SAPIENZA
UNIVERSITÀ DI ROMA

Performance evaluation of VLC-enabled mobile networks

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Informatica

Candidate

Luci Emiliano
ID number 1665528

Thesis Advisor

Petrioli Chiara

Academic Year 2016/2017

Performance evaluation of VLC-enabled mobile networks
Bachelor thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: luci.1665528@studenti.uniroma1.it

Contents

Summary

In this thesis we'll present an approach to the simulation and performance evaluation of VLC-enabled networks. We will first introduce the potentials and challenges of Visible Light Communication. We'll then take a look at the OMNeT++ simulator and our own set of modules, dwelling on some of the problems that are of particular interest. Finally, we'll go through a suite of tests designed to confirm the accuracy of our simulator as well as analyze the performance of VLC systems in plausible real life scenarios.

Acknowledgements

Thanks go to my thesis advisor, Chiara Petrioli. Special thanks are also due to Petrika Gjanci, Nupur Thakker and Georgia Koutsandria without whom it wouldn't have been possible to complete this work.

Chapter 1

Introduction

Over the past few years the number of devices connected to the Internet has grown exponentially [1] and an always growing portion of these devices do so wirelessly. The portion of the EM spectrum available for communication however has stayed pretty much the same. This is leading to an overcrowding of these bands that has motivated the research for new ways to connect more devices together without hindering the overall performance of the network.

One of such technologies is Visible Light Communication (**VLC**). As the name indicates VLC utilizes the portion of the spectrum from 390 to 700 nm (430 to 770 THz). This poses unique advantages and challenges inherent to the characteristics of this band. Deployment of visible light communications systems or of hybrid systems made of mobile devices equipped with both radio and VLC communications technologies, is considered one of the key directions for achieving the level of densification and performance expected for 5G and beyond 5G systems. VLC systems also offer the added bonus to be able to exploit existing light infrastructure for deploying a low cost network infrastructure.

In this thesis we have taken first steps into the direction of characterizing the performance of such a VLC based system. In particular, a first objective has been that of developing a simulation framework to evaluate the performance of both static and mobile VLC networks. The simulator has then been used to study and analyze a number of likely real life scenarios such as those of a confined room with both immobile and moving devices.

1.1 Modeling Visible Light Communications

An extremely comprehensive survey of possible potential and challenges that come with VLC systems has been made in [2]. We will now present just a quick overview focusing on what's most relevant for the purpose of this thesis. The first very evident characteristic of visible light is that it cannot pass through objects that we (humans) perceive as opaque: this feature is both desirable when considering that interference between, say, VLC Devices can be almost completely eliminated by

means of an opaque body such as a wall, and not as much so when having to design systems that, while not completely dependent on it, mostly rely on an uninterrupted Line of Sight (**LoS**) between devices for efficient communication.
Another major challenge must be faced when taking into account sources of noise that in some cases can completely disrupt a VLC link. A major source is, of course, the Sun.
As can be observed in Fig.1.1 the Sun has its peak emission right in the visible light band while the irradiance becomes negligible at lower frequencies such as those utilized for *Wi-Fi* transmission. This must be taken into account as daylight is almost ubiquitous in every space that has a window of some sort.
To make matters worse sunlight is not a nice and constant source of noise but can

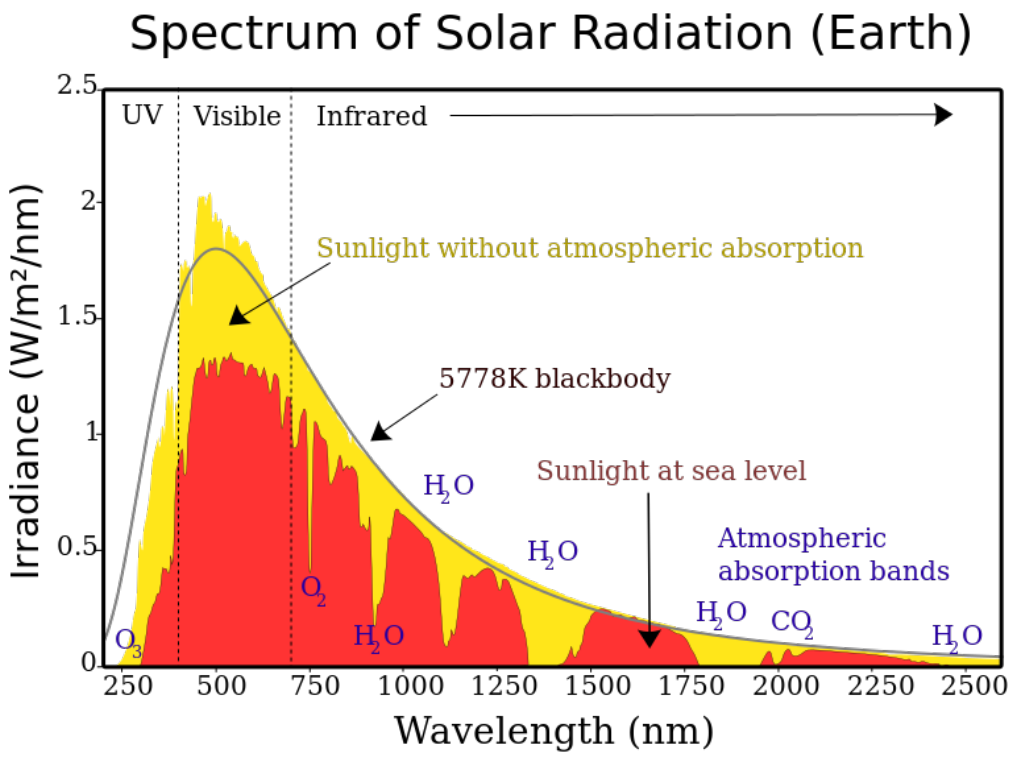

Figure 1.1. Sun Spectrum

be influenced by a number of factors: first and foremost the time of day and then of course the weather conditions. As displayed in Fig.1.2 the spectral power is subject to significant changes that are especially evident for wavelengths longer than 475 *nm* due to the effects of Mie scattering.

Another important factor to consider when designing a VLC system is that indoor areas are full of reflective surfaces that cause multipath interference. This is especially true if a VLC Access Point doubles as a lighting fixture as they're usually placed in an elevated place and are powerful enough that light from it still has significant power after bouncing 1 or 2 times on a surface, sometimes comparable to the intensity of the signal that one is trying to transmit. In their work,Tagliaferri

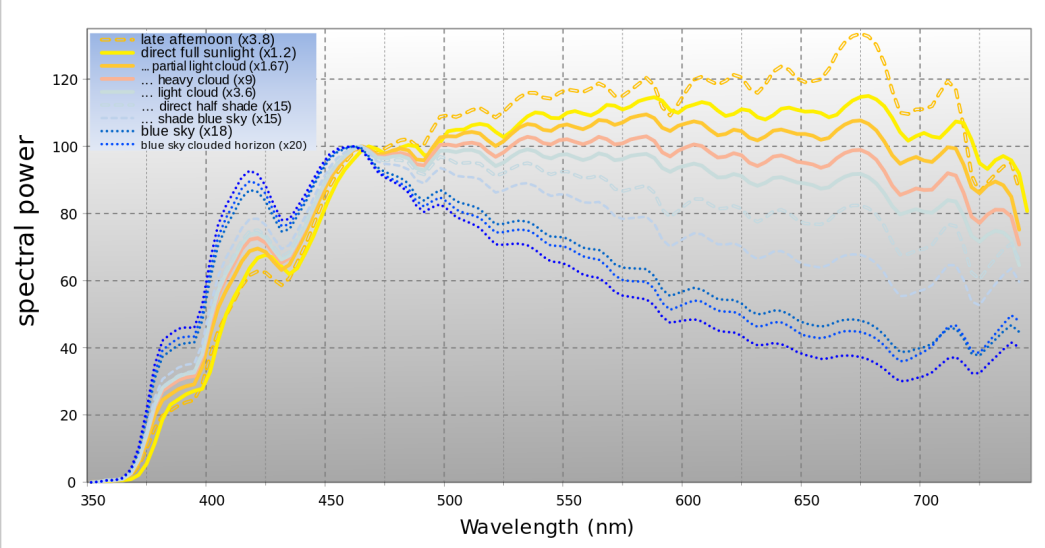

Figure 1.2. Spectrum in different conditions

and Capsoni [3] provide an implementation of a simulator written in MATLAB® that is able to simulate an indoor scenario with different light sources and furnitures each with its own reflection pattern.

The simulation however is concerned only on the modeling of the VLC channel and does not take into account the full protocol stack. In [4] instead an extension to ns3 is presented that simulates a VLC channel down to the physical layer. Their model, further confirmed by Ab-Rahman et al. [5], takes into account a good number of factors such as the relative orientation of the transmitter and receiver and their Field of View and can thus be used to simulate moving nodes using ns3 MobilityModel. It should be noted however that the scope of the simulation itself is very limited and only models point-to-point communication without considering the effects of external noises, multipath interference and interference between different transmitters. Furthermore it does not implement any type of Media Access Control (**MAC**).
Implementing a form of MAC for VLC network has proven itself to be a tricky challenge when considering pure VLC systems. This is justified by the fact that efficient access control requires a constant coordination between the elements of the network in order to maximize its performance and so far VLC system have been unable to provide that. Most of the current research in fact focus on simplex VLC connections between transmitters and receivers. One of the reason for this is that VLC itself is a relatively new technology and the details of the physical layer are still being figured out, but that's just a transitory limitation; the other obstacle is woven much deeper into physics itself and consist in the fact that producing a good VL signal is a relatively power intensive task, since the signal intensity has to match and overpower the aforementioned noise sources to yield a good SINR. At the current state of battery technology this proves to be too much of a toll for mobile devices. To provide access control most solutions [6] [7] [8] thus revolve around hav-

ing an upLink that is based on existing wireless technologies such as WiFi. While it might seem that relying on a separate radio channel to realize a duplex connection defeats the whole purpose of building a VLC system that's not the case: exception made for specific scenarios, most of the connections data flow is in downLink. If one could unburden most of the load from the existing wireless connections to a different channel that would still provide an enormous benefit in terms of goodput in a network. This is further accentuated by the inherently parallel nature of a VLC networks since each transmitter serves at most a handful of devices at a time and thus one can have lots of different independent data flows all running at the same time.

Chapter 2

The Alaula simulator

2.1 Contribution of the thesis

This thesis presents Alaula (Ke Alaula is Hawaiian for "morning light" or "dusk glow"), a module for the OMNeT++ simulator that provides a framework to build and simulate VLC-enabled networks up to the link layer of the protocol stack. The simulator handles node mobility and interference and provides a minimal MAC protocol that makes use of a separate wireless channel. The effort of this thesis has been put in the development of simulator itself as well as the ideation and analysis of a few case scenarios that has been deemed representative of a subset of the possible ones that can be found "in the wild".
We chose to focus on a few major components while developing the simulator while leaving for future iterations those that were not immediately essential. Namely, the greater part of our effort went into modeling the physical layer which include the channel and the way it interacts with the devices. We also made an attempt to tackle the problem of developing a MAC protocol based on a separate wireless channel which is described in greater detail later on.

2.2 VLC Communication Modeling

The same model as in [4] and [5] is used as it is moderately simple while still being sufficiently accurate. As illustrated in Fig.2.1 this model assumes that both the transmitter and receivers have very sharply constrained Field of Emission and View respectively and does not take into account optical phenomena such as diffusion that would soften these edges. Still, if we don't focus excessively on edge cases this model is "good enough" and more importantly computationally inexpensive. This is not much required when the number of devices in the network is restricted but can become a problem as it grows since recomputing the state of the network requires a non-negligible number of operations that would be better suited for a GPU rather than a CPU.

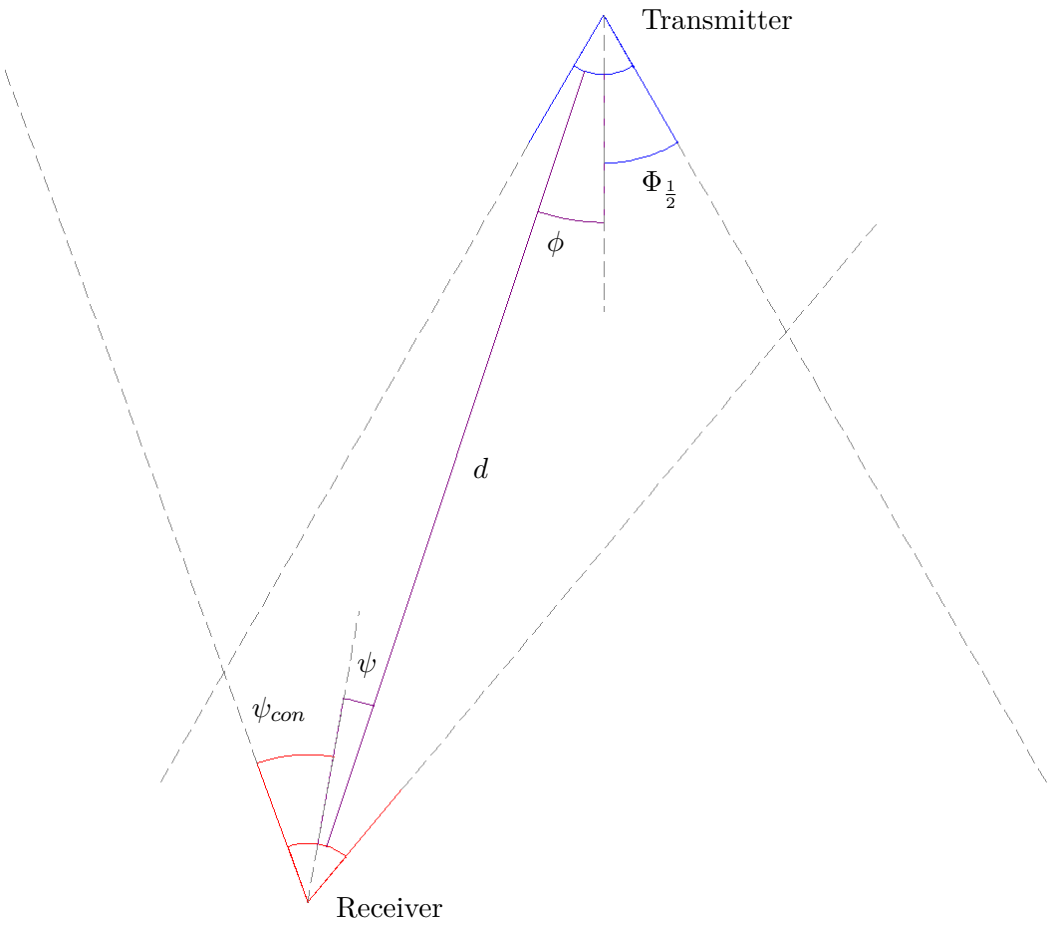

Figure 2.1. A representation of the model

We will now go through the model used in the simulator. Recall that for a transmitted signal it holds:

$$P_r(t) = Hs(t) + n(t) \tag{0}$$

Where $r(t)$ is the received power, H is the channel gain, $s(t)$ is the transmitted signal power and $n(t)$ is the noise term. We model our transmitters as Lambertian surfaces, illustrated in Fig. 2.2.

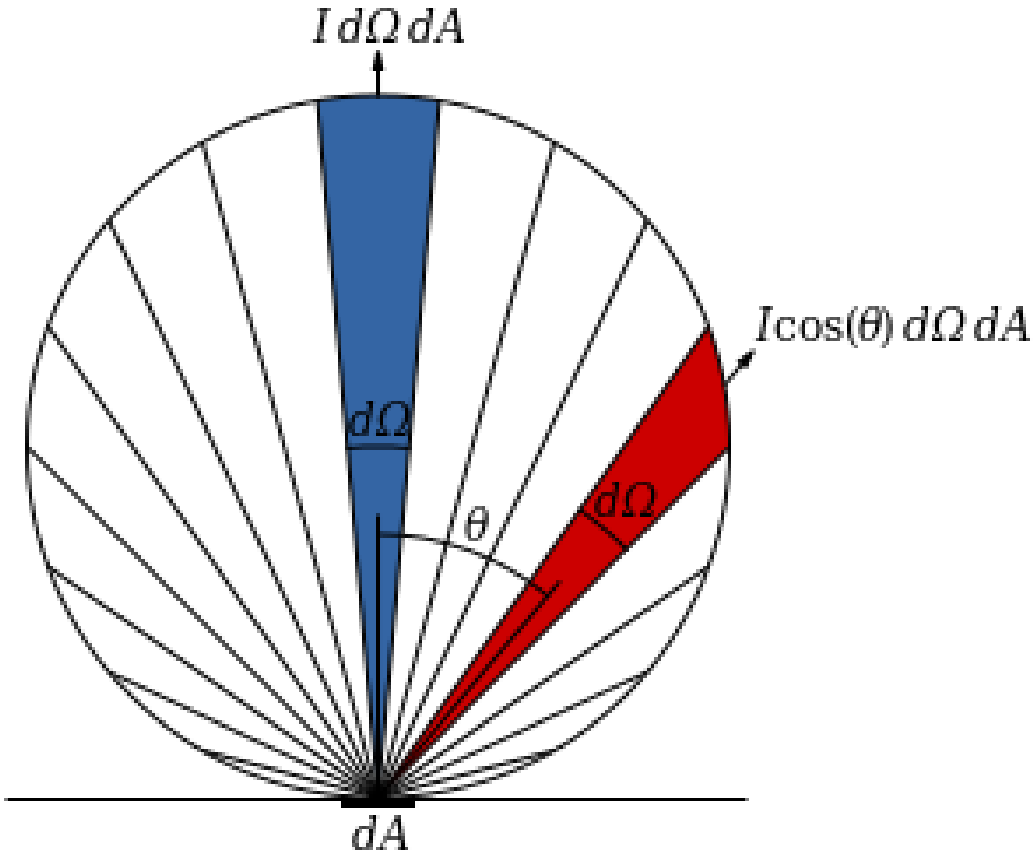

Figure 2.2. A Lambertian emitter

Since this is the same assumption made in [2] we can thus follow their derivation of the equations for the optical channel gain. If we consider a dominant LoS path we then have:

$$H = \frac{(m_l + 1)A}{2\pi d^2} cos^{m_l}(\phi) T_s(\psi) g(\psi) cos(\psi) \tag{1}$$

Where m_l is the Lambertian order of the transmitter given by:

$$m_l = -\frac{ln(2)}{ln(cos(\Phi_{\frac{1}{2}}))} \tag{2}$$

with $\Phi_{\frac{1}{2}}$ transmitter semi-angle, A is the photodetector area, d is the distance between the transmitter and receiver, $T_s(\psi)$ is the optical filter gain and $g(\psi)$ is the gain of the receiver optics. This is given by:

$$g(\psi) = \begin{cases} \frac{n^2}{sin^2(\psi_{con})} & \text{if } \psi \leq \psi_{con} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where n is the refractive index of the concentrator and ψ_{con} is the photodetector FoV semi-angle.

We are now able to compute the Signal plus Interference to Noise Ratio for a receiver. That is:

$$SINR = \frac{(P_r R)^2}{{\sigma_n}^2 + \sum_{i=0}^{|E_{FoV}|} \{(P_r(e_i)R)^2 + {\sigma_{shot}}^2(P_r(e_i))\}} \tag{4}$$

With P_r being the received power, R the responsivity of the photodetector, that is the current generated per watt, ${\sigma_n}^2$ the total noise variance and $|E_{FoV}|$ the cardinality of the set of active emitters that affect the receiver because they are in each other's FoV. The total noise variance is the sum of the shot noise variance and thermal noise variance:

$${\sigma_n}^2 = {\sigma_{shot}}^2 + {\sigma_{thermal}}^2 \tag{5}$$

$${\sigma_{shot}}^2 = 2qRP_rB + 2qI_BI_2B \tag{6}$$

$${\sigma_{thermal}}^2 = \frac{8\pi kT_kC_{pd}AI_2B^2}{G_{ol}} + \frac{16\pi^2kT_k\Gamma{C_{pd}}^2A^2I_3B^3}{g_m} \tag{7}$$

Please refer to the table of symbols for the meaning of each of the constants in the equations above.
We are now able to calculate the **B**it **E**rror **R**ate of a transmission. The BER is a function of the $SINR$ and the modulation scheme adopted. Two modulations schemes have been implemented in the simulator: **P**ulse **A**mplitude **M**odulation and **V**ariable **P**ulse **P**osition **M**odulation. For VPPM it holds:

$$BER_{VPPM} = \begin{cases} Q\left(\sqrt{\frac{SINR}{2b\alpha}}\right) & \text{if } \alpha < 0.5 \\ Q\left(\sqrt{\frac{SINR(1-\alpha)}{2b\alpha^2}}\right) & \text{if } \alpha \geq 0.5 \end{cases} \tag{8}$$

Where α is the transmitter duty cycle , b is a factor relating noise bandwidth to signal bandwidth and $Q(x)$ is the tail probability of the standard normal distribution, that is:

$$Q(x) = 1 - \Phi(x) = \frac{1}{2}erfc\left(\frac{x}{\sqrt{2}}\right)$$

Where $\Phi(x)$ is the cumulative distribution function of the SND and $erfc$ is the complimentary error function.
For PAM we instead consider the Symbol Error Rate:

$$SER_{PAM} = \frac{2(M-1)}{M}Q\left(\frac{\sqrt{SINR}}{(M-1)}\right) \tag{9}$$

Where M is the modulation order, that is, the number of symbols. Notice that if $M = 2$ we actually got On Off Keying modulation.
We can now compute the Packet Error Rate for both VPPM:

$$PER_{VPPM} = 1 - (1 - BER_{VPPM})^{8p} \tag{10}$$

and PAM:

$$PER_{PAM} = 1 - (1 - SER_{PAM})^{\frac{8p}{\log_2(M)}} \tag{10}$$

With p being the packet size in bytes and 8 being the conversion constant between bytes and bits.
Finally we are able to compute the throughput as:

$$\text{throughput} = \frac{8pN_{rx}}{T} \tag{11}$$

With N_{rx} being the number of received packets in the time period T. Notice that we don't make difference between goodput and throughput since the encoding of the data is not taken into consideration.
This concludes the description of the model.

2.3 Introduction to OMNeT++

OMNeT++ (**O**bjective **M**odular **Ne**twork **T**estbed in C++) is "an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators". OMNeT++ only provides the tools to build simulation frameworks. Many of these, such as INET/INETMANET, MiXiM and Castalia (and in its own small way Alaula!) happen to be related to computer networks simulations. OMNeT++ is a discrete event simulator, this essentially means that time does not flow by itself, as one is hopefully used to, but instead is made to "tick" by events happening in the simulator and *only* by them. This is particularly useful since one does not want to be bounded, for example, by the computing power of the machine one's running the simulation on. By tying the passage of time to the events hundreds of year could pass to respond correctly to an event and still no time would've passed in the simulator!
The core concepts of OMNeT++ are those of modules, gates and messages. A module is an active component of a simulation, modules can be "atomic" and are called simple modules or they can be grouped together to form compound modules (with any hierarchy level). Modules can also extend each other and, in a Java-like fashion, implement any number of modules interfaces.
Gates are the windows on the simulation world of the modules. While not prohibited to do otherwise the standard approach for having modules communicate is to link their gate together with a channel and then send messages via the gate. Sending and receiving messages are two examples of what is considered to be an event.
As said, OMNeT++ and its modules are written in Ctexttt++ but simulation developers are provided with two other useful tools: the NED (**NE**twork **D**escription) language and the configuration language of the .ini files that is used to help specify the values of the parameters of a simulation in an easy and assisted manner. The Network in NED is to be intended in a broad sense as a network of modules. It is in fact the language used to describe the structure of the modules, implement the aforementioned concept of module inheritance, composition and interface, describe the network topology and additionally provide a neat way to organize the modules in packages.
That being said we're now ready to dive in the specifics of Alaula.

2.4 The Simulator

We start from an overview of the modules with the package diagram, represented in Fig. 2.3.

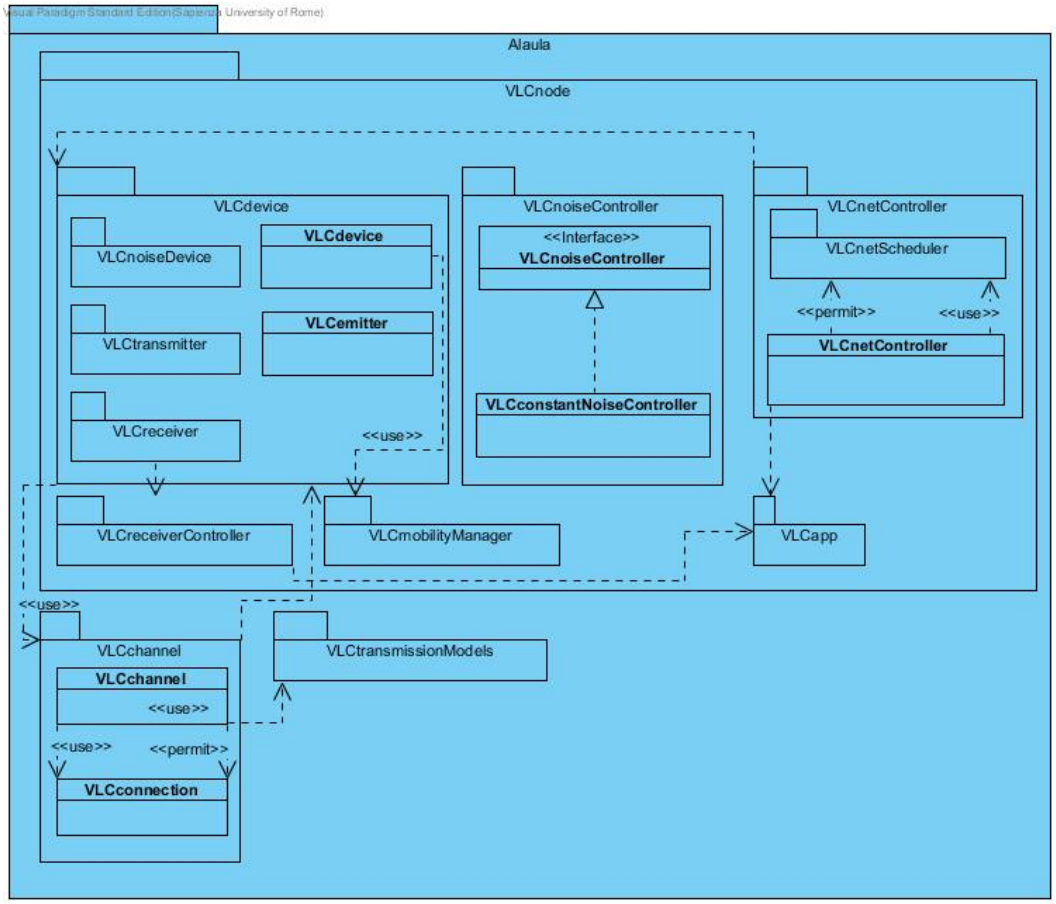

Figure 2.3. Alaula Package Diagram

We will now proceed to take a look at each package individually and see how they interact with each other.

2.4.1 VLCnode

The first package we'll look at is the **VLCnode** package whose "innards" are represented in Fig. 2.4

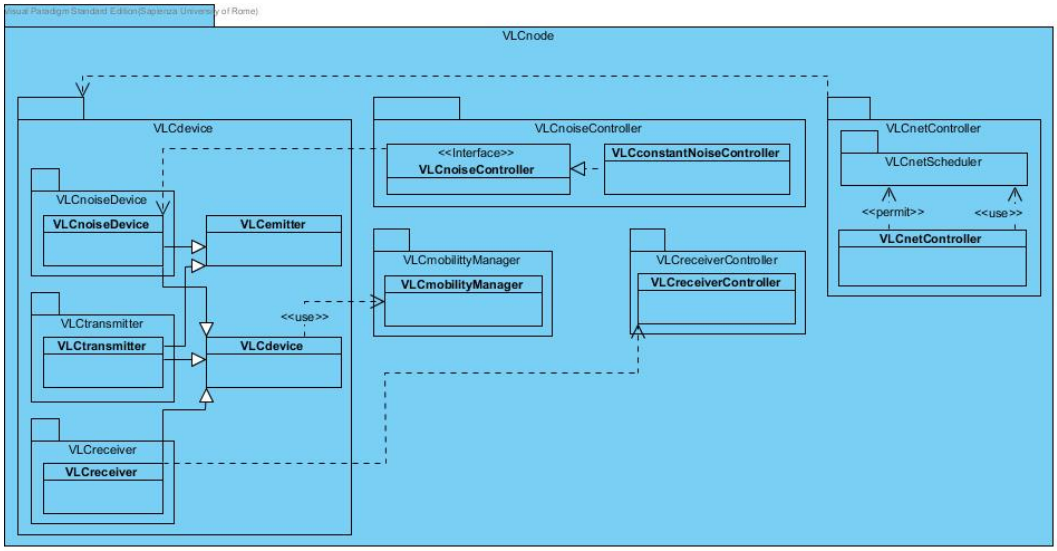

Figure 2.4. VLCnode Class Diagram

A VLCnode represents a standalone device in the network that can interact with both entities from the upper layers, namely **VLCapp**s, and with the physical layer which in our case is represented by the **VLCchannel** package. Each node is composed at least of a **VLCdevice** and a **VLCmobilityManager**. Such a node is however pretty useless since to actually interact with the network it needs a controller to act as the "brain" of the node and bridge between layers. Let us dive deeper into the hierarchy to see how to put together something useful to construct our networks. Each class is followed by its diagram that show its specification.

2.4.1.1 VLCdevice

A VLCdevice is a virtual class that every other device extends and hold information that are common to all of them, in particular an enum for the device type, the device semiAngle and pointers to its mobilityManager and channel.

VLCdevice	
Attributes	
VLCdeviceType	deviceType
double	semiAngle
VLCchannel*	channel
VLCmobilityManager*	mobilityManager

2.4.1.2 VLCreceiver

A VLCreceiver interfaces with a channel to receive packets sent via visible light. It holds the information needed to compute the gain of the transmission, i.e., the area

of the photodetector and the refractive index and has an address that represent the physical address (MAC) of the device. It also holds some information about the current transmission such as the address of the transmitter it is currently attached to and the last value of the noise variance for memoization purposes. It also holds a set of all the connections that have an end in this receiver.

VLCreceiver	
Attributes	
double	photoDetectorArea
double	refractiveIndex
double	noiseVariance
int	address
int	currentTransmitterAddress
set<VLCconnection*>	connectionEnds

2.4.1.3 VLCemitter

A VLCemitter is any device that can emit visible light. It holds its lambertian order and a map of the current transmission information such as the transmission power, the modulation type etc. Both VLCnoiseDevice and VLCtransmitter extend this class since they both emit light.

VLCemitter	
Attributes	
double	lambertianOrder
map<transmissionKeys, double>	currentTransmissionInfo

2.4.1.4 VLCtransmitter

A VLCtransmitter interfaces with a channel to send packets via visible light. It holds some information about its state, that is, if it's busy transmitting for example, the last data packet sent, the maximum transmission power and its physical address. It also holds a set of all the connections that start with this transmitter.

VLCtransmitter	
Attributes	
dataPacket	lastPacket
double	maxTransmissionPower
int	address
set<VLCconnection*>	connectionStarts

2.4.1.5 VLCnoiseDevice

A VLCnoiseDevice interfaces with a channel and, as the name suggests, acts as noise source for all the receivers it affects. It has no field except for those inherited from VLCdevice and VLCemitter.

2.4.1.6 VLCmobilityManager

A VLCmobilityManager interfaces with a channel to inform it of any change in the position of the device it manages. A mobility manager is attached to only one device, to which it holds a pointer. Node movements in Alaula are handled according to random waypoint (the system can then be extended to incorporate additional mobility models). When the mobility manager is initialized it reads from a configuration file the initial position of the node and an optional list of waypoints. After an optional initial delay it starts moving the node towards the next position with the speed specified in the waypoint. The position is updated at a fixed interval. When the final position is reached the node stops.

VLCmobilityManager

Attributes	
VLCdevice*	device
VLCchannel*	channel
VLCnodePosition	nodePosition
VLCnodePosition	targetPosition
vector<VLCwayPoint>	wayPoints
int	currentWayPoint
int	totalWayPoints
double	linearVelocity
double	xStep, yStep, zStep
double	alphaStep, betaStep
double	updateInterval
double	initialDelay

These elements are enough to build a module, in fact they do: respectively VLCtransmitterNode, VLCreceiverNode and VLCnoiseNode. However, the core of the logic is in another network element: the controller.

2.4.1.7 VLCreceiverController

A VLCreceiverController interfaces with a VLCreceiverNode, a VLCapp and a VLCnetController. It hold some information about its device, namely the address and the Id, and also whether the wireless channel is available to communicate with other controllers. It acts as a bridge between the components it is connected to and a manager for the state of the receiver.

VLCreceiverController	
Attributes	
int	receiverAddress
int	receiverId
VLCreceiver*	receiver
simtime_t	lastBeaconTime
double	beaconInterval
bool	WirelessAvailable

2.4.1.8 VLCnoiseController

A VLCnoiseController is a virtual class that interfaces with a VLCnoiseNode to control its noise level every updateInterval milliseconds. A class that extends this one should come with a function $f(t)$ for the noise power of the node.

VLCnoiseController	
Attributes	
double	updateInterval

2.4.1.9 VLCconstantNoiseController

VLCconstantNoiseController is the only implementation out of the box of the VLCnoiseController. Not much is to be said about it since all it does is set the noise power to a constant value and nothing else.

VLCconstantNoiseController	
Attributes	
double	noisePower

2.4.1.10 VLCnetController

A VLCnetController is a very special type of controller that interfaces with VLCtransmitterNodes, VLCreceiverControllers and VLCapps. It is somewhat different from the other controllers since it acts as a coordinator for the entire network. It has its own address separate from the devices it controls (that can be many) and is the module that implements the MAC protocol. Every VLCapp that is in transmission mode, that is, it generates and send out packets, interfaces with the VLCnetController. This makes use of a VLCnetScheduler that implements a scheduling algorithm to handle the packet flows. We'll talk about the MAC protocol and the scheduler later.

VLCreceiverController	
Attributes	
int	address
bool	WirelessAvailable, useWirelessDl
double	beaconInterval
VLCnetScheduler*	netScheduler
map<int, set<link>>	linkTable
map<int, transmitterInfo>	transmitterTable

2.4.1.11 VLCnetScheduler

A VLCnetScheduler is a helper class for the VLCnetController of which it holds a pointer to that is passed when the class is instantiated. When the net controller is informed of an event such as a new packet arriving or a transmitter having ended transmitting it asks the scheduler to process the next packet waiting in line. The scheduler runs the scheduling algorithm and comes up with the next set of packets to be transmitted on a respective set of interfaces.

VLCnetScheduler	
Attributes	
VLCnetController*	netController
vector<int>	destinations
int	currentDestination
map<int, queue<dataPacket*>>	packetQueues
map<int, <set<int>>	servedQueues
map<int, int>	transmitterServes

2.4.1.12 VLCapp

A VLCapp is a packet source or sink. If it's a source it generates a packet every transmissionInterval milliseconds for one of its destination addresses and sends it to the controller. If it's a sink it just sits there waiting for packets to arrive, periodically updating its average throughput computed over the interval from the first packet received to the current simulation time.

VLCapp	
Attributes	
int	address
int	packetSize
double	transmissionInterval
vector<double>	destinationAddresses
int64	totalReceivedData
double	startThroughputTime

2.4.1.13 VLCchannel

The next package we'll look at is the VLCchannel package, represented in Fig. 2.5. Though it consist of only 2 classes they are the heart and soul of the simulator.

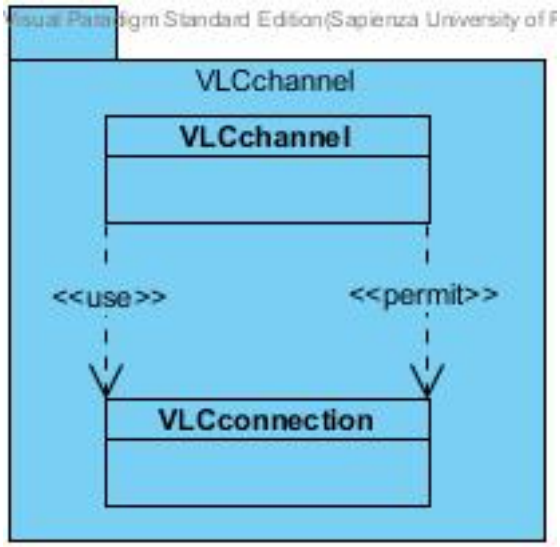

Figure 2.5. VLCchannel Class Diagram

2.4.1.14 VLCchannel

A VLCchannel handles the physical modelling of the VLC channel. Currently it is implemented as a singleton for simplicity but it can be easily modified to be able to have many channels run in parallel. Each VLC*Node communicates with this class via an ideal channel that is created upon the initialization of the simulation. The VLCchannel is informed of any change that happens in the simulation such as the movement of a node or the beginning and end of a transmission. Whenever such an event happens it recomputes the state of the network and responds to the event by, for example ending a transmission and sending the packet to the nodes that must receive them, the channel also updates the state of the network every updateInterval milliseconds. The VLCchannel class holds pointer to all the VLCdevices in the network as well as the connection between them. It also holds

a map of all the current "views" of the devices, a data structure consisting of an ordered pair of devices with the angles they see each other at and their distance.

VLCchannel	
Attributes	
cSimulation*	sim
double	updateInterval
set<VLCdevice*>	VLCdevices
map<VLCdevice*, VLCpacket*>	transmitterMessages
map<VLCdevice*, int>	VLCdeviceGates
set<VLCconnection>	VLCconnections
map<VLCdevice*, set<VLCdevViewInfo*»>	VLCcurrentViews

2.4.1.15 VLCconnection

A VLCconnnection represent a LoS link between a transmitter and a receiver. It holds information about the state of the connection over time and it is used at the end of a transmission to determine whether the packet should be transmitted. It implements the equations necessary to compute the SINR using the data provided by the channel.

VLCconnection	
Attributes	
static long unsigned int	connectionCounter
long unsigned int	connectionId
VLCtransmitter*	transmitter
VLCreceiver*	receiver
VLCchannel*	channel
double	SINRthreshold
double	totalNoisePower
double	connGainConstantPart
double	connectionGain
double	SINRdB
VLCdevViewInfo	lastView
vector<VLCtimeSINR>	SINRtrend
map<VLCdevice*, sourceInfo>	noiseSources
map<transmissionKeys, double>	transmissionInfo

2.4.2 Common Package

In the common package are files and classes shared and used by every other class these include:

2.4.2.1 VLCtransmissionModels

These files implement the functions that decide whether a packet gets transmitted or dropped. The function expects a VLCconnection from which it gets the SINRtrend and the modulation type. It then calls the appropriate function to calculate the Packet Error Rate, "flips a coin" and if the result is smaller than the PER it drops the packet.

2.4.2.2 VLCcommons

These are utility function such as those used to perform basic algebraic operations, logging and other random functions that did not belong anywhere else in particular.

2.4.2.3 VLCpacket

Packets in Alaula are organized in a tree hierarchy, mainly because multiple inheritance is not permitted for messages in OMNeT++. The root of the tree is the VLCpacket class. This class only holds the type of packet. There are 3 packet types in Alaula: control packet, data packets, and MAC packets.

VLCpacket	
Attributes	
VLCmsgType	messageType

The VLCctrlMsg is used to signal a module of a specific event or control a module. It always carry the id and address of the node that sent it as well as the control code.

VLCctrlMsg	
Attributes	
int	nodeId
int	nodeAddress
VLCctrlCode	ctrlCode

The VLCnoiseControlMsg is used to control a noise device. It extends VLCctrlMsg and only adds the value of the power at which the noise device must be set.

VLCnoiseControlMsg	
Attributes	
double	noisePower

The VLCMACMsg is used to exchange Media Access Control information. It holds the code of the MAC message as well as 4 other fields: the transmitter and receiver ids and addresses. Not all these fields are always set and functions know which of these will hold a value by looking at the MAC code.

VLCMACMsg

Attributes	
VLCMACCode	MACCode
int	transmitterNodeId
int	transmitterNodeAddress
int	receiverNodeId
int	receiverNodeAddress

The VLCbeaconMsg is-a VLCMACMsg with MACcode set to BEACON_MSG. As the name suggests it is used by transmitter to inform receivers of their presence. The net controller establishes the rate of transmission of this messages. If a receiver correctly receives a beacon it usually asks the net controller to "subscribe" him to that particular transmitter, that is, use that transmitter to send it data.

VLCbeaconMsg

Attributes	
VLCmodulationType	modulationType
double	transmissionPower
int	modulationOrder
double	dutyCycle

The dataPacket is the last of the packets types. It lacks the VLC prefix as it is not bound to be transmitted via VLC . It still always carries the information about how we would like to be transmitted (power, modulationType etc.) but the controllers are free to ignore them. Despite their name they don't actually carry any meaningful data. They are just defined by their length and we can think of them as random strings.

dataPacket	
Attributes	
VLCmodulationType	modulationType
double	transmissionPower
int	modulationOrder
double	dutyCycle
double	transmissionStartTime
int	sourceAddress
int	destinationAddress

This concludes the overview of the classes in Alaula.

2.4.3 Network control

We will now go through the MAC protocol implemented as well as the network logic embedded in the controllers. We designed the protocols to be as bare as possible. We also made the following assumptions:

- The controllers have access to a wireless channel to communicate amongst them.
- The receivers are able to sense when the SINR has reached a low threshold.

Each receiver is associated at any time with 0 to n different transmitters. An associations exists when, according to the last known state of the network, a receiver can correctly receive packets sent by the associated transmitter. The association is formed with a subscribe operation which sends a message via the wireless channel to the netController which in turn adds an entry in the MAC address table. The unsubscribe operation happens similarly but of course has the effect of removing the entry for that (transmitter, receiver) pair from the table.
These associations are used by the netControllers to route packets to their destination and control the access to the shared medium.

If more than one receiver is associated with a single transmitter the access to the channel will be regulated by a *kind-of-fair* algorithm. Each receiver will be served a packet in a round-robin fashion. This approximates an *actually* fair algorithm when the size of each packet is roughly similar but obviously favors the receivers that have been sent bigger packets.

On the controller side the association are kept in a modified version of a MAC address table, with each record being in the form:

linkType	transmitterAddress	receiverAddress	gate

The linkType field holds the type of channel that's connected to that gate, currently link types are `VLC_LINK` and `WIFI_LINK`.

The transmitterAddress holds the MAC address of the transmitter.
The receiverAddress holds the MAC address of the receiver.
The gate holds a pointer to a `cGate` object. This is equivalent to the address of the interface on which to send the packets.
The netController periodically instructs the transmitters to send a beacon packet. The packets contain the address of the transmitter. The following pseudocode illustrates the behavior of the controller:

```
fun processPacket( pkt ):
  links <- getAllLinks( pkt.receiverAddress )
  link <- select a link from links prioritizing VLC_LINK

  if ( link.linkType == VLC_LINK AND
    the transmitter should be sending a beacon ):
    sendBeacon( link.transmitter )
    return
  else:
    # some other code

  sendPacket( link, pkt )

this->onEvent( "beaconInterval", fun{
  foreach( transmitter in transmitters ):
    sendBeacon( transmitter )
})
```

The beacons are sent periodically when a timer fires but the controller will not interrupt a transmission to send a beacon; thus when processing a packet it will check whether the VLCtransmitter hasn't sent a beacon in a while and if that's the case it will postpone the transmission of the packet to send a beacon from that transmitter.

On the receiver side the following will happen:

```
this->onEvent( "beaconReceived", fun{
  if( beacon.transmitterAddress != currentTransmitterAddress):
    unsubscribeFromCurrentTransmitter()
    currentTransmitterAddress = beacon.transmitterAddress
    subscribeTo(beacon.transmitterAddress)
    lastBeaconTime = now()
})

this->onEvents( ["beaconTimeout", "SINRtooLow"], fun{
  unsubscribeFromCurrentTransmitter()
})
```

Since the receivers are mobile we also have to handle the handover issue. This is solved by using the beacons as a keep alive for the associations. When a beacon is received, the receiver can find itself in one of two states: either it was not associated to any transmitter, in which case it will send a subscribe request to the netController, or it was already associated to some transmitter, it will then ask to be subscribed to the latter and the netController will perform the handover.

The receiver expects to receive a beacon from the transmitter loosely every beaconInterval seconds. If no beacon is received it means that the receiver has moved out of the range of the transmitter and will ask the controller to unsubscribe. The receiver will also unsubscribe from the transmitter if it detects that the SINR is too low.

This concludes the description of the control algorithms implemented.

2.4.4 A complete example

In order to understand how the simulator works one could dive deep into the code and look at each function and method call and class instantiation etc. But that's very time consuming and not particularly fun. Therefore we're just going to follow the execution path of a complete scenario that summarizes most of the ones we can build so strap in!
We start by defining the geometry of our scenario as illustrated in Fig. 2.6.

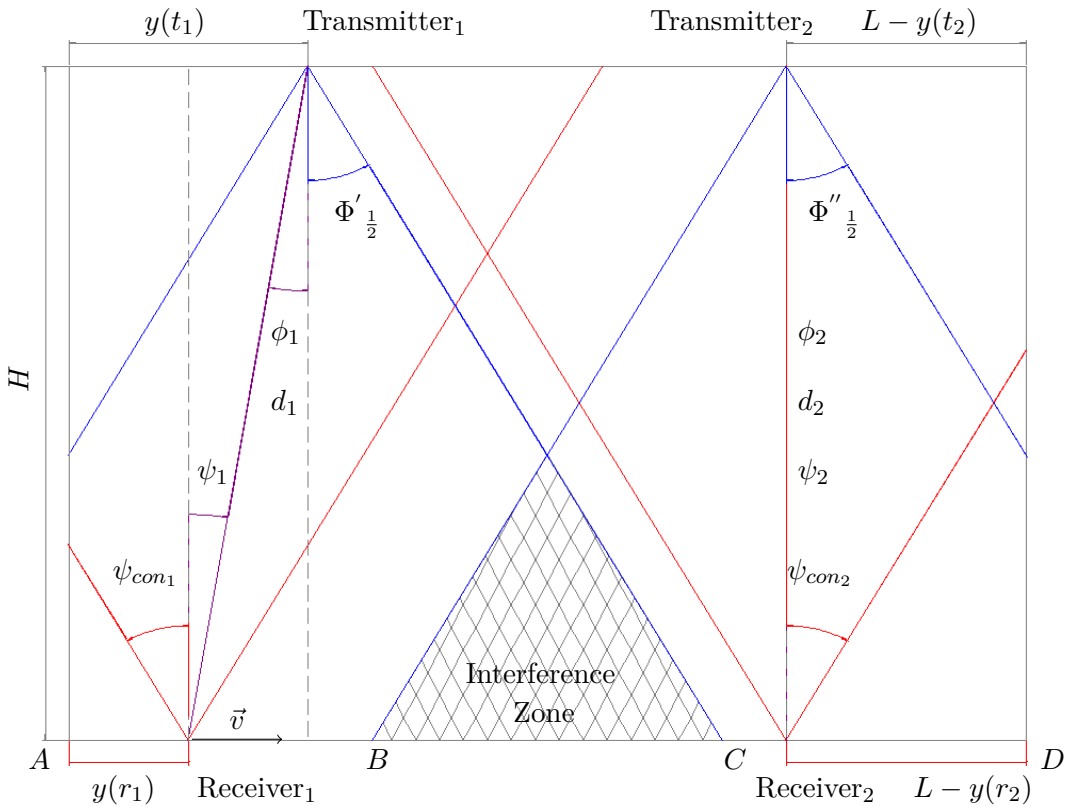

Figure 2.6. A complete example

We represent our space as bounded as may be the case of a room but the simulator has no knowledge of it since currently it doesn't model the concept of opaque bodies. This is quite a limitation and a successive version will probably implement a basic ray tracing algorithm to consider at least the presence of walls. We place our transmitters on the ceiling facing directly down with the same spacing as that which may be used for light fixtures since a real world application is exactly to have light fixtures act as Access Points. The receivers are facing directly up and are placed at floor level. In reality this may coincide with desk or arm height depending on when the devices' lowest will be reached.

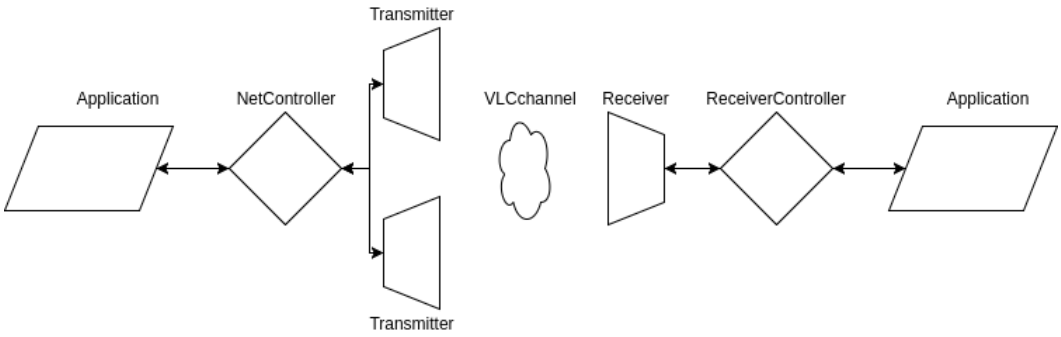

Figure 2.7. The network topology

When the network is initialized a few things happen: the devices establish their connections to the VLCchannel, the transmitters register themselves to the net controller they're attached to, the controllers poll their attached receiver to ask for their addresses. The mobility managers then initialize. They will read configuration files and parse the values used to move the nodes around. They will then inform the channel of the position of the nodes, which in turn will perform the first update of the state of the network.
The simulation is now ready to start. Depending on whether or not the external wireless channel is available we'll have a "smart" or "dumb" scenario. We will go through the former first, since it's the most interesting.

2.4.4.1 Smart Scenario

The "smartness" of this scenario lies in it using the form of access control we went through. As soon as the simulation starts the apps will start producing packets and sending them to the netController. This will just pass them to the scheduler which will sort them into queues depending on their address. When `VLCnetScheduler::processNextPacket()` is called the scheduler follows the following algorithm:

```
fun processNextPackets():
 foreach( transmitter in transmitters AND transmitter is not busy):
  transmitterQueues<-get the queues served by this transmitter
  while( currentIndex<-get the index of the next
                                queue to be served ):
   if ( transmitterQueues[currentIndex] not empty ):
    if ( processPkt(transmitterQueues[currentIndex].front() ) ):
     transmitterQueues[currentIndex].pop()
     break
```

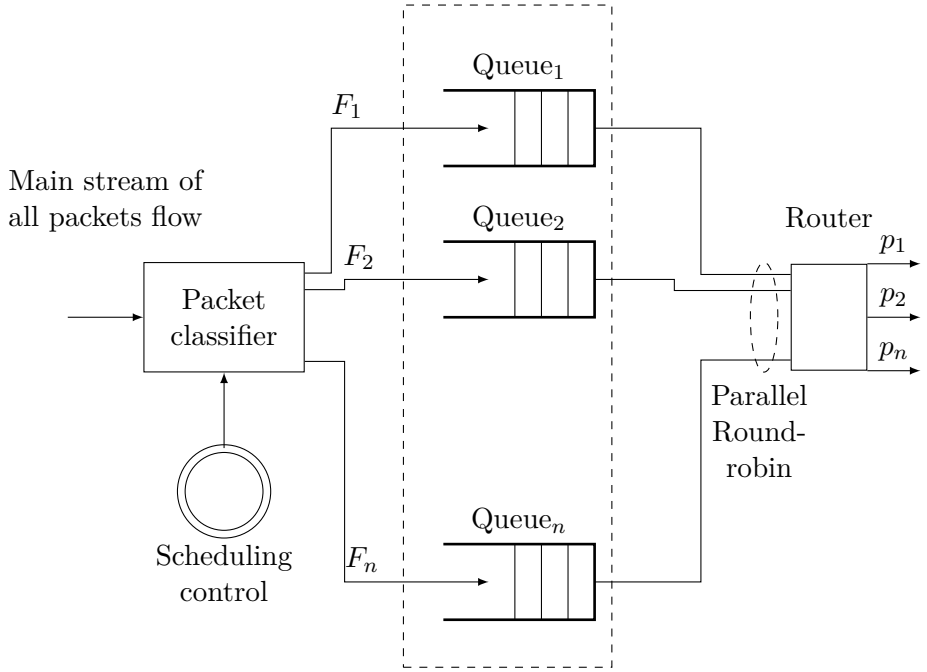

Figure 2.8. A representation of the scheduler

A transmitter serves at any moment a number of queues determined by the associations table. The scheduler keeps an index for each transmitter that indicates what queue it should serve next. When an event such as a packet arriving or a transmitter finishing transmitting is fired, it triggers the processing of the next front of packets. Note that in this case the term transmitter is used to indicate any type of network interface and is not limited to VLC devices.

2.4.4.2 Dumb Scenario

If the additional wireless channel is not available the net controller can't pull any clever tricks to try to maximize the overall efficiency of the network. When packets arrive they get pushed into their respective queues from which they then get popped one by one (cycling through the addresses) and effectively broadcast on all transmitters at the same time. This may seem very "unintelligent" (it is) but it's really the only thing the controller can do. Since the receivers have no way of communicating with the transmitters coordination is impossible. This setup could

still find a possible application if, for example, routing tables are set up manually. If I know Receiver$_a$ is always in view of Transmitter$_a$ I don't really have the need to handle mobility but it's still a very limiting factor and therefore from now on we will consider to be in the "smart" scenario.

2.4.4.3 An unexpacket journey

Suppose we have completed our initialization phase, the routing tables are filled and we just plucked a fresh packet from one of our queues, since it's for Receiver$_1$ it will be transmitted from Transmitter$_1$. We will now follow it until it reaches its destination app. The first thing that happens is the transmitter sending a ctrlMsg to the channel with control code `TRANSMISSION_BEGIN` and a self message to itself. The channel will immediately receive the former (again this channels are ideal and only used by the simulator for coordination purposes) which will call `VLCchannell::startTransmission(VLCdevice* transmitter)`.
This method will in turn call the one to create a connection from the transmitter to each receiver device in its view, even those which didn't want anything to do with it! This is of course justified by the fact that if a new transmitter with much more transmitting power, say a spotlight, enters the FoV of a receiver that was communicating with say, a bedside lamp, the spotlight will prevail and not only completely scramble the lamp transmission but also be probably able to correctly transmit to the receiver.
When a connections is created the channel is wary of this fact and thus also checks for all other active transmitters and noise sources and adds them indistinctly as noise sources for that connection. When all the connections have been created the channel issues a first update, this in turn calls the `VLCconnect::updateConnection()` for all active connections which will calculate the SINR at that instant and add it, paired with the current simulation time, to a vector.
Now, recall that our node was moving: when a step is made the mobility manager will signal the channel that the device it is attached to has changed and thus a state update is due. Since it is a receiver that moved we will see what happens in this case but the transmitter's one is almost identical. First thing first the channel will recompute what's in view of the receiver now. If the transmitter it was currently receiving from has gone out of its view it will abort the connection and be done with it. Notice that this does not mean that the transmitter will stop transmitting: it is done solely to avoid recomputing the state of that connection when a state update is issued. If instead the transmitter is still in view, the channel will take the receiver's new view and compare it with what was last in its view. It will then produce 3 sets: the one of the devices that have come into view, the one of the devices that stayed in view and the one of the devices that left the view.
Being a receiver we care about the new devices if they are transmitters or noise sources since they affect our current connection, we will simply add them as noise sources.
The devices that stayed in our view just need to be updated, that is, we need to recompute how they affect our connection.
Lastly the devices that sadly left our FoV need to be removed. Though we will no longer consider them when computing it their contribution to our SINR trend will

last forever.
If a transmitter or noise source moved it would have affected all the receivers (and their connections) in its FoV but again the logic is pretty similar.
Like all good things our transmission too will eventually come to an end. That's when that self message comes into play. It arrived at transmissionStartTime + $\frac{\text{packetSize}}{\text{channelDatarate}}$ ms and is a ctrlMsg with code `TRANSMISSION_END`. The transmitter will thus immediately stop transmitting and forward this message to the VLCchannel. The channel will respond by calling
`VLCchannell::endTransmission(VLCdevice* transmitter)`
this method loops through all the connection in which the transmitter was involved and checks whether they were successful or not by calling
`bool VLCconnection::getOutcome()`. This will just return the result of the `bool packetTransmitted(VLCconnection * conn)` function that is defined in `VLCtransmissionModels`. This function uses the connection specifics and SINRtrend data to return the outcome of this transmission. Climbing back up the call stack the channel takes the outcome and if it was negative it just drops the packet otherwise it sends it to the receiver(s) via an ideal channel. At the receiver side, the packet's destination address will be read, and if it was matches the receiver's one the packet willl be forwarded to the receiverController, which in turn will send it to the connected App which finally will use it to recompute the throughput and then just throw it away, blissfully unaware of all it had to go through to get there.
Of course things could have gone worse. When our receiver reaches point B for example it will find an interference zone. We assume that our receiver is able to measure the SINR and detect whether it goes below a certain threshold. If it does, it will then ask the controller to unsubscribe it from the transmitter it was attached to. The receiverController will generate a MACMsg and send it via the wireless channel to the netController which in turn will remove the association between that transmitter and receiver, severing the route. If other routes to that receiver exist, the scheduler will then use them instead, otherwise the receiver is cut from all communications. Not to worry however since transmission will start again when our recevier reaches point C whence it will ask to be subscribed to Transmitter_2. Since now 2 receivers share a transmitter the overall throughput of the network will of course go down but that's unavoidable.
This concludes our bird's eye view of the operation mode of the network.

Chapter 3

Performance evaluation

Now that we have an overall understanding of how the simulator works and the model behind it we are ready to perform some tests. We have chosen a handful of scenario that we deem to be representative of possible real life situations to perform a simple analysis of the network performance.
All the variables are computed according to the equations presented in the model. The table of the values of the variables used is presented below:

Table of variables and constants

Transmitter power, P	$1W$
Receiver Area, A	$5 \times 10^{-4} m^2$
Filter gain, T_s	1
Transmitter semi angle, $\Phi_{\frac{1}{2}}$	$30°$
Photodiode reponsivity, γ	$0.54 \frac{A}{W}$
Photodiode FoV semi angle, Ψ_{con}	$30°$
Boltzmann's constant, k	$1.3806488 \times 10^{-23} \frac{J}{K}$
Absolute temperature, T_k	$300K$
Fixed capacitance of photodiode, C_{pd}	$1.12 \frac{pF}{cm^2}$
Charge of the electron, q	$11.6 \times 10^{-19} C$
Open loop voltage gain, G_{ol}	10
FET channel noise factor, Γ	1.5
FET transconductance, g_m	30mS
Background current, I_B	$5100 \times 10^{-6} A$
Equivalent noise bandwidth, B	$100 MHz$
Noise bandwidth factor, I_2	0.562
Circuit constant, I_3	0.0868
Room height, H,	$3.0m$
Room length, L	$4.0m$
Room width, W	$3.0m$

3.1 Scenarios

We divide our tests in two parts: one set aimed purely to testing the implementation of the model in the simulator and the other to analyzing the performance of the network in a plausible real life scenario.

3.1.1 Model tests

In this suite of tests we will explore the relations between simple parameters such as device distance, angles of incidence and radiance and noise and our "output" variables, that is, SINR, BER and throughput.

3.1.1.1 Distance Test

The first test we are going to perform is a simple distance test. The test geometry and network topology are illustrated in Fig. 3.1 and Fig.3.2. This test just aims to measure the relation between distance, SNR, Bit Error Rate and the resulting throughput. The receiver moves at constant speed from the transmitter and they both face directly each other (the angle between their direction vector is 0°).

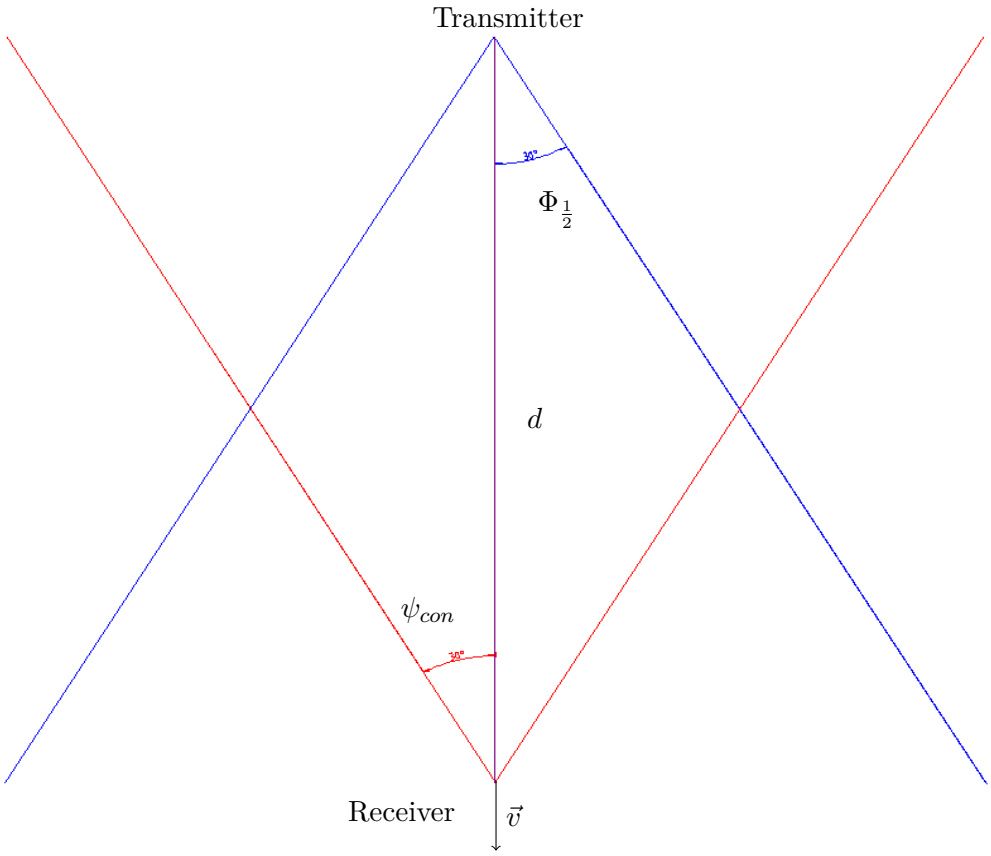

Figure 3.1. Distance test

Test parameters	
Transmitter position, $P(T_1)$	$(0, 0, 0)m$
Receiver initial position, $P_i(R_1)$	$(0, 0, 0)m$
Receiver final position, $P_f(R_1)$	$(0, 0, 50)m$
Receiver speed, $\vec{v}$	$(0, 0, 0.5)\frac{m}{s}$

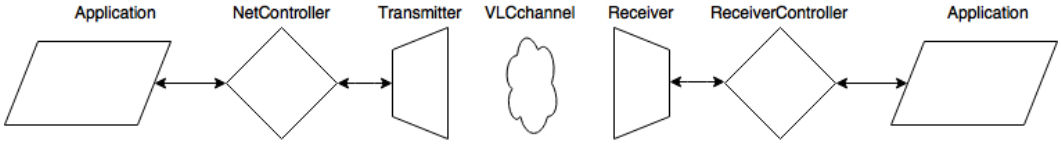

Figure 3.2. Single transmitter and receiver

The results of the test are presented in Fig.3.3 and Fig.3.4. As we expected from the model the SINR goes down as the inverse square of the distance. The average throughput stays stable until the receiver reaches the distance $d_{threshold} \approx 6.76m$. At that point it stops receiving packets completely since the SINR has gone below the threshold of $15dB$ and the average throughput starts declining.

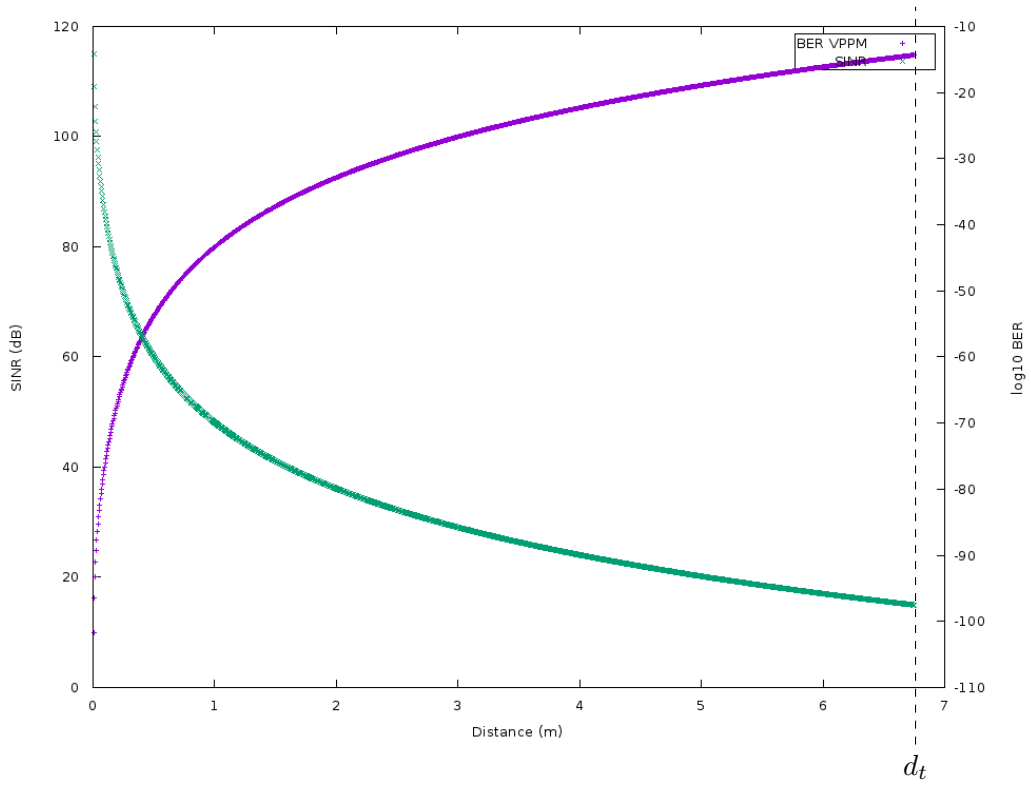

Figure 3.3. Distance test SINR and BER

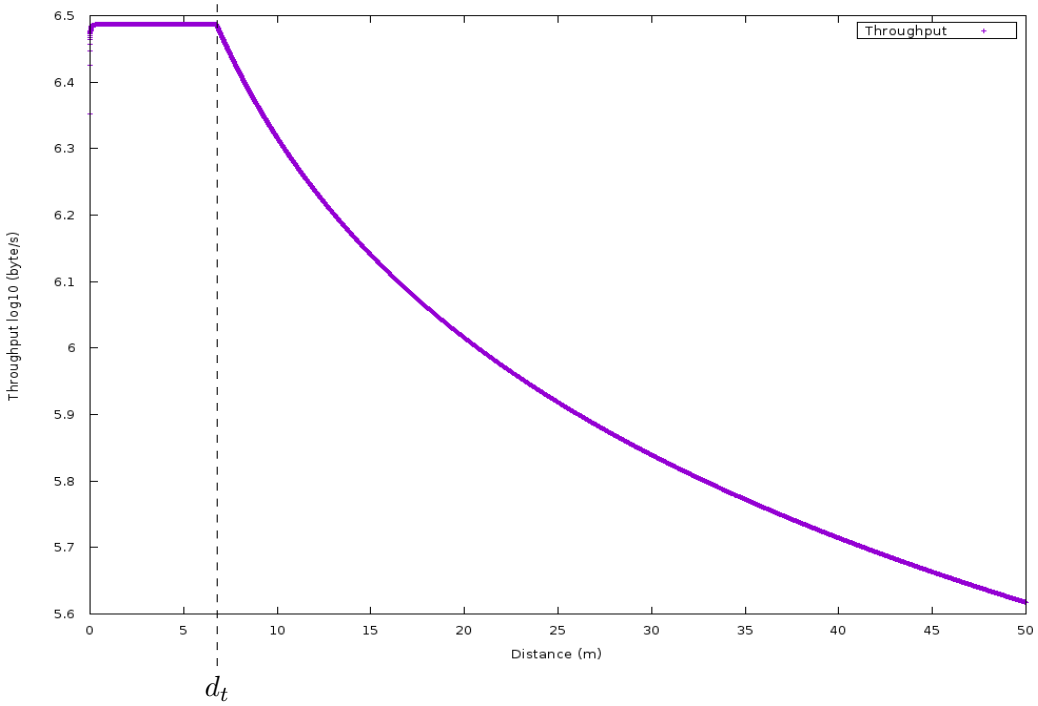

Figure 3.4. Distance test throughput

3.1.1.2 Sliding Test

The second test is a sliding test, illustrated in Fig. 3.5 the topology of the network remains the same as in Fig.3.2. The test mainly aims to measure the relation between angle of incidence and radiance, SNR, Bit Error Rate and the resulting throughput. Of course the distance varies too but we can account for the effects of that. The receiver moves at constant speed from point A, to point E, out of the FoV of the transmitter, with constant speed.

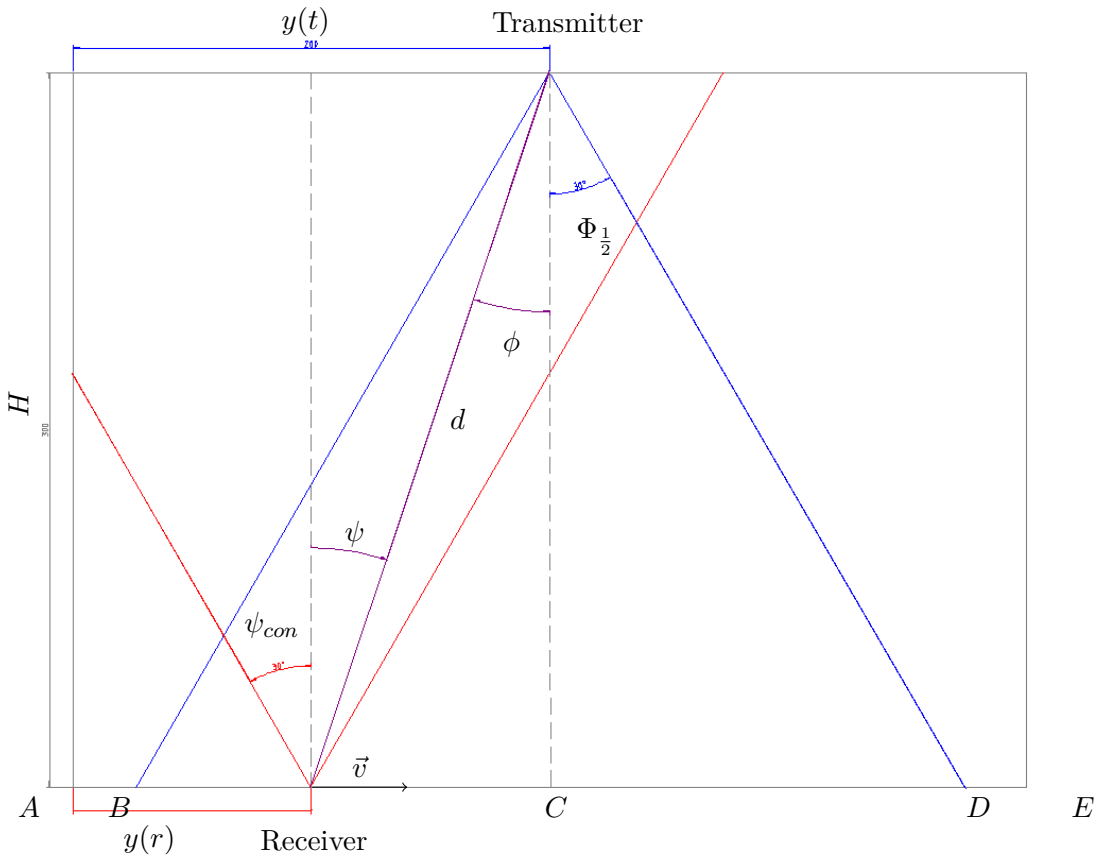

Figure 3.5. Sliding test

Test parameters	
Transmitter position, $P(T_1)$	$(1.5, 2, 3)m$
Receiver initial position, $P_i(R_1)$	$(1.5, 0, 0)m$
Receiver final position, $P_f(R_1)$	$(1.5, 4, 0)m$
Receiver speed, $\vec{v}$	$(0, 0.5, 0)\frac{m}{s}$

The results of the test are presented in Fig.3.7 and Fig.3.8. The receiver enters the FoV of the transmitter at $y(r) \approx 0.27m \approx 2 - 3tan(30°)m$ and exits at $y(r) \approx 3.73m \approx 2 + 3tan(30°)$. We know that the function for the SINR is:

$$f(d, \psi, \phi) = Kd^{-2}cos(\psi)cos(\phi)$$

For some constant K. Our distance is:

$$d = dist(\{|2 - y(r)|, 0\}, \{2, 3\})m$$

with $dist(P_1, P_2)$ the Euclidean distance function. Our angles are the same throughout the test since the direction vectors of the transmitter and receiver are always

opposite to each other. It follows that:

$$\phi = \psi = arctan(\frac{d}{3m})$$

thus our SINR goes like:

$$f(d) = Kd^{-2}cos^2(arctan(\frac{d}{3m}))$$

pictured below.

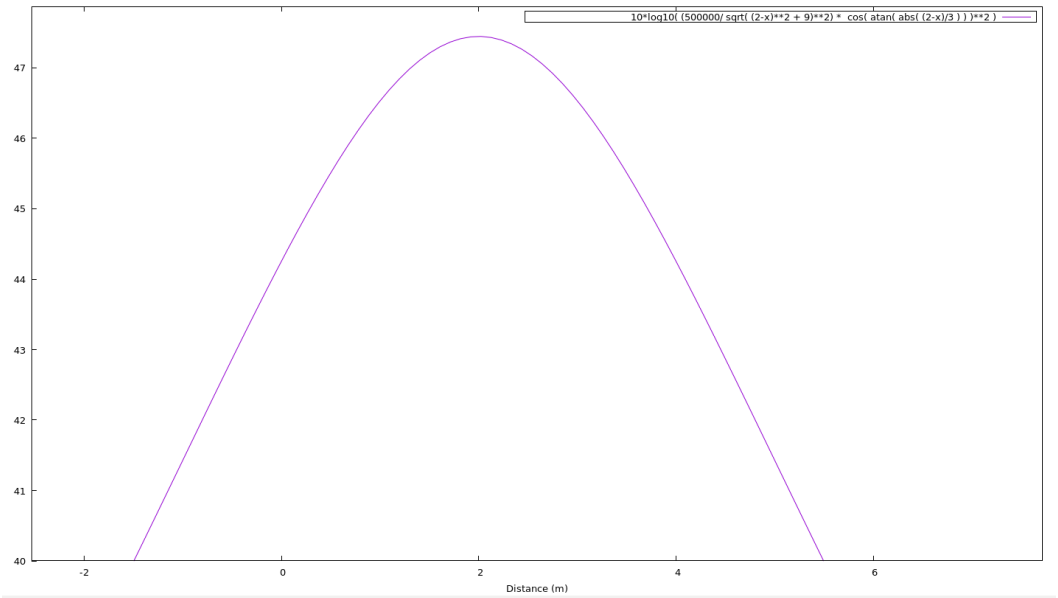

Figure 3.6. The trend of the SINR

This curve is comparable to the one we obtained in our tests given the appropriate choice of K.

We can observe that the throughput rapidly increases as soon as the receiver enters the FoV and only starts declining when it leaves it suggesting that transmission is stable for all the duration of the transit.

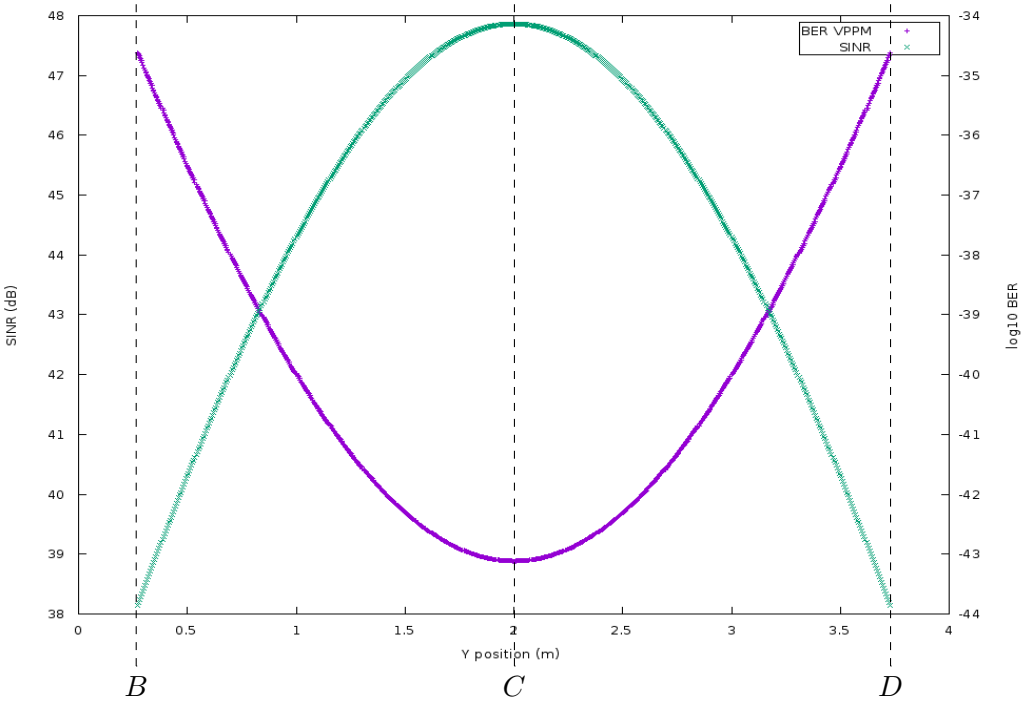

Figure 3.7. Sliding test SINR and BER

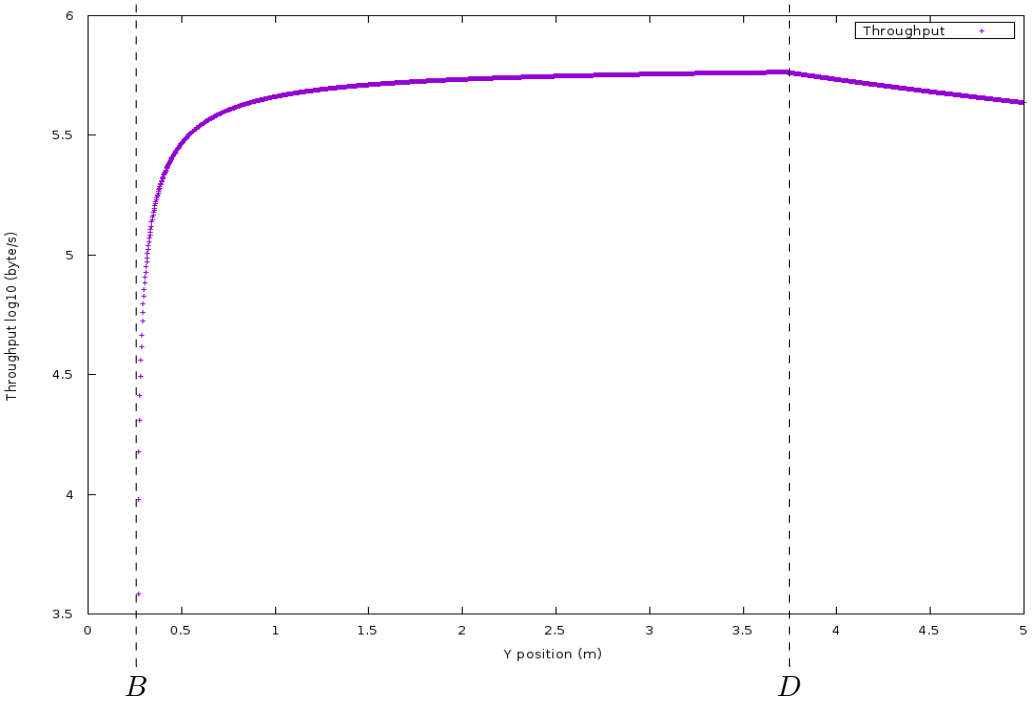

Figure 3.8. Sliding test throughput

3.1.1.3 Interference Test

This it the first in which more than 1 transmitter is present and we can start "appreciating" the effects of interference. The test setup is illustrated in Fig. 3.9 and Fig.3.10. The test aims to measure the relation between time, SNR, Bit Error Rate and the resulting throughput. The receiver moves at constant speed from point A, out of the FoV of the transmitter to point $E = (1.5, 5, 0)m$ (out of the room) with constant speed.

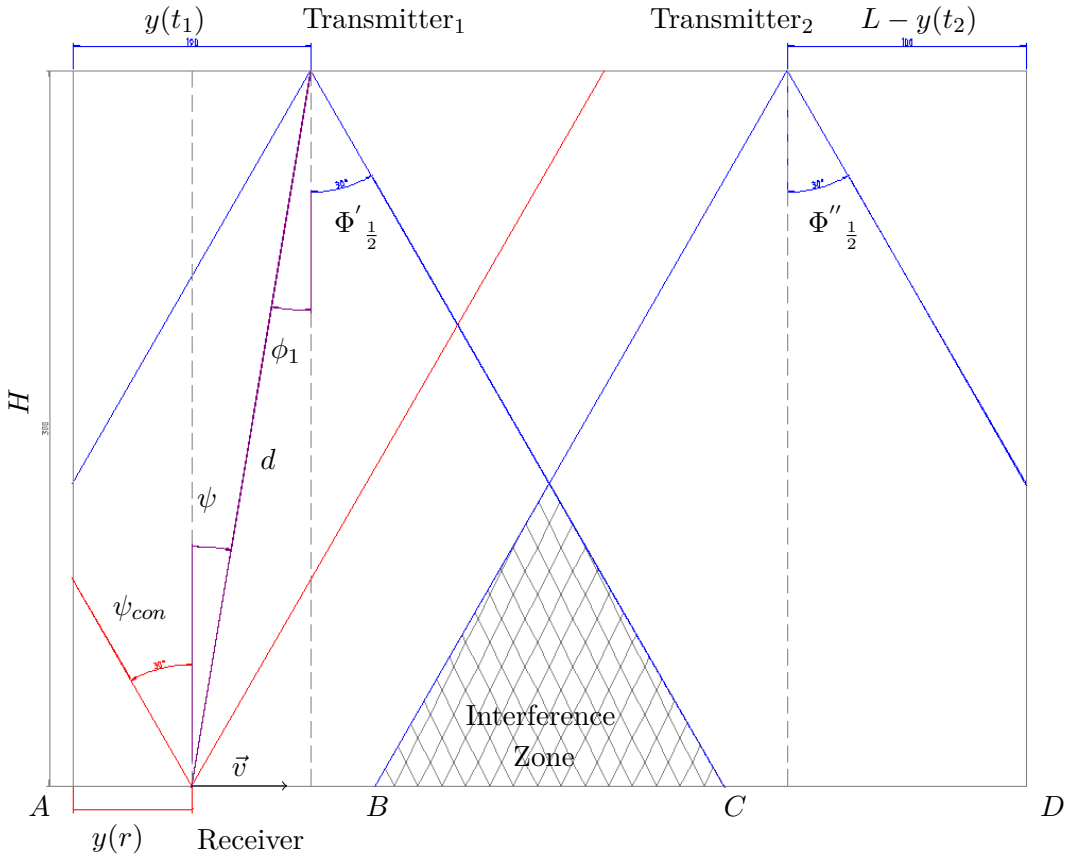

Figure 3.9. Interference test

Test parameters	
Transmitter 1 position, $P(T_1)$	$(1.5, 1, 3)m$
Transmitter 2 position, $P(T_2)$	$(1.5, 3, 3)m$
Receiver initial position, $P_i(R_1)$	$(1.5, 0, 0)m$
Receiver final position, $P_f(R_1)$	$(1.5, 5, 0)m$
Receiver speed, $\vec{v}$	$(0, 0.5, 0)\frac{m}{s}$

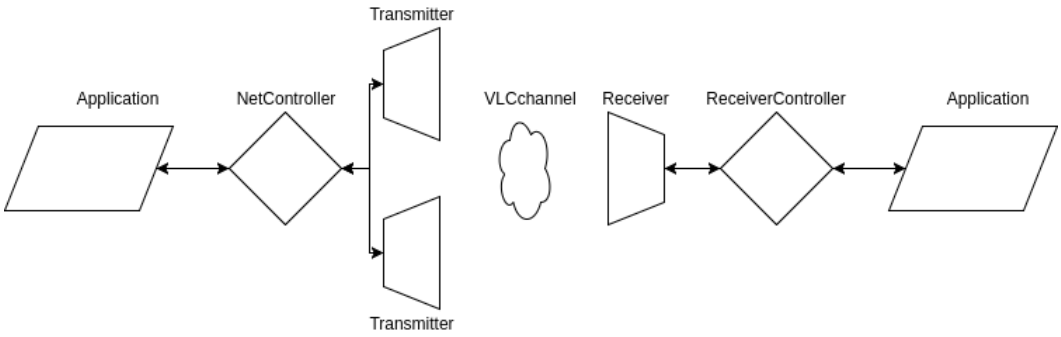

Figure 3.10. Single receiver two transmitters

The results of the test are presented in Fig.3.11 and Fig.3.12. The receiver starts in the FoV of the first transmitter at $y(r) = 0m$ it reaches the interference zone at $y(r) \approx 1.26m \approx 3 - 3tan(30°)$ and the throughput starts rapidly declining since no packets ever reach their destination, it then exits the interference zone at $y(r) \approx 2.73m \approx 2 + 3tan(30°)$ the throughput increases lags behind ever so slightly as the receiver waits for a new beacon from the second transmitter and send the subcribe message to the controller but then quickly rises as it's now being served by T_2. The receiver then crosses the boundaries of our imaginary room at $y(r) = 4m$ and the FoV of the second transmitter at $y(r) \approx 4.73m \approx 3 + 3tan(30°)$.

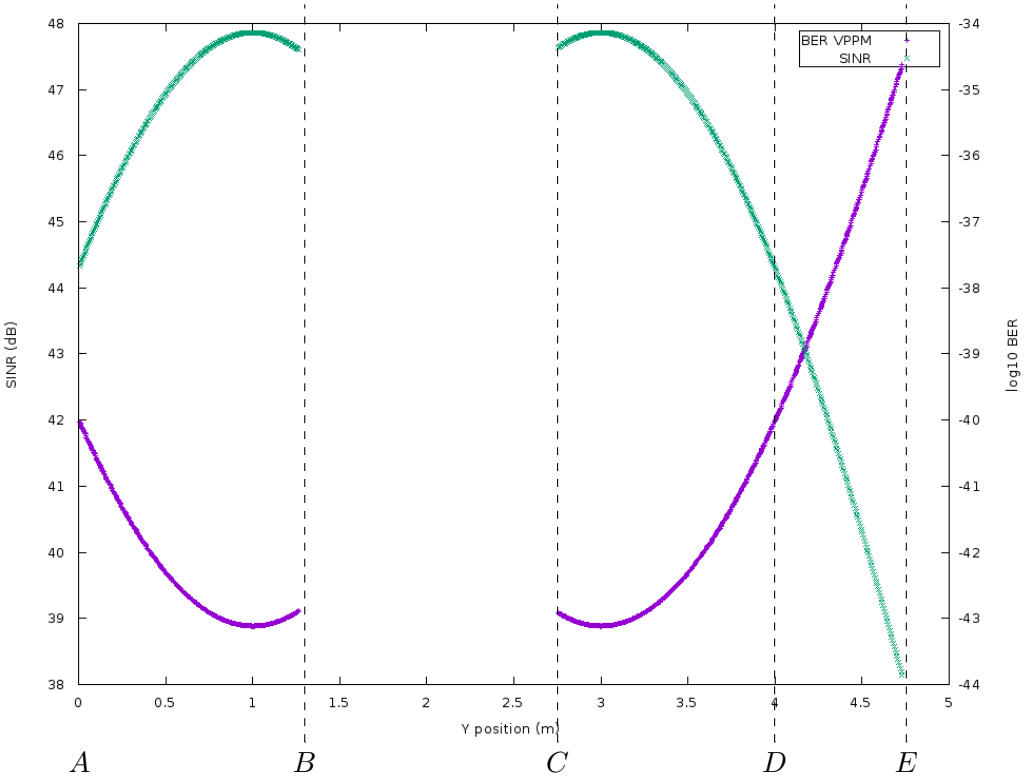

Figure 3.11. Interference test SINR and BER

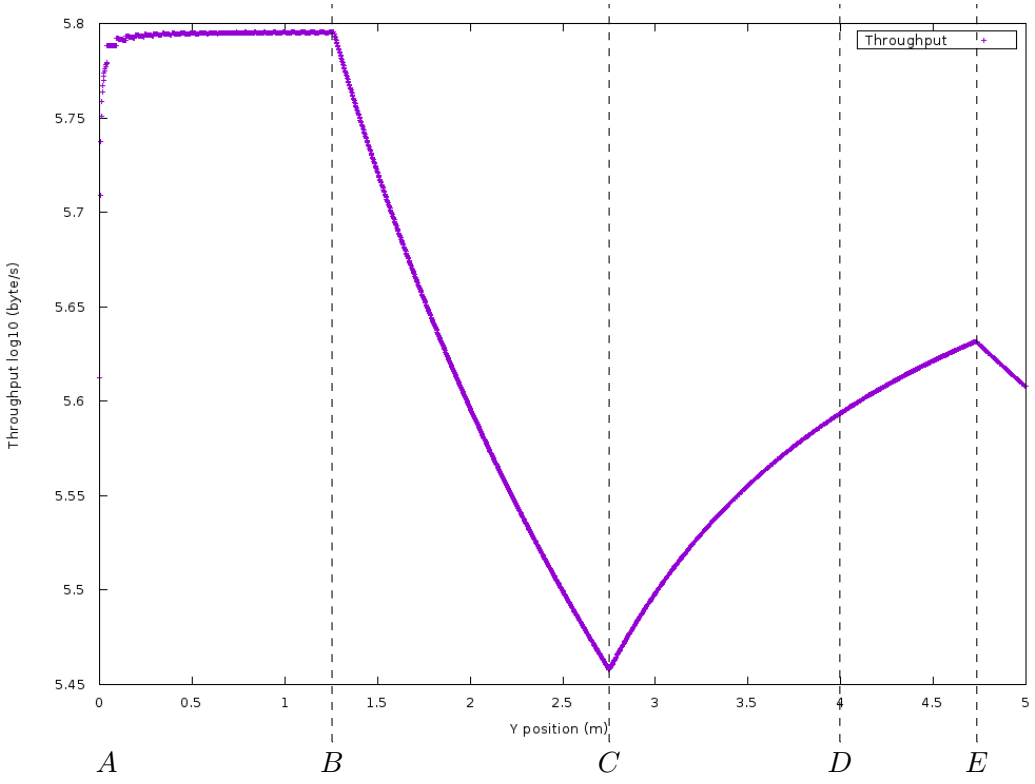

Figure 3.12. Interference test throughput

3.1.2 Performance tests

We will now define some more realistic scenarios that we'll use to test the efficiency of our hybrid solution by comparing it against a pure VLC case. Since we are dealing with low power devices for the receivers we chose to use a wireless channel rated at 250 kbit/s such as that defined in IEEE 802.15.4 . This suite of protocols is already widely implemented by a number of off-the-shelf devices used for example for sensor networks and other IoT applications.

3.1.2.1 Double No Wireless Test

In this test we have 2 transmitters and 2 receivers and we can appreciate all the combined effects of a full network. The test is illustrated in Fig. 3.13 and Fig.3.14. We aim to measure the relation between time, SINR, Bit Error Rate and the resulting throughput both for the individual connections and the cumulative one. The first receiver moves at constant speed from point A to point $E = (1.5, 5, 0)m$ and then back to A with constant speed. The second receiver remains stationary as to better focus on the effects of having the other receiver moving around. Also, in this scenario the wireless channel is cut loose so the nodes can't use it to perform any form of coordination. The scheduler thus proceeds in a simple round robin fashion where the next packet is broadcast on all transmitters.

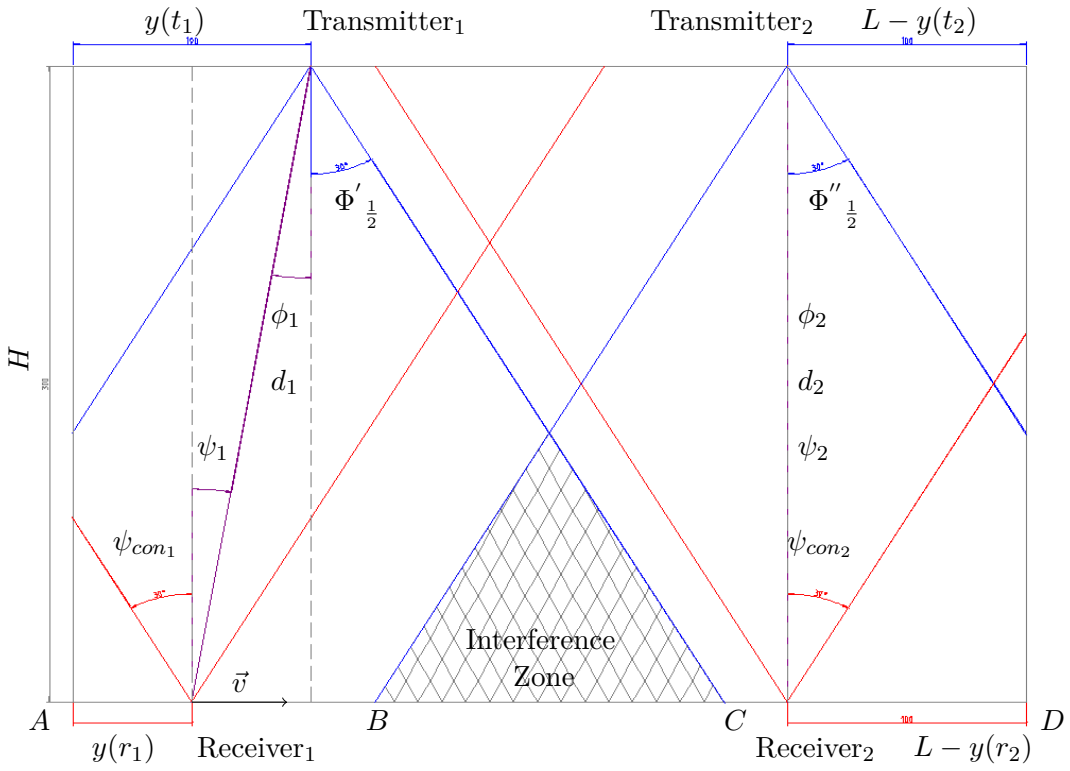

Figure 3.13. Double receiver and transmitter setup

Test parameters	
Transmitter 1 position, $P(T_1)$	$(1.5, 1, 3)m$
Transmitter 2 position, $P(T_2)$	$(1.5, 3, 3)m$
Receiver 1 initial position, $P_i(R_1)$	$(1.5, 0, 0)m$
Receiver 1 first intermediate position, $P_t(R_1)$	$(1.5, 5, 0)m$
Receiver 1 final position, $P_f(R_1)$	$(1.5, 0, 0)m$
Receiver 2 position, $P(R_2)$	$(1.5, 3, 0)m$
Receiver 1 speed A to E, $\vec{v}$	$(0, 0.5, 0)\frac{m}{s}$
Receiver 1 speed E to A, $\vec{v}$	$(0, -0.5, 0)\frac{m}{s}$

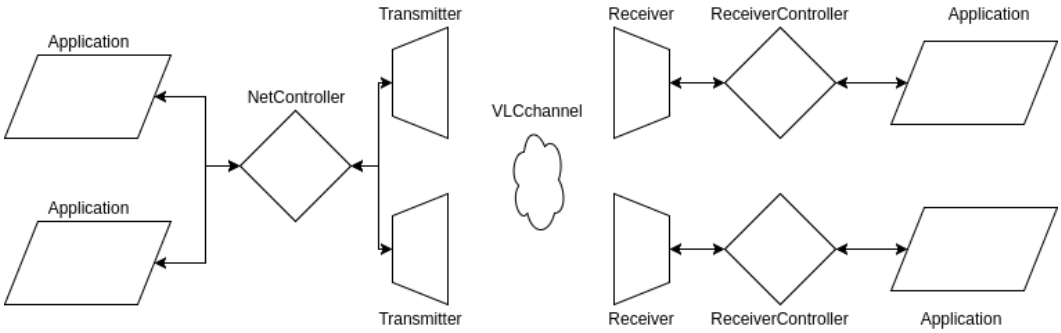

Figure 3.14. Two transmittes and two receivers

The results of the test are presented in Fig. 3.15 and Fig. 3.16. The second receiver R_2 stays still directly below the second transmitter T_2, its SINR, BER and throughput are stable throughout the test. The first receiver R_1 starts in the FoV of the first transmitter T_1 at $y(r) = 0m$ it reaches the interference zone at $y(r) \approx 1.26m \approx 3 - 3tan(30°)$ and the throughput starts rapidly declining since no packet ever reaches its destination. It then exits the interference zone at $y(r) \approx 2.73m \approx 2 + 3tan(30°)$ and the throughput starts increasing again. Notice how the two receivers are completely independent from each other. R_1 then exits the FoV of T_2 at $y(r) \approx 4.73m \approx 3 + 3tan(30°)$ and its throughput starts declining again. Finally R_1 then reverses its speed at $y(r) = 5m$ and the whole thing repeats itself.

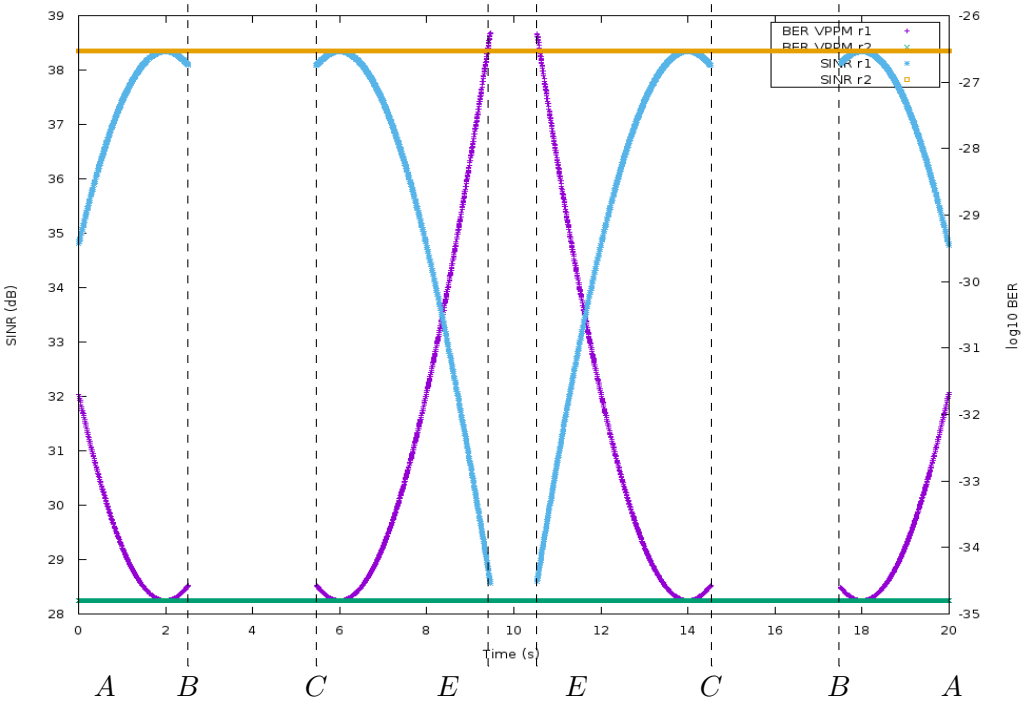

Figure 3.15. Double receiver and transmitter setup SINR and BER

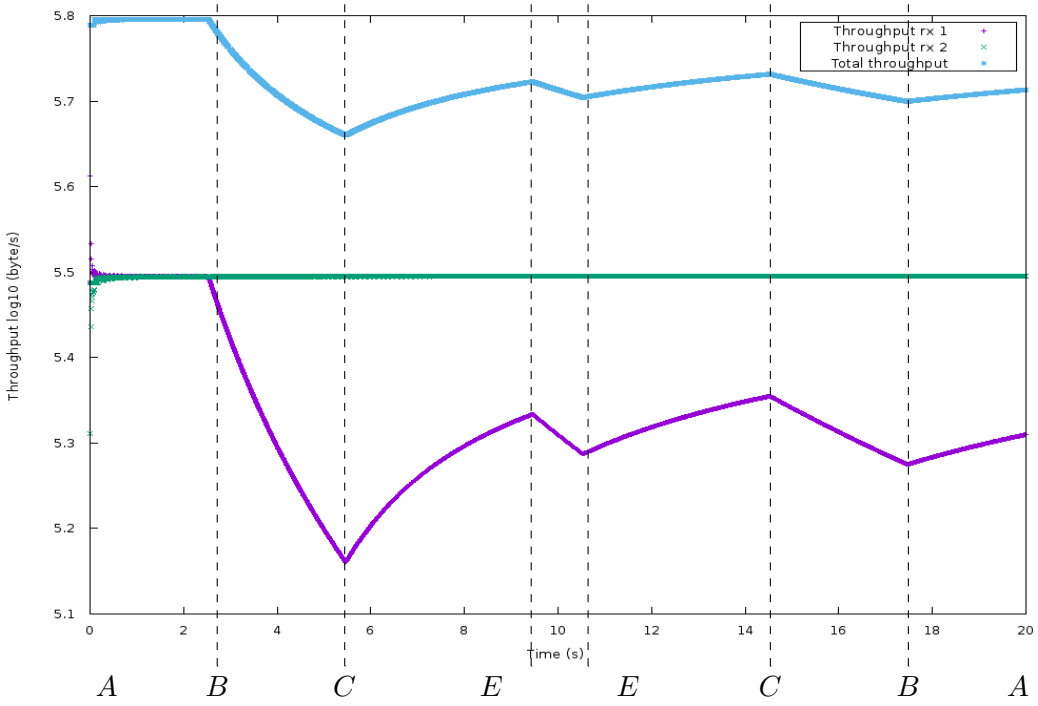

Figure 3.16. Double No Wireless test throughput

3.1.2.2 Double Hybrid Test

In this test we have the same geometry, topology and movements of the nodes but we restored the wireless channel.
The results of the test are presented in Fig. 3.17 we don't picture the results of the SINR and BER since they are the same as earlier. Now the two receivers can influence each other and we can immediately notice it from the graph. Compared to the previous case in fact now when R_1 moves into the FoV of the second transmitter the throughput of R_2 start declining since the transmitter must be shared.
However this setup never performs worse than the previous one and in fact it does way better. Now that the wireless channel was available the total received data amounted to $T_{wireless} \approx 16 \times 10^6$ bytes while in this last scenario we only received $T_{noWireless} \approx 10 \times 10^6$ bytes. This means we got a 50% increase! And in this case one transmitter idled for most of the time, since no receiver was in its FoV. A bit counterintuitively greater benefits can be reaped in some cases when the number of receivers increases since all the transmitter can be kept busy.

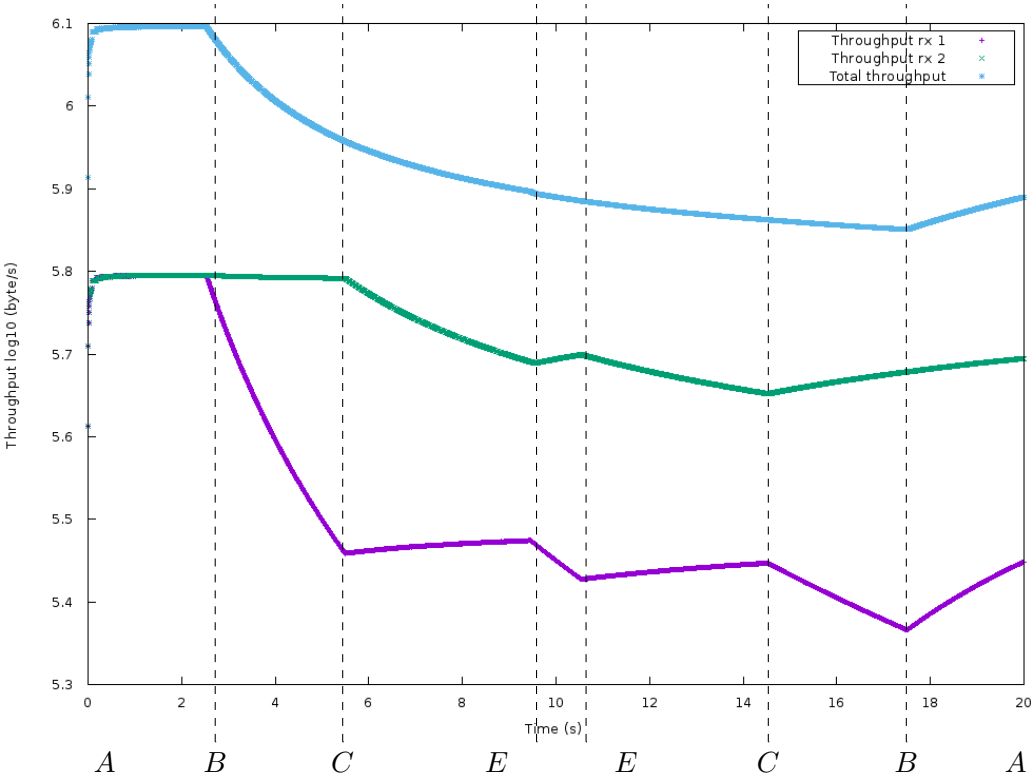

Figure 3.17. Double Hybrid test throughput

3.1.3 Physical test

Simulations are really nice since one has complete control over their models and variables. The real world on the other hand is messy and complicated, and unfortunately the one we live in. This thesis was written alongside [9] where they built a testboard for VLC. We will go quickly over some characteristics of the physical devices used and and compare theoretical to actual data points.
The first simplification that we made in our model is that we assumed our light sources are perfectly monochromatic, i.e., they emit light at a single wavelength. The only thing that comes close to that in the real world are lasers and they are only found as light fixtures in places such as discos. The almost totality of light sources emit light in a spectrum that most of the times is tailored to appear white to our eyes. In Fig. 3.18. for example we can observe the spectral distribution of a *what-we-perceive-white* LED.

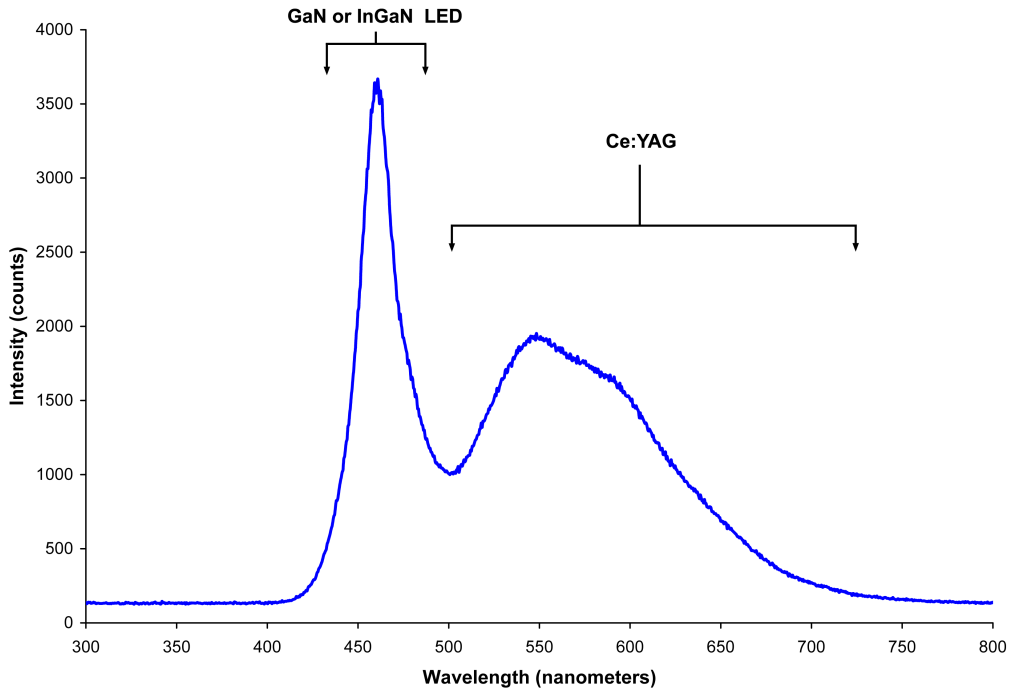

Figure 3.18. White Led Spectrum

If we wanted to compute the power received by our photoreceiver we would then have to integrate over the portion of the spectrum to which the receiver is sensible. This is trivial when the radiation intensity distribution is given but the LED utilized, a L-813SRC-J4, unfortunately only comes with the Relative Radiation Intensity pictured in Fig. 3.19.

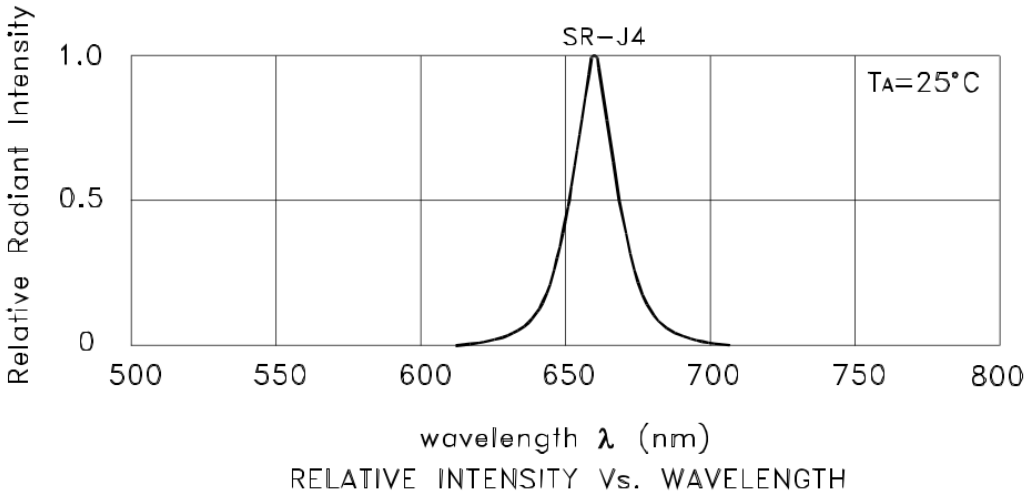

Figure 3.19. Our Led Spectrum

Fortunately, along with the photodiode BPW34 it also provides some other information, gathered in the table below:

LED info	
lv @ 50mA, I_v	$20000mcd$
Peak Wavelength, λ_{peak}	$660nm$
Half angle, $\Phi_{\frac{1}{2}}$	$0.131rad$
Photodiode info	
Photodiode area, A	$7.5 \times 10^{-6}m^2$
Short Circuit current, C with $E_e = 1mW/m^2$, $\lambda = 950nm$	$4.7nA$
Peak Wavelength, λ_{peak}	$900nm$
Scenario info	
Device distance, d	$1m$
Incidence angle, ψ	$0rad$
Radiance angle, ϕ	$0rad$

The photodiode also provides its Relative Responsivity Curve

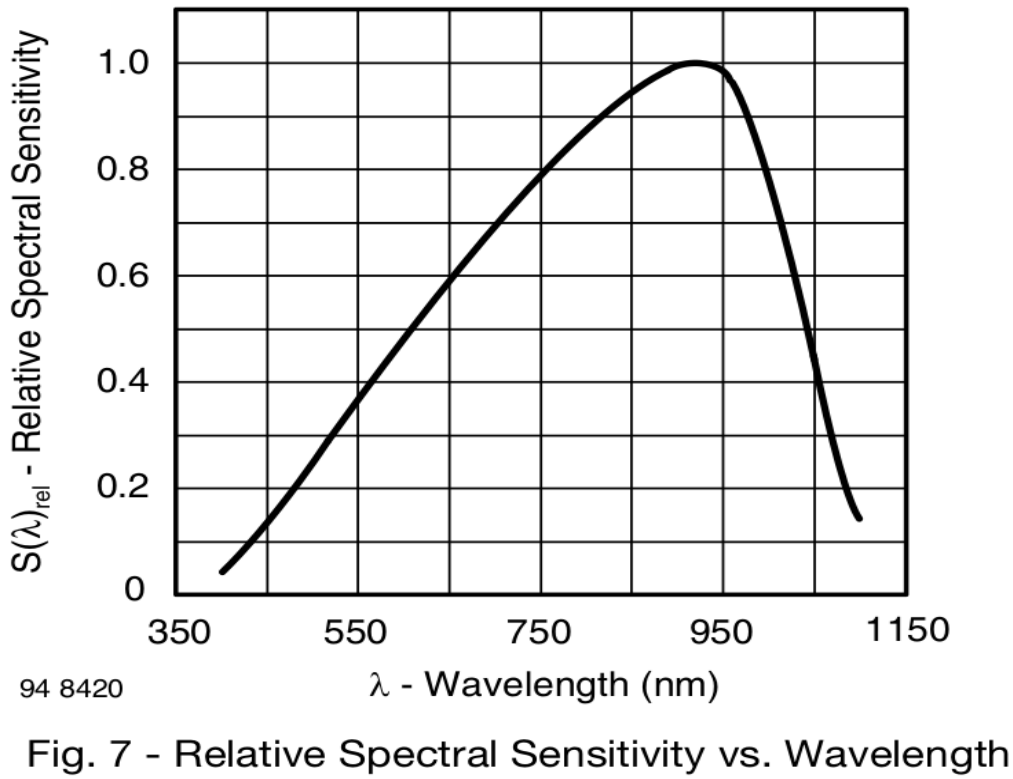

Figure 3.20. Photodiode Responsivity Curve

This ultimately translates to how much current is produced for the same amount of power received at wavelength λ compared to how much would have been produced if the light was at λ_{peak}. Back to the LED, we know that the luminous intensity is equal to:

$$I_v = 683.002 \frac{lm}{W} \int_{\lambda_{min}}^{\lambda_{max}} \bar{y}(\lambda) \cdot \frac{dI_e(\lambda)}{d\lambda} d\lambda$$

Where $K = 683.002 \frac{lm}{W}$ is a constant related to human photopic vision, $\bar{y}(\lambda)$ is the standard human luminosity function and $I_e(\lambda)$ is the Radiant Intesity function of our light source. We can make an educated guess and say that:

$$I_e(\lambda) = R(\lambda) \cdot I_0 \text{ for some constant } I_0 \text{ and thus}$$
$$I_0 = \frac{I_v}{683.002 \frac{lm}{W} \int_{\lambda_{min}}^{\lambda_{max}} \bar{y}(\lambda) \cdot \frac{dR(\lambda)}{d\lambda} d\lambda}$$

Where $R(\lambda)$ is the Relative Radiant Intensity function. We could then have the value of I_0 if we had the function $R(\lambda)$. Since all we have are a *finite set* (thanks to [10]!) of values of $R(\lambda)$ however, the best we can do is use finite differences to approximate the derivative at each point. This give us an $I_0 = 0.338 W/sr$ at peak intensity λ_{peak} .

We must now compute the amount of power on the surface of the photodiode. Now, recall that our LED behaves like a Lambertian Emitter, with the radiant intensity that depends on the angle of radiance. Since our transmitter and receiver are placed directly in front of each other however, we can simplify our equations a lot and we get the following for the short circuit current at the photodiode:

$$C_p = I_0 C \cdot \frac{A}{d^2} \int_{\lambda_{min}}^{\lambda_{max}} R(\lambda) R_f(\lambda) d\lambda$$

With $R_f(\lambda)$ the responsivity function of our photodiode of which, again we only have a finite set of values. In a terribly crude approximation we get a current of about $C_p = 3.74 \mu A$ which, considering the sparsity of our datasets is still pretty close to the $4.0 \mu A$ reported in [9].

3.2 Conclusions

We provided an easily extensible framework for OMNeT++ that permits to simulate VLC-enabled networks. We then validated the model used in the simulator comparing our results to other works in the same area of research that have been collected through means of both computer simulations and physical testing. Furthermore we explored the possibility of enhancing a pure VLC system by means of existing wireless technologies, that we used to implement our own toy version of a MAC protocol that nonetheless proved to be sufficient to give a net increase in the overall performance of the network.

Bibliography

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update," February 2017.

[2] P. H. Pathak, X. Feng, P. Hu, and P. Mohapatra, "Visible light communication, networking, and sensing: A survey, potential and challenges," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 2047–2077, Fourthquarter 2015.

[3] D. Tagliaferri and C. Capsoni, "Development and testing of an indoor vlc simulator," *2015 4th International Workshop on Optical Wireless Communications (IWOW)*, pp. 122–126, Sept 2015.

[4] A. Aldalbahi, M. Rahaim, A. Khreishah, M. Ayyash, R. Ackerman, J. Basuino, W. Berreta, and T. D. C. Little, "Extending ns3 to simulate visible light communication at network-level," *2016 23rd International Conference on Telecommunications (ICT)*, pp. 1–6, May 2016.

[5] M. S. "Ab-Rahman, N. I. Shuhaimi, L. A. H. Azizan, and M. R. Hassan, "Analytical study of signal-to-noise ratio for visible light communication by using single source," *Journal of Computer Science*, vol. 8, pp. 141–144, 2012.

[6] X. W. Yunlu Wang and H. Haas, "Load balancing game with shadowing effect for indoor hybrid lifi/rf networks," *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS*, vol. 16, April 2017.

[7] K. J. Aarthi. H., "A novel protocol design in hybrid networks of visible light communication and ofdma system," *IEEE Conference Publications*, pp. 1–5, 2015.

[8] E. H. E. K. Sharan Naribole, Shuqing Chen, "Lira: a wlan architecture for visible light communication with a wi-fi uplink," *IEEE Conference Publications*, pp. 1–9, 2017.

[9] P. C. Ageev Artem, "Progettazione di un sistema visible light communication per iot," 2017.

[10] A. Rohatgi, "Webplotdigitizer," June 2017.

Image sources

1.1 By Nick84 - http://commons.wikimedia.org/wiki/File:Solar_spectrum_ita.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24648395
1.2 By Txbangert - Data is from X-Rite i1Pro (118 sensor values between 350-740nm). Original file:
http://www.eecs.qmul.ac.uk/ tb300/pub/Images/Spectrum_of_Sunlight_en.pdf, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=35719991
2.2 By Inductiveload - Own work, Public Domain,
https://commons.wikimedia.org/w/index.php?curid=5290437
2.8 By Gonzalo Medina -
https://tex.stackexchange.com/questions/230979/how-to-draw-a
-parallel-network-of-queues
3.18 By Deglr6328 at the English language Wikipedia, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=3242448
3.19 http://www.farnell.com/datasheets/1701184.pdf
3.20 https://www.vishay.com/docs/81521/bpw34.pdf