll. All other features like IFO (Intelligent Frequency Optimisation), ICO (Intelligent Cellular Optimisation), and system level simulation etc. need to use path loss data.

3. Perspectives

The integration of MR-FDPF engine into iBuildNet makes the engine benefit from various resources and features that are maturely developed in iBuildNet. On the other side, the new integrated engine enriches the functionality of iBuildNet to provide user with more choice for a better prediction. This section will present the software developments for the integration of MR-FDPF engine. There are some functions, which is provided by iBuildNet previously but modified after the integration, already described in the D3.1 report. Reader shall refer to the D3.1 report for detailed operation, while this section will focus on the integration result.

Export the building model information

iBuildNet has built up a complete system for 3D building modelling. Typically a skilled iBuildNet user can construct a building model within 10 minutes. There are also several import method been developed to further facilitate the model construction, e.g. background image importing and semi-automatic CAD file importing etc. As the MR-FDPF engine is integrated into iBuildNet, we have enabled iBuildNet to export its building information into a rasterized file, which could be recognised by the engine.

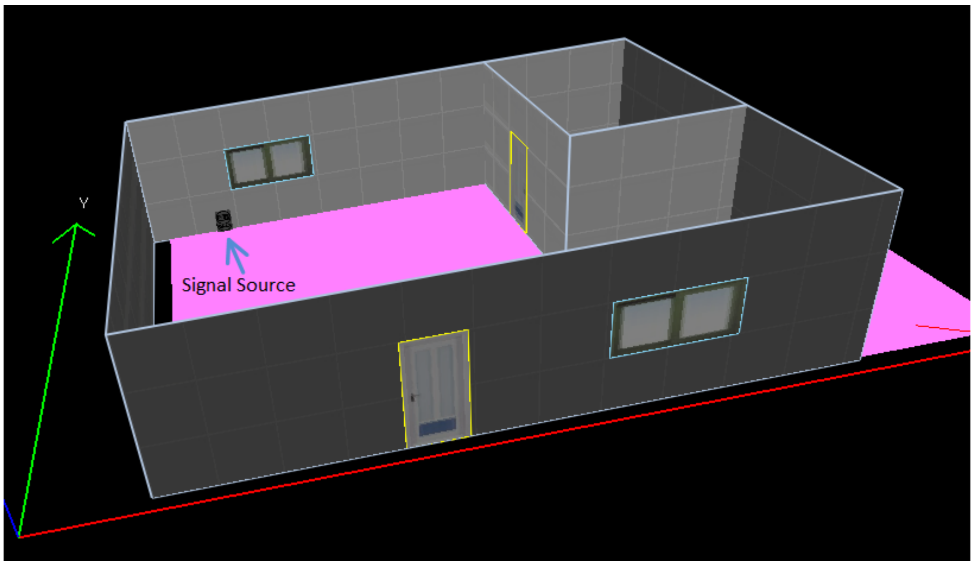
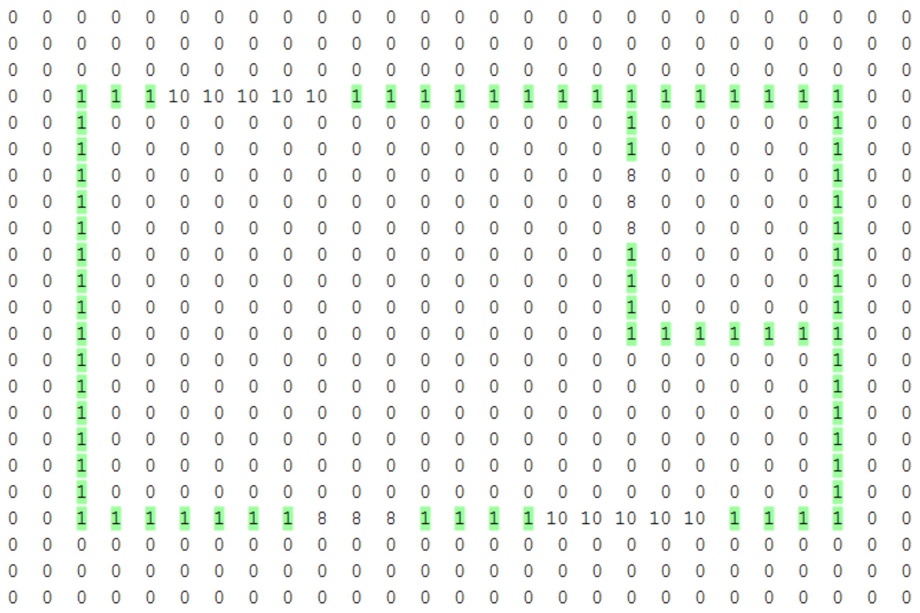


Figure 18 3D building model in iBuildNet

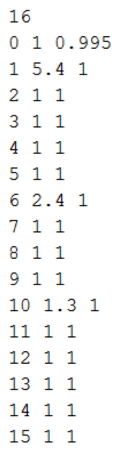
Figure 18 presents a simple 3D building model in iBuildNet. Walls, doors, windows and a signal source can be seen in the plot. To export this model into the engine, firstly the model needs to be pre-processed into a rasterised format at a given resolution. The resolution should be chosen very carefully, as a large resolution may cause the inaccuracy of the result and a small resolution may cause the prediction process to be very slow. There is a constraint that the resolution should be less than 1/6 of the wavelength to avoid anisotropy errors [3].



```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  1  1  1  10 10 10 10 10 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  1  1  1  1  1  1  8  8  8  1  1  1  1  10 10 10 10 10 1  1  1  1  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 19 Rasterized cross section

Figure 19 is a part of the rasterized file, generated by iBuildNet, for the use of MR-FDPF engine. This file contains the entire building's geometrical information. For the convenience of demonstration, only the 2D form, which is a cross section of the building at the antenna height, is shown here. In the figure, the numbers stand for different materials in the environment. The highlighted "1" represents concrete for the wall, the "8" represents wood for the door, the "10" represents glass for the window and the "0" is for the air. The rasterized structure matches the 3D building model in Figure 19.



```
16
0 1 0.995
1 5.4 1
2 1 1
3 1 1
4 1 1
5 1 1
6 2.4 1
7 1 1
8 1 1
9 1 1
10 1.3 1
11 1 1
12 1 1
13 1 1
14 1 1
15 1 1
```

Figure 20 Material properties

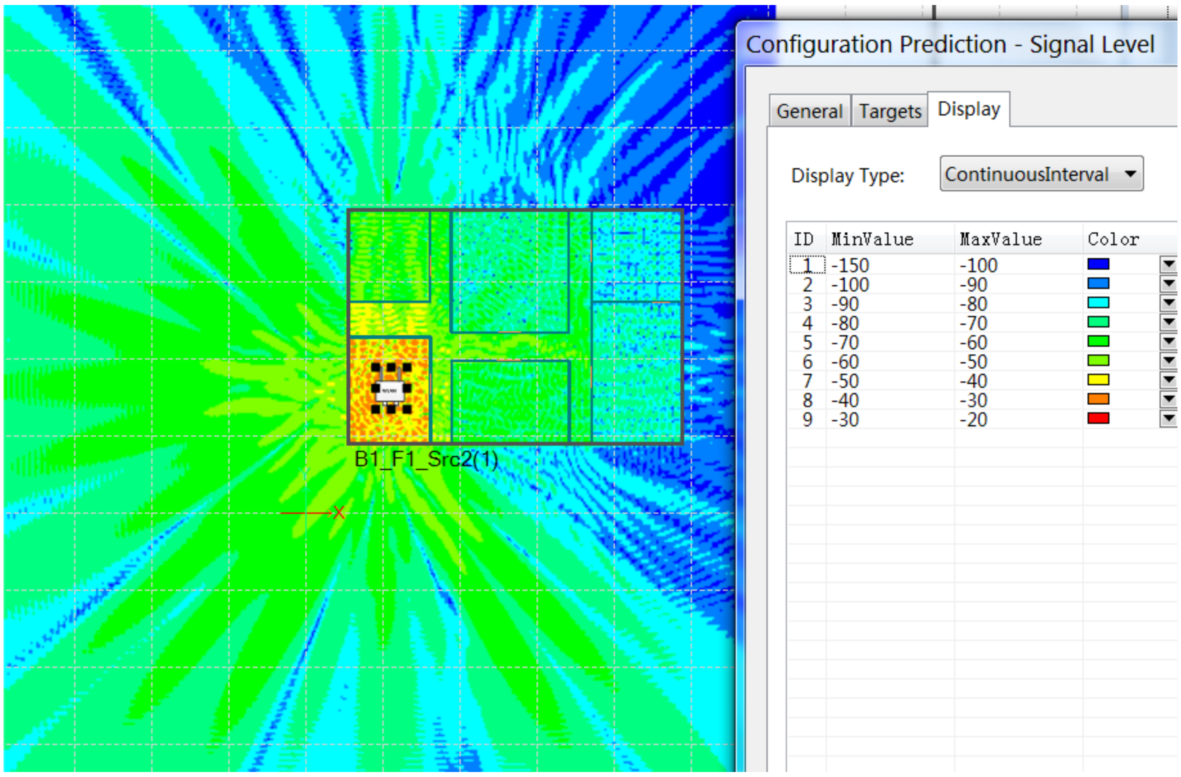
Figure 20 shows the material properties contained in the same rasterized file. The first number "16" means there are 16 types of material listed. In each of the following 16 rows, the starting number, which matches the numbers in cross section plot above, represents one particular type of material, and the rest two numbers are the refraction index and attenuation coefficient for this material.

Thus far, we have ensured the correctness of the exporting information from the iBuildNet into the MD-FDPF engine. One is the geometrical structure which is shown in Figure 19; the other is the material properties as shown in Figure 20. The same procedure follows for the RPMLib engine, only that its exporting file is hexadecimal based and can't be demonstrated here.

Visualization of the prediction result

After exporting the building model into the radio propagation engine, the prediction process shall take place inside the engine. The process may usually consume several seconds up to tens of minutes, depending on the size of the building model, the calculation resolution and the algorithm used. After finished, a set of prediction result for each signal source will be returned to the iBuildNet, where the prediction will be visualized to the user.

The return values from MR-FDPF engine are in a form of 2-D matrix, in which each cell contains a number indicating the signal level at the corresponding position in the building model. To display the result, iBuildNet will first have to match the matrix back to the building according to the chosen resolution, then plot the signal level according to a legend, which is defined by the user.

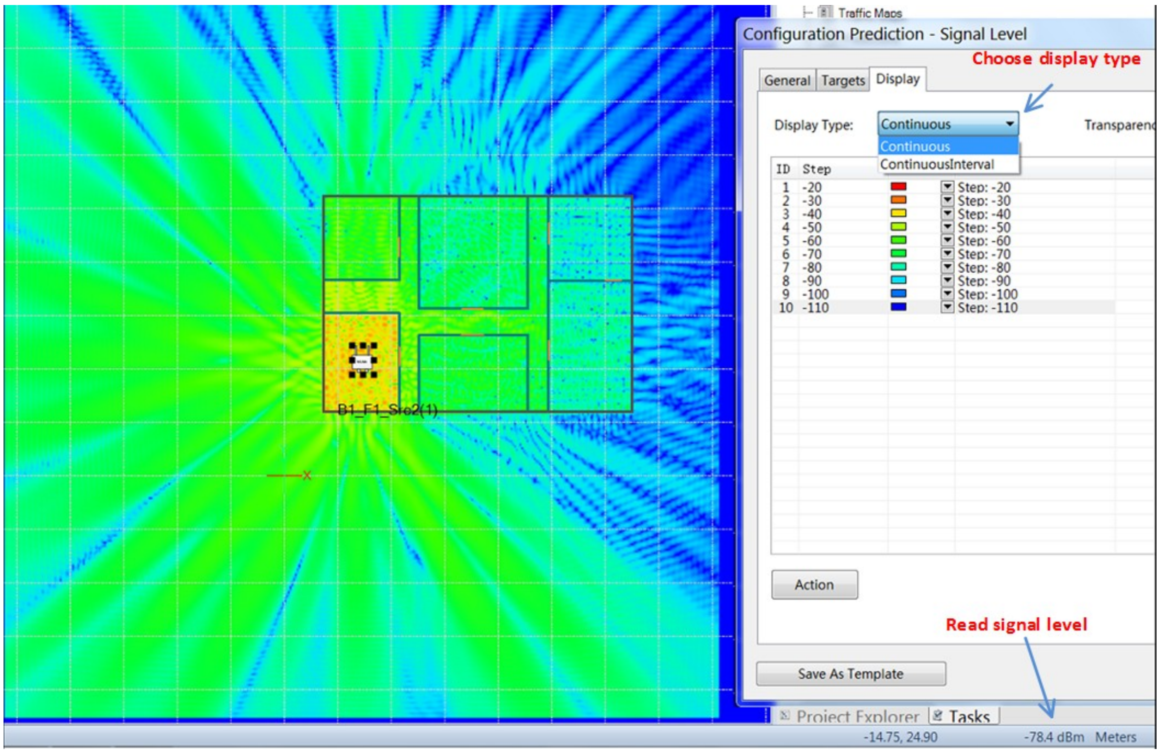


Display Type: ContinuousInterval

ID	MinValue	MaxValue	Color
1	-150	-100	🟦
2	-100	-90	🟦
3	-90	-80	🟦
4	-80	-70	🟩
5	-70	-60	🟩
6	-60	-50	🟩
7	-50	-40	🟨
8	-40	-30	🟧
9	-30	-20	🟥

Figure 21 Display of prediction from MR-FDPF

Figure 21 shows the visualization of prediction result from MR-FDPF. The legend can be customized by user to choose different colors for displaying different signal levels. Noted that the signal levels within the same range are displayed by the same color, this would be less accurate and can cause sharp changes in the signal level plot, as can be seen in Figure 21. To avoid this problem, the iBuildNet also provides user with option to display the prediction continuously.



Display Type: Continuous

ID	Step		
1	-20	🟥	Step: -20
2	-30	🟧	Step: -30
3	-40	🟨	Step: -40
4	-50	🟩	Step: -50
5	-60	🟩	Step: -60
6	-70	🟩	Step: -70
7	-80	🟩	Step: -80
8	-90	🟦	Step: -90
9	-100	🟦	Step: -100
10	-110	🟦	Step: -110

-14.75, 24.90 -78.4 dBm

Figure 22 Display the prediction continuously

In Figure 22, the display type is chosen to be “Continuous” and the signal level plot looks much smoother than the Figure 21. After all, no matter which display type is chosen, it is still the same set of signal levels getting displayed, and the accurate signal level value can always be read at the bottom right corner, as pointed in the Figure 22.

Sometimes, the user may want to know the path loss instead of signal strength. Such option is provided in iBuildNet for the MR-FDPF engine by further manipulating the prediction result with the signal source information, as shown in Figure 23.

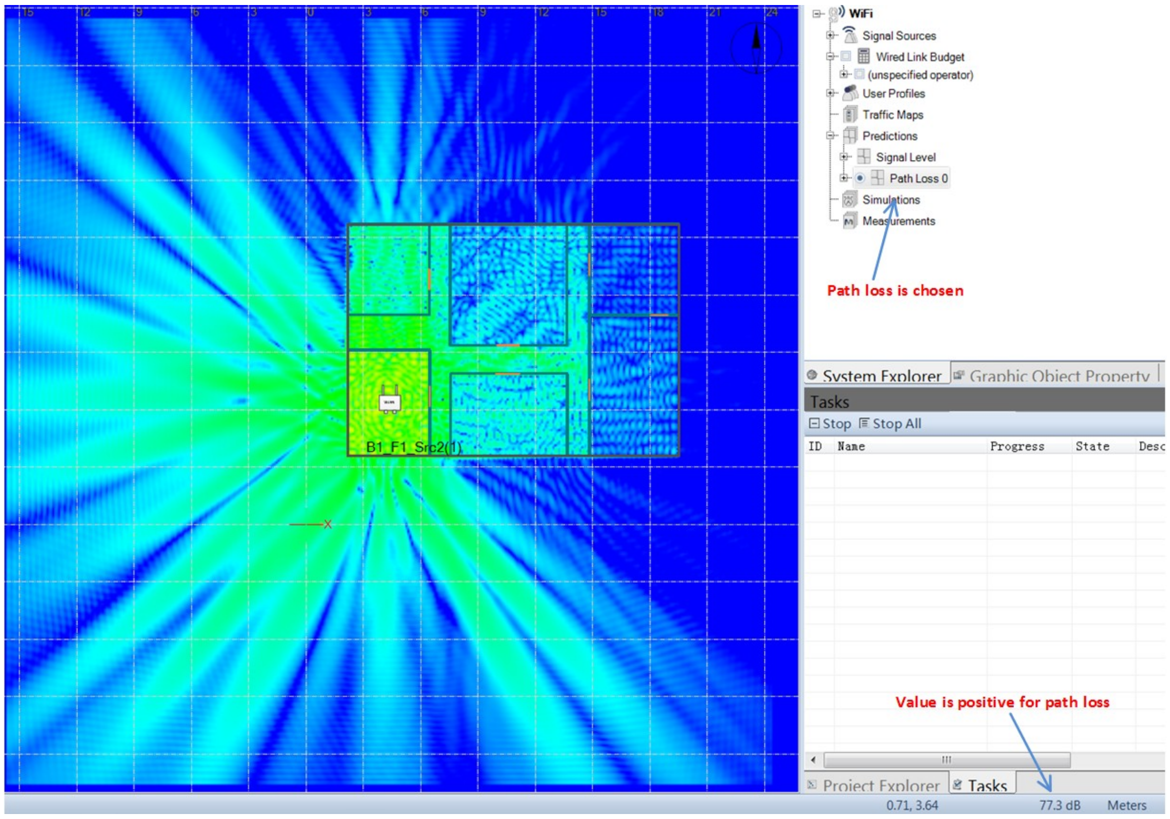


Figure 23 Display of MR-FDPF prediction by path loss

In Figure 23, the path loss is displayed instead of signal strength. The difference between them can be seen by the comparison from Figure 22. It may also be noticed that the read value at the bottom right corner is positive and in dB, as it is for path loss.

In case several signal sources are deployed in one scenario, the prediction result will be given per signal source independently. User can also choose to display the result by best signal level, which means that at one position only the highest received signal level will be displayed among all the signal sources, as shown in Figure 24.

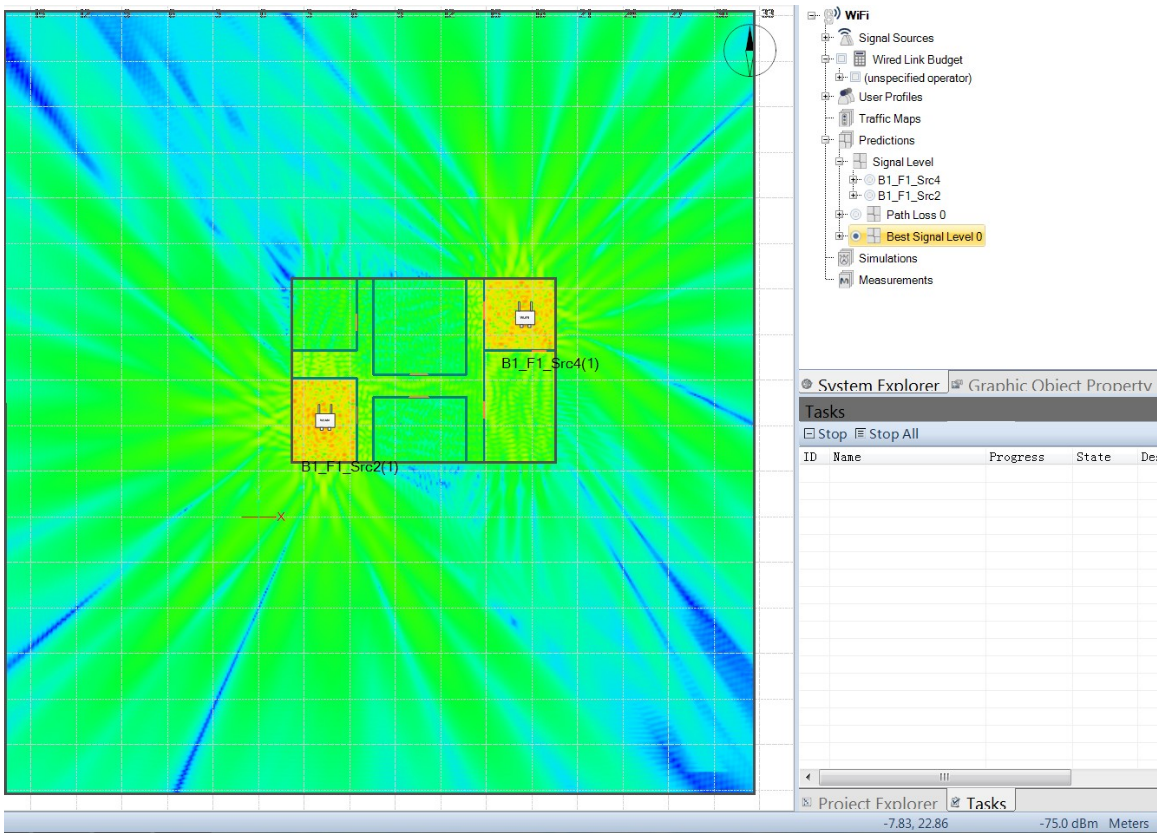


Figure 24 Display of multiple signal sources by best signal level

In Figure 24, two signal sources are deployed and the prediction is displayed by best signal level. So the plot of the signal level at each point is the highest level that can be received from these two sources, according to the prediction result return from MR-FDPF.

Validation of the integration of MR-FDPF engine

The above part shows various ways that are enabled to display and interact with the prediction from MR-FDPF engine. However, as MR-FDPF is integrated as a plugin of the RPMLib engine to work in iBuildNet, of which the detail is given in section 2, we want to make sure that the displayed result in iBuildNet is representing the prediction from MR-FDPF correctly.

To test the correctness, we have enabled iBuildNet to export its prediction data into a text file. Then the same building model is given to the MR-FDPF, and the simulation is processed in MR-FDPF independently without iBuildNet. The raw data received from MR-FDPF, together with the exported data from iBuildNet, are turned into matrix and plotted in MATLAB, as shown in Figure 25.

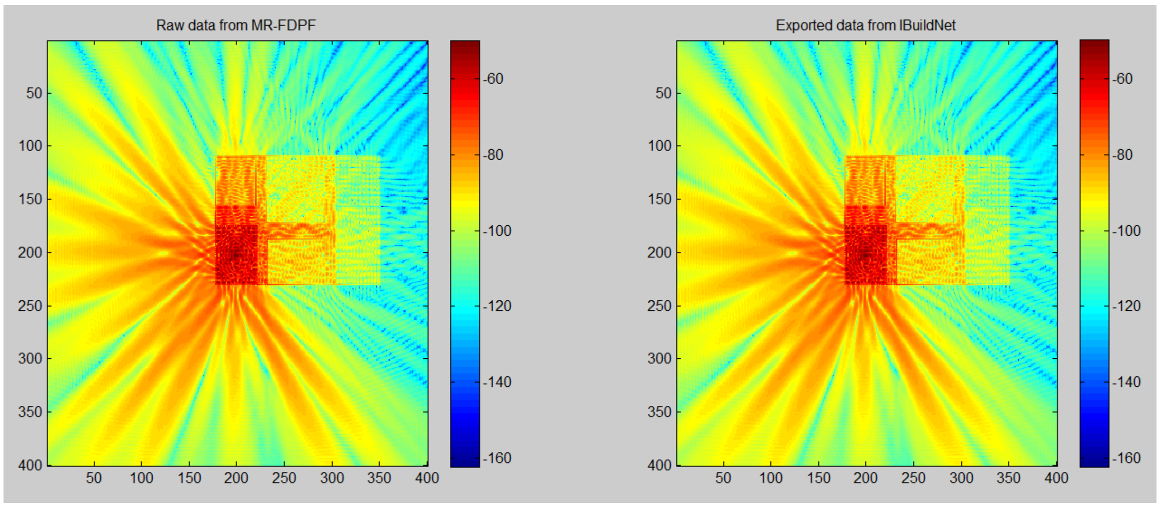
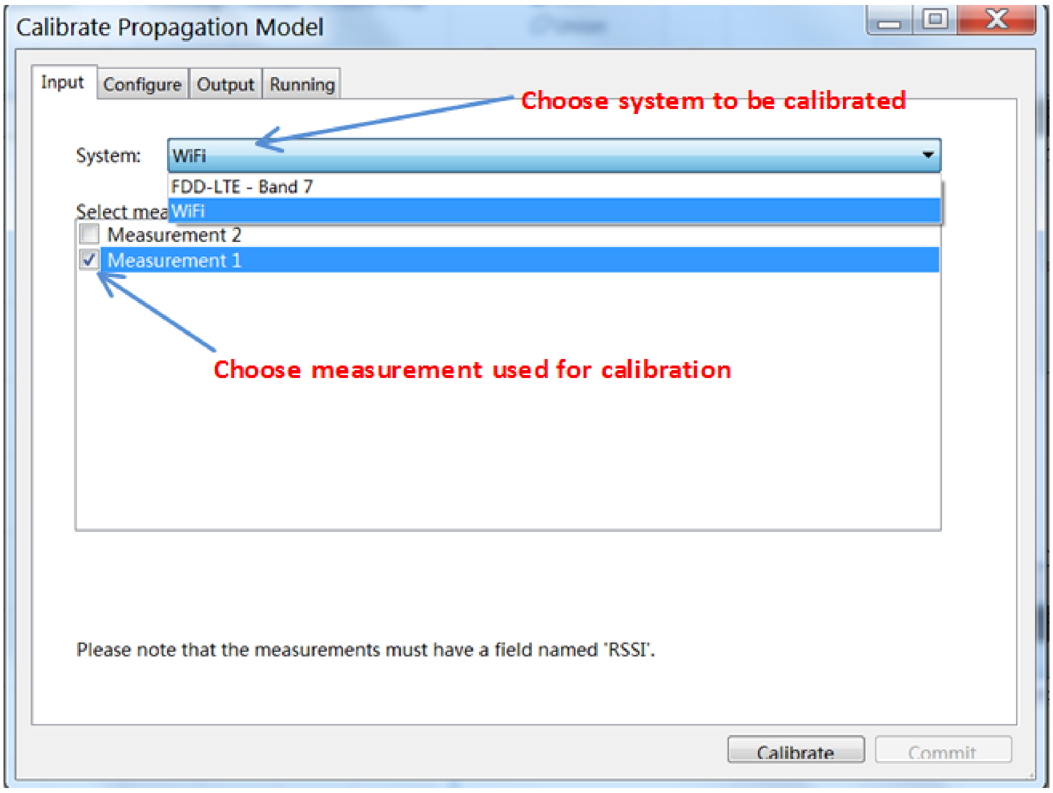


Figure 25 Comparison of result from MR-FDPF and iBuildNet

In Figure 25, the raw data from MR-FDPD is shown on the left and the exported data from iBuildNet is shown on the right. The comparison between them barely shows any difference, and the RMSE is calculated to be $1.899*10^{-4}$. The same process is repeated in other four different scenarios, and the RMSEs are calculated as $1.823*10^{-4}$, $1.965*10^{-4}$, $1.789*10^{-4}$ and $1.849*10^{-4}$. These small RMSEs are caused by conversion from float number to decimal, and thus they are negligible. In conclusion, the two sets of result are identical from the comparison, which means the prediction from MR-FDPF is originally represented in iBuildNet. Or in another word, the integration of MR-FDPF is proved to be correct.

Calibration of the Material for MR-FDPF engine

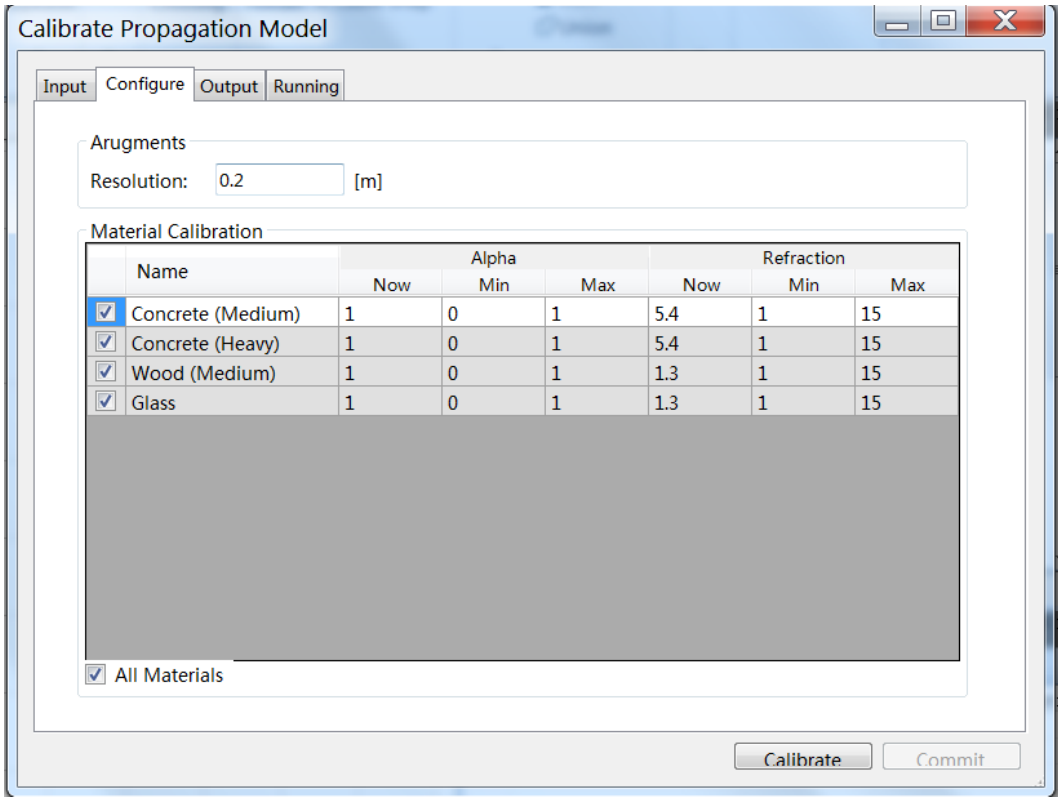
To guarantee the accuracy of the radio propagation prediction, not only we need to make sure the engine is integrated properly, but also we need to ensure that the right material information is given to the engine. For that, we have developed the calibration module in iBuildNet to cope with the calibration of MR-FDPF engine.



Please note that the measurements must have a field named 'RSSI'.

Figure 26 Input tab of calibration form

As shown in Figure 26, on opening the calibration form, the input tab will let user choose the system to be calibrated and the measurement used for calibration.



Arugments

Resolution: 0.2 [m]

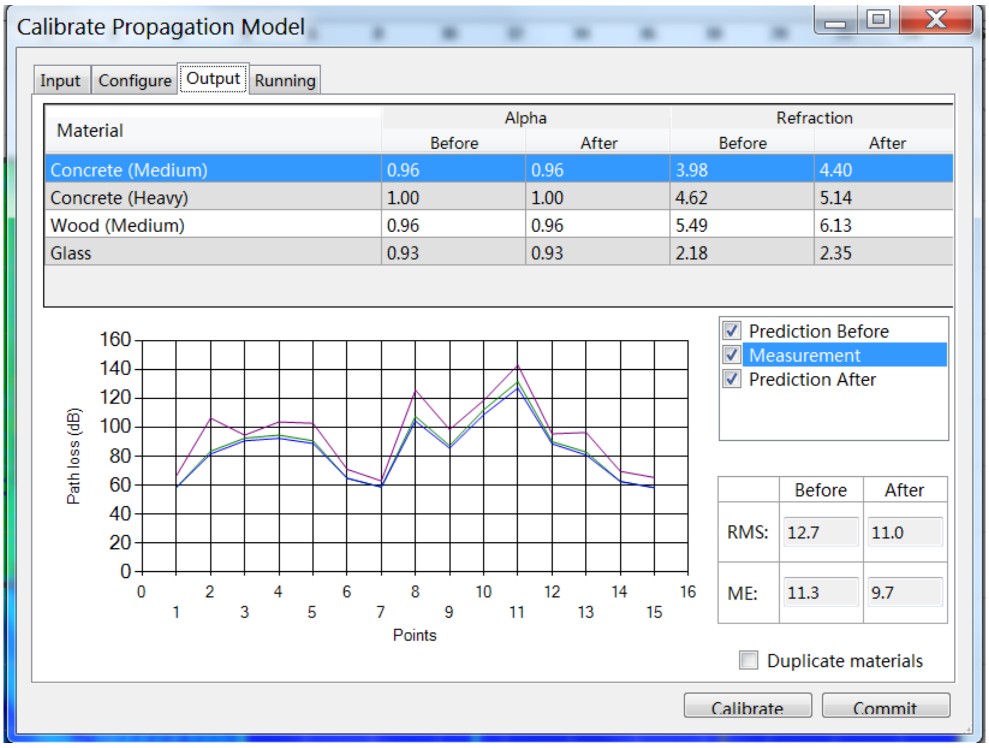
Material Calibration

Name	Alpha			Refraction		
	Now	Min	Max	Now	Min	Max
✓ Concrete (Medium)	1	0	1	5.4	1	15
✓ Concrete (Heavy)	1	0	1	5.4	1	15
✓ Wood (Medium)	1	0	1	1.3	1	15
✓ Glass	1	0	1	1.3	1	15

✓ All Materials

Figure 27 Configure tab of calibration form

Then in the configure tab, the material used in the current project will be listed with their parameters. In Figure 27, as the MR-FDPF engine is used, the relevant alpha and refraction are displayed for the material. In case RPMLib engine is selected, another set of parameters will be displayed for calibration automatically. Also in this tab, user can set the value range of parameters after calibration. For example the alpha is limited between 0 and 1 here due to its nature. The resolution displayed in the form will be the rasterization resolution for processing the calibration, so it needs to be adjusted carefully.



Material	Alpha Before	Alpha After	Refraction Before	Refraction After
Concrete (Medium)	0.96	0.96	3.98	4.40
Concrete (Heavy)	1.00	1.00	4.62	5.14
Wood (Medium)	0.96	0.96	5.49	6.13
Glass	0.93	0.93	2.18	2.35

	Before	After
RMS:	12.7	11.0
ME:	11.3	9.7

Figure 28 Output tab of calibration form

After finishing the calibration, the result will be displayed in the output tab, as shown in Figure 28. On top of the tab, the parameters of each material before and after calibration are listed. And the bottom part plots the comparison between the prediction and measurement. The relevant RMSE will be calculated automatically. After commit the calibration result, it is recommended that user repeats the same process several times for a possible better result. Also reduce the calibration range, as shown in Figure 27, according to certain knowledge of the value will improve the result as well. This is due to the generic algorithm used in the calibration process, as described in section 2. In the end, the RMS is reduced to 6.9dB after 5 repeats.

4. Publications and software results

Simulation test at CITI lab, INSA

In this section, a simulation performance test is carried based on the geometry information of CITI lab at INSA-Lyon. A set of received signal strength is measured at the 3rd floor of the lab to evaluate the prediction accuracy. The simulation shall test the performance of the RPMLib and the MR-FDPF engines, demonstrating the calibration process and give comparison between the engines.

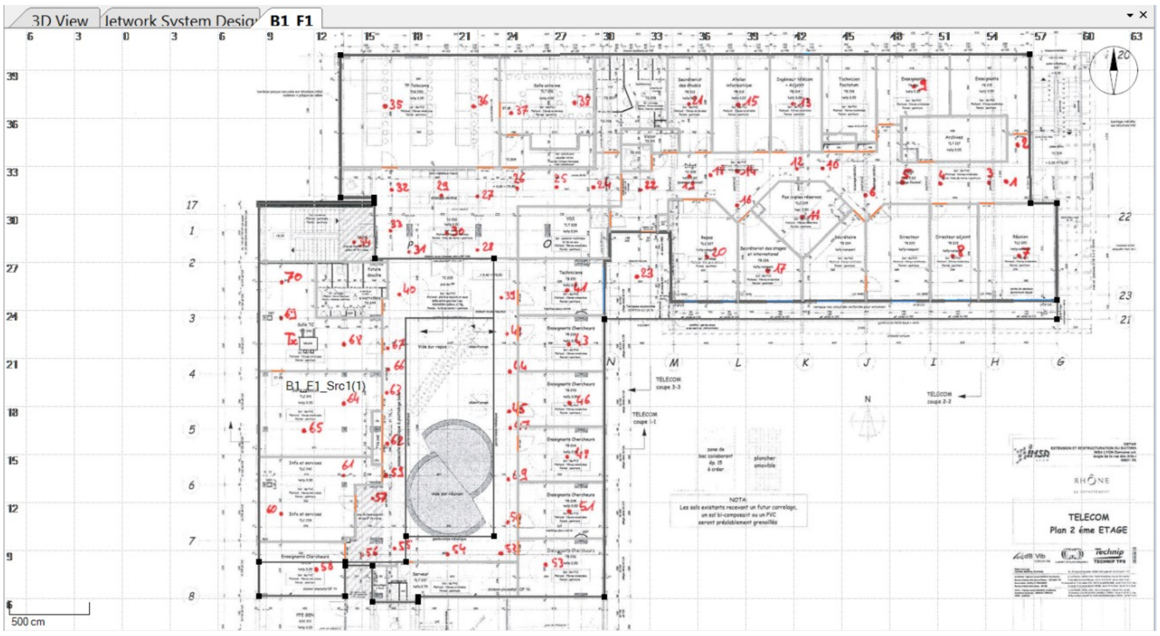


Figure 29 Floor plan of the 3rd floor and model skeleton in iBuildNet

Figure 29 is the floor plan of the 3rd floor shown in the iBuildNet. The building model is sketched based on the floor plan with 1:1 scale to the real object. Different materials are picked up for different part of the building, e.g. concrete for the wall and wood for the door etc. The various materials used could help to truly restore the building, and more importantly improve the calibration performance. This is due to that more types of material could increase the freedom of adjustment while calibrate the material.

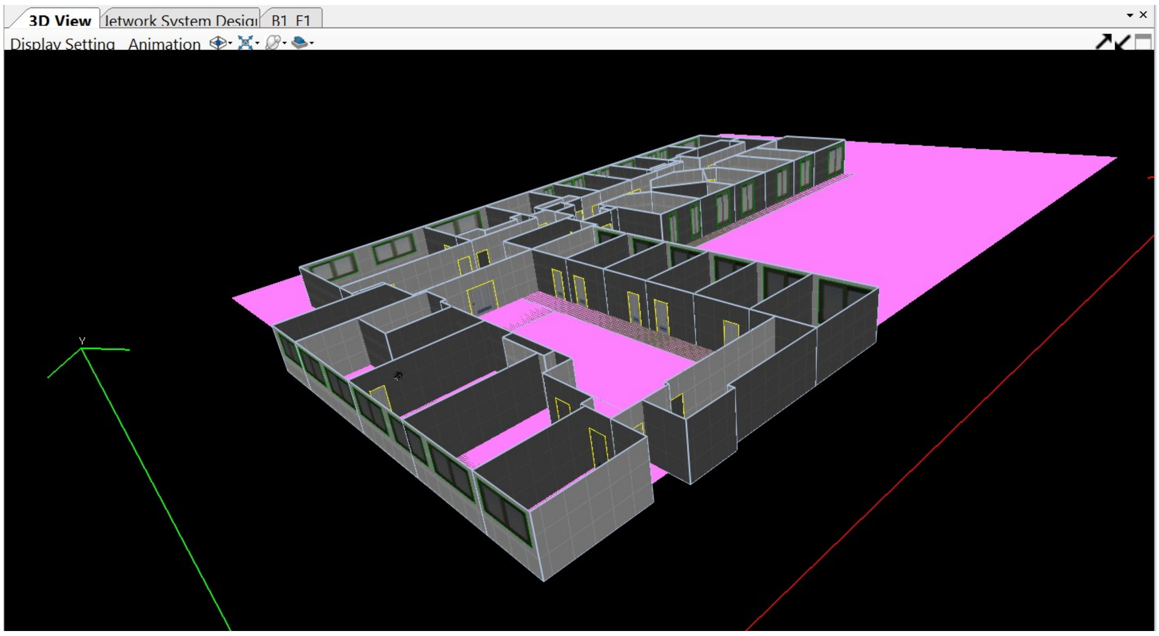
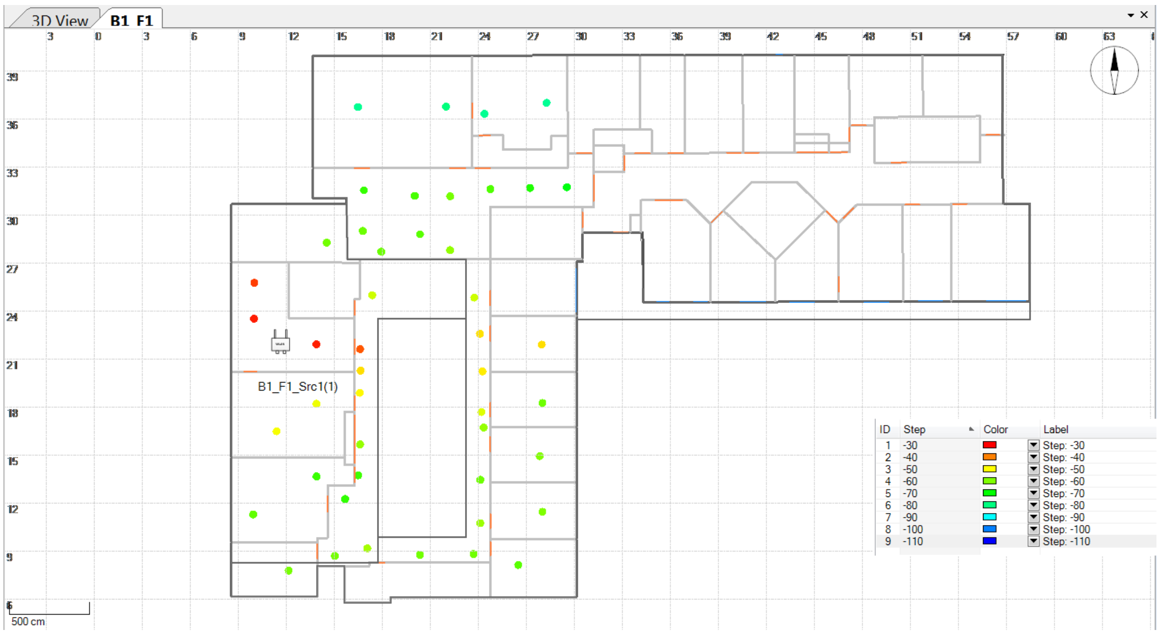


Figure 30 3D building model of the 3rd floor

Figure 30 is the representation of the building in 3D model. The construction of this model in iBuildNet would take less than 10 minutes, if it is operated by a user with sufficient knowledge of the software and the target building environment.

As for the measurement, it is done by using the equipment FSH spectrum analyser [4]. The EIRP is set to 20dBm and the central frequency is set to 3.4GHz to avoid potential interference from Wi-Fi signal. The sensitivity of the analyser is -141dBm/Hz, and the measured bandwidth is set to 300 kHz. Thus the sensitivity in this case equals to $\left(-141 + 10log(3 \times 10^5)\right) \approx -86dBm$. To avoid the inaccuracy, only the signal level, which is measured above -83dBm, is imported into iBuildNet. The set of measurement data imported into iBuildNet is shown below in the Figure 31.



ID	Step	Color	Label
1	-30	🟥	Step: -30
2	-40	🟧	Step: -40
3	-50	🟨	Step: -50
4	-60	🟩	Step: -60
5	-70	🟩	Step: -70
6	-80	🟩	Step: -80
7	-90	🟦	Step: -90
8	-100	🟦	Step: -100
9	-110	🟦	Step: -110

Figure 31 Imported measurement data

The simulation setting is as following. The source power is 20dBm with 0dB noise figure and the transmitting frequency is 3.4GHz, such that the signal source can represent the transmitter in the measurement. For the MR-FDPF engine, due to its nature described in [3], a trade-off is made and the frequency is assumed to be 340MHz. According to the λ/6 limit, the resolution should be less than 0.147m and 0.1m is chosen. For the RPMLib engine, the 0.1m resolution is used as well. Their prediction results are shown below.

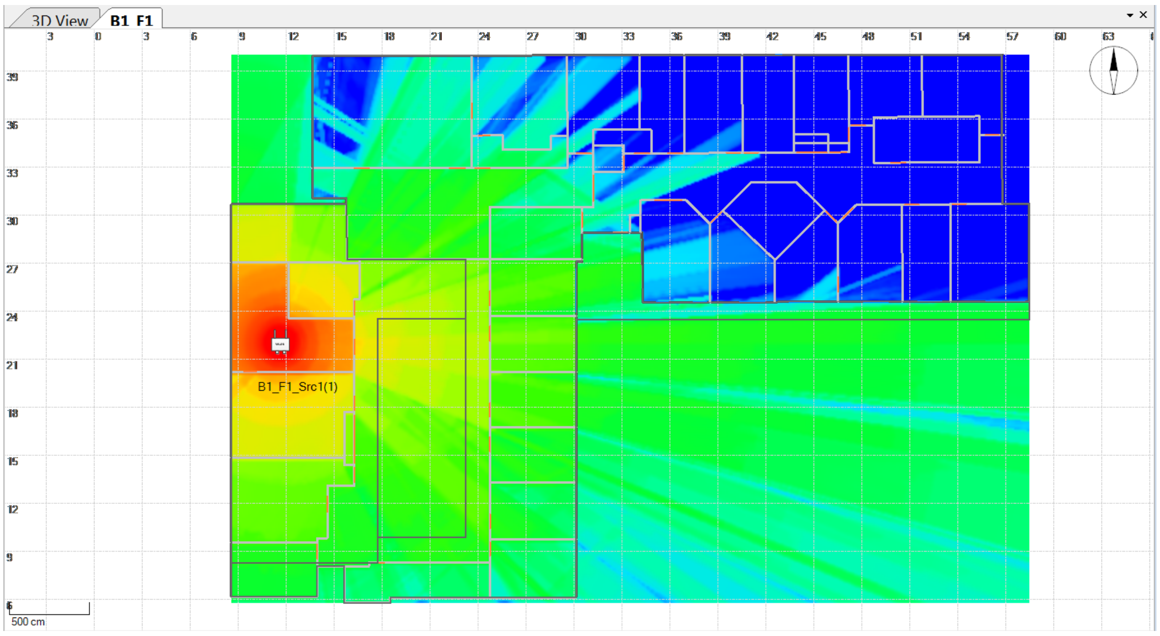


Figure 32 Prediction result of RPMLib

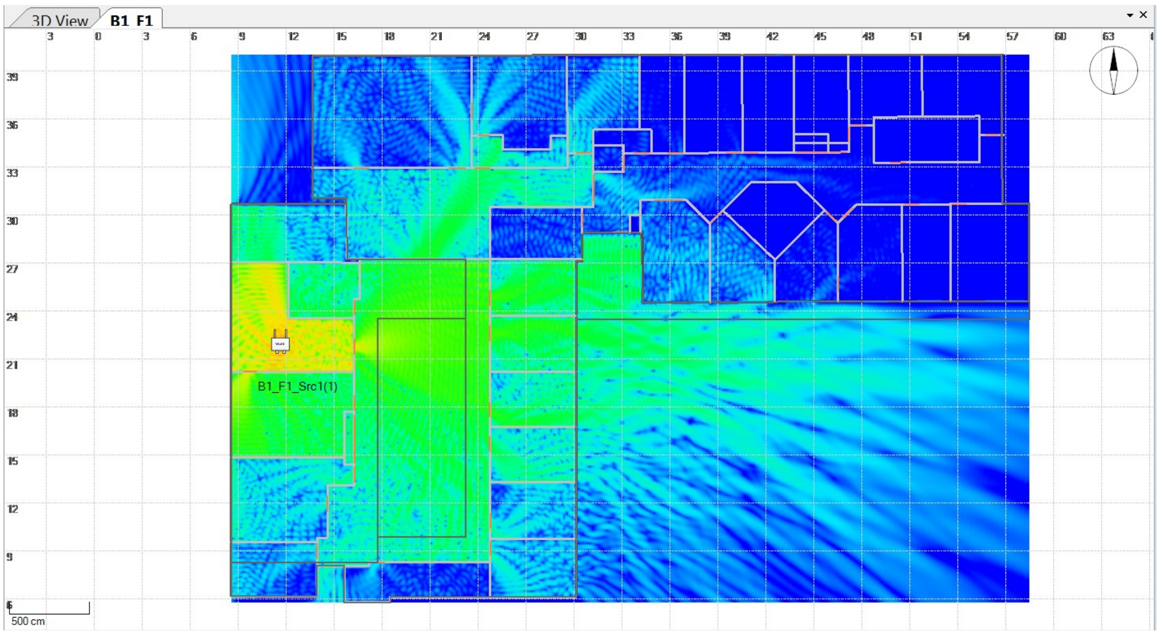
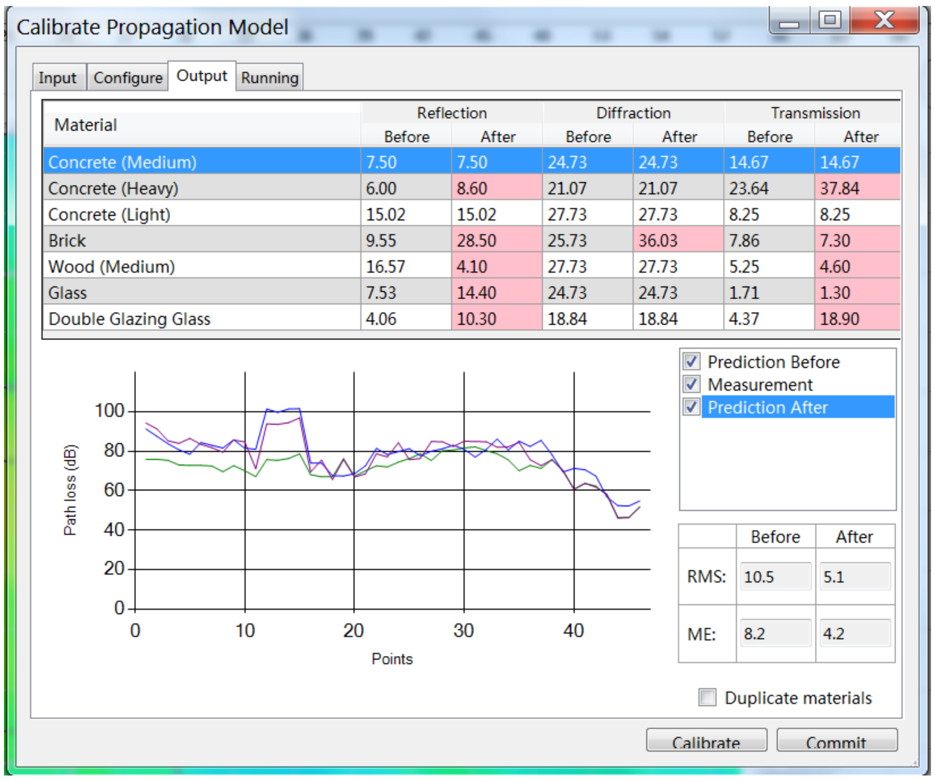


Figure 33 Prediction result of MR-FDPF

From Figure 32 and Figure 33, it can be seen that RPMLib tends to have a more optimistic prediction than MR-FDPF, especially at the range close to the signal source. However, it should be noted that the default material library is based on other scenario and may not represent the CITI environment very well. Calibration shall be done to adjust the material parameters, and the results are shown below.



Calibrate Propagation Model

Input | Configure | Output | Running

Material	Reflection		Diffraction		Transmission	
	Before	After	Before	After	Before	After
Concrete (Medium)	7.50	7.50	24.73	24.73	14.67	14.67
Concrete (Heavy)	6.00	8.60	21.07	21.07	23.64	37.84
Concrete (Light)	15.02	15.02	27.73	27.73	8.25	8.25
Brick	9.55	28.50	25.73	36.03	7.86	7.30
Wood (Medium)	16.57	4.10	27.73	27.73	5.25	4.60
Glass	7.53	14.40	24.73	24.73	1.71	1.30
Double Glazing Glass	4.06	10.30	18.84	18.84	4.37	18.90

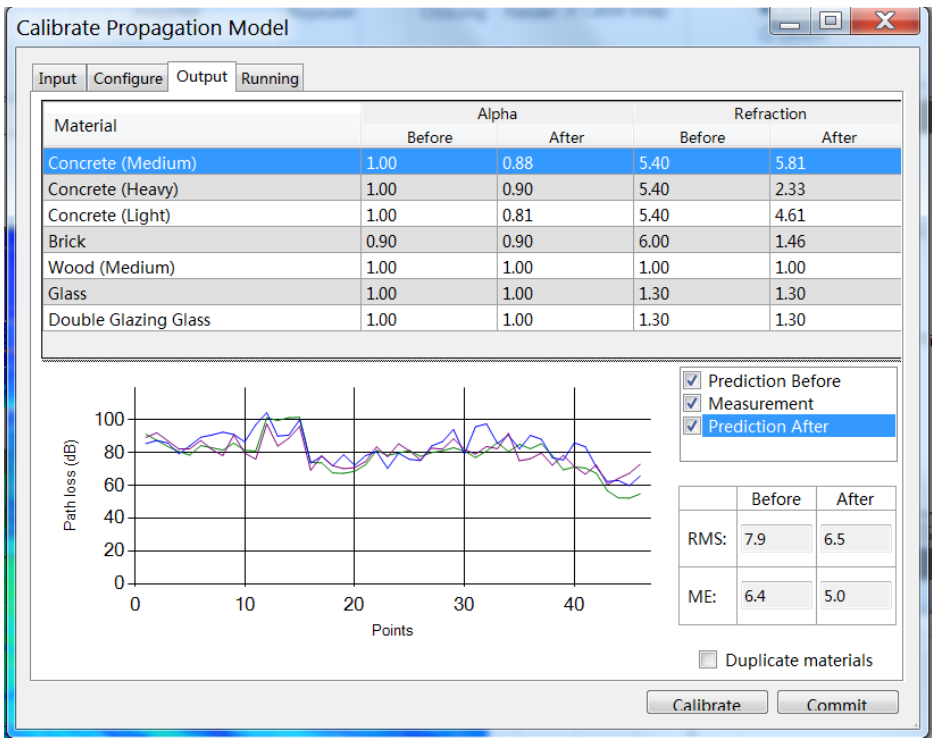
☑ Prediction Before
☑ Measurement
☑ Prediction After

	Before	After
RMS:	10.5	5.1
ME:	8.2	4.2

☐ Duplicate materials

Calibrate | Commit

Figure 34 Calibration result of RPMLib engine



Calibrate Propagation Model

Input | Configure | Output | Running

Material	Alpha		Refraction	
	Before	After	Before	After
Concrete (Medium)	1.00	0.88	5.40	5.81
Concrete (Heavy)	1.00	0.90	5.40	2.33
Concrete (Light)	1.00	0.81	5.40	4.61
Brick	0.90	0.90	6.00	1.46
Wood (Medium)	1.00	1.00	1.00	1.00
Glass	1.00	1.00	1.30	1.30
Double Glazing Glass	1.00	1.00	1.30	1.30

☑ Prediction Before
☑ Measurement
☑ Prediction After

	Before	After
RMS:	7.9	6.5
ME:	6.4	5.0

☐ Duplicate materials

Calibrate | Commit

Figure 35 Calibration result of MR-FDPF engine

Figure 34 and Figure 35 show the calibration results of the two engines, together with statistical results of their predictions. It can be seen that the RMS of RPMLib is 10.5dB before calibration and

5.1dB after, while the RMS of MR-FDPF is 7.9dB before calibration and 6.5dB after. Predictions are improved after calibration and both engines perform quite good results, while RPMLib has a slightly better accuracy in this specific scenario. Below figures are the predictions after calibration shown together with the measurement.

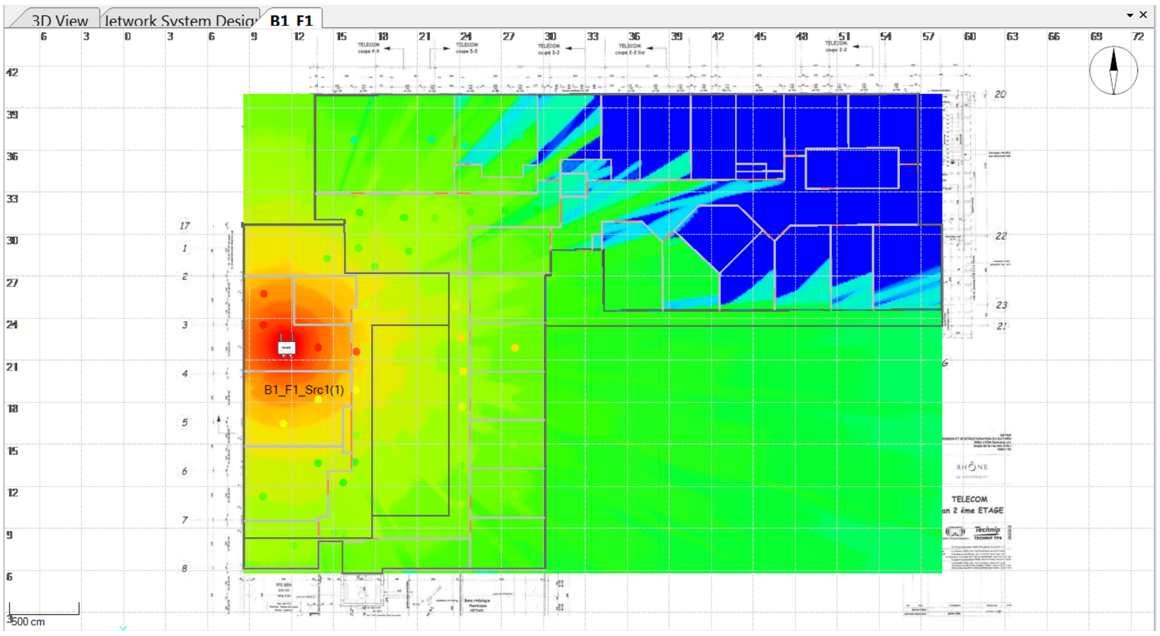


Figure 36 Prediction of RPMLib after calibration

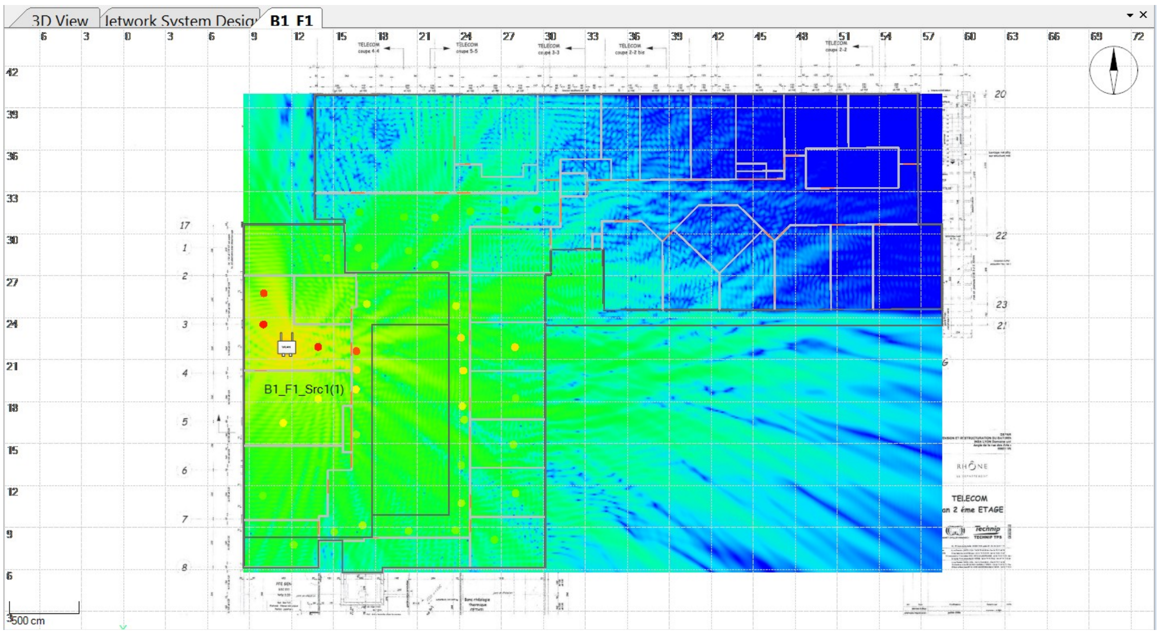


Figure 37 Prediction of MR-FDPF after calibration

Note that the prediction of signal levels from both engines is overall increased after calibration with the measurement. Then the simulation is repeated 4 times with different resolutions to compare the speed of these two engines and their accuracy. The results are shown in the following Table 1:

	0.1m	**0.2m**	**0.3m**	**0.5m**
Used time(RPMLib)	9min44sec	43sec	8sec	3sec

Used time(MR-FDPF)	38sec	8sec	4sec	2sec
RMSE(RPMLib)	5.1dB	5.4dB	5.8dB	6.9dB
RMSE(MR-FDPF)	6.5dB	6.8dB	8.6dB	11.5dB

Table 1 Comparison between the two engines at different resolution

The calculation in Table 1 is performed by a laptop with Intel i7-3610QM CPU and 8GB RAM. The scenario approximately represents a $50 \times 34 \times 3m^3$ space. And the source frequency is adjusted accordingly to the resolution for the MR-FDPF engine. From the Table 1 Comparison between the two engines at different resolution, it can be seen that the time consuming decreases significantly when the resolution is reduced from 0.1m to 0.2m, while both engines still keep good accuracy. The prediction of MR-FDPF starts to corrupt quickly when the resolution goes above 0.3m, while the RPMLib maintains good prediction among all the resolutions. As for the speed, MR-DFPF always consume less time than the RPMLib, especially at the 0.1m resolution. But it should be noted that the current MR-FDPF only performs 2D simulation while RPMLib performs 3D simulation.

Calculation in MATLAB shows that the correlation coefficient of the predictions from the two engines is 0.899 at 0.1m resolution. Below the Figure 38 shows the best linear fitting for these two sets of predictions.

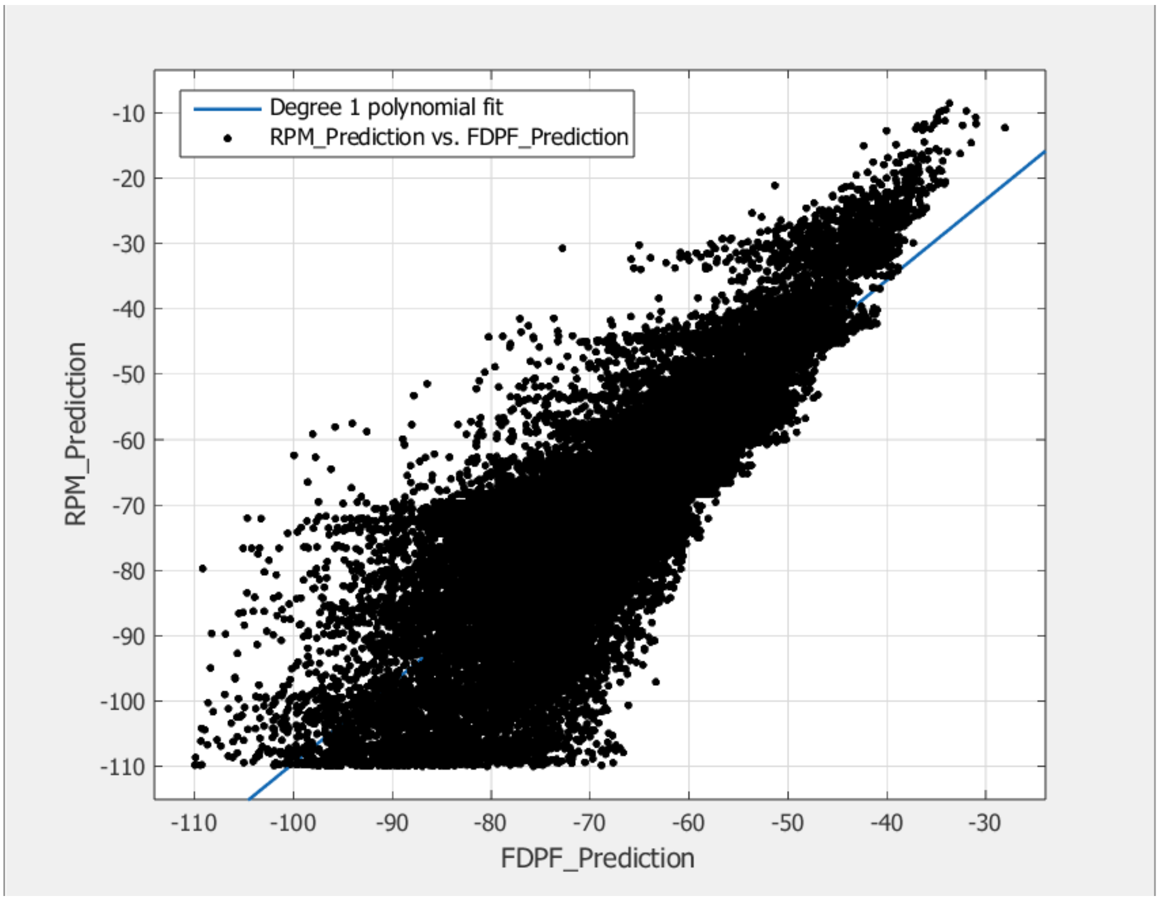


Figure 38 Linear fit of the predictions

In Figure 38, the X axis represents the prediction of MR-FDPF and the Y axis represents the prediction of RPMLib. The fitting polynomial with least RMSE is given as $f(x) = 1.232x + 13.71$. The function indicates that the RPMLib tends to predict higher signal strength than MR-FDPF when it is above 59dBm while vice versa. Look back at Figure 36 and Figure 37, it can be seen that RPMLib does have a higher prediction at close range, which fits better the measurement. While at far range when the number of transmission is becoming large, 8 for example, the calculation load of RPMLib will increase significantly and its prediction is likely to be several tens of dB lower than the measurement. However, the MR-FDPF engine will barely be affected by the number of transmission due to its different algorithm. Although lack of theory, the above result may indicate a way of combining the result from these two engines to achieve a possible better prediction. But it is beyond the scope of this task, and could be left for future research.

5. References

[1] J.-M. Gorce, K. Jaffres-Runser, and G. de la Roche, “Deterministic approach for fast simulations of indoor radio wave propagation,” *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, p. 938,948, 2007.

[2] M. Melanie, “An introduction to genetic algorithms,” *Cambridge, Massachusetts London, England, Fifth printing*, vol. 3, 1999.

[3] G. Roche, K. Jaffres-Runser, and J. Gorce, “On predicting in-building WiFi coverage with a fast discrete approach,” *International Journal of Mobile …*, vol. 2, pp. 3–12, 2007.

[4] “R&S®FSH4 / R&S®FSH8 / R&S®FSH13 / R&S®FSH20 Spectrum Analyzer - Rohde & Schwarz.” [Online]. Available: http://www.rohde-schwarz.com/en/product/fsh4-8-productstartpage_63493-8180.html. [Accessed: 30-Apr-2013].

6. Appendix

Sample Plugin for RPMLib.dll

```
/*
        A Sample Plugin for RPMLib.dll: Free Space Calculation
*/

#include <math.h>
#define sqr(x) (x)*(x)
#define C_FLAG_STOPPING 4

/* helper dist fun */
inline float
__dist(
        const int p1[3],
        const int p2[3],
        const float res[3]
)
{
        return (
                sqrt(
                        sqr((p1[0] - p2[0]) * (res[0])) +
                        sqr((p1[1] - p2[1]) * (res[1])) +
                        sqr((p1[2] - p2[2]) * (res[2]))
                )
        );
}

/* information structure */
struct infotype
{
        int versionmajortracer;
        int versionminortracer;
        int versionmajorutility;
        int versionminorutility;
        char *dllver;
        char *dllpath;
};

/* app structure */
struct apptype
{
        float eval;
        float temp;
        int currentstage;
        int totalstage;
        int currentid;
        int totalid;
        int currenttx;
        int totaltx;
        int flag;
        bool pausesimu;
        char msg[256];
```

```
};

/* plugin entry */
int
RPMLib(
        bool *contd,
        void *par,
        apptype *app,
        void *ranges,
        void *rangesmax,
        void *mat,
        void *objlst,
        void *paths,
        void *vc,
        void *pl,
        int *dim,
        float *res,
        int *tx,
        float *txp,
        float **ant,
        float bearing,
        float etit,
        float mtit,
        float gain,   // dBi
        float noise,
        float dbm,   // dBm
        float freq,  // GHz
        infotype *info
)
{
        /* free space computation starts here */
        // compute some values
        float fspl = 20.00f * log10(freq * 1000.0f) + 32.44f;
        float ***pt = (float ***)pl;

        /* let the caller knows the progress */
        int x = dim[0];
        int y = dim[1];
        int z = dim[2];
        int total = x * y * z;
        app->totalid = total;
        app->currentid = 0;

        // loop each pixel in 3D path loss matrix
        for (int i = 0; i < x; i ++)
        {
                for (int j = 0; j < y; j ++)
                {
                        // progress information
                        (*app).currentid += (z);
                        // exit the simulation if 'cancel' is clicked
                        if ((app->flag & C_FLAG_STOPPING) > 0)
                        {
                                break;
                        }
                        for (int k = 0; k < z; k ++)
```

```
                    {
                            // current pixel
                            int cur[3] = {i, j, k};
                            // compute the path loss
                            float dist = __dist(cur, tx, res);
                            if (dist < 0.001f)
                            {
                                    dist = 0.001f;
                            }
                            float p = fspl + 20 * log10(0.001f * dist);
                            if (p < 0)
                            {
                                    p = 0;
                            }
                            // output: update the pixel value in 3D-matrix
                            pt[i][j][k] = p;
                    }
            }
    }
    // tell RPMLib.dll task is accomplished
    *contd = false;
    return (0);
}
```