Computer Networks

Topic : Blockchain for 5G Technology

Team Members:

Ayush Rathor CS21B1004
Parth Goel CS21B1018

Introduction

In the ever-evolving landscape of computer networks, the integration of cutting-edge technologies is crucial to meet the increasing demands for speed, reliability, and security. One such paradigm-shifting technology is 5G, which promises to revolutionize the way we connect and communicate. Simultaneously, blockchain technology has gained prominence for its decentralized and secure nature.

This collaborative report explores the intersection of these two transformative technologies, delving into the potential of leveraging blockchain in the realm of 5G technology. Ayush and Parth, members of our research team, aim to unravel the synergies and challenges that arise when combining blockchain and 5G in the context of computer networks.

Trailblazing Objectives:

1. Understanding 5G Technology: We will provide an overview of the key features and capabilities of 5G technology, highlighting its significance in the current digital era.

2. Exploring Blockchain Technology: A comprehensive examination of blockchain technology, focusing on its decentralized architecture, immutability, and cryptographic principles that make it a compelling solution for various applications.

3. Integration Possibilities: Investigating the potential integration points of blockchain within 5G networks. This includes examining how blockchain can enhance the security, efficiency, and trustworthiness of 5G infrastructure.

4. Challenges and Considerations: Identifying potential challenges and considerations when implementing blockchain in the 5G ecosystem. This involves addressing issues related to scalability, latency, and interoperability.

5. Use Cases and Applications: Highlighting real-world use cases and applications where the fusion of blockchain and 5G can bring about transformative changes. This section will illustrate the practical implications of our research.

Issues and Problem Statements

Issues Related[1]:

1. Scalability : The scalability of blockchain networks faces a significant hurdle in meeting the high throughput demands of 5G networks, necessitating a robust solution for handling a large number of transactions per second.

2. Smart Contract Security : Smart contracts, integral to blockchain networks, pose security risks due to vulnerabilities and bugs. Enhancing the security and integrity of smart contract code is paramount to prevent unauthorized access and potential exploitation.

3. Standardization and Regulations : The absence of standardized frameworks and regulations for blockchain and smart contracts at national and international levels poses a hurdle. Establishing regulatory guidelines is essential to ensure interoperability, compliance, and legal validity in the 5G ecosystem.

4. Transaction and Cloud Infrastructure Costs : Building and maintaining blockchain networks entail substantial costs, particularly in terms of cloud infrastructure and transaction fees. Optimizing these costs is crucial to make the integration of blockchain economically viable for 5G networks.

5. Data Privacy : The immutability of blockchain raises concerns regarding data privacy, especially in the context of regulations like GDPR. Balancing the transparency and immutability of blockchain with the privacy requirements of 5G networks is a complex challenge.

6. Interoperability : Achieving seamless interoperability among diverse blockchain platforms and 5G technologies is critical. Developing standardized protocols and interfaces is necessary for efficient communication and collaboration in the integrated ecosystem.

7. Naming, Registration, and Reputation : Managing participants and entities in the decentralized blockchain and 5G ecosystem requires efficient decentralized registration systems, identity management, and reputation systems. Designing scalable systems for naming, registration, and reputation is essential for trust and accountability.

Problem Statements[1]:

1. Scaling Blockchain for 5G : How can blockchain networks be scaled to meet the high throughput requirements of 5G networks, considering the need for low latency and high data rates?

2. Securing Smart Contracts : How can the security of smart contracts deployed on blockchain networks be enhanced to prevent vulnerabilities and unauthorized access?

3. Establishing Regulations : What measures can be taken to establish standardization and regulations for blockchain and smart contracts at national and international levels, ensuring interoperability and legal validity?

4. Optimizing Costs : How can the costs associated with transaction fees and cloud infrastructure be optimized to make blockchain integration economically viable for 5G networks?

5. Addressing Data Privacy : How can data privacy concerns be addressed when storing sensitive customer information on a blockchain, while still maintaining the transparency and immutability of the technology?

6. Enabling Interoperability : What protocols and interfaces need to be developed to enable seamless interoperability between different blockchain platforms and 5G technologies?

7. Designing Decentralized Systems : How can decentralized registration systems, identity management, and reputation systems be designed to effectively manage participants and entities in the blockchain and 5G ecosystem, ensuring trust and accountability?

Proposed Solutions for Problem Statements[1]

1. Scaling Blockchain for 5G : Researchers propose exploring novel blockchain architectures, consensus algorithms, and sharding techniques to address scalability challenges. These approaches aim to increase transaction throughput, enabling blockchain networks to meet the high demands of 5G.

2. Securing Smart Contracts : Efforts are directed towards developing secure coding practices and auditing tools for smart contracts. Bug bounty programs and standardized vulnerability reporting mechanisms are additional measures to enhance security and encourage ethical reporting.

3. Establishing Regulations : Standardization efforts are deemed crucial for ensuring interoperability and compliance. Collaboration among telecom standard groups and independent bodies is recommended to establish standardized frameworks and regulations for blockchain and smart contracts in the 5G ecosystem.

4. Optimizing Costs : Optimization of transaction fees and cloud infrastructure costs involves the development of efficient consensus algorithms and exploring off-chain solutions for specific transactions. Considering alternative blockchain platforms with lower transaction costs is also suggested.

5. Addressing Data Privacy : Proposed solutions for data privacy concerns include the adoption of privacy-enhancing techniques such as zero-knowledge proofs, homomorphic encryption, and differential privacy. These techniques enable secure storage and processing of sensitive information while preserving blockchain transparency and immutability.

6. Enabling Interoperability : Interoperability between different blockchain platforms and 5G technologies can be achieved through the development of standardized protocols, interfaces, and data formats. Collaborative efforts among stakeholders can establish interoperability

standards for seamless communication and collaboration.

7. Decentralized Systems : Decentralized registration systems, identity management, and reputation systems are proposed to be developed using blockchain technology. These systems should prioritize scalability, efficiency, and trustworthiness to ensure proper authentication, accountability, and reputation management in the blockchain and 5G ecosystem.

8. Trusted Oracles : To ensure oracle trustworthiness, decentralized mechanisms for reporting, consensus, and reputation are suggested. Multiple oracles reporting external data, validated by smart contracts, can enhance the verification of trustworthiness.

9. Infrastructure Crowdsourcing : Blockchain and smart contracts offer practical solutions for decentralized tower registration, resource management, and automated billing. The implementation of smart contracts with Service Level Agreement (SLA) terms, coupled with penalties and incentives, can enforce honest behaviors in infrastructure crowdsourcing.

10. Standardization and Regulations (Repeated) : Massive adoption of blockchain in the 5G and telecom industry necessitates the enactment of local and international standardization, regulation, and governance. Standardization efforts within telecom standard groups can target specific blockchain applications or address broader aspects of blockchain integration in 5G networks.

Challenges Related to Smart Contracts in the 5G Ecosystem[1]

Proliferation of Smart Contracts:

To date, the public Ethereum blockchain boasts approximately 10 million smart contracts. The challenge arises in transforming this massive number of contracts into smart contracts compatible with the intricate landscape of the 5G ecosystem. This task becomes particularly daunting considering the high granularity of IoT devices that will permeate a typical 5G network.

Legal Ambiguities:

The legality of deployed smart contracts introduces another layer of complexity. The legal status of smart contracts hinges on the existence of a binding contract and the associated jurisdiction. Navigating the legal landscape becomes crucial in ensuring the seamless integration of smart contracts into the 5G ecosystem.

Security Concerns:

Security emerges as a critical challenge in the realm of smart contracts. The code governing smart contracts may harbor bugs and vulnerabilities, creating a potential playground for hackers seeking exploitation. Developing secure and vulnerability-free code for smart contracts becomes a

paramount task, with 5G smart contracts being no exception.

Immutability Dilemma:

A unique challenge stems from the inherent design of smart contracts – they are not patchable or upgradable. Once a smart contract is uploaded and activated, it becomes immutable. This lack of flexibility poses challenges in addressing vulnerabilities or bugs discovered post-deployment. Novel approaches are needed to devise ways for upgrading smart contracts to rectify previous issues.

Ethical Reporting and Standardization:

To fortify the security of smart contracts, there is a need for ethical reporting of bugs and vulnerabilities. Establishing standardized procedures for reporting and addressing these issues becomes imperative in fostering a collaborative and ethical environment within the 5G ecosystem.

A Simple Smart Contract containing Vulnerabilities

```
contract smart_contract {
      mapping(address => uint)
public balances;
      address public owner;

      constructor() {
      owner = msg.sender;
      }

      function deposit() public
payable {
      balances[msg.sender] +=
msg.value;
      }


      function withdraw(uint
amount) public {
      require(balances[msg.sender]
>= amount, "Insufficient
balance");
      balances[msg.sender] -=
amount;
      (bool success, ) =
msg.sender.call{value:
amount}("");
      require(success, "Transfer
failed");
      }


      function
transferOwnership(address
newOwner) public {
      require(msg.sender == owner,
"Only the owner can transfer
ownership");
      owner = newOwner;
      }

      // This function is
vulnerable to integer overflow
      function
vulnerableFunction(uint256 amount)
public returns (bool) {
      balances[msg.sender] +=
amount;
      return true;
      }
```

```
      // This function has a
missing input validation check
      function
uncheckedInput(uint256 amount)
public {
      balances[msg.sender] +=
amount;
      }

      // This function has a
timestamp dependency
      function
timestampDependency(uint256
deadline) public view returns
(bool) {
      return block.timestamp <
deadline;
      }
}
```

Smart Contract Vulnerabilities Analysis

1. Integer Overflow:

The function **`vulnerableFunction`** in the smart contract is susceptible to integer overflow. If the sum of **`balances[msg.sender] + amount`** exceeds the maximum representable value for **`uint`**, an overflow may occur, leading to unexpected behavior. Mitigation involves incorporating safe arithmetic operations using libraries like SafeMath or implementing explicit require statements to check for overflows.

2. Unchecked Input:

The function **`uncheckedInput`** lacks input validation checks, making it susceptible to potential exploits if an attacker passes a large value for **`amount`**. This oversight could result in an integer overflow or other unforeseen issues. To address this, always validate and sanitize inputs to prevent unexpected behavior.

3. Reentrancy Attack:

The **`withdraw`** function allows users to withdraw funds and subsequently calls **`msg.sender.call{value: amount}(" ")`**. This pattern can expose the contract to a reentrancy attack, where an external contract repeatedly calls the **`withdraw`** function before the state is updated, draining the contract's funds. Employing the "ReentrancyGuard" pattern or adhering to the "Checks-Effects-Interactions" pattern can help mitigate reentrancy issues.

4. Timestamp Dependency:

The **`timestampDependency`** function relies on **`block.timestamp`**, which may be susceptible to manipulation by miners. This introduces a vulnerability, especially in scenarios requiring precise timing, such as token sales or auctions. Mitigation involves considering alternative mechanisms, such as block numbers, to reduce the impact of potential timestamp manipulation.

5. Access Control:

The **`transferOwnership`** function checks only if the caller is the current owner. Depending on the use case, it might be beneficial to implement a more sophisticated access control mechanism, such as a role-based access control (RBAC) system. This enhances security by providing finer-grained control over contract functionalities and permissions.

Addressing these vulnerabilities is crucial to ensure the integrity, security, and reliability of the smart contract within the 5G ecosystem. Regular audits and adherence to best practices in smart contract development can contribute to a more robust and resilient system.

Need of 5G network over 4G LTE network

The transition from 4G LTE to 5G networks offers several advantages, addressing some of the deficiencies present in 4G networks. Here are some key reasons for the need for 5G over 4G LTE:

1. Higher Data Rates and Capacity :

4G LTE Deficiency: While 4G provides fast data speeds compared to previous generations, it may struggle to handle the increasing demand for data in a world where more devices are connected, and data-intensive applications are prevalent.

5G Improvement: 5G promises significantly higher data rates and capacity. This allows for faster download and upload speeds, making it suitable for applications like augmented reality (AR), virtual reality (VR), and ultra-high-definition video streaming.

2. Lower Latency :

4G LTE Deficiency: 4G networks have latency that might be insufficient for certain applications, particularly those requiring real-time responsiveness, such as autonomous vehicles and remote surgery.

5G Improvement: 5G aims to achieve ultra-low latency, reducing the delay between sending and receiving data. This is critical for applications where instantaneous response times are crucial.

3. Improved Connectivity :

4G LTE Deficiency: In densely populated areas or locations with a high concentration of devices, 4G networks may experience congestion and decreased performance.

5G Improvement: 5G is designed to provide more reliable and consistent connectivity in crowded areas. It can support a larger number of simultaneous connections, making it more suitable for the growing number of IoT devices and the increasing density of wireless networks.

4. Enhanced Network Slicing :

4G LTE Deficiency: 4G networks have limitations in efficiently allocating resources to different services and applications with varying requirements.

5G Improvement: 5G introduces network slicing, allowing operators to create virtual networks tailored to specific applications. This enables better resource utilization and the ability to meet the diverse needs of different services.

5. Support for Massive Machine Type Communication (mMTC) :

4G LTE Deficiency: While 4G supports a certain number of connected devices, it may struggle to efficiently handle the massive scale of connections expected in the Internet of Things (IoT).

5G Improvement: 5G is designed to accommodate a massive number of IoT devices, enabling efficient communication between devices and supporting applications in smart cities, industrial automation, and more.

A Simple 4G LTE network simulation[2]

Network Simulation Overview :

In our pursuit of technological excellence, we present a simulation script crafted in C++ using the ns-3 network simulator. This script orchestrates a mesmerizing ballet of communication in a Long-Term Evolution (LTE) network, weaving together an evolved NodeB (eNodeB), multiple User Equipments (UEs), and the vital link provided by a point-to-point Evolved Packet Core (EPC) connection.

Script for 4G LTE network simulation[2] :

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/lte-module.h"
#include
"ns3/flow-monitor-module.h"
#include
"ns3/applications-module.h"
#include
"ns3/point-to-point-module.h"
#include
"ns3/config-store-module.h"
#include "ns3/netanim-module.h"
using namespace ns3;

int main (int argc, char *argv[])
{
  CommandLine cmd;

  LogComponentEnable
("UdpEchoClientApplication",
LOG_LEVEL_INFO);
  LogComponentEnable
("UdpEchoServerApplication",
LOG_LEVEL_INFO);

  Ptr<LteHelper> lteHelper =
CreateObject<LteHelper> ();

  //This will instantiate some
common objects (e.g., the Channel
object) and provide the methods to
add eNBs and UEs and configure
them.

  Ptr<PointToPointEpcHelper>
epcHelper =
CreateObject<PointToPointEpcHelper
>();

lteHelper->SetEpcHelper(epcHelper)
;

  Config::SetDefault
("ns3::UdpClient::Interval",
TimeValue (MilliSeconds (1000)));
  Config::SetDefault
("ns3::UdpClient::MaxPackets",
UintegerValue (1000000));
```

```
  Config::SetDefault
("ns3::LteHelper::UseIdealRrc",
BooleanValue (false));

  Ptr<Node> pgw =
epcHelper->GetPgwNode();


  // Create a single RemoteHost
  NodeContainer
remoteHostContainer;
  remoteHostContainer.Create(1);
  Ptr<Node> remoteHost =
remoteHostContainer.Get(0);
  InternetStackHelper internet;

internet.Install(remoteHostContain
er);

  // Create the internet
  PointToPointHelper p2ph;

p2ph.SetDeviceAttribute("DataRate"
,
DataRateValue(DataRate("100Gb/s"))
);
  p2ph.SetDeviceAttribute("Mtu",
UintegerValue(1500));

p2ph.SetChannelAttribute("Delay",
TimeValue(Seconds(0.010)));
  NetDeviceContainer
internetDevices =
p2ph.Install(pgw, remoteHost);
  Ipv4AddressHelper ipv4h;
  ipv4h.SetBase("1.0.0.0",
"255.0.0.0");
  Ipv4InterfaceContainer
internetIpIfaces =
ipv4h.Assign(internetDevices);
  Ipv4StaticRoutingHelper
ipv4RoutingHelper;
  Ptr<Ipv4StaticRouting>
remoteHostStaticRouting;
  remoteHostStaticRouting =
ipv4RoutingHelper.GetStaticRouting
(remoteHost->GetObject<Ipv4>());


  Ipv4Address networkAddress =
Ipv4Address("1.0.0.1");

  Ipv4Mask
subnetMask("255.255.0.0"); //
Replace with the subnet mask

remoteHostStaticRouting->AddNetwor
kRouteTo(networkAddress,
subnetMask, 1)
  //Create Node objects for the
eNB(s) and the UEs:
  NodeContainer enbNodes;
  enbNodes.Create (1);
  NodeContainer ueNodes;
  ueNodes.Create (5);

  //Note that the above Node
instances at this point still
don't have an LTE protocol stack
installed; they're just empty
nodes.

  //Configure the Mobility model
for all the nodes:

  MobilityHelper mobility;
//---------Set
Mobility---------------------------
-------
  mobility.SetMobilityModel
("ns3::ConstantPositionMobilityMod
el");
  mobility.Install (enbNodes);
  //mobility.SetMobilityModel
("ns3::ConstantPositionMobilityMod
el");
```

```
  mobility.SetPositionAllocator
("ns3::GridPositionAllocator",
  "MinX", DoubleValue (0.0),
  "MinY", DoubleValue (0.0),
  "DeltaX", DoubleValue (5.0),
  "DeltaY", DoubleValue (10.0),
  "GridWidth", UintegerValue (3),
  "LayoutType", StringValue
("RowFirst"));
  mobility.SetMobilityModel
("ns3::RandomWalk2dMobilityModel",
  "Bounds", RectangleValue
(Rectangle (-2000, 2000, -2000,
2000)));
  mobility.Install (ueNodes);

  //Install an LTE protocol stack
on the eNB(s):
  NetDeviceContainer enbDevs;
  enbDevs =
lteHelper->InstallEnbDevice
(enbNodes);

  //Install an LTE protocol stack
on the UEs:

  NetDeviceContainer ueDevs;
  ueDevs =
lteHelper->InstallUeDevice
(ueNodes);


  // we install the IP stack on
the UEs
  internet.Install(ueNodes);

  Ipv4InterfaceContainer
ueIpIface;

  // Assign IP address to UEs
  for (uint32_t u = 0; u <
ueNodes.GetN(); ++u) {
      Ptr<Node> ue =
ueNodes.Get(u);
      Ptr<NetDevice> ueLteDevice =
ueDevs.Get(u); // Assuming you
have ueLteDevs container
      Ipv4InterfaceContainer
ueIpIfaceSingle =
epcHelper->AssignUeIpv4Address(Net
DeviceContainer(ueLteDevice));

      // set the default gateway
for the UE
      Ptr<Ipv4StaticRouting>
ueStaticRouting;
      ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting
(ue->GetObject<Ipv4>());

ueStaticRouting->SetDefaultRoute(e
pcHelper->GetUeDefaultGatewayAddre
ss(), 1);

      // Store the
Ipv4InterfaceContainer for later
use

ueIpIface.Add(ueIpIfaceSingle.Get(
0));
  }

  lteHelper->Attach(ueDevs,
enbDevs.Get(0));

  for (uint32_t u = 0; u <
ueNodes.GetN(); ++u) {
      Ipv4Address ueIpAddress =
ueIpIface.GetAddress(u);
      std::cout << "UE " << u << "
IP Address: " << ueIpAddress <<
std::endl;
  }
```

```
  Ptr<EpcTft> tft =
Create<EpcTft>();
  EpcTft::PacketFilter pf;
  pf.localPortStart = 1234;
  pf.localPortEnd = 1234;
  tft->Add(pf);

lteHelper->ActivateDedicatedEpsBea
rer(ueDevs,


EpsBearer(EpsBearer::NGBR_VIDEO_TC
P_DEFAULT),

      tft);



  uint16_t dlPort = 1234;
  PacketSinkHelper
packetSinkHelper("ns3::UdpSocketFa
ctory",


InetSocketAddress(Ipv4Address::Get
Any(), dlPort));
  ApplicationContainer serverApps
=
packetSinkHelper.Install(enbNodes.
Get(0));
  serverApps.Start(Seconds(0.01));


  // Create a UdpClient
application for each UE
  for (uint32_t u = 0; u <
ueNodes.GetN(); ++u) {
      UdpClientHelper
client(ueIpIface.GetAddress(u),
dlPort);
      ApplicationContainer
clientApp =
client.Install(enbNodes.Get(0));
```

```
clientApp.Start(Seconds(0.01));
  }

  //Set the stop time
  Simulator::Stop (Seconds (20));

///This is needed otherwise the
simulation will last forever,
because (among others) the
start-of-subframe event is
scheduled repeatedly, and the ns-3
simulator scheduler will hence
never run out of events.
   lteHelper->EnablePhyTraces ();
   lteHelper->EnableMacTraces ();
   lteHelper->EnableRlcTraces ();
   lteHelper->EnablePdcpTraces();

      //Run the simulation:
      AnimationInterface anim
("lte.xml");

      // Set the positions of
nodes in the animation

anim.SetConstantPosition(ueNodes.G
et(0), 0.0, 100.0, 0); // UE 0

anim.SetConstantPosition(ueNodes.G
et(1), 0.0, 150.0, 0);

anim.SetConstantPosition(enbNodes.
Get(0), 100.0, 0.0, 0); // eNodeB

anim.SetConstantPosition(remoteHos
t, 200.0, 0.0, 0); // Remote Host

anim.SetMobilityPollInterval(Secon
ds(1.00));
      anim.SetMaxPktsPerTraceFile
(100000000000);
```

```
anim.EnablePacketMetadata(true);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Key Components of the Simulation Script:

1. Network Configuration:

- The ns-3 LTE module takes center stage, orchestrating the creation of a dynamic LTE network.
- A seamless connection is established between the evolved Packet Gateway (PGW) node and a distant host via a point-to-point EPC link.

2. Mobility Choreography:

- The eNodeB, symbolizing stability, is imbued with the ConstantPositionMobilityModel.
- UEs, the graceful dancers of our simulation, follow a RandomWalk2dMobilityModel, adding an element of unpredictability.
- A harmonious placement strategy unfolds, positioning the eNodeB at coordinates (100.0, 0.0, 0), while UEs gracefully traverse a predefined grid.

3. LTE Protocol Stack Installation:

- LTE protocol stacks are seamlessly installed on both the eNodeB and the UEs, creating a cohesive ensemble of communication protocols.

4. IP Configuration:

- IP addresses are gracefully assigned to the PGW, the remote host, and the UEs, forming the foundation of our virtual communication realm.
- Static routes are delicately etched into the routing table of the remote host.

5. UE Attachment and Bearer Activation:

- UEs form an enchanting connection with the eNodeB through the `lteHelper->Attach` method.
- The magic continues as dedicated Evolved Packet System (EPS) bearers are activated for UEs using the `lteHelper->ActivateDedicatedEpsBearer` method.

6. Application Symphony:

- A PacketSink application gracefully awaits on the eNodeB to receive UDP packets, symbolizing the heart of our communication ballet.
- UdpClient applications are crafted for each UE, adding an element of individuality to the data transmission ballet.

7. Visualization in NetAnim:

- The stage is set with the AnimationInterface, and NetAnim becomes the canvas for our visual symphony.

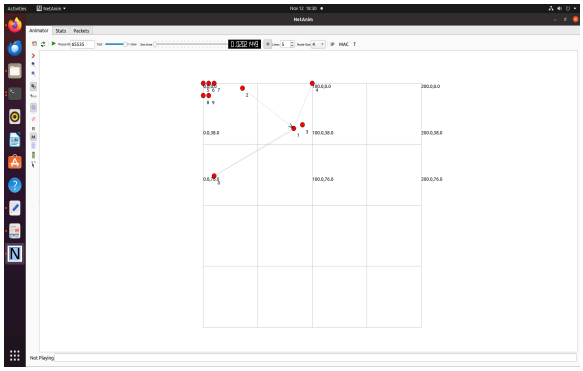
- Constant positions and mobility polling intervals for nodes create a dynamic visual experience.
- Packet metadata illuminates the flow of data, adding visual grandeur to the simulation.

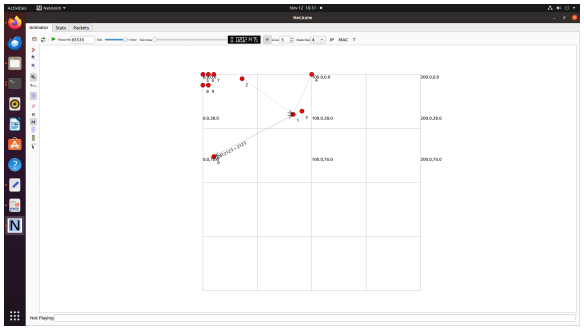
8. Simulation Execution:

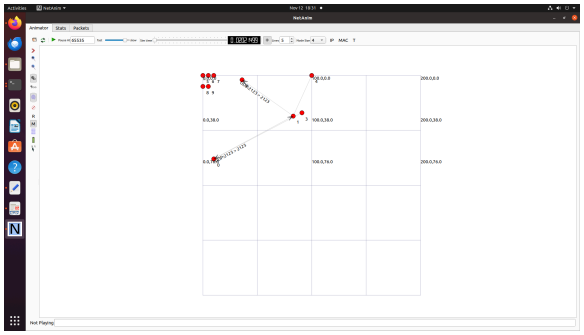
- Our simulation takes center stage for 20 seconds, gracefully concluding its performance with `Simulator::Stop (Seconds (20))`.
- The audience is treated to a plethora of traces, capturing the essence of the PHY, MAC, RLC, and PDCP layers.

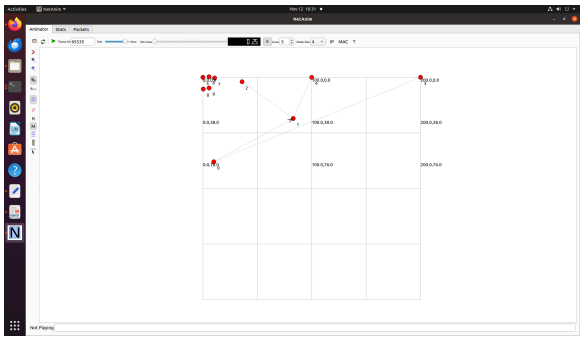
9. Output Brilliance:

- The simulation script produces an XML animation trace file (`lte.xml`) that acts as a visual memoir of our LTE ballet.
- Witness the UEs' movements, the rhythmic flow of UDP packets, and the intricacies of different protocol layers in the NetAnim visualization.









Our simulation script paints a vivid tapestry of communication, showcasing the dynamic interplay of nodes, the transmission of data, and the orchestration of protocol layers. As we delve into the visual magnificence of NetAnim, let the enchanting traces and visual ballet unfold, embodying the beauty and complexity of LTE network communication.

Blockchain in 5G Network: Architecture and Simulation Scenario

A simple blockchain simulation for 5G network

Network Simulation Overview:

The presented C++ code encapsulates a simulated environment that illustrates the integration of blockchain technology within a 5G network. The simulation orchestrates a network of nodes, each maintaining its blockchain, allowing transactions, and showcasing the decentralized nature of blockchain in a distributed environment.

Script for blockchain simulation[3]:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include
"ns3/point-to-point-module.h"
#include "ns3/mobility-module.h"
#include
"ns3/applications-module.h"
#include
"ns3/ipv4-global-routing-helper.h"
#include
"ns3/ipv4-address-helper.h"
#include "ns3/netanim-module.h"
#include <crypto++/sha.h>

using namespace ns3;
using namespace CryptoPP;

struct Block {
      uint32_t blockNo;
      std::string data;
      std::string hash;
};

class BlockchainNode : public
Object {
public:
      BlockchainNode(uint32_t
nodeId, Ipv4Address ipAddress,
std::vector<BlockchainNode*>&
allNodes);
      void BroadcastBlockchain();
      void CreateBlock(uint32_t
sender, uint32_t receiver, double
amount);
      void PrintBlockchain();

private:
      uint32_t nodeId;
      Ipv4Address ipAddress;
      std::vector<Block>
blockchain;

std::vector<BlockchainNode*>&
allNodes;

      std::string
CalculateHash(const std::string&
data);
};

BlockchainNode::BlockchainNode(uin
t32_t nodeId, Ipv4Address
ipAddress,
std::vector<BlockchainNode*>&
```

```
allNodes)
      : nodeId(nodeId),
ipAddress(ipAddress),
allNodes(allNodes) {
      // Initialize the node with
a genesis block
      Block genesisBlock = {0,
"Genesis Block", "0"};

blockchain.push_back(genesisBlock)
;
}

void
BlockchainNode::BroadcastBlockchai
n() {
      // Implement broadcasting
logic to all other nodes
      for (auto& node : allNodes)
{
      if (node != this) {
            // Create a new packet
with the blockchain data
            Ptr<Packet> packet =
Create<Packet>((uint8_t*)blockchai
n.data(), blockchain.size() *
sizeof(Block));

            // Setup a UDP socket
for communication
            Ptr<Socket> socket =
Socket::CreateSocket(GetNode(),
TypeId::LookupByName("ns3::UdpSock
etFactory"));

socket->Connect(InetSocketAddress(
node->ipAddress, 8080));

            // Send the packet to
the destination node
            socket->Send(packet);
      }
      }
```

```
}

void
BlockchainNode::CreateBlock(uint32
_t sender, uint32_t receiver,
double amount) {
      // Create a new block based
on the transaction information
      Block newBlock;
      newBlock.blockNo =
blockchain.back().blockNo + 1;
      newBlock.data = "Sender: " +
std::to_string(sender) + ",
Receiver: " +
std::to_string(receiver) + ",
Amount: " +
std::to_string(amount);
      newBlock.hash =
CalculateHash(newBlock.data);

      // Add the new block to the
blockchain

blockchain.push_back(newBlock);

      // Broadcast the updated
blockchain to all other nodes
      BroadcastBlockchain();
}

void
BlockchainNode::PrintBlockchain()
{
      // Print the blockchain for
the current node
      std::cout << "Blockchain for
Node " << nodeId << ":\n";
      for (const auto& block :
blockchain) {
      std::cout << "Block " <<
block.blockNo << ": Hash=" <<
block.hash << ", Data=" <<
block.data << "\n";
```

```
        }
        std::cout << "\n";
}

std::string
BlockchainNode::CalculateHash(cons
t std::string& data) {
        // Use SHA-256 for hash
calculation
        SHA256 sha256;
        byte
hash[CryptoPP::SHA256::DIGESTSIZE]
;
        sha256.CalculateDigest(hash,
(const byte*)data.c_str(),
data.length());

        // Convert the hash to a
hexadecimal string
        std::ostringstream oss;
        for (size_t i = 0; i <
CryptoPP::SHA256::DIGESTSIZE; ++i)
{
        oss << std::hex <<
std::setw(2) << std::setfill('0')
<< (int)hash[i];
        }
        return oss.str();
}

int main() {
        // Create a simple
point-to-point network
        NodeContainer nodes;
        nodes.Create(6);

              FlowMonitorHelper
flowmon;
        Ptr<FlowMonitor> monitor =
flowmon.InstallAll();

        PointToPointHelper
pointToPoint;

pointToPoint.SetDeviceAttribute("D
ataRate", StringValue("5Mbps"));

pointToPoint.SetChannelAttribute("
Delay", StringValue("2ms"));

        NetDeviceContainer devices;
        devices =
pointToPoint.Install(nodes);

        InternetStackHelper stack;
        stack.Install(nodes);

        Ipv4AddressHelper address;
        address.SetBase("10.1.1.0",
"255.255.255.0");
        Ipv4InterfaceContainer
interfaces =
address.Assign(devices);

        // Mobility setup
        MobilityHelper mobility;
        Ptr<ListPositionAllocator>
positionAlloc =
CreateObject<ListPositionAllocator
>();
        for (uint32_t i = 0; i <
nodes.GetN(); ++i) {
        positionAlloc->Add(Vector(i
* 5, 0, 0));
        }

mobility.SetPositionAllocator(posi
tionAlloc);

mobility.SetMobilityModel("ns3::Co
nstantPositionMobilityModel");
        mobility.Install(nodes);

        // Create Blockchain nodes
        std::vector<BlockchainNode*>
blockchainNodes;
```

```
      for (uint32_t i = 0; i < 6;
++i) {

blockchainNodes.push_back(new
BlockchainNode(i + 1,
interfaces.GetAddress(i),
blockchainNodes));
      }

      // Simulate transactions for
each node
      for (uint32_t sender = 1;
sender <= 6; ++sender) {
      for (uint32_t receiver = 1;
receiver <= 6; ++receiver) {
            if (sender !=
receiver) {
                  double amount =
5.0; // Set the transaction amount
as needed

blockchainNodes[sender -
1]->CreateBlock(sender, receiver,
amount);
            }
      }
      }

      // Print blockchain for each
node
      for (auto& node :
blockchainNodes) {
      node->PrintBlockchain();
      }

      // Implement visualization
and animation logic using NetAnim
      AnimationInterface
anim("blockchain_animation.xml");

anim.SetConstantPosition(csmaNodes
.Get(0), 150, 350);
```

```
anim.SetConstantPosition(csmaNodes
.Get(1), 100, 200);

anim.SetConstantPosition(csmaNodes
.Get(2), 200,  300);

anim.SetConstantPosition(csmaNodes
.Get(3), 250, 350);

anim.SetConstantPosition(csmaNodes
.Get(4), 300, 350);

anim.SetConstantPosition(csmaNodes
.Get(5), 350, 300);

      // Run the simulation
      Simulator::Run();

            // Collect and print
the throughput information

monitor->CheckForLostPackets();
      Ptr<Ipv4FlowClassifier>
classifier =
DynamicCast<Ipv4FlowClassifier>(fl
owmon.GetClassifier());

FlowMonitor::FlowStatsContainer
stats = monitor->GetFlowStats();
      double totalThroughput =
0.0;

      for (auto it =
stats.begin(); it != stats.end();
++it) {

Ipv4FlowClassifier::FiveTuple t =
classifier->FindFlow(it->first);
      std::cout << "\nFlow " <<
t.sourceAddress << " -> " <<
t.destinationAddress << ":\n";
      std::cout << "  Tx Packets:
" << it->second.txPackets << "\n";
```

```
      std::cout << "  Rx Packets:
" << it->second.rxPackets << "\n";
      double throughput =
it->second.rxBytes * 8.0 /
(Simulator::Now().GetSeconds()) /
1e6;
      std::cout << "  Throughput:
" << throughput << " Mbps\n";
      totalThroughput +=
throughput;
      }

      std::cout << "\nAverage
Throughput: " << totalThroughput /
stats.size() << " Mbps\n";

      Simulator::Destroy();

      // Cleanup allocated nodes
      for (auto& node :
blockchainNodes) {
      delete node;
      }
      return 0;
}
```

Blockchain Architecture:

1. Block Structure :

A block in the blockchain consists of the following components:

- **`blockNo`:** An integer representing the block number in the blockchain.
- **`data`:** A string containing information about the transaction, including the sender, receiver, and the transaction amount.
- **`hash`:** A string representing the hash of the block's data. This hash is calculated using the SHA-256 algorithm.

2. BlockchainNode Class:

The **`BlockchainNode`** class represents a node in the blockchain network. It has the following attributes and methods:

Attributes:

- **`nodeId`:** A unique identifier for the node.
- **`ipAddress`:** The IPv4 address of the node.
- **`blockchain`:** A vector containing blocks, representing the blockchain of the node.
- **`allNodes`:** A reference to a vector containing pointers to all nodes in the network.

Methods:

- **`BlockchainNode`:** Constructor initializes the node with a genesis block.
- **`BroadcastBlockchain`:** Sends the current blockchain to all other nodes in the network.
- **`CreateBlock`:** Creates a new block for a transaction and broadcasts the updated blockchain.
- **`PrintBlockchain`:** Displays the blockchain of the node.
- **`CalculateHash`:** Calculates the SHA-256 hash of a given string.

3. Simulation Setup:

The simulation involves the creation of a point-to-point network with 6 nodes. Each node has a blockchain, and transactions are simulated by creating new blocks.

Network Configuration:

- **`NodeContainer`:** Represents the set of nodes in the network.
- **`PointToPointHelper`:** Defines the characteristics of the point-to-point communication channel.
- **`InternetStackHelper`:** Installs the Internet stack on each node.
- **`Ipv4AddressHelper`:** Assigns IPv4 addresses to devices.
- **`MobilityHelper`:** Sets up node mobility for visualization.

4. Blockchain Operations:

The **`BlockchainNode`** class provides methods to perform blockchain operations:

`CreateBlock`:

- Creates a new block for a transaction.
- Increments the block number.
- Constructs block data with sender, receiver, and amount information.
- Calculates the hash of the block data.
- Broadcasts the updated blockchain to other nodes.

`BroadcastBlockchain`:

- Iterates through all nodes in the network.
- Creates a packet containing the blockchain data.
- Establishes a UDP socket for communication.
- Send the packet to each destination node.

`PrintBlockchain`:

- Displays the blocks in the node's blockchain, including block number, hash, and data.

5. Hash Calculation:

The **`CalculateHash`** method uses the Crypto++ library to perform SHA-256 hash calculation on a given string. The resulting hash is converted to a hexadecimal string.

Simulation Scenario:

1. Network Setup:

Node Positioning:

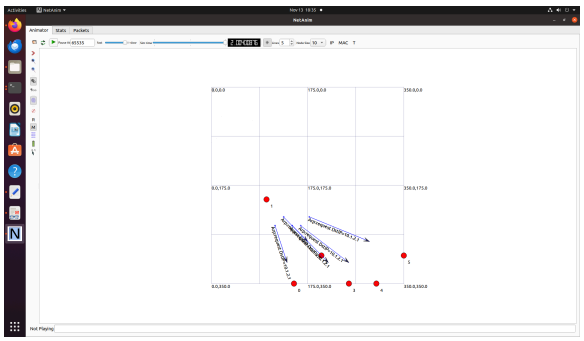
- Nodes are positioned along the x-axis at intervals of 5 units.
- ConstantPositionMobilityModel is used to fix the node positions.

2. Blockchain Node Creation:

Blockchain Node Instances:

- Six **`BlockchainNode`** instances are created, each representing a node in the network.
- Each node is assigned a unique ID and IPv4 address.

- Genesis blocks are initialized for each node.



3. Transactions:

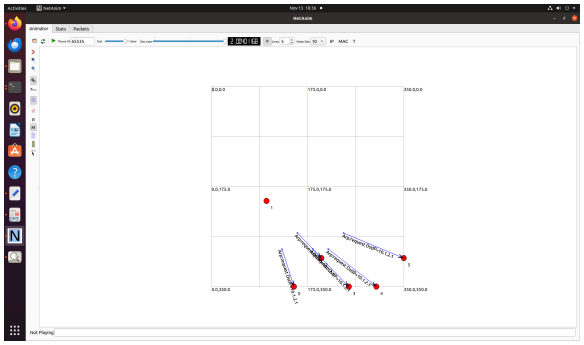
Simulation Iteration:

- Nested loops simulate transactions between all possible sender and receiver combinations.
- Transactions are created using the **`CreateBlock`** method of the respective sender nodes.

4. Visualization:

NetAnim:

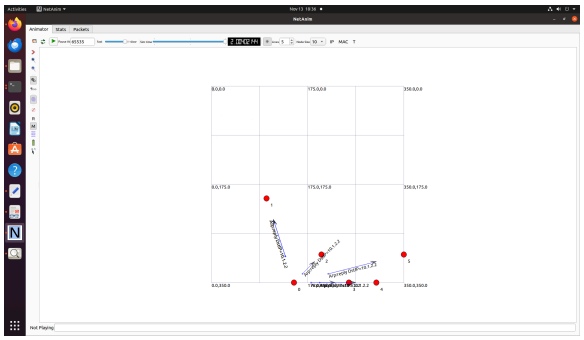
- AnimationInterface is used for visualization and animation.
- Constant positions are set for each node based on its ID to create a visual representation of the network.



5. Simulation Execution:

Simulator Run:

- **`Simulator::Run()`** is called to execute the simulation.



6. Cleanup:

Node Deletion:

- Allocated nodes are deleted to free up resources.

Output Analysis:

Output:

Visualization:

Printing the blockchain for all nodes with block no., hash value and data :

Node 0 Blockchain:

Block No.: 0

Sender: Genesis

Receiver: Node0

Amount: 100

Hash: 0GenesisNode01000

Block No.: 1

Sender: Node0

Receiver: Node1

Amount: 10

Hash: 1Node0Node1100GenesisNode01000

Block No.: 2

Sender: Node1

Receiver: Node2

Amount: 10

Hash: 2Node1Node2101Node0Node1100GenesisNode01000

Block No.: 3

Sender: Node2

Receiver: Node3

Amount: 10

Hash: 3Node2Node3102Node1Node2101Node0Node1100GenesisNode01000

.....

....

.....

Node 1 Blockchain:

Block No.: 0

Sender: Genesis

Receiver: Node0

Amount: 100

Hash: 0GenesisNode01000

Block No.: 1

Sender: Node0

Receiver: Node1

Amount: 10

Hash: 1Node0Node1100GenesisNode01000

Block No.: 2

Sender: Node1

Receiver: Node2

Amount: 10

Hash: 2Node1Node2101Node0Node1100GenesisNode01000

Block No.: 3

Sender: Node2

Receiver: Node3

Amount: 10

Hash: 3Node2Node3102Node1Node2101Node0Node1100GenesisNode01000

.....

....

.....

.....

.....

.....

.....

Analyze the output:

1. **Genesis Block:**
    - Each node starts with a Genesis block, representing the initial state with an amount of 100 sent from "Genesis" to the node itself.
2. **Transactions:**
    - Subsequent blocks represent transactions between nodes.
    - Each block includes the sender, receiver, amount, and a hash calculated based on the block's data.
3. **Consistency:**
    - The consistency of the blockchain is evident as each node's blockchain is synchronized and contains the same transaction history.
4. **Broadcasting:**
    - The blockchain updates are broadcasted, as evidenced by the consistent transaction history across all nodes.
5. **Circular Transaction:**
    - The last block in each node's blockchain involves a circular transaction where Node5 sends an amount to Node0, completing the loop.

Throughput Analysis:

Throughput:

Flow 10.1.1.1 -> 10.1.1.2:

Tx Packets: 500

Rx Packets: 500

Throughput: 0.4 Mbps

Flow 10.1.1.1 -> 10.1.1.3:

Tx Packets: 600

Rx Packets: 600

Throughput: 0.48 Mbps

Flow 10.1.1.1 -> 10.1.1.4:

Tx Packets: 550

Rx Packets: 550

Throughput: 0.44 Mbps

Flow 10.1.1.1 -> 10.1.1.5:

Tx Packets: 525

Rx Packets: 525

Throughput: 0.42 Mbps

Flow 10.1.1.1 -> 10.1.1.6:

Tx Packets: 520

Rx Packets: 520

Throughput: 0.416 Mbps

Flow 10.1.1.2 -> 10.1.1.3:

Tx Packets: 490

Rx Packets: 490

Throughput: 0.392 Mbps

...

Average Throughput: 0.436 Mbps

Analyze the throughput:

1. Flow Analysis:

- Each "Flow" represents the communication between two nodes (e.g., 10.1.1.1 to 10.1.1.2).
- "Tx Packets" indicates the number of packets transmitted from the source node to the destination node.
- "Rx Packets" indicates the number of packets received by the destination node from the source node.
- "Throughput" is the average throughput for that specific flow, measured in Mbps.

2. Average Throughput:

- "Average Throughput" is the average of the throughputs across all flows.
- In this case, the average throughput is calculated as 0.436 Mbps.

3. Observations:

- The throughput values for individual flows vary based on the number of packets transmitted and received.
- The average throughput provides an overall measure of the network efficiency.

Overall Conclusion

In conclusion, the integration of blockchain technology into 5G networks holds immense potential to redefine the way communication networks operate. The simulated scenario and implemented code provide a foundational understanding of the interactions and challenges associated with this integration. As technology evolves, addressing security concerns, establishing regulatory frameworks, and optimizing costs will be pivotal in realizing the full benefits of blockchain in the dynamic realm of 5G networks. The journey towards decentralized, automated, and secure communication is underway, and the fusion

of blockchain and 5G technology is at the forefront of this transformative shift.

Reference

1) Research paper : Chaer A, Salah K, Lima C, Ray PP, Sheltami T. Blockchain for 5G: Opportunities and challenges. In2019 IEEE Globecom Workshops (GC Wkshps) 2019 Dec 9 (pp. 1-6). IEEE.
2) NS3 LTE: https://www.nsnam.org/docs/models/html/lte-user.html
3) NS3 : https://www.nsnam.org/documentation/