Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

RESEARCH INTERNSHIP

Simulating Inter-Cell Interference in 5G NR

Author:	YALCINTAS,Yagiz
Address:	Notburgastrasse 23
	80639 Munich
	Germany
Matriculation Number:	03721475
Supervisor:	Yash Deshpande
Begin:	04. April 2023
End:	15. November 2023

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, 23.11.2023

Place, Date

Signature

München, 23.11.2023

Place, Date

Signature

Abstract

5G NR is the fifth-generation technology standard for cellular networks that has already begun to be deployed globally. It promises faster data rates, serving more users with improved quality of service in crowded areas. Yet synchronization among cells of inter/intra operators are still a topic to be further discovered.

This paper uses the standards mentioned in the recommendation [2] as a foundation. Based on [2], this paper creates a framework for a deeper look into aspects of 5G NR TDD systems using the 5G toolbox [3]. The implementation of this framework is explained step by step. Using this framework, the network's behavior is examined under diverse conditions for validation purposes of this framework. These tests include UE positions, gNB positions, the number of UEs per cell, channel bandwidths, subcarrier spacing, and transmit power of cells. Subsequently, the investigation extends to the impact of time offsets applied to PDSCH and/or CSI-RS packets of the interfering cell, aiming to understand the intricate interplay between time synchronization and network performance.

The anticipated outcome involves observing a decrease in achieved throughput with larger time offsets due to increased base station-to-base station interference. However, the obtained results exhibit inconsistency, prompting consideration of factors such as the lack of UE-to-UE interference and cell selection implementation.

In essence, this internship lays the foundation for a versatile 5G application testing environment. A potential avenue for future research includes the addition of UE-to-UE interference implementation and a deeper exploration of different channel types used in simulations. The intricate interdependencies between synchronization and various network parameters pave the way for nuanced insights and optimizations in 5G NR TDD systems.

Contents

Chapter 1

Introduction

The advent of 5G technology heralds a transformative era in communication, promising unprecedented levels of speed, connectivity, and reliability. The potential for enhanced communication quality opens doors to innovative applications and seamless connectivity experiences.

5G networks are cellular networks that segment the service area into smaller areas defined as cells. All 5G wireless devices in a cell communicate with a cellular base station via radio waves on frequencies that are assigned by the base station.

However, as the number of base stations expands to meet the demands of this high-speed network, the significance of precise time synchronization becomes paramount. Synchronization is not only essential between cells of the same operator but also between different operators. Since synchronization offset can cause interference in neighboring cells regardless of the operator resulting in decreased performance. In this era of rapid technological advancement, understanding and addressing the challenges posed by synchronization become pivotal for unlocking the full potential of 5G communication networks.

This paper focuses on the synchronization characteristics outlined in ITU-T Recommendation [2]. The imperative nature of adherence to these specifications is underscored by the necessity for interoperability among equipment from diverse manufacturers and the optimization of network performance. Synchronization, both in terms of time and phase, proves to be a linchpin for the seamless operation of telecommunication networks.

While [2] outlines the time synchronization requirements for 5G NR TDD systems, there is still no extensive testing data gathered on this topic. This paper raises an open question about the precise impact of various time offsets at the application layer. To address this gap, the paper establishes a comprehensive framework, leveraging the 5G toolbox from MathWorks ([3]). The validation of this toolbox involves a series of tests on a simple network topology, featuring two interfering neighboring cells—one acting as the interfered cell and the other as the aggressor cell, both operating on the same frequency.

Chapter 2

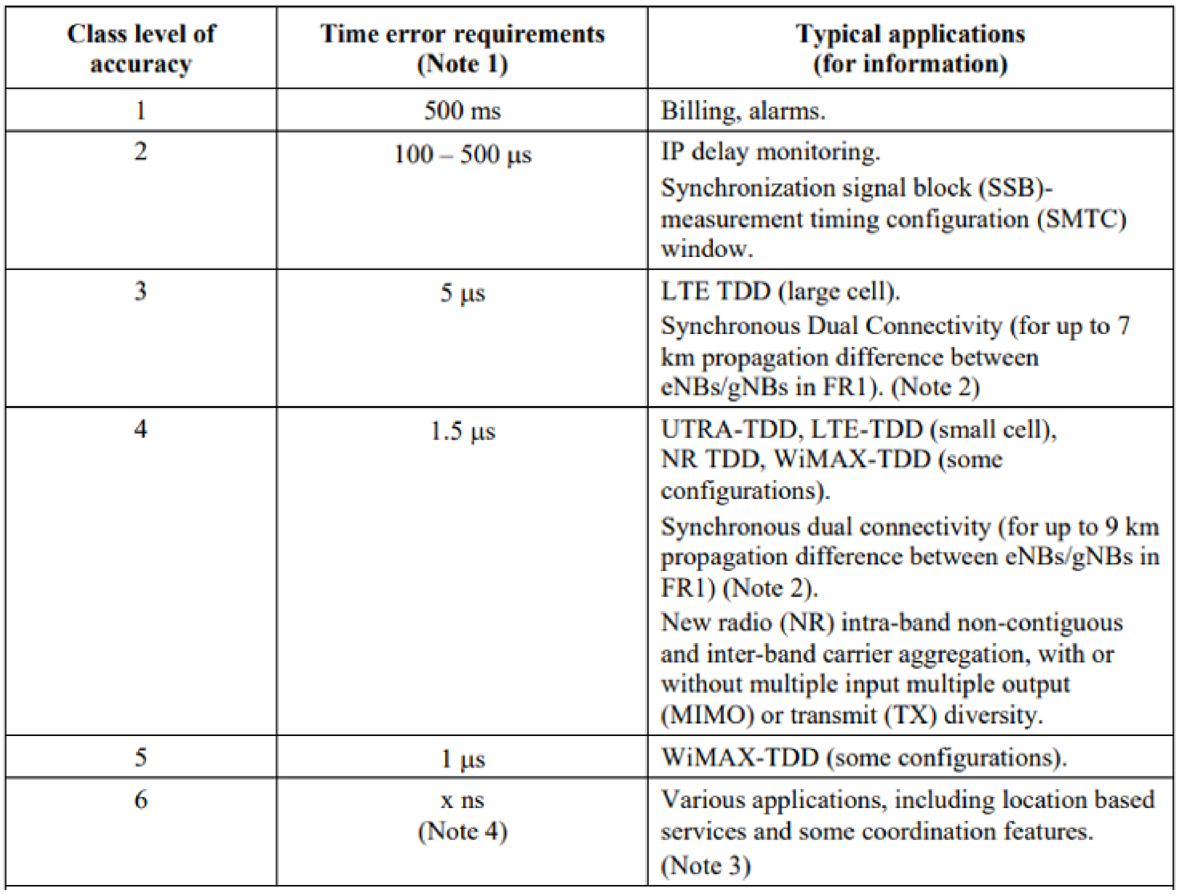
Background

2.1 Recommendation G.8271/Y.1366

The International Telecommunication Union (ITU) is the United Nations specialized agency in telecommunications, information, and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating, and tariff questions and issuing Recommendations to standardize telecommunications worldwide. The requirements for the synchronization characteristics that are specified in this Recommendation([2]) must be adhered to ensure interoperability of equipment produced by different manufacturers and a satisfactory network performance

Time synchronization is traditionally required to support billing and alarm functions (maintenance or fault isolation). In this context, synchronization must in general be accurate to within hundreds of milliseconds. Another time synchronization application is the monitoring of delays in Internet protocol (IP) networks. In this case, the requirement is accuracy to within some hundreds of microseconds (the actual requirement depends on the application) Stringent time synchronization requirements (i.e., in the range of a few microseconds) apply to the generation of signals over the air interface of some mobile systems, such as code division multiple access (CDMA)2000 or long-term evolution frequency division duplexing (LTE FDD) unicast, when it is required to support synchronous CDMA2000 interworking.

Phase synchronization is often needed to support requirements for the air interface of some mobile systems, as in the case of time division duplexing (TDD) systems (for instance, LTE TDD), or when supporting multimedia broadcast/multicast service (MBMS). Note that ordinary wideband CDMA (WCDMA) MBMS does not require accurate phase synchronization since it has been specified and designed to work properly in networks that satisfy the 50 ppb frequency accuracy requirement. This requirement, guaranteed by the WCDMA node synchronization function (see [1]), limits phase drift between 10 and 20 ms. But when MBMS is based on single-frequency network (MBSFN) mode, timing must be accurate within a few microseconds. This is because identical waveforms are transmitted simultaneously from multiple cells. The signals from these cells are then combined as the multipath components of a single cell. Terminals must thus perceive the signals of an entire group of transmitting cells as though they came from a single cell. Therefore, all transmissions must be very tightly synchronized and deliver exactly the same content to each base station.



Class level of accuracy	Time error requirements (Note 1)	Typical applications (for information)
1	500 ms	Billing, alarms.
2	100 – 500 μs	IP delay monitoring. Synchronization signal block (SSB)-measurement timing configuration (SMTC) window.
3	5 μs	LTE TDD (large cell). Synchronous Dual Connectivity (for up to 7 km propagation difference between eNBs/gNBs in FR1). (Note 2)
4	1.5 μs	UTRA-TDD, LTE-TDD (small cell), NR TDD, WiMAX-TDD (some configurations). Synchronous dual connectivity (for up to 9 km propagation difference between eNBs/gNBs in FR1) (Note 2). New radio (NR) intra-band non-contiguous and inter-band carrier aggregation, with or without multiple input multiple output (MIMO) or transmit (TX) diversity.
5	1 μs	WiMAX-TDD (some configurations).
6	x ns (Note 4)	Various applications, including location based services and some coordination features. (Note 3)

Figure 2.1: Class Level Accuracy Requirements ([2])

NOTE 1 – The requirement is expressed in terms of time error with respect to a common reference. Some of the original requirements were expressed in terms of relative time error.
NOTE 2 – FR1: 410 MHz – 7.125 GHz; FR2: 24.25 – 52.6 GHz

2.2 Time and phase synchronization methods

Packet-based methods (typically using the network time protocol (NTP)) without timing support from the network are traditionally used to support applications with less strict time and phase synchronization requirements (class 1 according to 2.1). This Recommendation([2]) focuses on applications corresponding to classes 4, 5, and 6 according to 2.1.

For these applications, the following options are considered in this Recommendation([2]): a distributed primary reference time clock (PRTC) approach, implementing a global navigation satellite system (GNSS) receiver in the end application (a global positioning system (GPS) receiver, for example); – packet-based methods with timing support of intermediate nodes.

Distributed PRTC:

Distributed PRTC method refers to distributing synchronization signal to each clock in the network. This method suits more to radio network since cabling to each clock in network can become unfeasible due to complexity and maintenance. The distribution is typically done with GNSS (eg. GPS).

Advantages of this architecture is the globally availability of time reference signals in case of GNSS. A less complex, flat hierarchy with no risk of timing loops.

Disadvantage of this method can be said that being prone to have interference.

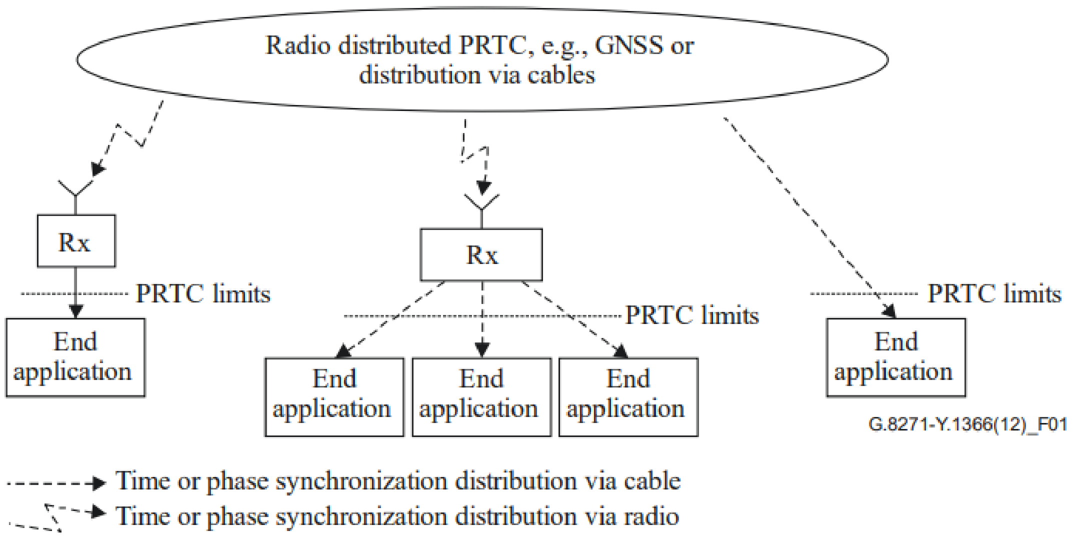

Radio distributed PRTC, e.g., GNSS or distribution via cables
Rx
PRTC limits
End application
Rx
PRTC limits
End application
End application
End application
PRTC limits
End application
G.8271-Y.1366(12)_F01
Time or phase synchronization distribution via cable
Time or phase synchronization distribution via radio

Figure 2.2: Distributed PRTC Example ([2])

Packet Based Methods with Timing Support of Intermediate Nodes:

Time synchronization can be distributed via timing protocols. [2] only considers the case where the timing reference is carried by the network. The timing support in the intermediate nodes can correspond to telecom boundary clock (T-BC) which terminates and regenerates timestamp messages.

The Figure 2.3 shows an example of phase/time synchronization distributed via packet-based methods with timing support from the network. The packet timing distribution is initiated by a packet master clock function in a telecom grandmaster (T-GM) with access to reference timing signal, and every transport node implements a T-BC.

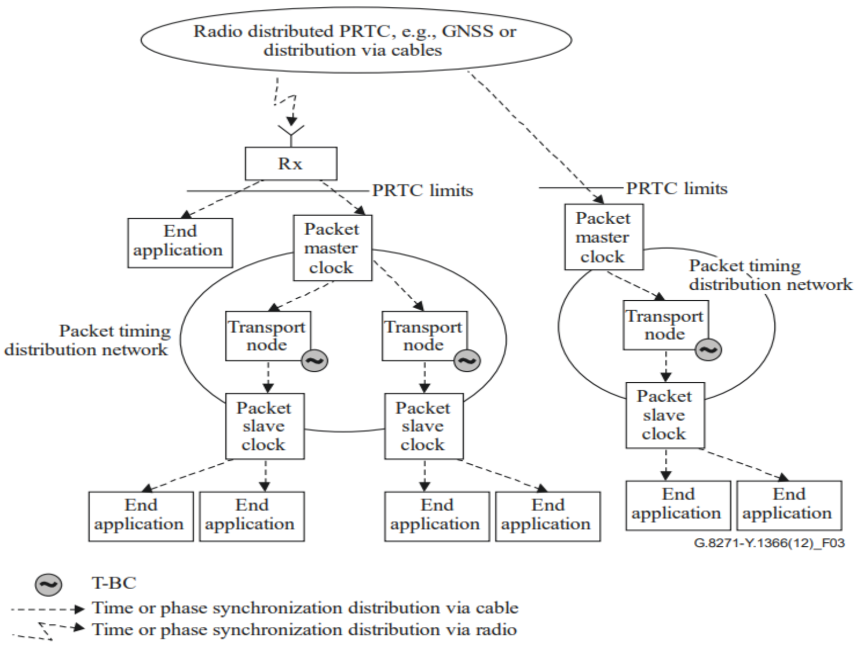

Radio distributed PRTC, e.g., GNSS or distribution via cables
Rx
PRTC limits
End application
Packet master clock
Packet timing distribution network
Transport node
Packet slave clock
PRTC limits
G.8271-Y.1366(12)_F03
T-BC
Time or phase synchronization distribution via cable
Time or phase synchronization distribution via radio

Figure 2.3: Packet Based Methods with Timing Support of Network ([2])

Packet Based Methods with Timing Support of Network architecture requires much fewer GNSS receivers yet the network complexity is higher. Also it should be noted that the noise accumulation and time error due to asymmetries in the network needs to be taken into account.

2.3 Time synchronization aspects in TDD-based mobile communication systems

1. Interference in TDD systems occur when signals from adjacent cells overlap a UL time slot at a base station or DL time slot at a UE. Synchronization errors at a single base station can lead to performance degradation of neighboring base stations, whether the base

stations are of the same operator or different operators. This is due to the interference caused by the neighboring base station. Hence, in TDD systems, it is essential that base stations receive accurate synchronization.

If TDD systems are operated without guard frequency bands, not only intra-operator time synchronization but also inter-operator time synchronization to a common timing standard is required. One possibility is to make it explicit that the reference must be traceable to a specific common recognized time standard source. Coordinated Universal Time (UTC) might be a good candidate.

As shown in Figure 2.4, TDD systems have two types of interference conditions: BS-to-BS and UE-to-UE. Interference occur in TDD systems due to propagation delay of signals in each scenario described below. The propagation delay of a DL signal is higher in large cells than in small cells.

Moreover, the propagation delay of a DL signal is not proportional to the cell radius in inter-operator interference, unlike in intra-operator interference.

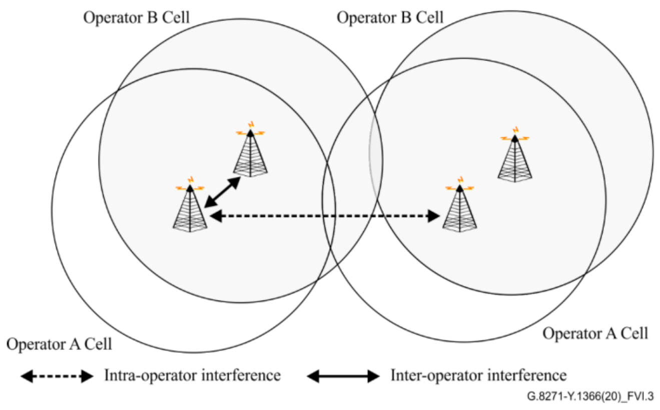


Figure 2.4: Interference between inter- and intra-operator ([2])

Guard periods (GPs) in the signal frame format of TDD systems are provided at the switching point around DL-to-UL and UL-to-DL switching as shown in 2.5. Guard periods are not used for signal transmission but contribute to reducing interference during DL to UL switching.

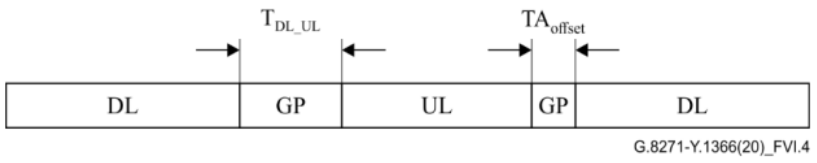


Figure 2.5: Guard Band ([2])

Downlink to Uplink Switching Point:

The time synchronization error between base stations (TSync) is defined as the timing error at the Antenna Reference Point (ARP). If the propagation time between interfering and interfered base stations is represented as $TpropBS2BS$, and power ramp-down time at the base station transmitter as $TbtsRampdown$, BS-to-BS interference at DL-to-UL switching occurs under the following condition:

$$TSync > TDL/UL - TpropBS2BS - TbtsRampdown \tag{2.1}$$

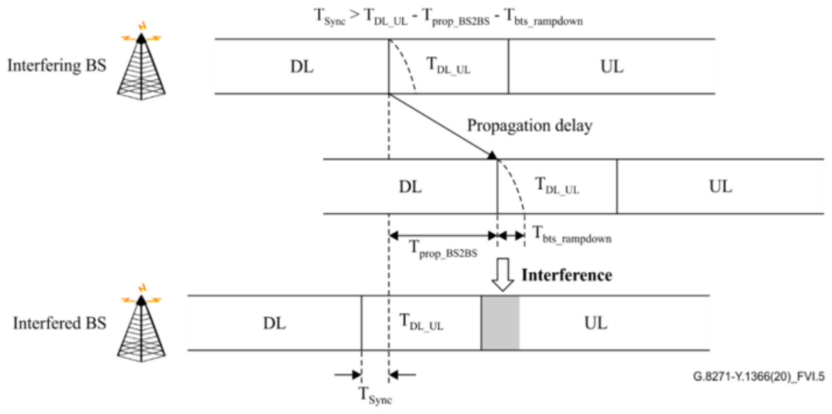


Figure 2.6: Interference Between Base Station during DL to UL Swiching ([2])

Note that the interference decays as the path loss increases with increased $TpropBS2BS$, so that at after certain distance it will not be significant.

Uplink to Downlink Switching Point:

BS-to-BS interference at the UL-to-DL switching point occurs under the following condition where $TbtsRampup$ is the power ramp-up time at the base station transmitter.

$$Tsync > TAoffset + TpropBS2BS - TbtsRampup \tag{2.2}$$

Propagation delay may take a smaller value in inter-operator interference than in intra-operator interference, so this type of interference may have a greater possibility of occurring.

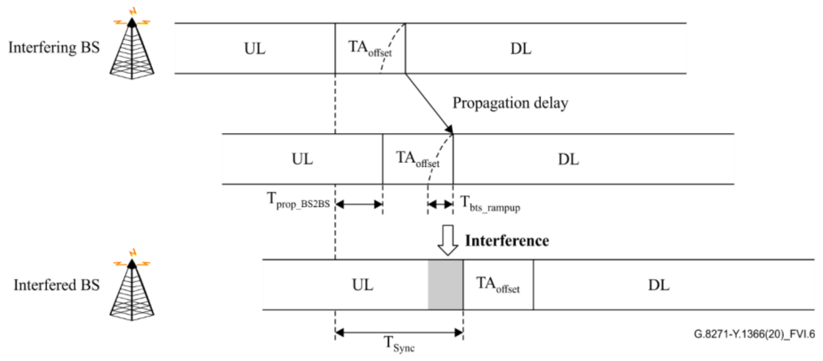


Figure 2.7: Interference Between Base Station during UL to DL Switching ([2])

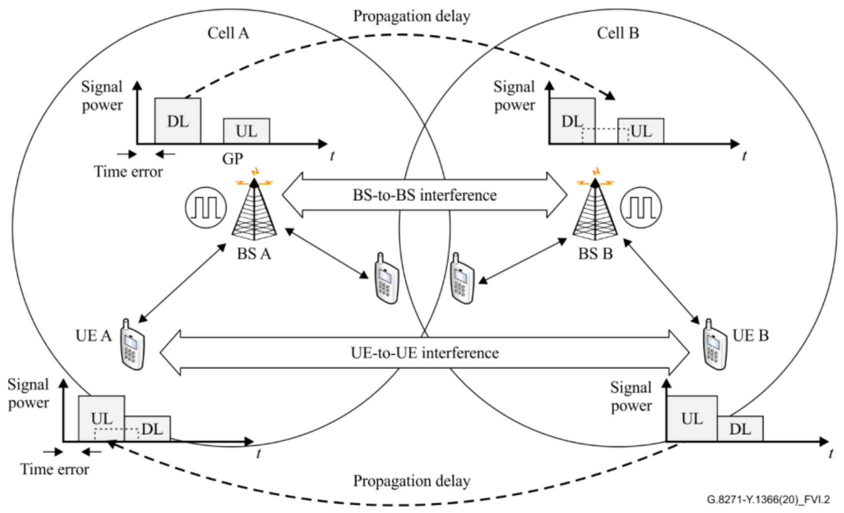


Figure 2.8: Interference Patterns in TDD Systems ([2])

User equipment to user equipment interference:

Reachability of the signal from interfering UE to interfered UE is lower in the case of UE-to-UE interference than in the case of BS-to-BS interference. Thus, only the condition under which both UEs are located close to the cell edge can cause intra-operator interference. On the other hand, UE-to-UE interference may occur anywhere in the cell in the case of inter-operator interference. Similar relationship can be derived for BS-to-BS interference, where time synchronization error between base stations exceeding a certain threshold would cause interference between the UEs. Interference between UEs is out of the scope of this internship.

Chapter 3

Implementation/Results

3.1 Implementation

As mentioned in Introduction, we used the 5G toolbox of Mathworks ([3]). We initially used MATLAB R2023a version, but it failed to implement full physical functionality. Despite defining the abstraction flag as ”false”, the simulation could only use the default channel model without any interference. Hence we switched to version R2023b, which could implement the features of full physical transmission and reception.

Our main classes of interest in our simulation are wirelessNetworkSimulator, wirelessNode, nrNode, nrGNB and nrUE as it can be seen in figure A.1 .

Initialization Script

Our initialization script creates the objects (gNBs, UEs, wirelessNetworkSimulator, channel model etc.) with the relevant configuration (center frequency, bandwidth, position, subcarrier spacing etc.)

The initialization script creates and assigns the custom full physical channel links between gNBs and UEs. The channel config struct has properties DelayProfile and DelaySpread. The delay profile provides multiple versions of the transmitted signal with different delays and phase shifts due to multipath. Delay spread is the difference between the earliest and latest reception components in the time domain. For these properties, ”CDL-C” and 300e-9 values are used. The initialization script finally invokes the networkSimulator object that starts the simulation.

nrGNB.m

gNBs are instances of nrGNB class which is defined in nrGNB.m. it has the properties are: Name, Position, DuplexMode(TDD in our case), CarrierFrequency, ChannelBandwidth, SubcarrierSpacing, NumResourceBlocks, NumTransmitAntennas, NumReceiveAntennas, TransmitPower, PHYAbstractionMethod("none" in our case), DLULConfigTDD, NoiseFigure, ReceiveGain, NumHARQ, ID and ConnectedUEs. The method of creating an instance, obj = nrGNB(varargin)method in nrGNB.m is where the MAC entity and PHY entity is defined. Within nrGNB(varargin) method the MAC and PHY entity intarfaces are binded to nrGNB object with the setLayerInterfaces(obj) method that resides in nrGNB.m. setLayerInterfaces() registers Phy interface functions to MAC for sending packets, receive, DL and UL requests to Phy. It also registers MAC callback function at PHy for sending packets and UL channel quality data to MAC. All the transmissons and receptions are done according to the interlayer interface bindings that are set in this script.

A similar process of interlayer binding happens for UE in nrUE.m

wirelessNetworkSimulator.m

wirelessNetworkSimulator class is defined in wirelessNetworkSimulator.m. Which is the main class where all the other objects such as gNBs and UEs are bound to. The main functionality of this class is to Simulate a multimode wireless network for a specified period and handle or cancel the actions that are scheduled to advance the simulation. Methods of interest of this class include: addChannelModel(obj, customChannelFcn), addNodes(obj, nodes), run(obj, simulationDuration), dt = nextInvokeTime(obj), distributePackets(obj, recentlyRunNodesIdx).

addChannelModel(obj, customChannelFcn): Adds a custom channel and path-loss model. Obj is an instance of wirelessNetworkSimulator type object. customChannelFcn is a function handle with signatures: RXINFO and TXPACKET. RXINFO has fields: ID, Position, Velocity (representing receiver node position), and NumReceiveAntennas. TXPACKET has the fields: Type (0,1,2,3,4 which corresponds to nvalid packet, WLAN, 5G, Bluetooth LE, Bluetooth BR/EDR packets), TransmitterID, TransmitterPosition, TransmitterVelocity, NumTransmitAntennas, StartTime (transmission starting time or reciaval starting time in seconds), Duration(duration of the packet), CenterFrequency, Bandwidth, Abstraction(abstracted PHY or FULL PHY), SampleRate, DirectToDestination(decide if the packet is sent through the channel or bypasses and reaches the receiving node directly such as control packets which are out of band transmission), Data (contains time samples for full PHY case or frame information in case of abstracted PHY) The fields we see for TXPACKET are also present as the packet information throughout the simulation flow. It is not specific to this method only.

addNodes(obj, nodes): It binds the gNB and UE nodes into the wirelessNetworkSimulator.

run(obj, simulationDuration): this method in wirelessNetworkSimulator can be said to be the starting point of all the actions to be processed at their scheduled times and advancing the simulation time for the simulation duration. Invoking all the nodes for transmission, receival, distributing packets(with distributePackets()), updating the next invoke times of nodes, invoking the actions for updating stats and visualizers, and advancing simulation time (with dt=nextInvokeTime(obj)) is done here. This method belongs to the nrNode class which is defined in nrNode.m

distributePackets(obj, recentlyRunNodesIdx): The main functionality of this method is to distribute data from transmitter nodes to receiving nodes. It checks the direcToDestionation value to decide if the channel should be applied or if there is an out-of-band transmission. It checks if the packet is relevant to other nodes according to their center frequency setting with the help of the isPacketRelevant(obj, packet) method in nrNode.m. In case the transmission is found to be relevant the packet is directed to the receiving buffer of the relevant node. Finally, it updates the nextInvokeTime of the receiving node(if the packet is found to be relevant) to the current time of simulation for instant reception.

dt = nextInvokeTime(obj): Checks the next invoke times of all the nodes and the scheduled actions and picks the earliest one to advance the simulation time.

nrNode.m

nrNode class contains the properties and components that are common for both gNB and UE nodes. It is the umbrella class for gNB and UE nodes. It has properties that are of interest: TrafficManager, MACEntity, PhyEntity, CurrentTime, and PHYAbstraction. The main methods we will go over are: nextInvokeTime = run(obj, currentTime), isPacketRelevant(obj, packet), nextInvokeTime = runLayers(obj, currentTime, packets).

isPacketRelevant(obj, packet): It checks if the packet is relevant to the receiving node. It uses the isPacketRelevant(obj, packet) method of the wirelessNode class that is defined in wirelessNode.m

run(obj, currentTime) function is the main invoking function of the node in our simulation. The run() method in wirelessNetworkSimulator.m points to the run() method in nrNode.m. It runs the 5G Node at the current time and returns the next invoke time. It checks if there is Rx data to be processed or not and runs the runLayers() method in nrNode.m. If it has rX data to be processed it puts the receiveBuffer object as a parameter in the runLayers() method.

nextInvokeTime = runLayers(obj, currentTime, packets): Runs the node with received packets and returns the next invoke time. This is the main method where the application, MAC, and physical layers are run. After each layer is run it chooses the earliest next invoke time of the 3 layers as the next invoke time. Runlayers has 3 run() methods: nextAppTime = run(obj.TrafficManager, currentTime), nextMACTime = run(obj.MACEntity, currentTime, packets), nextPHYTime = run(obj.PhyEntity, currentTime, packets).

trafficManager.m

nextAppTime = run(obj.TrafficManager, currentTime) method resides in trafficManager.m. This class runs all network traffic objects that generate traffic at the scheduled time with relevant packet information and forwards it to lower layers. Finally updates the statistics. and return the next invoke time. nextMACTime = run(obj.MACEntity, currentTime, packets): Calls the run function of nrGNBMAC class in nrGNBMAC.m.

nrGNBMAC.m

nrGNBMAC class implements the MAC layer interaction with PHY for transmission and receive chains of gNB nodes. This class contains a scheduler entity that handles UL and DL scheduling. UL and DL assignments, the physical downlink control channel (PDCCH) are sent out-of-band directly from the MAC entity. Since the Physical uplink control channel (PUCCH) is not modeled control packets from UEs, PDSCH and CSI are also handled out-of-band. Run() method runs the MAC layer operations and updates the next invoke time.

nrPHY.m

nextPHYTime = run(obj.PhyEntity, currentTime, packets): Calls the run function of nr-PHY in nrPHY.m which acts as a base class for all the physical layer types. In this method, the duration completed in the current slot is calculated. The current AFN, slot, and symbol are calculated. Phy transmission is done with the phyTx method within nrGNBPHY class defined in nrGNBPHY.m. This is followed by storing the received packets with the storeReception() method in nrPHY.m, physical reception with Phy Rx() method in nrGNBPHY.m and getting the next invoke physical layer invokes time with getNextInvokeTime() method in nrPHY.m.

nrGNBPHY.m

nrGNBPHY class is the base class for physical layer of gNB nodes. It has the necessary methods for Tx request, to start PDSCH transmission (txDataRequest(obj, pdschInfo, macPDU, timingInfo)), Rx request to start PUSCH reception(rxDataRequest(obj, puschInfo, timingInfo)), Downlink control (non-data) transmission request(dlControlRequest(obj, pduType, dlControlPDU, timingInfo)), Uplink control (non-data) reception request from MAC to PHY(ulControlRequest(obj, pduType, ulControlPDU, timingInfo)), Physical layer transmission and reception(phyTx(obj, currTimingInfo) and phyRx(obj, currTimingInfo) respectively), PDSCH transmission(pdschTx(obj, currTimingInfo)), CSI-RS transmission(csirsTx(obj, currTimingInfo)), PUSCH reception(puschRx(obj, currTimingInfo)), SRS reception(srsRx(obj, currTimingInfo)). With the phTx method being called, it checks if any transmission is scheduled now. If that is the case it calls the pdschTx method. pdschTx method creates the PDSCH data using pdschData() method in nrGNBFullPHY class defined in nrGNBFullPHY.m. pdschData() method fills the slot grid with PDSCH symbols using populatePDSCH(obj, pdschInfo, macPDU) and applies OFDM modulation with nrOFDMModulate(obj, CarrierConfig, pdschGrid) methods that lies in nrGNBFullPHY.m. pdschData() finally applies Tx power and returns the PDSCH waveform. Then pdschTx() creates pdsch packets by calling the pdschPacket() method that is also in nrGNBFullPHY. This is where the fields of packets are filled. We add our time offsets for the interferer gNB node here to the “packet.StartTime” variable. pdschTx() places the packets into the Tx buffer using the addToTxBuffer(obj, packet) method in nrNode.m and clears the Tx contexts.

nrGNBFullPHY.m

nrGNBFullPHY class implements full PHY processing for gNB. It has the methods: pdschData(obj, pdschInfo, macPDU), csirsData(obj, csirsConfig), decodePUSCH(obj, puschInfoList, startTime, endTime, carrierConfigInfo), decodeSRS(obj, startTime, endTime, carrierConfigInfo), pdschPacket(obj, pdschInfoList, pdschDataList, txStartTime), csirsPacket(obj, csirsInfoList, csirsDataList, txStartTime), populatePDSCH(obj, pdschInfo, macPDU), populateCSIRS(obj, csirsInfo), puschRxProcessing(obj, rxWaveform, puschInfo, packetInfo, carrierConfigInfo, numSampleChannelDelay), srsRxProcessing(obj, , packet), applyRxGain(obj, waveformIn), applyThermalNoise(obj, waveformIn). This is followed by a similar process for CSI-RS packets.

In nrGNBPHY.m in phyTx, csirsTx(obj, currTimingInfo) is called after pdschTx(obj, currTimingInfo)is finished as it is explained above. csirsData(), populateCSIRS(), and csirsPacket() are called in a similar fashion compared to pdsch transmission. The csi-rs packet offset is added in the csirsPAcket() method to the “packet.StartTime” variable.

Then the reception process starts in the storeReception() method in nrPHY.m. It skips directToDestination packets and puts the rest into the reception buffer. Then with the phyRx() method gNB node receives the PUSCH and SRS data from UEs which is out of the scope of this internship. Finally the run() method in nrPHY.m decides the next invoke time but chooses the minimum duration for the next Tx and Rx time.

On the UE side after the run() method in WirelessNetworkSimulator.m directs to the run() method in nrNode.m which returns the next invoke time. The nextInvokeTime is the return value of the runLayers() method. After the App and MAC layer run() methods are done in runLayers(), the simulation continues with the run() method for the Physical layer which directs to the run() method in nrPHY.m. As in gNB after calculating the current AFN, slot, and symbol phyTx() method is called but for UEs, this method directs to the phyTX method in nrUEPHY.m.

nrUEPHY.m

nrUEPHY is the base class for all the physical layer types of UE nodes. The methods of interest of this class are: rxDataRequest(obj, pdschInfo, timingInfo), phyTx(obj, currTimingInfo), phyRx(obj, currTimingInfo), pdschRx(obj, currTimingInfo), csirsRx(obj, currTimingInfo).

If there is a scheduled transmission for that moment in simulation, PUSCH and SRS information is sent to the receive buffer of the relevant gNB node in the cell with the phyTx() method in nrUEPHY.m.

As in the gNB case it continues with the storeReception method in run() method in nrPHY.m which places the packets in the receive buffer except the ones with directToDestination flag with value true. Returning back to run() in nrPHY.m the physical layer reception is again done with phyRx() yet this method directs to the phyRx() method in nrUEPHY.m. This method is the parent method where we will see interference between the packets of multiple gNBs. In case of a scheduled reception phyRx() runs pdschRx() and csirsRx() methods. pdschRx(obj, currTimingInfo) method receives scheduled pdsch for the current time. It gets the context for pdsch for current time from the DataRxContext of relevant UE. Updates macPDUInfo and calculates the time window of PDSCH to be received now. Then decodes the PDSCH to extract MAC PDU(s) and crc results with the method [macPDUInfo.MACPDU, macPDUInfo.CRCFlag] = decodePDSCH(obj, pdschInfo, pktStartTime, pktEndTime, carrierConfigInfo)which points to the decodePDSCH() method in nrUEFullPHY.m.

nrUEFullPHY.m

nrUEFullPHY class is the class that implements the full physical features of UE PHY. A nrUEFullPHY object has the fields: RNTI, NCellID, DuplexMode, SubcarrierSpacing, NumResourceBlocks, NumHARQ, ChannelBandwidth, DLCarrierFrequency, ULCarrierFrequency, CSIReportConfiguration which is the same essential fields with non-full-phy object.

The methods it contains are: data = puschData(obj, puschInfo, macPDU), data = srsData(obj, srsConfig), [macPDU, crcFlag] = decodePDSCH(obj, pdschInfo, pktStartTime, pktEndTime, carrierConfigInfo), [dlRank, pmiSet, cqiRBs, precodingMatrix] = decodeCSIRS(obj, csirsConfig, pktStartTime, pktEndTime, carrierConfigInfo), txGrid = populatePUSCH(obj, puschInfo, macPDU), txGrid = populateSRS(obj, srsConfig), [macPDU, crcFlag] = pdschRxProcessing(obj, rxWaveform, pdschInfo, packetInfo, carrierConfigInfo, numSampleChannelDelay), [rank, pmiSet, cqiRBs, precodingMatrix] = csirsRxProcessing(obj, , csirsConfig, packet, carrierConfigInfo), waveformOut = applyThermalNoise(obj, waveformIn), waveformOut = applyRxGain(obj, waveformIn). The methods of interest will be briefly explained within the code flow.

decodePDSCH() method in nrUEFullPHY.m first reads the relevant packets. The relevant packet can be either a packet of interest or another packet that has been sent with the same carrier frequency. It filters the relevant packets with the help of the packetList() method in interferenceBuffer.m.

interferenceBuffer.m

InterferenceBuffer is a key class that handles the important full physical feature of inter-cell interference. For abstracted PHY, power calculations done in this class assume the same center frequency and bandwidth for all packets yet we used full phy. The methods we will mention are: bufferIdx = addPacket(obj, packet), [rxWaveform, numPackets, sampleRate] = resultantWaveform(obj, startTime, endTime, varargin), packets = packetList(obj, startTime, endTime, varargin), currentPower = receivedPacketPower(obj, currentTime, varargin), packets = retrievePacket(obj, bufferIdx), rxWaveform = combineWaveforms(obj, startTime, endTime, centerFrequency, packetIndices, sampleRate), [packetIdxList, numPackets, acprRequiredFlag] = getOverlappingPackets(obj, startTime, endTime, centerFrequency, bandwidth), removeObsoletePackets(obj, endTime).

The packetList() method in nrUEFullPHY.m returns the list of packets that are overlapping in time and/or frequency. It first validates the Name-value pair, start and end times. It sets the end time more than the minimum overlap time after the start time. Minimum overlap time is set with MinTimeOverlap property and its default value is 1e-9. The packet indices are then returned from the getOverlappingPackets() method called in packetList(). getOverlappingPackets() method returns the indices, the number of overlapping packets, and a flag that indicates the packets filtered are of the same center frequency and bandwidth. This is done by first finding the active packets at the time, and filtering the overlapping packets based on their time period and frequencies, in case there is an overlap. And returns the packet ID list. Finally in packetList() only the packets selected in getOverlappingPackets() are filtered among all the received packets at that time. Below you can see a sample of the filtered packets.

These packets are returned to decodePDSCH() in nrUEFullPHY.m. Then pdsch of interest is searched among these packets. The combined waveform received during the packet duration is calculated with the resultantWaveform(obj.RxBuffer, pktStartTime, pktStartTime+packet.Duration) in interferenceBuffer.m. resultantWaveform() collects the indices of overlapping packets with the help of getOverlappingPackets(obj, startTime, endTime, centerFrequency, bandwidth) method again in interferenceBuffer.m. After receiving the indices of overlapping packets resultantWaveform() calculates the sample rate and calls the combineWaveforms(obj, startTime, endTime, centerFrequency, packetIndices, sampleRate) within interferenceBuffer.m which handles the main part of actually combining overlapping waveforms. combineWaveforms() method initializes a waveform using the duration (hence the waveform length by multiplying duration and sample rate) and the number of antennas. Then for each packet it verifies that all packets are full physical packets, and calculates the number of overlapping samples. “Calculates the overlapping start and end index of the resultant waveform time domain samples”. “Calculates the overlapping start and end index of the interferer waveform time-domain samples”. It shifts the frequency of interfering waveform in case the center frequency does not match the required center frequency and finally combines the time-domain samples(rxWaveform(soiStartIdx:soiEndIdx, 1:nRxAnts) + interfererWaveform(:,1:nRxAnts)) and returns it to the resultantWaveform().

Processing of Returned Values

Now having the received waveform with interference decodePDSCH() method within nrUE-FullPHY.m calls the pdschRxProcessing(obj, rxWaveform, pdschInfo, packetOfInterest, carrierConfigInfo, numSampleChannelDelay) method. The main function of pdschRxProcessing() is to extract pdsch data from the waveform. First, it applies to receive gain and thermal noise on the waveform with applyRxGain(obj, rxWaveform) and applyThermalNoise(obj, rxWaveform) methods. Then it initializes slot-length waveform. Populates the received waveform at the corresponding indices in the slot-length waveform. Calculates the offset with the nrPerfectTimingEstimate(pathGains,pathFilters) method within nrPerfectTimingEstimate.m which "performs perfect timing estimation. To find the peak of the channel impulse response, the function first reconstructs the impulse response from the channel path gains array PATHGAINS and the path filter impulse response matrix PATHFILTERS. The function returns the estimated timing offset OFFSET in samples and the channel impulse response magnitude MAG.". But after input validation, the perfect time estimation is mainly done with the method called channelDelay(pathGains,pathFilters.') at the end of the nrPerfectTimingEstimate() method. channelDelay() method resides in chennaleDelay.m.

With pdschRxProcessing having the offset value it is applied to the slot-length waveform that was populated in the previous step (slotWaveform = slotWaveform(1+offset: end, :)). The next step is performing OFDM modulation on the data to recreate the resource grid. pdschRxProcessing() then calls the nrPerfectChannelEstimate(pathGains,packetInfo.Metadata.Channel.PathFilters.', carrierConfigInfo.NSizeGrid,carrierConfigInfo.SubcarrierSpacing,carrierConfigInfo.NSlot,offset, packetInfo.Metadata.Channel.SampleTimes) method that resides in nrPerfectChannelEstimate.m.

nrPerfectChannelEstimate() "performs perfect channel estimation, producing a perfect channel estimate H, by reconstructing the channel impulse response from information about the propagation channel and then performing OFDM demodulation"([4]). Which returns the estimated channel grid to pdschRxProcessing(). pdschRxProcessing() then applies precoding and equalization and calls the nrPDSCHDecode(pdschEq, pdschInfo.PDSCHConfig.Modulation, pdschInfo.PDSCHConfig.NID, pdschInfo.PDSCHConfig.RNTI, noiseEst) method.

nrPDSCHDecode() method which resides in nrPDSCHDecode.m, "returns a cell array CWS of soft bit vectors (codewords) and cell array SYMBOLS of received constellation symbol vectors resulting from performing the inverse of physical downlink shared channel (PDSCH) processing as defined in TS 38.211 Sections 7.3.1.1 - 7.3.1.3. The decoding consists of layer demapping, symbol demodulation, and descrambling."([5])(The return variable is dlschLLRs (downlink shared channel log likelihood ratios) and rxSymbols. The LLRs are scaled by CSI and lastly bit stream is converted to byte steam which is the returned variable of pdschRxProcessing(), macPDU. macPDU and crcFlag are returned to decodePDSCH() in nrUEFullPHy.m and decodePDSCH() returns them to pdschRx() in nrUEPHY.m. pdschRx() updates the macPDUInfo, MACPDU and macPDUInfo, CRCFlag values and increments the number of received packets for UE and the number of decode failures for UE (if any). Finally makes an Rx callback to MAC and clears the context. After phyRx() in nrUEPHY.m is finished with pdschRx(), it calls the csirsRx(obj, currTimingInfo) which receives the CSI-RS. csirsRx() Works essentially the same way as the pdschRx(), only with different types of packets.

Next Invoke Time and Packet Distribution

As phyRx() finishes run() function in nrPHY.m then calls the getNextInvokeTime() to decide the next invoke time of the UE node. getNextInvokeTime() does this by choosing and returning the minimum value between the next receive and transmission time. Run() finally updates the last run time of the node and returns to the runLayers() in nrNode.m with the nextInvokeTime value as the nextPHYTime. runLayers() method in nrNode.m finally decides the next invoke time by choosing the minimum value between nextAppTime, nextMACTime, and nextPHYTime and returns this value to the run() method in nrNode.m. And the run() in nrNode.m returns this next invoke time value to the wirelessNetworkSimulator script as the next invoke time of the node of the simulator object (obj.NodeNextInvokeTimes(nodeIdx) = run(obj.NodesnodeIdx, obj.CurrentTime)). In wirelessNetworkSimulator, every gNB and UE node is run in this fashion in a loop. When the loop is complete and every node that is scheduled to run in the current time is run, the recently run nodes are updated. And the packets are sent to the recently run nodes with the distributePackets() method in wirelessNetworkSimulator.m.

In distributePackets() the transmitted data is directed from the transmission buffer to the reception buffer of receiving nodes if the packet is relevant to the receiving node. If the directToDestionation flag is true then there is no channel applied and the packets are sent out-of-band. If not, the channel is applied to support packet drops in the channel. In our case, we used our custom channel model instead of the default free-path-loss model. Finally, the next invoke times of the nodes that received data are updated to the current time so they can have immediate reception when they go back into the loop in wirelessNetworkSimulator.m.

In wirelessNetworkSimulator.m the next iteration time is calculated and the simulation time is incremented to that point in time in the main loop and the simulation continues until the currentTime variable is incremented to the simDuration variable value.

3.2 Results

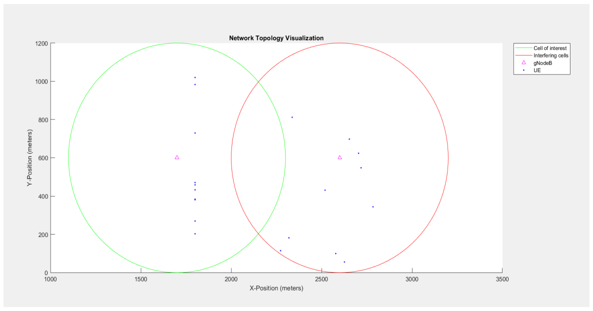
We have looked into the time and phase synchronization requirements for LTE-TDD. In the Recommendation [2] it is explained that in TDD systems snyhronization error in one base station can negatively affect the performance of not only neighboring base stations of the same operator but also the base stations that belongs to different operators due to interference it creates.

In this internship we aimed to create a platform to analyze this phenomena. We chose the 5G tool of MathWorks (https://de.mathworks.com/help/5g/index.html).

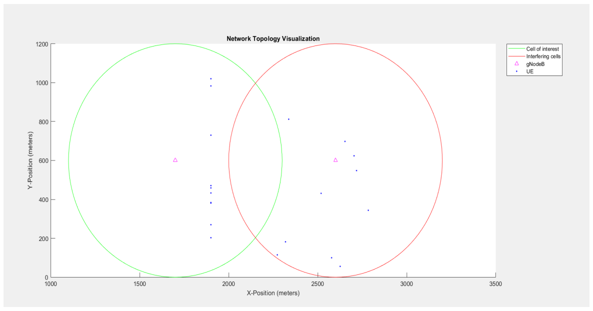
First we validated the framework by doing various tests. These tests include analyzing the results of using different: Positions of the UEs of the cell of interest, Number of UEs per cell, Subcarrier spacing, Transmit powers.

Impact of UE Positions:

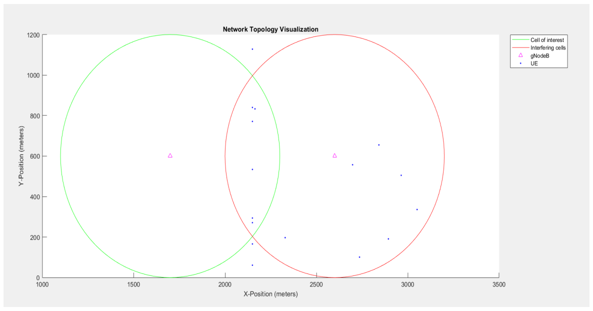
The cartesian position ([x,y,z]) of gNBs are [1700 600 0] and [2600 600 0]. We used 3 different x positions for the interfered UEs and kept the y,z positions fixed. The x-axis positions used are 1800, 1900, 2150 and 2200. So we observed how the distance to interfered and interfering base stations affect the total throughput obtained from UEs in the cell of interest.



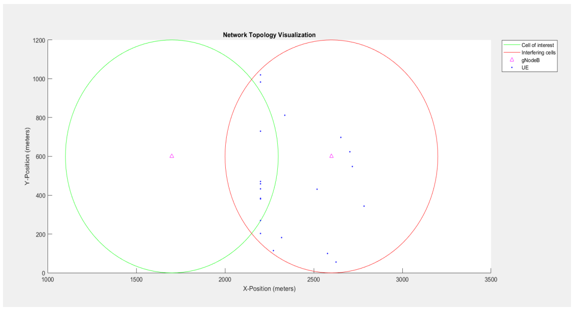
(a) UE x position: 1800



(b) UE x position: 1900



(c) UE x position: 2150



(d) UE x position: 2200

The gNB configuration (with 10 UE per cell) we used is as follows:

```
nrGNB(Position=gNBPositions,DuplexMode="TDD",CarrierFrequency
   =2.5e9,ChannelBandwidth=5e6,SubcarrierSpacing=30e3,
   PHYAbstractionMethod=phyAbstractionType,TransmitPower=18,
   ReceiveGain=11);
```

The Cartesian coordinates of gNBs are [1700 600 0](Cell of interest) and [2600 600 0](interfering Cell) with a coverage radius of 600. For the UE positions vary in y coordinates but we altered the x coordinates from the coordinate set [1800,1900,2150,2200]. As the UEs get further away from the gNB of interest and closer to the interfering cell, we observe a much lower achieved throughput and higher bit error rate. We used 10 frames as simulation duration for this part.

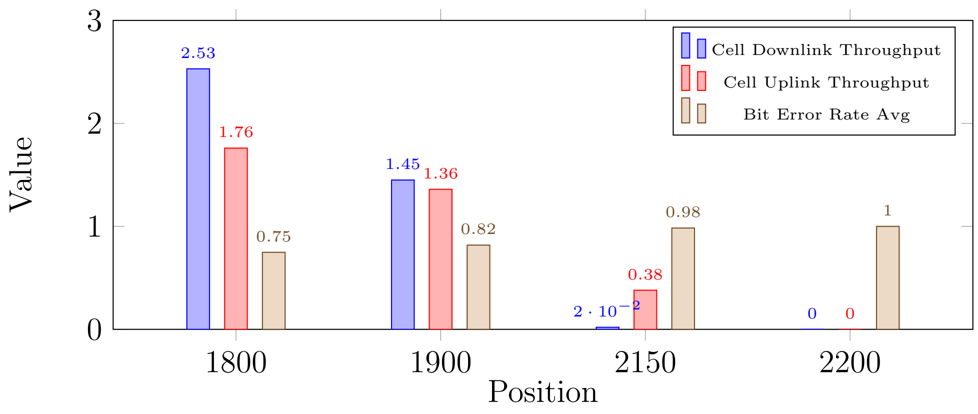


Figure 3.2: Performance Metrics vs. Position

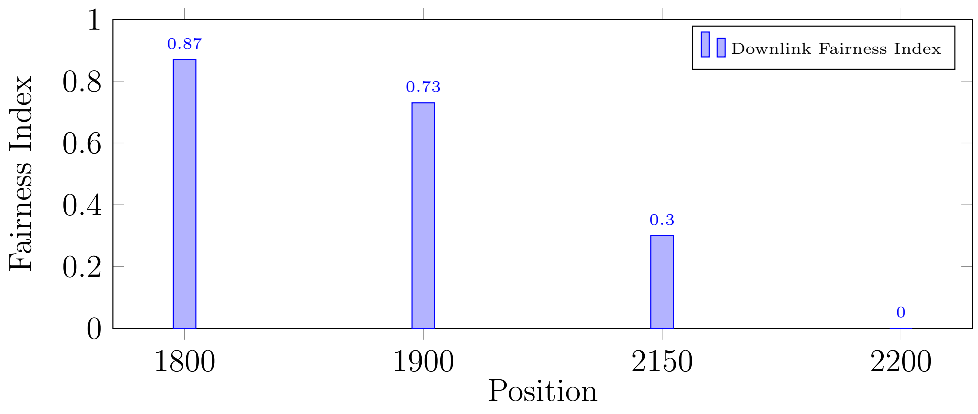
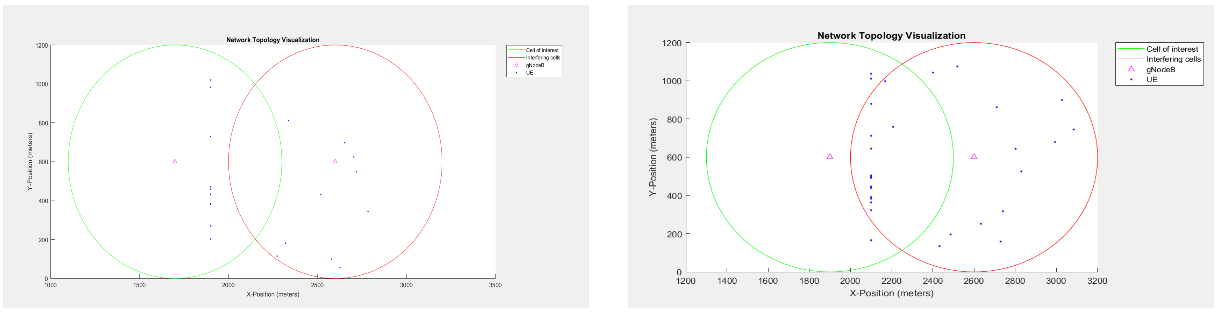


Figure 3.3: Fairness Index of Cell of Interest Downlink with different UE positioning

Impact of gNB Positions:

Next, we compare two cases in which, in both cases, the x-coordinates of UEs of the cell of interest remain the same relative to the gNB of the relevant cell but gNBs are closer to each other. For this test, the cell of interest is moved 200 in the positive x direction. So the coordinates of the gNB of interest is [1900 600 0] and UEs are set on 2100 position on x-axis. So relative distances of UEs to the gNB in the cell remain same compared to the case where gNB is at [1700 600 0] and UEs are at 1900 on x-axis(3.4). As it can be expected, while achieving less throughput, we observe a higher bit error rate due to increased interference(3.5). The simulation duration is 10 frames with 15 UEs per cell so we are comparing the results with 15 UE case in the figure 3.10. The gNB configuration is as follows:

```
nrGNB(Position=gNBPositions,DuplexMode="TDD",CarrierFrequency
    =2.5e9,ChannelBandwidth=10e6,SubcarrierSpacing=30e3,
    PHYAbstractionMethod=phyAbstractionType,TransmitPower=18,
    ReceiveGain=11);
```



(a) gNB x position: 1700 (b) gNB x position: 1900

Figure 3.4: Two different gNB positions with same relative UE positions in the cell

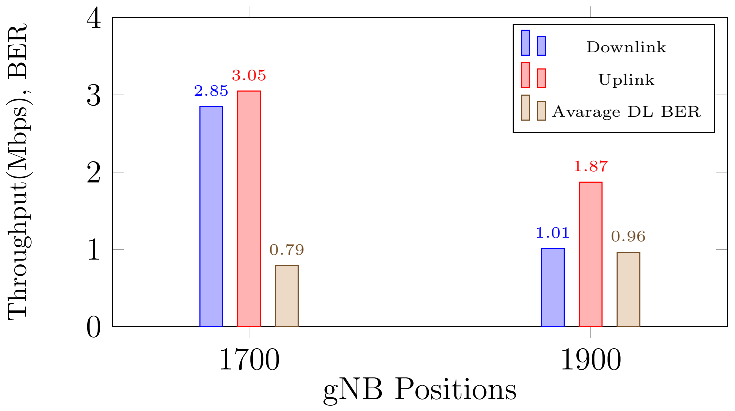


Figure 3.5: Cell performance with different gNB positions

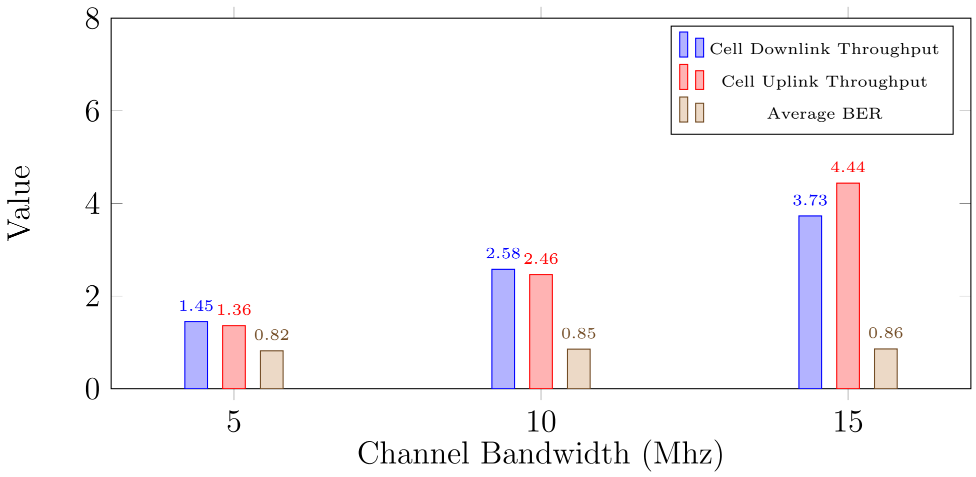


Figure 3.6: Cell of interest performance with different bandwidths (Both cell of interest and interfering cell bandwidth)

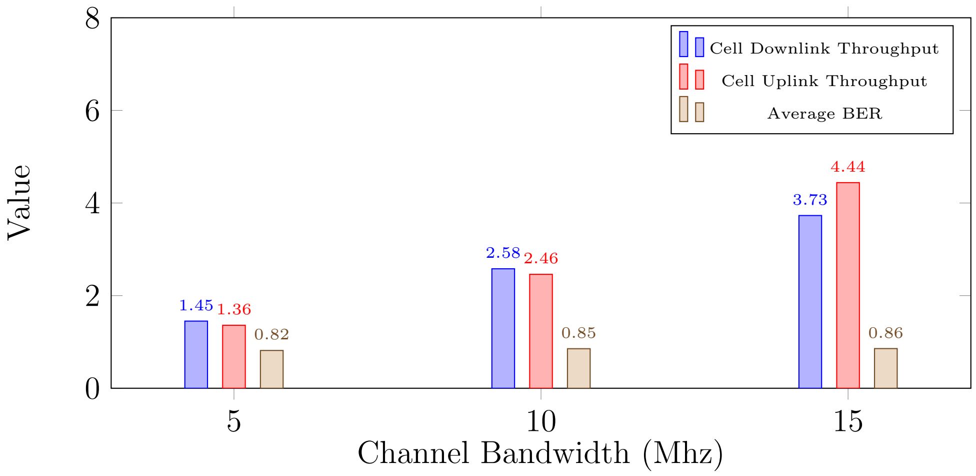


Figure 3.7: Cell of interest performance with different interfering cell bandwidths

Impact of Bandwidth:

When we use 5, 10, and 15 MHz channel bandwidths for both cells with the x-axis position 1900(3.6), with increased bandwidth, both Downlink and Uplink throughputs increase as more resources are available to the users while there is not a significant change in bit error rate.

When the same changes are applied to the interfering cell channel bandwidth only, we do not see any difference in the cell of interest performance(3.7).

Impact of Transmit Power:

When we look into the impact of the changes in the transmit power of the cells that vary between 15 to 35 dBm, firstly again, changed transmit power of both cells (fig. 3.8) we do not observe a consistent behavior with the increased transmit power. Also, the effect of transmit power on bit error rate can be considered negligible. The UEs of cell of interest are on 1900 on x-axis with 5 frame simulation duration.

And for changing the transmit power of only(transmit power of cell of interest fixed to 18 dBm) the interfering cell has no effect on the throughput despite the UEs of the cell of interest being within the radius of the interfering cell. For this test 2150 value for x-axis position for the UEs of the cell of interest is used. This is for testing if the UE position being closer to interfering cell makes the UEs more prone to interference with increased interfering cell transmit power. We used 10 frames for Simulatio duration.f the interfering cell. Transmit powers we use (fig. 3.9) .

For both part the cell configuration was: 5Mhz bandwidth, 30khz SCS with 10 UE per cell.

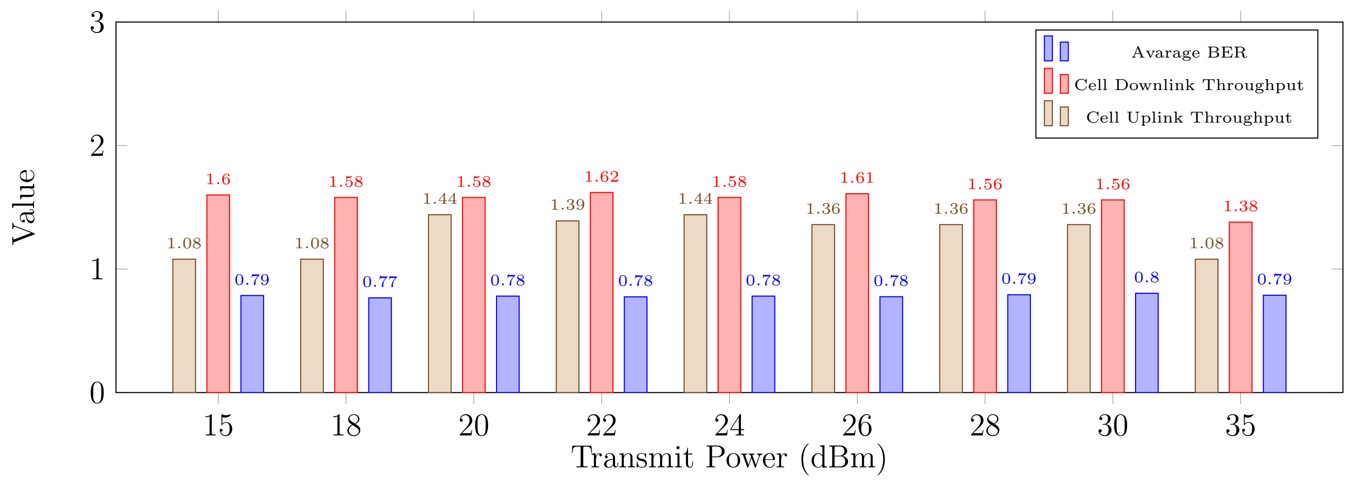


Figure 3.8: Cell of interest performance with different Cell Transmit Power (Both cell of interest and interfering cell transmit power

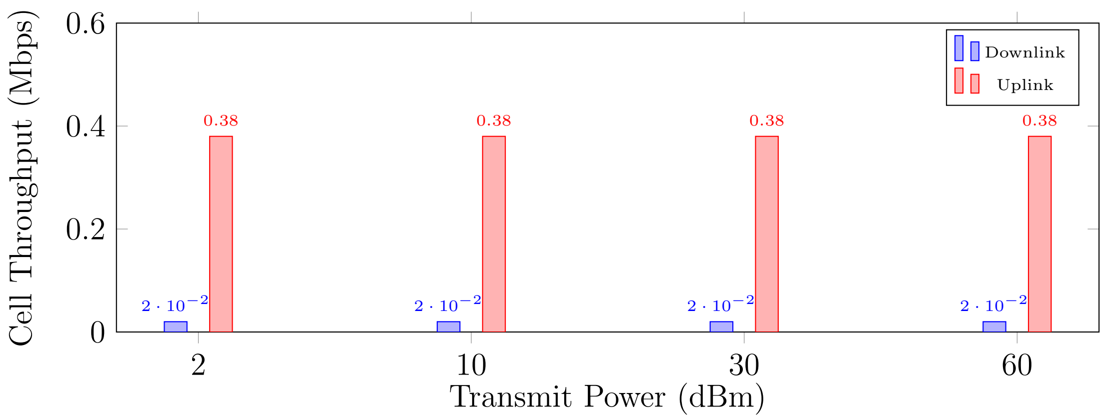


Figure 3.9: Cell of interest performance with different interfering Cell Transmit Power

Impact of number of UEs per Cell:

Next we vary the number of UEs per cell to see the correlation between the throughput and number of UEs per cell. 1900 on x-axis is selected as UE positions. But it should be noted that the interference created by UEs is out of the scope of this internship. We used 10 frames as simulation duration. In figure 3.10 we observe an increase in achieved cell throughput up to 15 UEs per cell and then it starts to decline again. We achieve less throughput using very few UEs due to unused resources. But after a certain number of UEs, the resources are no longer sufficient hence we observe a decrease in throughput. We do not see a consistent correlation between number of UEs per cell and the resulted Jains fairness index. The gNB configuration is as follows:

```
nrGNB(Position=gNBPositions,DuplexMode="TDD",CarrierFrequency
   =2.5e9,ChannelBandwidth=10e6,SubcarrierSpacing=30e3,
   PHYAbstractionMethod=phyAbstractionType,TransmitPower=18,
   ReceiveGain=11);
```

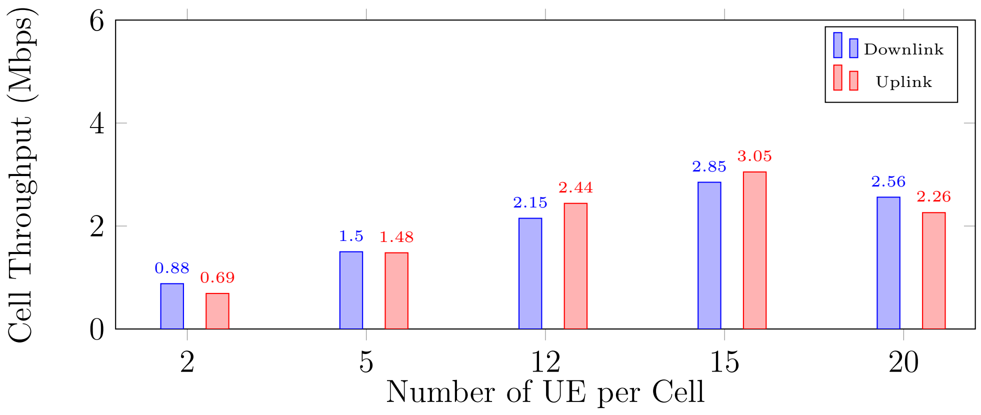


Figure 3.10: Cell Throughput with Different Numbers of UEs per Cell

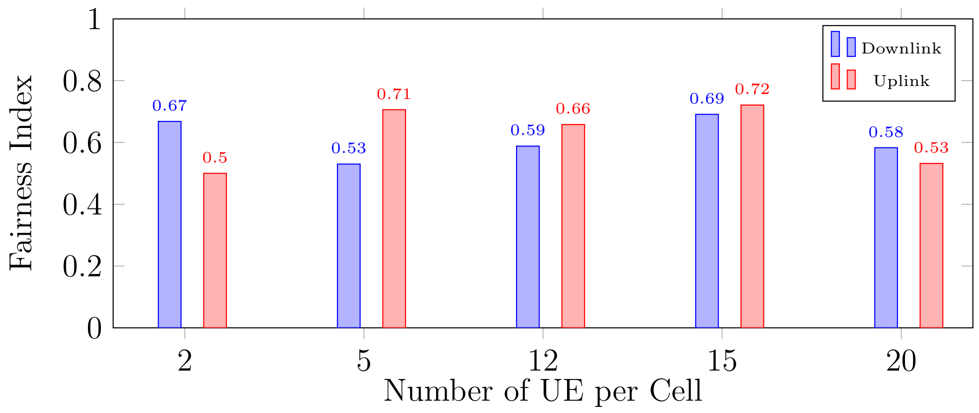


Figure 3.11: Fairness Index of Cell of Interest for Uplink and Downlink with different number of UE per cell

Impact of Subcarriers Spacing:

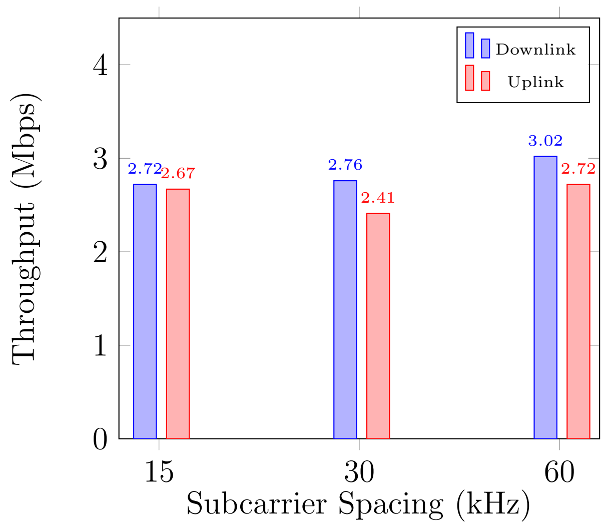
For this we used the gNB setup (with different SubcarrierSpacing values):

```
nrGNB(Position=gNBPositions,DuplexMode="TDD",CarrierFrequency
    =2.5e9,ChannelBandwidth=10e6,SubcarrierSpacing=15e3,
    PHYAbstractionMethod=phyAbstractionType,TransmitPower=18,
    ReceiveGain=11);
```

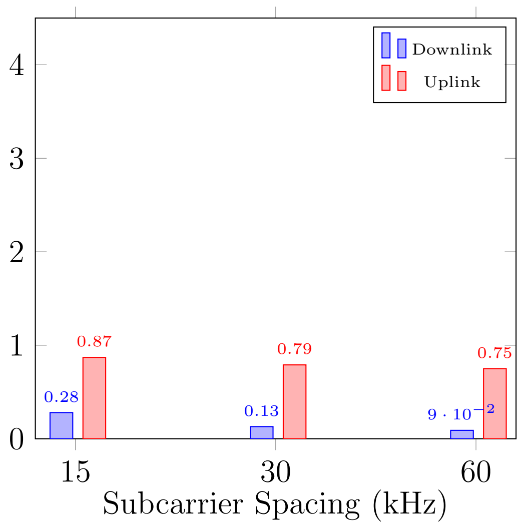
The larger subcarrier spacing results in lower latency and supports higher-frequency bands, along with a shorter symbol duration, enabling faster data transmission. However, a bigger subcarrier spacing is also more likely to have more interference. We used two different position. 1900 and 2100 with different interference levels. This for observing if bigger SCS have varying impacts on throughput depending on the interference level. We use 15, 30, and 60 kHz with 10 MHz channel bandwidth. The simulation duration was 5 frames with 10 UEs per cell.

At UE x-axis position 1900, there is little interference. So as expected, with increased subcarrier spacing, we observe a slight increase in downlink throughput but uplink throughput fluctuated(3.12a).

At UE x-axis position 2100, Both uplink and downlink throughputs decrease with the increasing subcarrier spacing due to increased interference level(3.12b).



(a) Position 1900



(b) Position 2100

Figure 3.12: Uplink and Downlink Throughput with various SCS in x-coordinates 1900 and 2100

Impact of Time Synchronization:

After the initial framework validation measurements we investigated the impact of the time offset of interfering cell. As it is discussed in [2] LTE TDD systems requires a time synchronization of 5μs. We first applied 1μs, 5μs, 10μs, 100μs and 150μs offset to both PDSCH and CSI-RS packets of the interfering cell. Later we applied the same offsets to only PDSCH and CSI-RS packets individually. We used the 1900 x-axis position and gNB configuration is as follows:

```
nrGNB(Position=gNBPositions,DuplexMode="TDD",CarrierFrequency
    =2.5e9,ChannelBandwidth=5e6,SubcarrierSpacing=30e3,
    PHYAbstractionMethod=phyAbstractionType,TransmitPower=18,
    ReceiveGain=11);
```

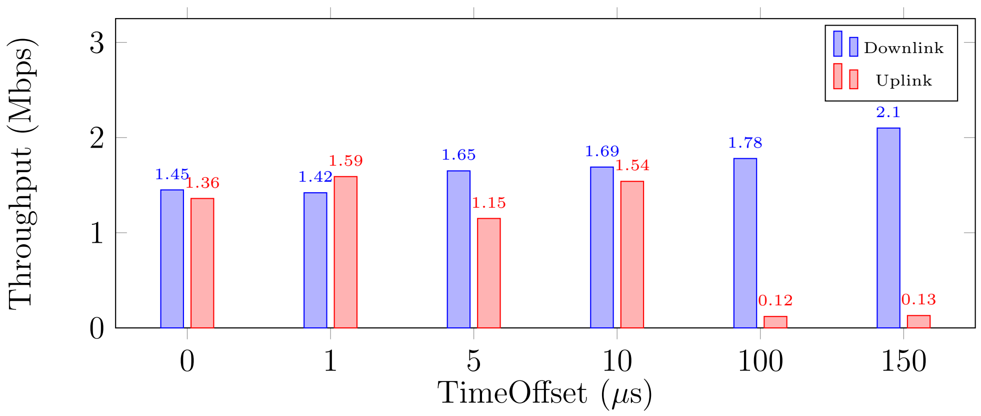


Figure 3.13: Cell of interest Throughput with different interfering Cell time offsets for both PDSCH and CSI-RS packets

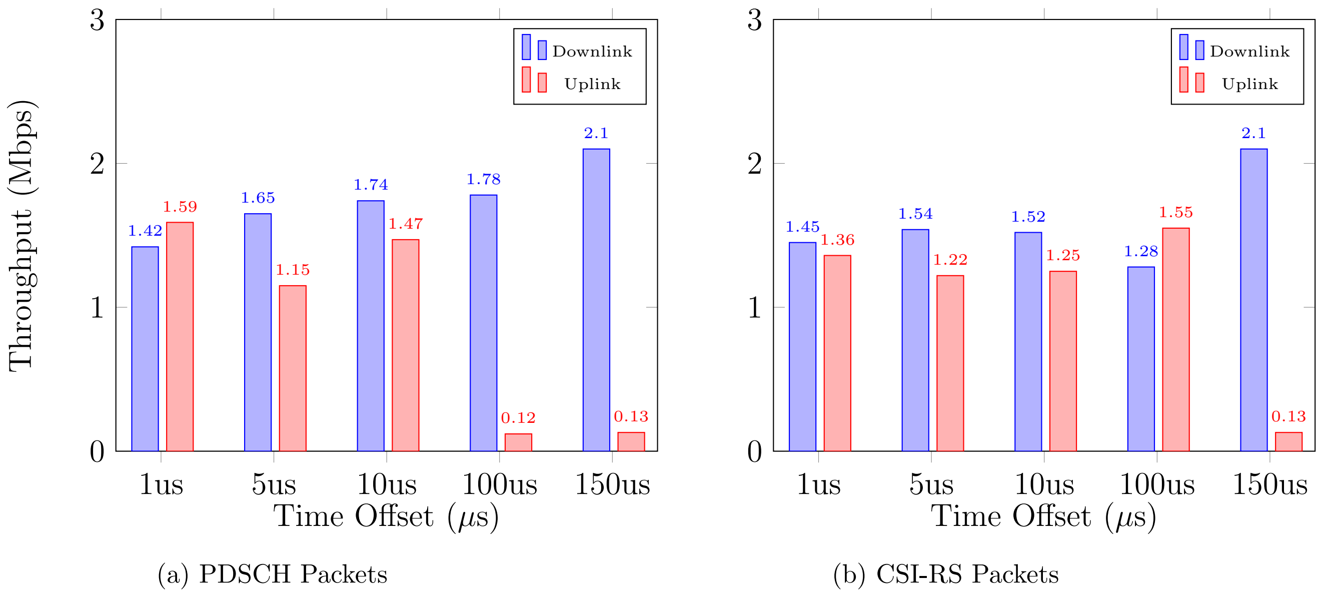


Figure 3.14: Cell of interest Throughput with different interfering Cell time offsets for PDSCH and CSI-RS packets

Offset for Both PDSCH and CSI-RS Packets:

With increased time offset, we do not observe a consistent decrease in throughput as expected(3.13). Only beyond 100 µs offset, the uplink throughput decreases but this offset for TDD systems in 5G is not practical since it is much beyond the requirements mentioned in [2].

Offset for PDSCH Packets:

For the time offset applied on PDSCH packets, we do not see any effect on cell performance except the 10 µs time offset(3.14a).

Offset for CSI-RS Packets:

We do not collect a consistent result with the time offsets applied on CSI-RS packets as the achieved cell throughput fluctuate and despite the time offset we achieved higher throughput compared to no offset apllied. One noticeable result is the drastic increase in uplink throughput with 100 µs offset but as it is mentioned this is not an applicable offset in the context of 5G TDD(3.14b).

Chapter 4

Conclusions and Outlook

Time and phase synchronization in communication technologies are required for various applications. It supports alarm and billing functions, monitors delays in IP networks, in the case of MBMS with a single-frequency setup, strict phase synchronization is needed to deliver the same content to each station etc.

In 5G NR TDD systems, synchronization is essential not only for intra-operator but also inter-operators. Synchronization errors in a single cell can cause interference to neighboring cells regardless of the operator, as the carrier frequencies overlap.

While the requirements of time synchronization are clearly laid out by [2], the exact effect of different time offsets on 5G NR TDD at the application layer remains an open question in this field. Hence, we aim to create a framework to enable a practical analysis of time and phase synchronization in TDD systems. We create this framework based on the 5G toolbox from Mathworks ([3]). We validate this toolbox by setting up a number of tests on a simple network topology that consists of two interfering neighboring cells. One cell takes the role of the interfered cell, and the second one is the interfering cell. Both operate on the same frequency. We compare the throughputs and bit error rates of the cell of interest using different network parameters. We observe the impact of the UE Positions, gNB Positions, number of UEs per cell, channel bandwidths, sub-carrier spacing, and transmit power of cells.

After validating the basic properties of the network, we analyze the interference behavior with the time offsets applied to PDSCH and/or CSI-RS packets of the interfering cell. The expected result was, the bigger offset apllied to the interfering cell packet the less achieved throughput due to increased BS2BS interference. But we get inconsistent results. This may be due to lack of UE2UE implementation since UL data transmission of UEs can also interfere with the DL transmission of neighboring base station. The purpose of this internship was to create a base environment for 5G application testing and a further study can be the addition of UE2UE interference implementation and to have a deeper look into different channel types used in the simulation.

Appendix A

A.1 5G toolbox Implementation Structure

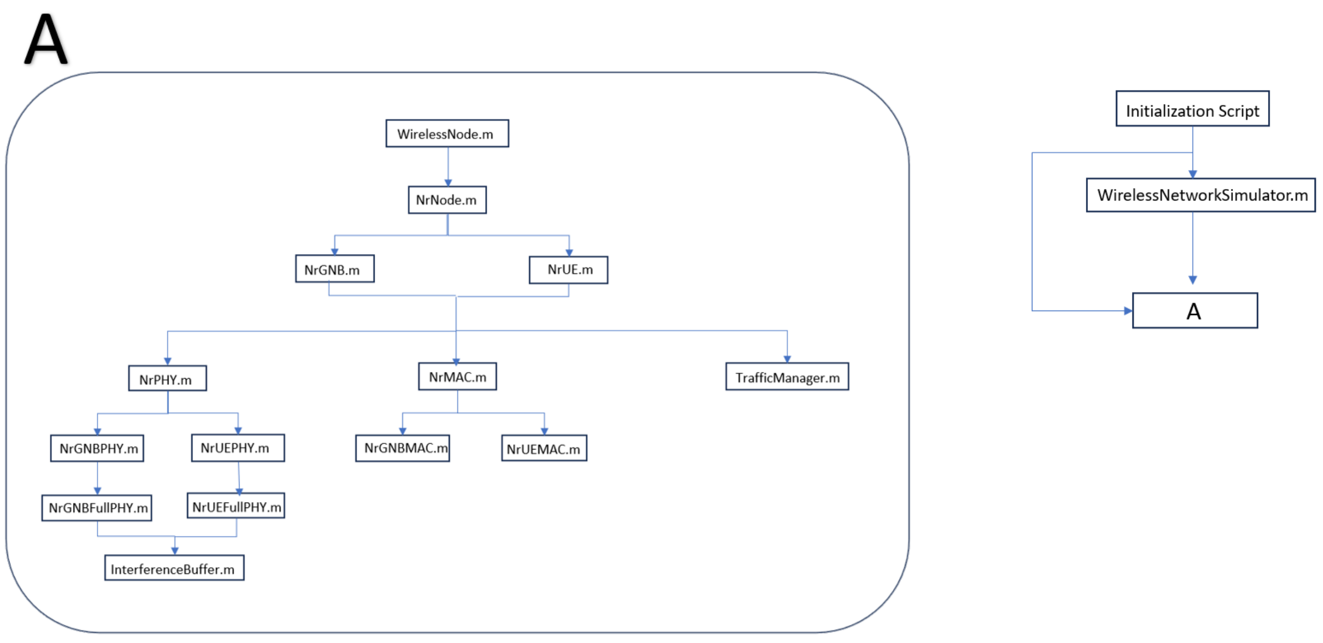


Figure A.1: Implementation Code Hierarchy [3]

A.2 Jain‘s Fairness Index Python Code

```python
downlink_Uplink_Position = "downlink with"
downlink2 = "0.75-0.13"
downlink5= "0.81-0.42-0.09-0.12-0.07"
downlink12= "0.02-0.16-0.08-0.21-0.05-0.1-0.02-0.3-0.39-0.36-0.02-0.44"
downlink15= "0.22-0.15-0.02-0.06-0.02-0.12-0.21-0.28-0.27-0.33-0.03-0.41-0.38-0.26-0.09"
downlink20= "0.02-0.2-0.02-0.03-0.05-0.04-0.01-0.24-0.18-0.23-0.01-0.24-0.36-0.15-0.19-0.01-0.1-0.19-0.01-0.3"
Uplink2= "0-0.69"
Uplink5="0.51-0.39-0.44-0.12-0.02"
Uplink12="0.05-0.38-0.22-0.26-0.02-0.3-0-0.36-0.12-0.24-0.05-0.45"
Uplink15="0.17-0.26-0.12-0.1-0.05-0.23-0.19-0.35-0.09-0.23-0-0.4-0.42-0.33-0.09"
Uplink20="0.02-0.19-0.14-0.12-0.02-0.05-0-0.33-0-0.18-0-0.23-0.28-0.21-0.05-0.02-0.02-0.1-0.02-0.26"
Pos1800="0.32-0.2-0.05-0.32-0.16-0.35-0.16-0.35-0.3-0.32"
Pos1900="0.19-0.11-0.03-0.17-0.05-0.28-0.02-0.12-0.21-0.25"
Pos2150="0-0-0.01-0.01-0-0-0.01-0-0-0"
downlinkList =[downlink2,downlink5,downlink12,downlink15,downlink20]
UplinkList=[Uplink2,Uplink5,Uplink12,Uplink15,Uplink20]
PositionList=[Pos1800,Pos1900,Pos2150]

def jains_fairness_index(throughputs):
    N = len(throughputs)
    if N == 0:
        return 0  # Avoid division by zero for empty lists
    num_users = len(throughputs)
    # Calculate the numerator (sum of throughputs)^2
    numerator = sum(throughputs)**2
    # Calculate the denominator (N * sum of squares of throughputs)
    denominator = num_users * sum(x**2 for x in throughputs)
    # Calculate the fairness index
    fairness_index = numerator / denominator if denominator != 0 else 0

    return fairness_index

Lists=[downlinkList,UplinkList,PositionList]
for list in Lists:

    if Lists.index(list)==1:
        downlink_Uplink_Position = "Uplink with"
    elif Lists.index(list)==2:
        downlink_Uplink_Position = "position"

    for x in list:
        Sum = 0
        SumofSquares = 0
        SquareofSums = 0
        number_strings = x.split('-')

    # Convert the list of strings to a list of floats (numbers)
        numbersArray = [float(num) for num in number_strings]
        JainFairnessIndex=jains_fairness_index(numbersArray)
        if downlink_Uplink_Position == "position":
            if list.index(x)==0:
                print(f'{downlink_Uplink_Position} 1800: {"{:.3f}".format(JainFairnessIndex)}')
            elif list.index(x)==1:
                print(f'{downlink_Uplink_Position} 1900: {"{:.3f}".format(JainFairnessIndex)}')
            elif list.index(x)==2:
                print(f'{downlink_Uplink_Position} 2150: {"{:.3f}".format(JainFairnessIndex)}')

        else:
            print(f'{downlink_Uplink_Position} {len(number_strings)} UE: {"{:.3f}".format(JainFairnessIndex)}')
```

A.3 Notation and Abbreviation

This chapter contains tables where all abbreviations and other notations like mathematical placeholders used in the thesis are listed.

AP	Access Point
CQI	Channel Quality Indicator
DCI	Downlink Control Information
D-SR	Dedicated Scheduling Request
D2D	device to device
eNodeB	evolved Node B or E-UTRAN Node B
FDD	Frequency Division Duplexing
H-ARQ	Hybrid-Automatic Repeat Request
IoT	Internet of Things
LTE	Long Term Evolution
MBMS	Multimedia Broadcast Multicast Services
MCS	Modulation and Coding Scheme
OFDM	Orthogonal Frequency Division Multiplexing
PDCCH	Physical Downlink Control Channel
PDSCH	Physical Downlink Shared Channel
PRB	Physical Resource Block
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
RACH	Random Access Channel
SC-FDMA	Single Carrier Frequency Division Multiple Access
SR	Scheduling Request
SRS	Sounding Reference Signal
TDD	Time Division Duplexing
UE	User Equipment

Bibliography

[1] 3rd Generation Partnership Project (3GPP). TS 25.346: Introduction of the Multimedia Broadcast/Multicast Service (MBMS) in the Radio Access Network (RAN); Stage 2. Technical Specification, 2018. Release 15.

[2] ITU-T. Recommendation G.8271/Y.1366: Time and Phase Synchronization Aspects of Telecommunication Networks. ITU-T Recommendation G.8271/Y.1366, November 2018.

[3] MathWorks. 5G Toolbox Documentation, 2023. Accessed: November 23, 2023. URL: https://de.mathworks.com/help/5g/.

[4] MathWorks. 5G Toolbox Documentation, 2023. Accessed: November 23, 2023. URL: https://de.mathworks.com/help/5g/ref/nrperfectchannelestimate.html.

[5] MathWorks. 5G Toolbox Documentation, 2023. Accessed: November 23, 2023. URL: https://de.mathworks.com/help/5g/ref/nrpdschdecode.html.