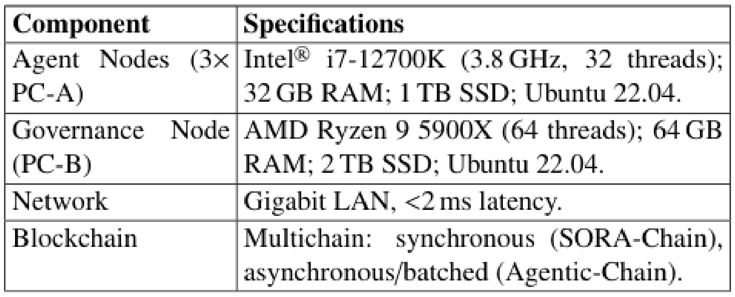
SORA-ATMAS: Adaptive Trust Management and Multi-LLM Aligned Governance for Future Smart Cities

➢ Blockchain network configuration

This guide provides step-by-step instructions to configure the MultiChain blockchain architecture in four different machine , allowing you to set up both global and local instances of the blockchain for interoperable services

1. **Setup Two Different Machines as Testbed**



Component	Specifications
Agent Nodes (3× PC-A)	Intel® i7-12700K (3.8 GHz, 32 threads); 32 GB RAM; 1 TB SSD; Ubuntu 22.04.
Governance Node (PC-B)	AMD Ryzen 9 5900X (64 threads); 64 GB RAM; 2 TB SSD; Ubuntu 22.04.
Network	Gigabit LAN, <2 ms latency.
Blockchain	Multichain: synchronous (SORA-Chain), asynchronous/batched (Agentic-Chain).

2. **Installation of Multichain Web Demo**

MultiChain Web Demo is a simple web interface for MultiChain blockchains, written in PHP.

https://github.com/MultiChain/multichain-web-demo

Install MultiChain Web Demo in all Four different machine inorder to visualize the transaction information in GUI interface when interoperable service Sharing data with each other

A. System Requirements

- A computer running web server software such as Apache.
- PHP 5.x or later with the curl and JSON extensions.
- MultiChain 1.0 alpha 26 or later, including MultiChain 2.x.

3. Create and launch a MultiChain Blockchain

Download MultiChain to install MultiChain on four different setup machines and create a chain named as Localchain1, Localchain2, Localchain3 For (Smart Service1, Smart Service2, Smart Service3) inorder to work as decentralized application as follows:

- **multichain-util create LocalChain //For creation**
- **multichaind** LocalChain **–daemon //For Running**
- **multichain-util create GlobalChain**
- **multichaind GlobalChain –daemon**

4. Configure the Web Demo For both Local and Global Blockchain

```
cat ~/.multichain/chain1/multichain.conf
grep rpc-port ~/.multichain/chain1/params.dat
```

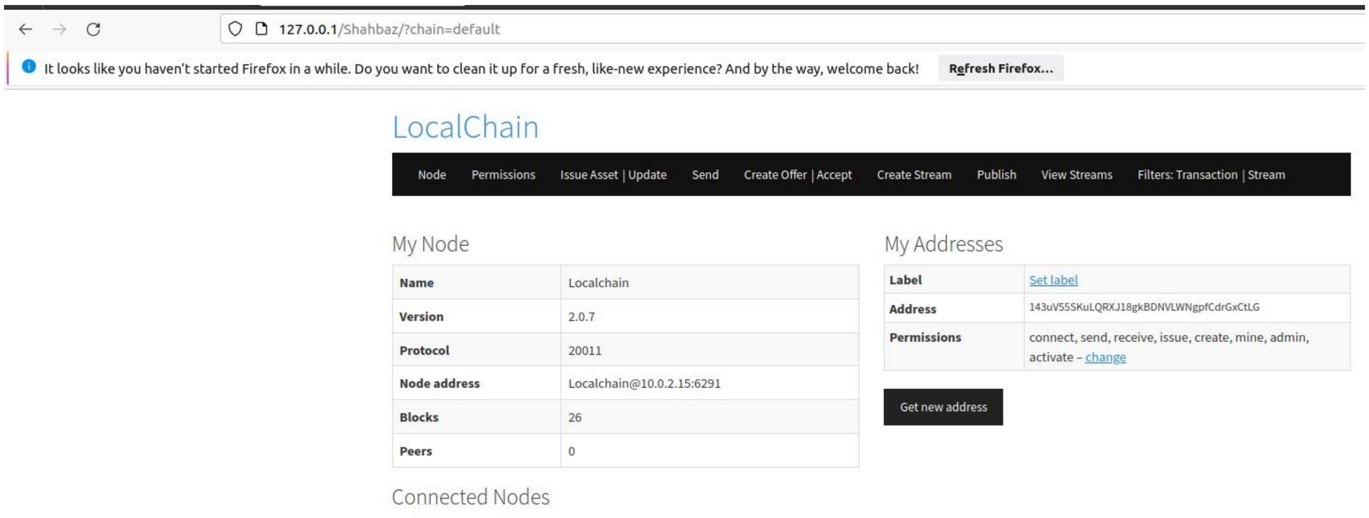
In the web demo directory, copy the config-example.txt file to config.txt:

```
cp config-example.txt config.txt
```

In the demo website directory, enter chain details in config.txt e.g.:

```
default.name=Default                  # name to display in the web interface
default.rpchost=127.0.0.1             # IP address or domain of MultiChain node
default.rpcsecure=0                   # set to 1 to access RPC via https (e.g. for MultiChain on
Azure)
default.rpcport=12345                 # usually default-rpc-port from params.dat
default.rpcuser=multichainrpc         # username for RPC from multichain.conf
default.rpcpassword=mnBh8a.......... # password for Local and Global Blockchain
```

LocalBlockchain Screen (Application-1)



127.0.0.1/Shahbaz/?chain=default
It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!
Refresh Firefox...
LocalChain
Node
Permissions
Issue Asset | Update
Send
Create Offer | Accept
Create Stream
Publish
View Streams
Filters: Transaction | Stream
Get new address

My Node

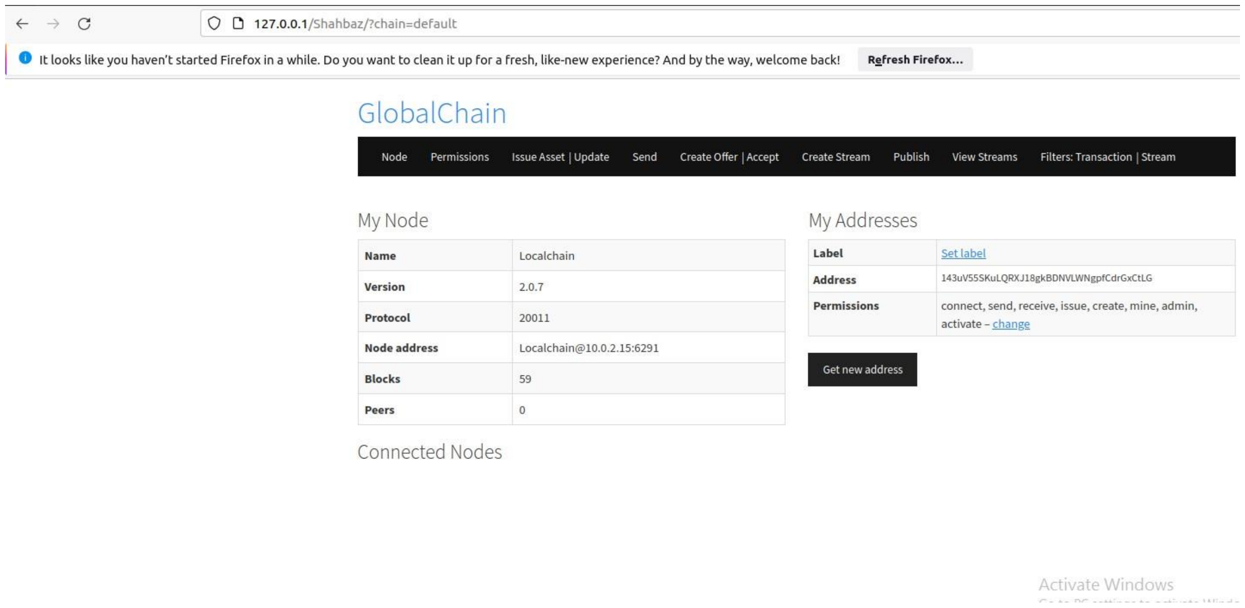
Name	Localchain
Version	2.0.7
Protocol	20011
Node address	Localchain@10.0.2.15:6291
Blocks	26
Peers	0

My Addresses

Label	Set label
Address	143uV55SKuLQRXJ18gkBDNVLWNgpfCdrGxCtLG
Permissions	connect, send, receive, issue, create, mine, admin, activate – change

Connected Nodes

Global Blockchain Screen



127.0.0.1/Shahbaz/?chain=default

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

GlobalChain

My Node

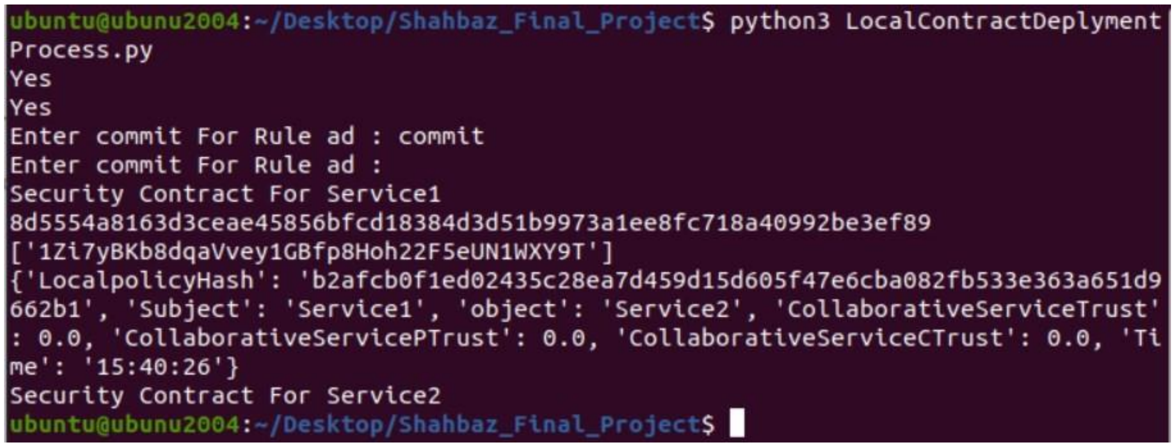
Name	Localchain
Version	2.0.7
Protocol	20011
Node address	Localchain@10.0.2.15:6291
Blocks	59
Peers	0

My Addresses

Label	Set label
Address	143uV55SKuLQRXJ18gkBDNVLWNgpfCdrGxCtLG
Permissions	connect, send, receive, issue, create, mine, admin, activate – change

Connected Nodes

5. **Deploy Default Service Security Smart Contract in all 3 interoperable services in order to load the basic Service Security Contract For interoperable services in local blockchain for this perform the Following steps in all machines**



```
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project$ python3 LocalContractDeplyment
Process.py
Yes
Yes
Enter commit For Rule ad : commit
Enter commit For Rule ad :
Security Contract For Service1
8d5554a8163d3ceae45856bfcd18384d3d51b9973a1ee8fc718a40992be3ef89
['1Zi7yBKb8dqaVvey1GBfp8Hoh22F5eUN1WXY9T']
{'LocalpolicyHash': 'b2afcb0f1ed02435c28ea7d459d15d605f47e6cba082fb533e363a651d9
662b1', 'Subject': 'Service1', 'object': 'Service2', 'CollaborativeServiceTrust'
: 0.0, 'CollaborativeServicePTrust': 0.0, 'CollaborativeServiceCTrust': 0.0, 'Ti
me': '15:40:26'}
Security Contract For Service2
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project$
```

✓ This will create a Rule repository in the Local machine and deploy the contract on the local Blockchain also update the smart contract in the Rule Data Stream of the Local Blockchain

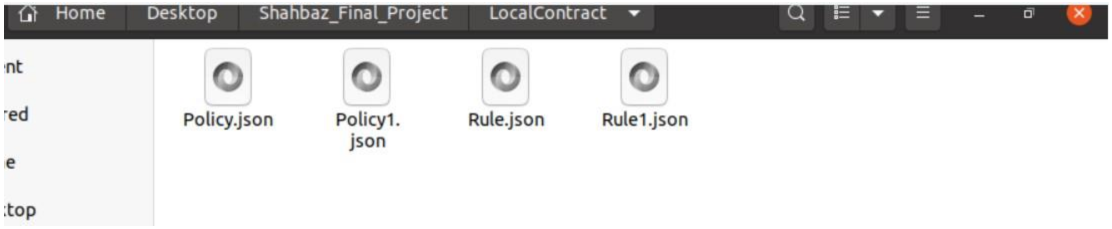
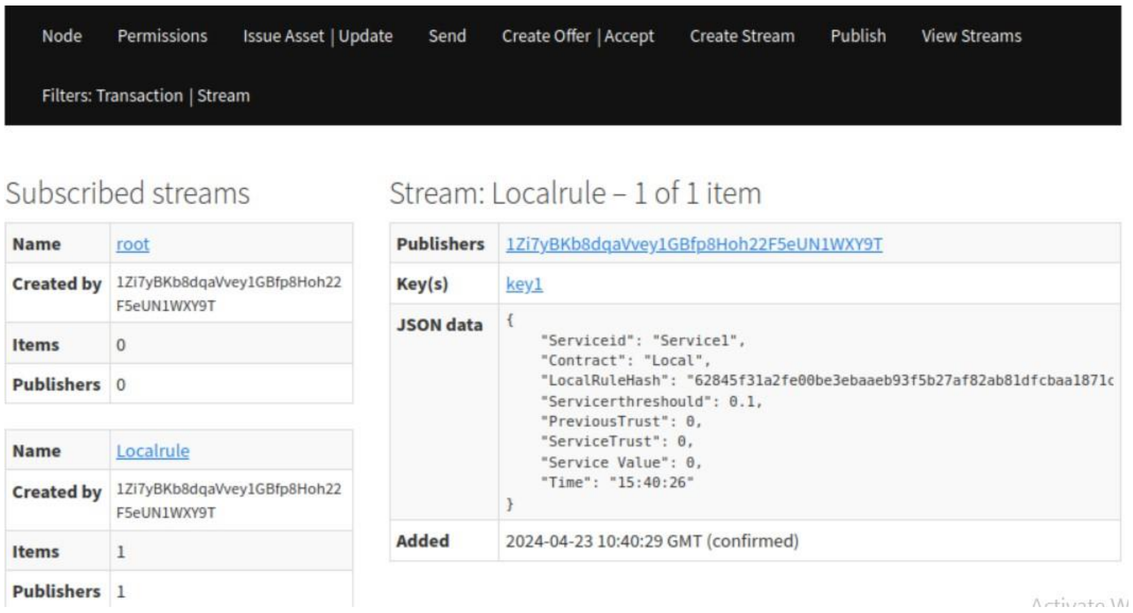


Figure of Local Blockchain Repository



Subscribed streams

Name	root
Created by	1Zi7yBKb8dqaVvey1GBfp8Hoh22F5eUN1WXY9T
Items	0
Publishers	0

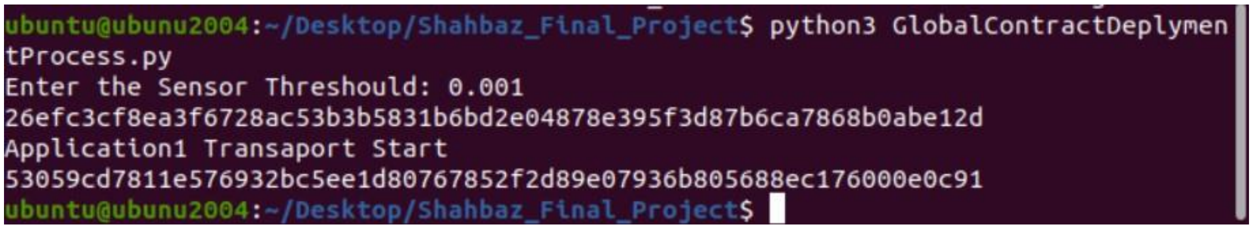
Name	Localrule
Created by	1Zi7yBKb8dqaVvey1GBfp8Hoh22F5eUN1WXY9T
Items	1
Publishers	1

Stream: Localrule – 1 of 1 item

Publishers	1Zi7yBKb8dqaVvey1GBfp8Hoh22F5eUN1WXY9T
Key(s)	key1
JSON data	see below
Added	2024-04-23 10:40:29 GMT (confirmed)

```
{
    "Serviceid": "Service1",
    "Contract": "Local",
    "LocalRuleHash": "62845f31a2fe00be3ebaaeb93f5b27af82ab81dfcbaa1871c
    "Servicerthreshould": 0.1,
    "PreviousTrust": 0,
    "ServiceTrust": 0,
    "Service Value": 0,
    "Time": "15:40:26"
}
```

6. **Deploy Default ServiceSecurity Smart Contract Format in Administrative Global Blockchain Node inorder to load the basic Global ServiceSecurity Contract**



```
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project$ python3 GlobalContractDeplymen
tProcess.py
Enter the Sensor Threshould: 0.001
26efc3cf8ea3f6728ac53b3b5831b6bd2e04878e395f3d87b6ca7868b0abe12d
Application1 Transaport Start
53059cd7811e576932bc5ee1d80767852f2d89e07936b805688ec176000e0c91
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project$
```

➢ Start the Client Nodes of Interoperable Services

To implement the clients of these interoperable smart decentralized applications in a simulated environment, we're utilizing Cooja along with SDN_WISE controller called as **SDIoT layer** , is a popular simulator. Cooja allows us to emulate a network of IoT devices, providing a realistic testing ground for our decentralized applications. To connect these simulated client nodes with our Python-based decentralized application, we've developed a bridge that interfaces between Cooja and our Python code. This bridge ensures seamless communication and interaction between the simulated IoT client nodes and our decentralized application, enabling us to evaluate its performance, reliability, and interoperability in a controlled setting before real-world deployment.

1. **SDN Installation**
   For Installation of contiki operating system follow the link and install the SDNWISEcontroller
   https://sdnwiselab.github.io/docs/guides/GetStarted.html

2. Management Repository of SDIoT layer

In both machines, the management repository of SDIoT layer is present in following path that

\Ada-Framework-Shahbaz1\GetStarted\

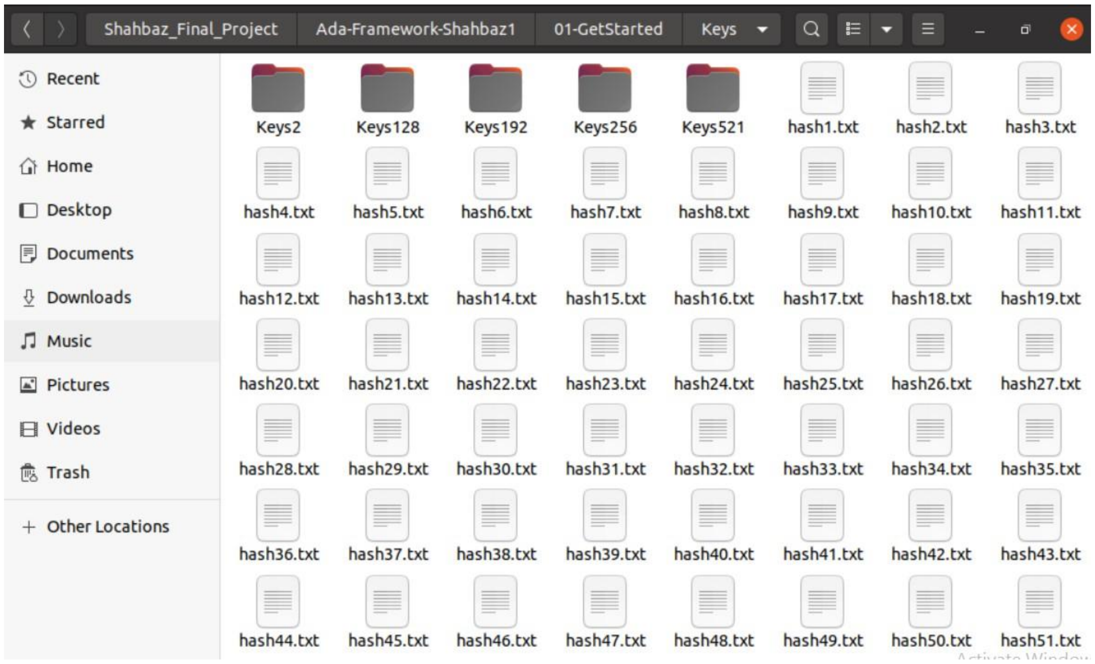
Following are the Management repository of individual application in different machine are

1. Keys (Responsible to manage the Key pair of Client nodes and SDN controller)
2. Trust management (Responsible to manage the Trust of Client nodes and SDN controller)
3. Policy Management (Responsible to manage the policies of Client nodes and SDN controller)

3. KeyManagement Execution

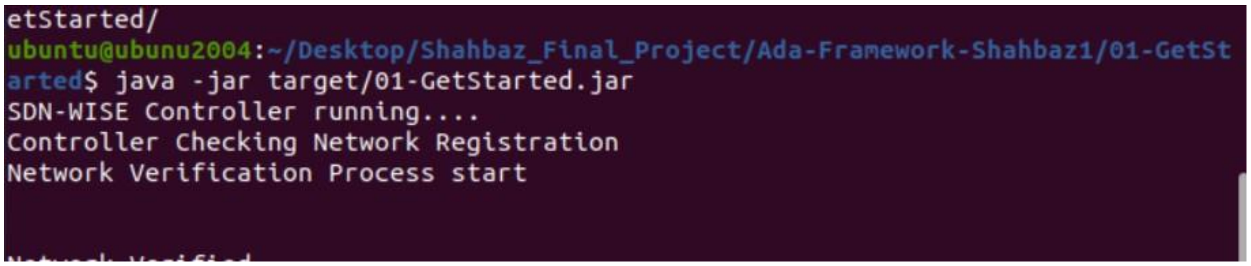
Run the Script Keys_sh that work as Key Management algorithm to Generate the Key pairs for SDN controller and the Client Nodes of the Interoperable Service

Location:



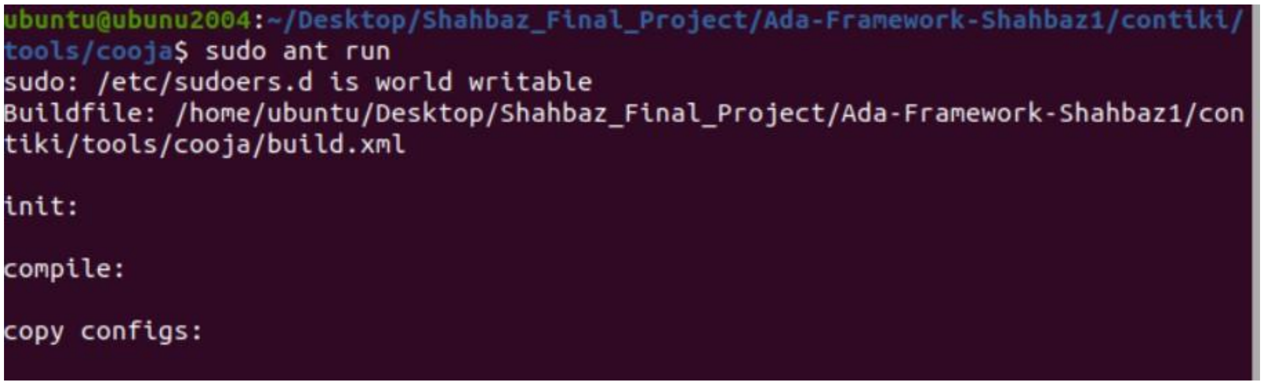
4. Run Client Nodes

In order to Run the Client First Start the SDN Controller First with the help of the following commands

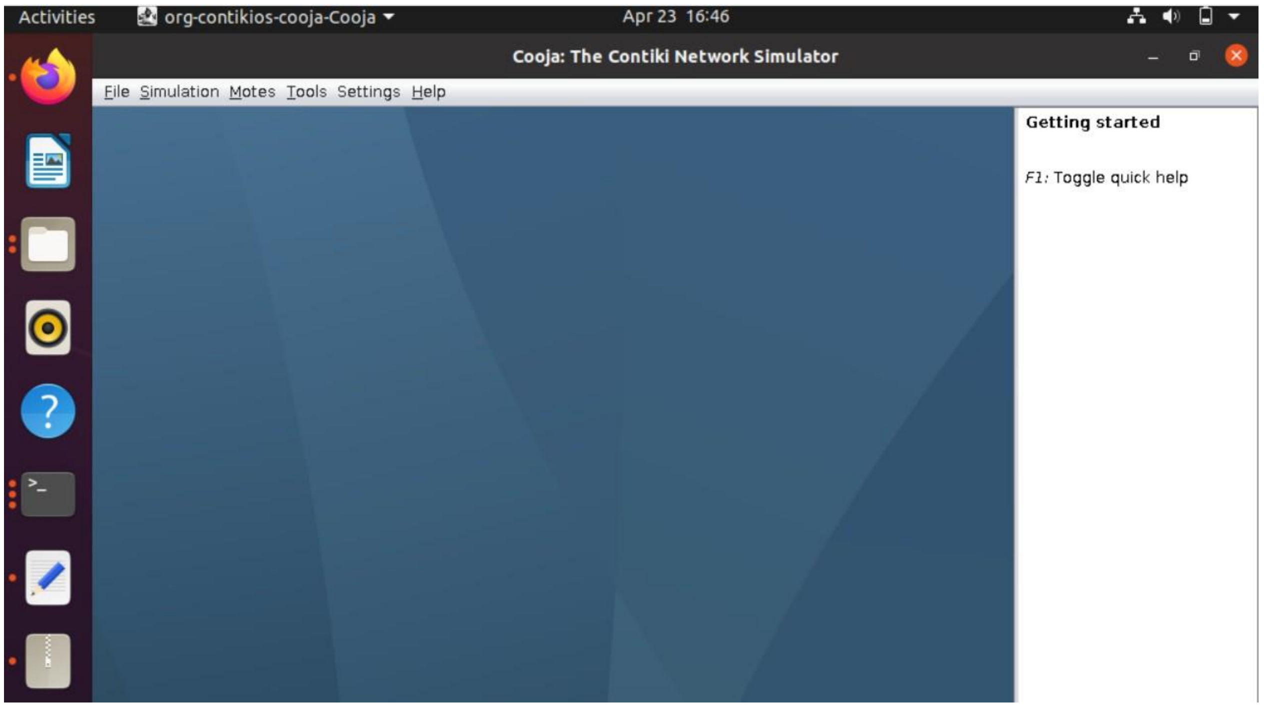


```
etStarted/
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project/Ada-Framework-Shahbaz1/01-GetSt
arted$ java -jar target/01-GetStarted.jar
SDN-WISE Controller running....
Controller Checking Network Registration
Network Verification Process start
```

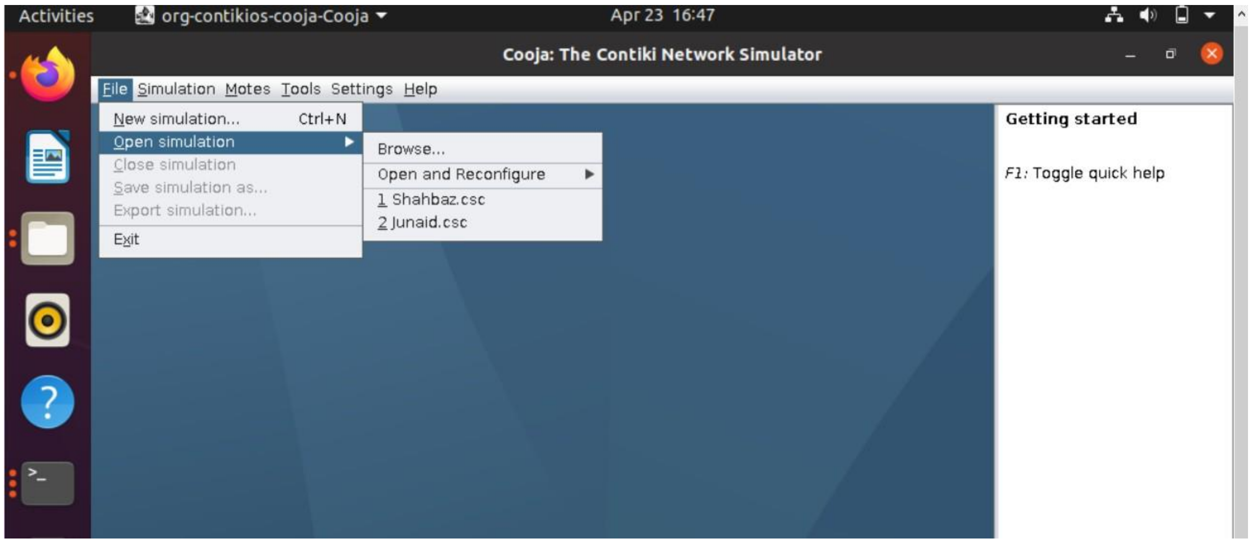
Then Run the Client Node

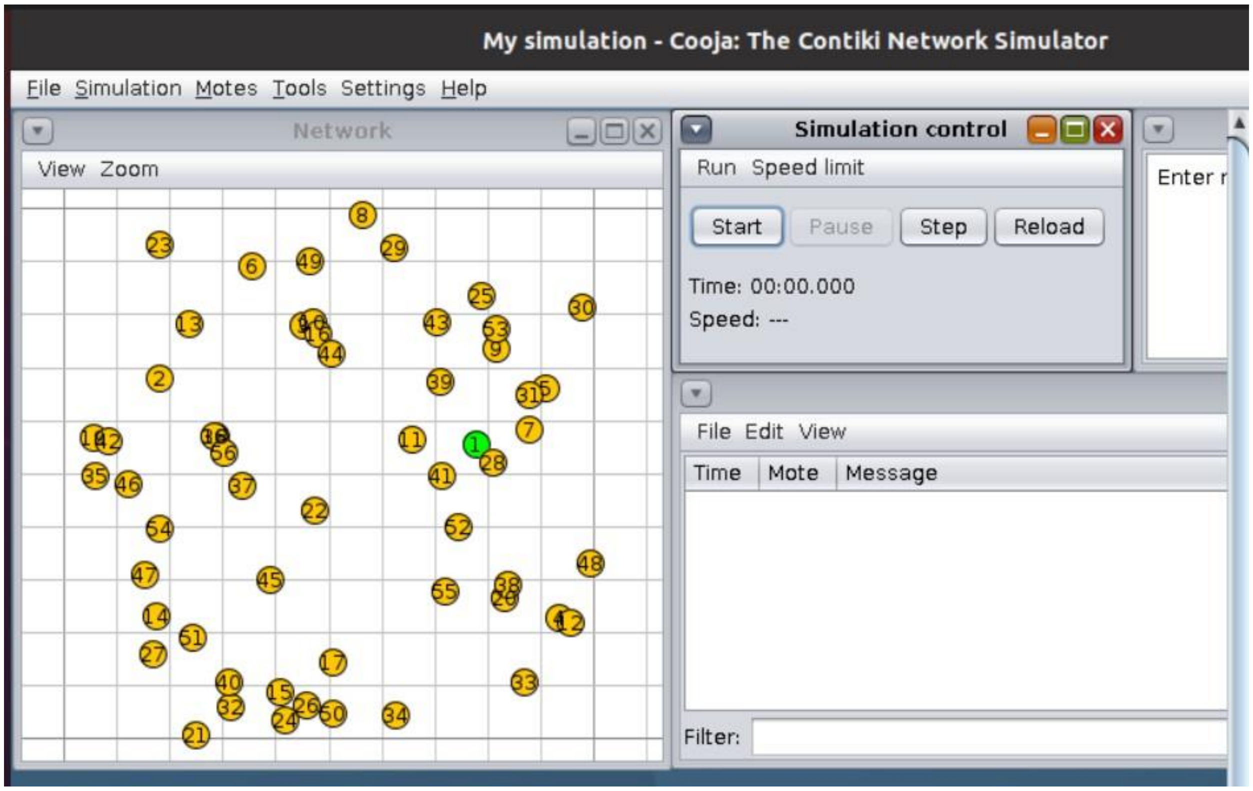


```
ubuntu@ubunu2004:~/Desktop/Shahbaz_Final_Project/Ada-Framework-Shahbaz1/contiki/
tools/cooja$ sudo ant run
sudo: /etc/sudoers.d is world writable
Buildfile: /home/ubuntu/Desktop/Shahbaz_Final_Project/Ada-Framework-Shahbaz1/con
tiki/tools/cooja/build.xml

init:

compile:

copy configs:
```

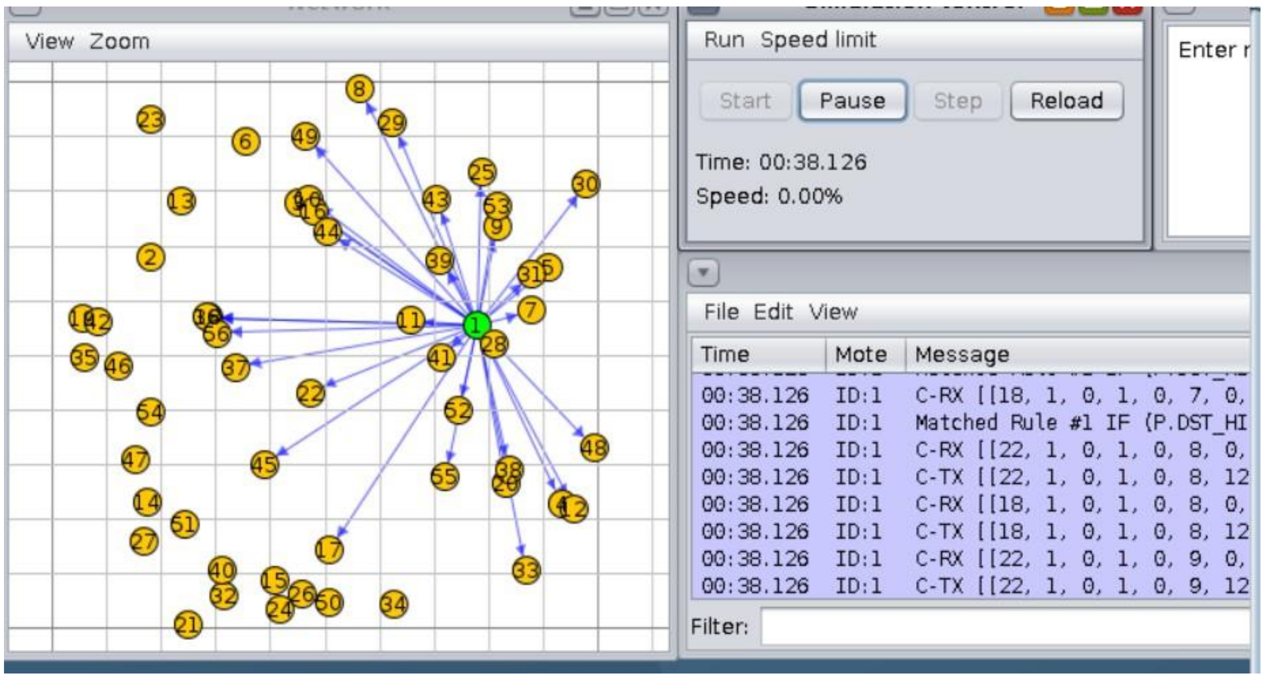

Activities
org-contikios-cooja-Cooja
Apr 23 16:46
Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Getting started
F1: Toggle quick help

Load Shahbaz.csv File For environment loading


Activities
org-contikios-cooja-Cooja
Apr 23 16:47
Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
New simulation... Ctrl+N
Open simulation
Close simulation
Save simulation as...
Export simulation...
Exit
Browse...
Open and Reconfigure
1 Shahbaz.csc
2 Junaid.csc
Getting started
F1: Toggle quick help


My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start
Pause
Step
Reload
Time: 00:00.000
Speed: ---
File Edit View
Time
Mote
Message
Filter:
Enter r

➢ Start the Simulation



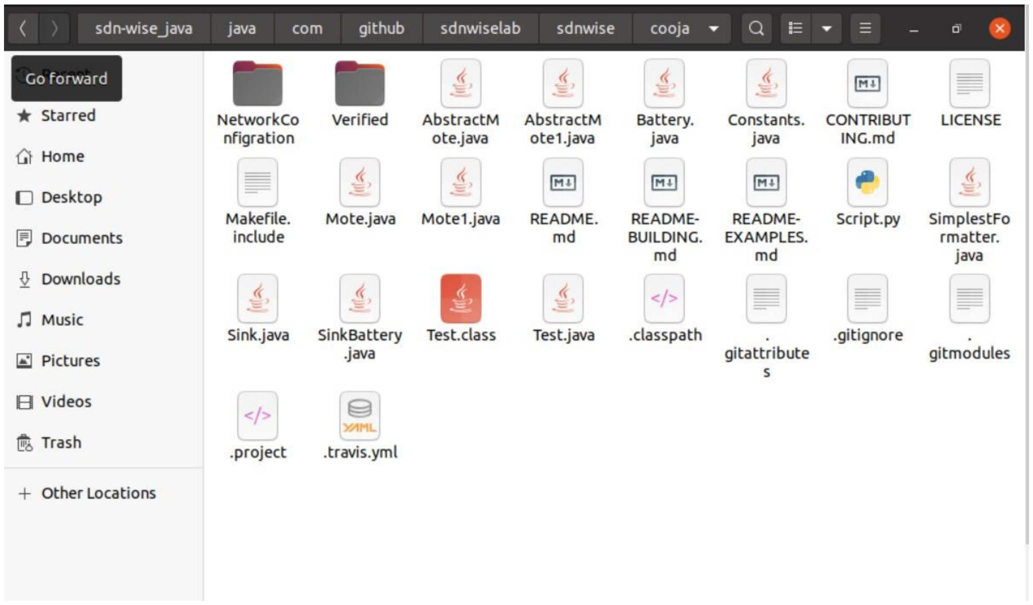
Time: 00:38.126
Speed: 0.00%

```
Time       Mote   Message
00:38.126  ID:1   C-RX [[18, 1, 0, 1, 0, 7, 0,
00:38.126  ID:1   Matched Rule #1 IF (P.DST_HI
00:38.126  ID:1   C-RX [[22, 1, 0, 1, 0, 8, 0,
00:38.126  ID:1   C-TX [[22, 1, 0, 1, 0, 8, 12
00:38.126  ID:1   C-RX [[18, 1, 0, 1, 0, 8, 0,
00:38.126  ID:1   C-TX [[18, 1, 0, 1, 0, 8, 12
00:38.126  ID:1   C-RX [[22, 1, 0, 1, 0, 9, 0,
00:38.126  ID:1   C-TX [[22, 1, 0, 1, 0, 9, 12
```

5. Service Management

Service Management Script is embedded in Client node (AbstractMote.java) code as a sub function reside in the following Folder

/home/Ubuntu/Desktop/Shahbaz_Final_Project/Conitki/…..



Application Design Demonstration

In this section, we demonstrate the integration of our Python-based decentralized agents with the client nodes of the SDIoT. We showcase how these agents interact seamlessly with the client nodes to facilitate various functionalities within the SDIoT ecosystem. By connecting the client nodes to our Python applications, we aim to highlight the adaptability and interoperability of our decentralized solutions. This demonstration serves to illustrate the potential of Python in developing robust and efficient decentralized agents that can effectively manage and control SDIoT networks for smart-city disaster management.

1. Implementation Discussion

In this section, we will delve into the implementation of Python client-server-based agents for weather disaster response management, leveraging the power of standard APIs and Multi-LLM reasoning.

✓ Weather Agent

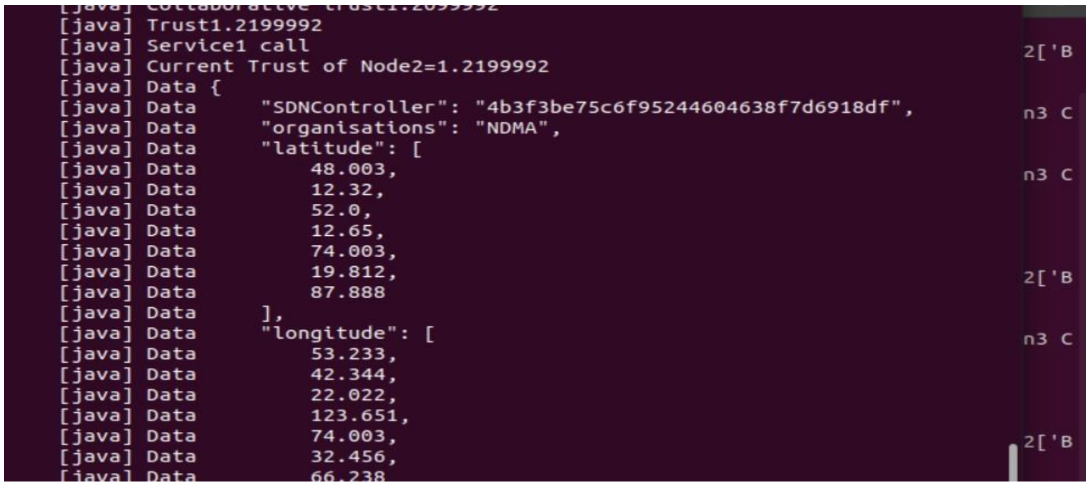
The Weather Agent plays a crucial role in our interoperable agent ecosystem within smart cities. It has been implemented using Python socket programming in a server-client code framework. As a client node, the Weather Agent connects not only with its own server but also with other administrative servers of smart agents, such as Traffic and Safety agents. This enables it to execute collaborative task requests seamlessly. Additionally, as a server node, the Weather Agent allows its clients and other smart agents to establish connections with it, facilitating smooth communication and seamless integration between different agents.

✓ Start Weather Agent

Run the Interoperable Weather Agent in PC-A with the help of this command:

```
python3 WeatherAgent.py
```

The Adaptive Trust Management Code is embedded within all Smart Interoperable Agent Codes. The Weather Agent Algorithm focuses solely on the implementation of the Smart Interoperable Agent Code, while the Adaptive Trust verification is integrated within this code.



```
[java] Collaborative trust1.2099992
[java] Trust1.2199992
[java] Service1 call
[java] Current Trust of Node2=1.2199992
[java] Data {
[java] Data     "SDNController": "4b3f3be75c6f95244604638f7d6918df",
[java] Data     "organisations": "NDMA",
[java] Data     "latitude": [
[java] Data         48.003,
[java] Data         12.32,
[java] Data         52.0,
[java] Data         12.65,
[java] Data         74.003,
[java] Data         19.812,
[java] Data         87.888
[java] Data     ],
[java] Data     "longitude": [
[java] Data         53.233,
[java] Data         42.344,
[java] Data         22.022,
[java] Data         123.651,
[java] Data         74.003,
[java] Data         32.456,
[java] Data         66.238
```

```
Weather Agent

Require: Meteorological Data (Temperature, Precipitation, Humidity, Wind, Cloud Cover, UV), Host,
    Port

Ensure: Classification (Normal/Rain/Storm/Heatwave), Risk-Trust Rationale

As Server Node

1: while True do

2: Hostname='Weather Agent'

3: Port='1234'

4: Listen(1)

5: Bind(Port,Host)

6: connect(Port,HOST) For Traffic Agent

7: connect(Port,HOST) For Safety Agent

8: Message[Temperature,Precipitation,Humidity,Wind,CloudCover,UV]=Received values from Open-Meteo
    API

9: if (Message[Precipitation]>=50 || Message[Temperature]>=35 || Message[Wind]>=50) then

10: Classify(Regime: Storm/Heatwave), Compute(Risk R_i(t), Trust T_i(t))

11: Send(Alert Message, Location, Risk-Trust)

12: Store Result in Weather Collaborative Agent.Json

13: end if

14: end while
```

✓ Traffic Agent

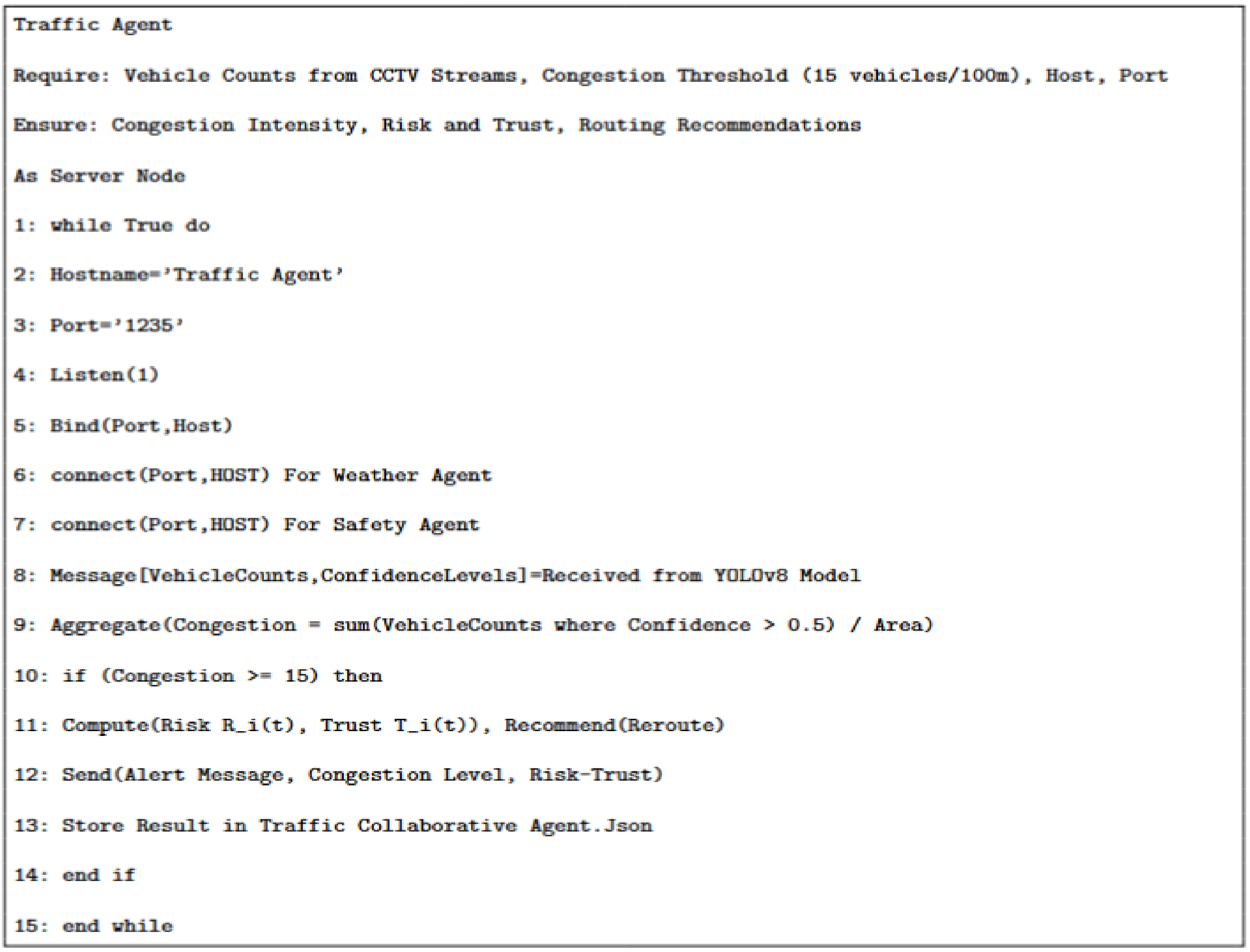
The Traffic Agent plays a vital role in our interconnected agent ecosystem in smart cities. Implemented using Python socket programming in a server-client framework, it acts as a client node, connecting not only with its own server but also with administrative servers of other smart agents like Weather and Safety agents. This seamless integration allows the Traffic Agent to execute collaborative tasks efficiently. Furthermore, as a server node, the Traffic Agent enables its clients and other smart agents to establish connections with it, ensuring smooth communication and effective coordination among the various agents.

✓ Start Traffic Agent

Run the Interoperable Traffic Agent in PC-A with the help of this command:

```
python3 TrafficAgent.py
```

The Adaptive Trust Management Code is embedded within all Smart Interoperable Agent Codes. The Traffic Agent Algorithm focuses solely on the implementation of the Smart Interoperable Agent Code, while the Adaptive Trust verification is integrated within this code.



```
Traffic Agent
Require: Vehicle Counts from CCTV Streams, Congestion Threshold (15 vehicles/100m), Host, Port
Ensure: Congestion Intensity, Risk and Trust, Routing Recommendations
As Server Node
1: while True do
2: Hostname='Traffic Agent'
3: Port='1235'
4: Listen(1)
5: Bind(Port,Host)
6: connect(Port,HOST) For Weather Agent
7: connect(Port,HOST) For Safety Agent
8: Message[VehicleCounts,ConfidenceLevels]=Received from YOLOv8 Model
9: Aggregate(Congestion = sum(VehicleCounts where Confidence > 0.5) / Area)
10: if (Congestion >= 15) then
11: Compute(Risk R_i(t), Trust T_i(t)), Recommend(Reroute)
12: Send(Alert Message, Congestion Level, Risk-Trust)
13: Store Result in Traffic Collaborative Agent.Json
14: end if
15: end while
```

- ✓ **Safety Agent**

The Safety Agent is implemented using Python socket programming within a server-client framework. This agent is responsible for receiving communications from other interconnected agents and subsequently searching for indicators of hazards like fire or smoke. For instance, the Weather Agent can transmit to the Safety Agent weather conditions containing rain and flood prediction values. The client nodes receive the message and examine the conditional parameters for alert messages. The response is then returned to the connected agent, which relays the alert. The Safety Agent is crucial to the coordination and administration of the emergency response. By seeking out hazard conditions, the agent is able to provide early warning and situational awareness to other interconnected agents, enabling a more coordinated and effective emergency response.

- ✓ **Start Safety Agent**

Run the Interoperable Traffic Agent in PC-A with the help of this command:

```
python3 SafetyAgent.py
```

The Adaptive Trust Management Code is embedded within all Smart Interoperable Agent Codes. The Safety Agent Algorithm focuses solely on the implementation of the Smart Interoperable Agent Code, while the Adaptive Trust verification is integrated within this code.

```
Safety Agent

Require: Fire/Smoke Detections from CCTV Streams, Host, Port

Ensure: Hazard Extent with Confidence-Based Annotations, Risk-Trust

As Server Node

1: while True do

2: Hostname='Safety Agent'

3: Port='1236'

4: Listen(1)

5: Bind(Port,Host)

6: connect(Port,HOST) For Weather Agent

7: connect(Port,HOST) For Traffic Agent

8: Message[FireDetections,SmokeDetections,Confidence]=Received from YOLO11 Model

9: if (any(Confidence > 0.5 for Fire or Smoke)) then

10: Compute(HazardExtent, Risk R_i(t), Trust T_i(t))

11: Send(Alert Message, Hazard Details, Risk-Trust)

12: Store Result in Safety Collaborative Agent.Json

13: end if

14: end while
```

B. SORA Governance

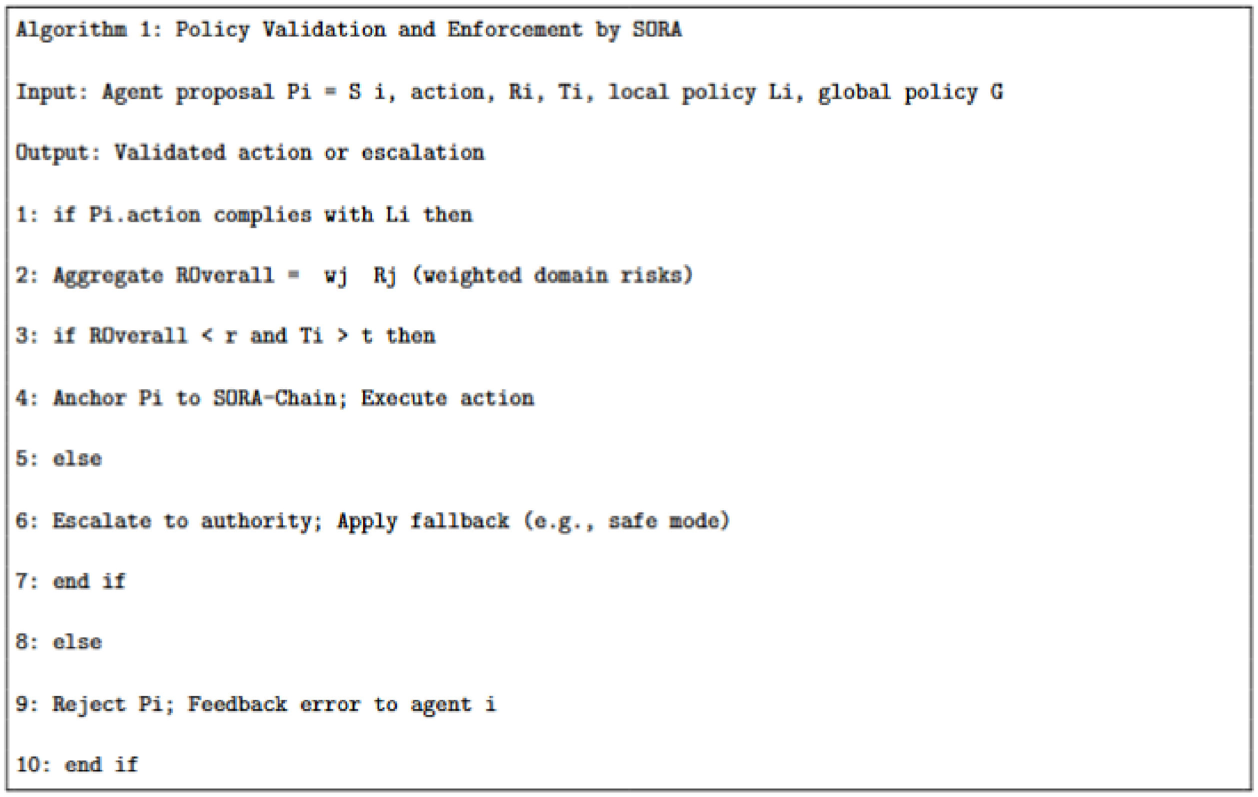
The SORA Governance layer provides centralized oversight and validation for the agent outputs. Implemented in Python, it integrates multi-LLM evaluation (GPT, Grok, DeepSeek) for policy alignment, risk-trust regulation, and compliance enforcement. It operates on the SORA Blockchain for auditability and receives inputs from the Agentic Blockchain.

✓ Start SORA Governance

Run the Interoperable Smart Ambulance Service in PC-3 with the help this command

python3 SORA.Py

The Multi-LLM Alignment and Governance Code is embedded within the SORA framework. The SORA Algorithm focuses on validation and enforcement, integrating adaptive trust management.



```
Algorithm 1: Policy Validation and Enforcement by SORA

Input: Agent proposal Pi = S i, action, Ri, Ti, local policy Li, global policy G

Output: Validated action or escalation

1: if Pi.action complies with Li then

2: Aggregate ROverall =  wj  Rj (weighted domain risks)

3: if ROverall < r and Ti > t then

4: Anchor Pi to SORA-Chain; Execute action

5: else

6: Escalate to authority; Apply fallback (e.g., safe mode)

7: end if

8: else

9: Reject Pi; Feedback error to agent i

10: end if
```

```
Algorithm 2: Policy Enforcement and Logging via Dual-Chain Governance

Input: Agent observation for S_j = {d_j(t)}, local policy L_j

Output: Validated {R_j(t), T_j(t)}; anchored decision

1: for each LLM m in {GPT, DeepSeek, Grok} do

2: Compute d_m(t) = LLM_m(S_j, L_j)

3: Evaluate MAE_m = |d_m(t) - baseline|

4: end for

5: Select m* = argmin MAE_m

6: if MAE_m* > _mae then

7: Feedback correction; Iterate

8: else

9: Compute R_j(t), T_j(t) from d_m*(t)

10: Anchor to Agentic-Chain

11: Transmit to SORA for global validation

12: end if
```