https://github.com/biblioteca1960/ChronaNet

June 6, 2025

ChronaNet: A Decentralized Network with Chronoresonant Consensus

1 Intellectual Property and Licensing

ChronaNet™ and ChronaCoin™ are trademarks of Luis Morató de Dalmases. The software is released under a dual license to promote open access while preserving commercial rights:

- **MIT License**: Free use for research, education, and personal experimentation.
- **ChronaCoin Commercial Use Agreement**: Required for any commercial use (e.g., exchanges, financial apps, or monetized services).

Attribution Required: ChronaCoin Protocol, developed by Luis Morató de Dalmases (2025). Powered by quantum-temporal cryptography and chrono-algorithmic architecture.

For licensing inquiries, please visit https://chronanet.org/license or email chronacoin.licensing@protonmail.com. Full license terms are available in the GitHub repository: https://github.com/ChronaNet.

Abstract

ChronaNet is an innovative decentralized network that leverages quantum physics principles, E8 geometry, and advanced cryptography to deliver a blockchain system with a novel consensus mechanism called **Proof of Coherence (PoC)**. This mechanism uses temporal resonance and spectral analysis to validate transactions and blocks, ensuring high security and energy efficiency. ChronaNet introduces **Chronacoins**, a native currency to incentivize validator nodes, and features a distributed architecture with HTTP communication, persistent storage, and a real-time web interface. This whitepaper describes the technical architecture, key components, and steps to join the public testnet.

2 Introduction

2.1 Motivation

Traditional blockchain networks, such as Bitcoin and Ethereum, rely on resource-intensive consensus mechanisms like Proof of Work or complex systems like Proof of Stake. ChronaNet proposes an alternative approach based on **spectral coherence** and temporal resonance, utilizing quantum physics and advanced mathematics to create a secure, scalable, and sustainable system.

2.2 Objectives

- Develop a decentralized network with low energy consumption.
- Implement a consensus mechanism based on the coherence of quantum tesseracts.
- Provide an intuitive interface for real-time network visualization.
- Launch an accessible and scalable public testnet.

3 ChronaNet Architecture

ChronaNet consists of interconnected nodes managing a persistent blockchain. Each node contains a set of **tesseracts** with E8 geometry, a **wallet** for Chronacoins, and an HTTP communication layer for synchronizing transactions and blocks. The following figure illustrates the global architecture:

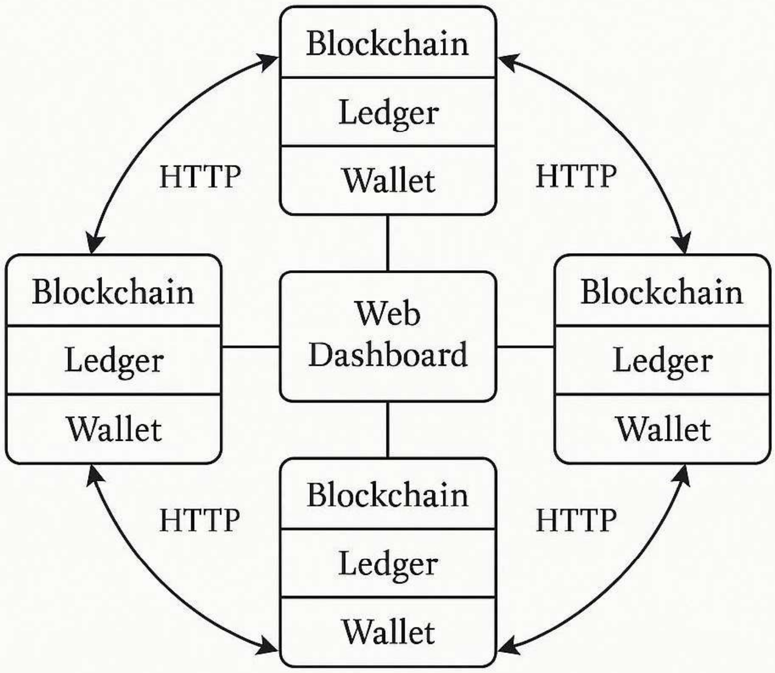


Figure 1: ChronaNet Architecture: Nodes are interconnected in a circular topology, each with a blockchain, ledger, and wallet. Bidirectional arrows represent HTTP communication. A central

Figure 1: ChronaNet Architecture: Nodes are interconnected in a circular topology, each with a blockchain, ledger, and wallet. Bidirectional arrows represent HTTP communication. A central web dashboard displays the network state.

3.1 Key Components

- **Tesseract**: A quantum entity with a triad (a, b, c), phase, and intensity, computing an angular frequency (`omega`) based on unique prime factors.
- **CronaWallet**: Manages identity, Chronacoin balance, and transaction history, signing transactions with a live phase key.
- **ChronaBlockChain**: Stores blocks with validated transactions, coherence, spectral hash, and previous hash.
- **ChronaNetNode**: Coordinates transaction validation, block creation, and inter-node communication.
- **Web Interface**: A real-time dashboard built with Streamlit for network visualization.

4 Consensus Mechanism: Proof of Coherence (PoC)

4.1 Foundations

The **Proof of Coherence (PoC)** mechanism relies on the spectral coherence of the central tesseract (T8) in each node. Coherence is calculated as:

$$C(t) = \frac{\left|\sum_{\text{active nodes}} \psi_{T8}(t)\right|}{\sum_{\text{active nodes}} |\psi_{T8}(t)|}$$

where $\psi_{T8}(t) = I \cdot e^{i(\omega t+\phi)}$ is the quantum state of the T8 tesseract at time t, with intensity I, angular frequency ω, and phase ϕ. A coherence $C \geq 0.707$ is required to validate transactions and blocks.

4.2 Spectral Validation

Each transaction and block includes a **spectral hash** computed via the Fast Fourier Transform (FFT) of the T8 tesseract's quantum state. The following code implements this:

```python
def spectral_hash(self, t, dt=0.01):
    t_vec = np.arange(0, t, dt)
    psi_vec = [self.t8.psi(ti) for ti in t_vec]
    return fft(psi_vec)
```

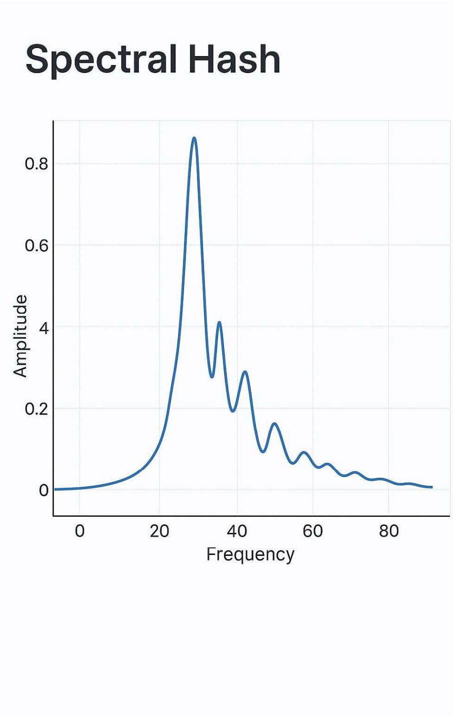


Figure 2: Spectral Hash: A line plot showing the amplitude of the spectral hash (FFT of ψ_{T8}) versus frequency, highlighting dominant spectral components.

Validation compares the received spectral hash with the locally computed one, with a tolerance of 10^{-3}.

5 Blockchain Structure

Each ChronaNet block has the following structure:

```
{
  "block_id": 1,
  "timestamp": 1622917200.12345,
  "transactions": [
    {
      "sender": "Node_1",
      "recipient": "Node_2",
      "value": 100,
      "timestamp": 0.12345,
      "tavari": "Hg-Ax-Ro",
      "hash": [...],
      "phase_key": 0.987
    }
  ],
  "coherence": 0.85,
  "t8_hash": [1.23, 0.45, ...],
  "prev_hash": "0",
  "block_hash": "a1b2c3..."
}
```

- **Persistence**: Blocks are stored in `blockchain_Node_X.json`.
- **Synchronization**: Upon startup, nodes query peers and adopt the longest chain with the highest total coherence.

6 Rewards System

Validator nodes receive **Chronacoins** for creating blocks. The reward is calculated as:

$$\text{Reward} = \text{REWARD\_BASE} \cdot C$$

where `REWARD_BASE = 10.0` and C is the block's coherence. The implementation is:

```
if len(self.blockchain.current_transactions) >= 3:
    prev_hash = self.blockchain.chain[-1]["block_hash"] if self.blockchain.chain else "0"
    block = self.blockchain.create_block(t, C, self.spectral_hash(t), prev_hash)
    reward = REWARD_BASE * C
    self.wallet.update_balance(reward, "reward", {"block_id": block["block_id"]})
```

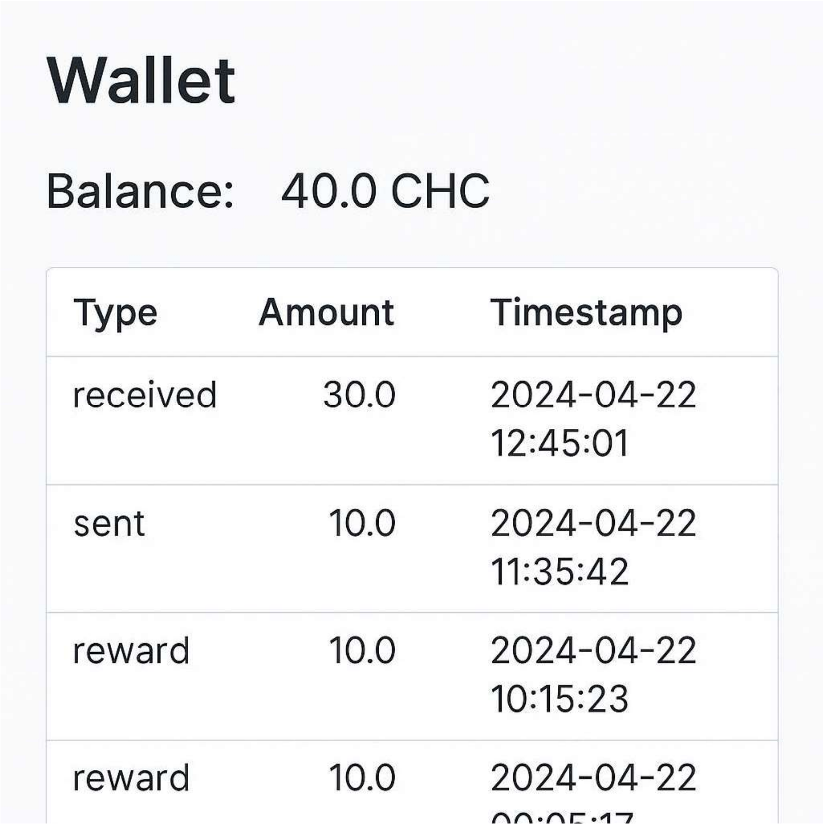
7 Wallet and Chronacoin Management

Each node has a **CronaWallet** with:

- **Balance**: Available Chronacoins.
- **History**: Record of sent, received, and reward transactions.
- **Persistence**: Stored in `wallet_Node_X.json`.

Example wallet file:

```json
{
  "balance": 925.0,
  "history": [
    {
      "type": "sent",
      "amount": -100,
      "txn": {...},
      "timestamp": 1622917200.12345
    },
    {
      "type": "reward",
      "amount": 8.5,
      "txn": {"block_id": 1},
      "timestamp": 1622917201.23456
    }
  ]
}
```



Wallet

Balance: 40.0 CHC

Type	Amount	Timestamp
received	30.0	2024-04-22 12:45:01
sent	10.0	2024-04-22 11:35:42
reward	10.0	2024-04-22 10:15:23
reward	10.0	2024-04-22 00:05:17

Figure 3: Wallet History: A table displaying the current balance and transaction history, with columns for type, amount, and timestamp.

8 Inter-Node Communication

Nodes communicate via HTTP using Flask, with the following endpoints:

- `POST /transaction`: Receives and validates transactions.
- `POST /block`: Receives and adds propagated blocks.
- `GET /chain`: Returns the current blockchain state.

Example block propagation:

```python
def propagate_block(self, block):
    for peer in self.peers:
        try:
            response = requests.post(f"{peer}/block", json=block)
            logger.info(f"Block␣propagated␣to␣{peer}")
        except requests.RequestException as e:
            logger.error(f"Connection␣error␣with␣{peer}:␣{e}")
```

9 Web Interface with Streamlit

The ChronaNet dashboard, built with Streamlit, displays:

- **Node Status**: Active/inactive, balance, number of blocks.
- **Blocks**: Details of the latest block per node.
- **Transaction History**: Table of wallet transactions.
- **Coherence**: Real-time line chart of network coherence.

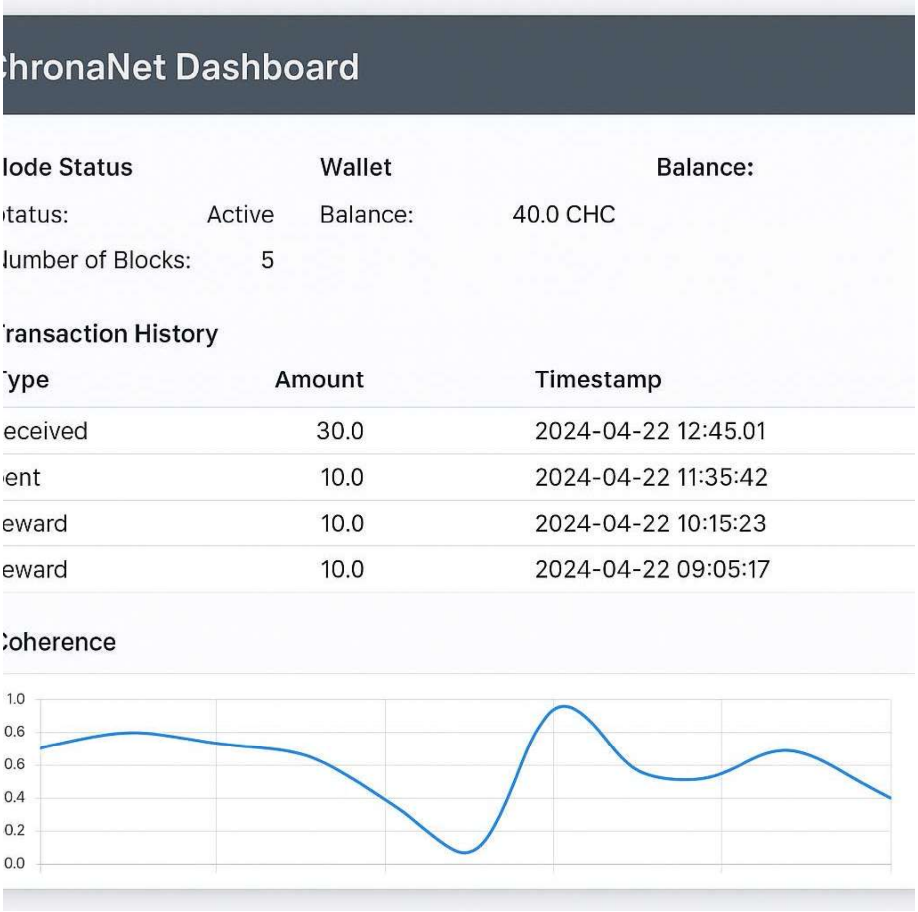
Implementation:

```python
def start_web_interface(nodes):
    st.title("ChronaNet␣Dashboard")
    st.header("Network␣Status")
    for node in nodes:
        st.subheader(f"Node␣{node.node_id}")
        st.write(f"Status:␣{'Active'␣if␣node.is_active␣else␣'Inactive'}
            ")
        st.write(f"Balance:␣{node.wallet.balance}␣Chronacoins")
        st.write(f"Number␣of␣Blocks:␣{len(node.blockchain.chain)}")
        if node.blockchain.chain:
            st.write("Latest␣Block:")
            st.json(node.blockchain.chain[-1])
        st.write("Transaction␣History:")
        st.json(node.wallet.history)
        t = datetime.now().timestamp()
        coherence = node.coherence(t, nodes)
        st.write(f"Current␣Coherence:␣{coherence:.3f}")
        st.line_chart([coherence])
```

10 Public Testnet Deployment

ChronaNet is ready for public deployment on Google Cloud Platform (GCP) using Cloud Run and Docker. Steps include:

1. **Build Docker Image**:



hronaNet Dashboard

lode Status

tatus: Active

lumber of Blocks: 5

Wallet

Balance: 40.0 CHC

Balance:

ransaction History

ype	Amount	Timestamp
eceived	30.0	2024-04-22 12:45.01
ent	10.0	2024-04-22 11:35:42
eward	10.0	2024-04-22 10:15:23
eward	10.0	2024-04-22 09:05:17

oherence

Figure 4: ChronaNet Dashboard: A web interface showing node sections with balance and history tables, and a coherence line chart.

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "chronanet.py"]
```

2. **Push to Google Container Registry**:

```
gcloud auth configure-docker
docker tag chronanet gcr.io/[PROJECT-ID]/chronanet
docker push gcr.io/[PROJECT-ID]/chronanet
```

3. **Deploy Nodes to Cloud Run**:

```
gcloud run deploy chronanet-node1 \
  --image gcr.io/[PROJECT-ID]/chronanet \
  --platform managed \
  --region us-central1 \
  --port 5000 \
  --allow-unauthenticated
```

4. **Deploy Web Interface**:

   ```
   gcloud run deploy chronanet-web \
     --image gcr.io/[PROJECT-ID]/chronanet \
     --platform managed \
     --region us-central1 \
     --port 8501 \
     --allow-unauthenticated \
     --command "streamlit run chronanet.py"
   ```

5. **Public Repository**: Publish code on GitHub with a detailed README.

11 Security and Scalability

- **Security**: Transactions are signed with live phase keys. Blocks are validated using spectral hashes and coherence. HTTP endpoint authentication is recommended for production.
- **Scalability**: Cloud Run auto-scales nodes based on load. Automatic chain synchronization ensures consistency in large networks.

12 Conclusions and Roadmap

ChronaNet introduces a groundbreaking approach to decentralized networks using quantum-inspired consensus. Future milestones include:

- **Q3 2025**: Public testnet launch.
- **Q4 2025**: Enhanced consensus and authentication mechanisms.
- **2026**: Mainnet launch with decentralized application support.

13 Joining the Testnet

1. Clone the repository: `git clone https://github.com/ChronaNet`.
2. Install dependencies: `pip install -r requirements.txt`.
3. Run a node: `python chronanet.py`.
4. Add initial peers: `https://node1-xyz.run.app:5000`, `https://node2-xyz.run.app:5001`.
5. Access the dashboard: `https://chronanet-web-xyz.run.app`.

14 Acknowledgments

We thank the developer and research community for inspiring this project, as well as open-source tools (Python, Flask, Streamlit) that made ChronaNet possible.

15 Appendix: Full Code

The complete code is available at GitHub. Below is the key snippet for spectral validation and block creation:

```python
def validate_transaction(self, txn, t, network_nodes):
    if not self.is_active:
        logger.error(f"Node␣{self.node_id}␣inactive.␣Validation␣failed.")
        return False
    C = self.coherence(t, network_nodes)
    if C < COHERENCE_THRESHOLD:
        logger.warning(f"Insufficient␣coherence:␣C={C:.3f}␣<␣{COHERENCE_THRESHOLD}")
        return False
    expected_hash = self.spectral_hash(t)
    hash_diff = abs(np.array(txn[’hash’]) - expected_hash).sum()
    if hash_diff > 1e-3:
        logger.error(f"Spectral␣hash␣mismatch:␣diff={hash_diff}")
        return False
    phase_key_valid = abs(txn[’phase_key’] - abs(self.wallet.phase_key)) < 1e-2
    if not phase_key_valid:
        logger.error("Invalid␣phase␣key.")
        return False
    logger.info(f"Transaction␣validated:␣{txn[’sender’]}␣->␣{txn[’recipient’]}")
    self.save_transaction(txn, valid=True)
    self.blockchain.add_transaction(txn)
    if len(self.blockchain.current_transactions) >= 3:
        prev_hash = self.blockchain.chain[-1]["block_hash"] if self.blockchain.chain else "0"
        block = self.blockchain.create_block(t, C, self.spectral_hash(t), prev_hash)
        reward = REWARD_BASE * C
        self.wallet.update_balance(reward, "reward", {"block_id": block["block_id"]})
        logger.info(f"Block␣created␣by␣node␣{self.node_id}:␣{block[’block_id’]}␣with␣{len(block[’transactions’])}␣transactions.")
        self.propagate_block(block)
    return True
```