Lamassu Lightning Network Integration Analysis

Overview

This document provides a comprehensive technical analysis of how the Lamassu Bitcoin ATM server integrates with the Lightning Network to process Bitcoin Lightning transactions. The analysis covers the complete flow from hardware interaction to Lightning invoice creation and payment processing.

Lightning Network Architecture

Lamassu implements Lightning Network support through the **Galoy wallet plugin**, which acts as a Lightning service provider rather than running a local Lightning node.

Key Architecture Decision: Despite the presence of a `LIGHTNING_NETWORK_DAEMON` environment variable, Lightning functionality is actually handled via **Galoy's hosted GraphQL API**, not a local Lightning daemon.

Core Components

- **Transaction Coordinator**: Routes Lightning transactions through the cash-in/cash-out pipeline
- **Galoy Wallet Plugin**: Interfaces with Galoy's Lightning infrastructure via GraphQL
- **Invoice Management**: Creates, monitors, and processes Lightning invoices
- **Payment Probing**: Tests Lightning payment routes and limits

Lightning Invoice Creation Flow

When a customer wants to purchase $100 of Bitcoin Lightning, the following sequence occurs:

1. Machine Initiates Transaction Request

The ATM machine sends a POST request to the server:

```
POST /tx
Content-Type: application/json

{
  "direction": "cashIn",
  "cryptoCode": "LN",
  "fiat": 10000,
  "cryptoAtoms": "546448",
  "isLightning": true,
  "deviceId": "<machine-id>",
  "txVersion": 1
}
```

Parameters Explained: - `fiat`: Amount in cents ($100.00 = 10000 cents) - `cryptoAtoms`: Equivalent amount in satoshis - `cryptoCode`: "LN" indicates Lightning Network - `isLightning`: Boolean flag for Lightning transactions

2. Server Processing Pipeline

Route Handler **File**: `packages/server/lib/routes/txRoutes.js:13-49`

```
function postTx(req, res, next) {
  const pi = plugins(req.settings, req.deviceId)
  return Tx.post(_.set('deviceId', req.deviceId, req.body), pi)
    .then(tx => res.json(tx))
```

```
    .catch(next)
}
```

Transaction Coordinator **File**: packages/server/lib/tx.js:13-18

```
function process(tx, pi) {
  const mtx = massage(tx)
  if (mtx.direction === 'cashIn') return CashInTx.post(mtx, pi)  // Lightning routing
  if (mtx.direction === 'cashOut') return CashOutTx.post(mtx, pi)
  return Promise.reject(new Error('No such tx direction: ' + mtx.direction))
}
```

Cash-In Handler **File**: packages/server/lib/cash-in/cash-in-tx.js:154-155

```
return pi
  .newAddress(r.tx)  // Creates Lightning invoice
  .then(txObj => {
    // Process the invoice response
    if (txObj.batched) {
      return txBatching.addUnconfirmedTx(r.tx)
    }
    // Handle immediate processing
  })
```

3. Lightning Invoice Generation

Plugin System Routing **File**: packages/server/lib/plugins.js:427-437

```
function newAddress(tx) {
  const info = {
    cryptoCode: tx.cryptoCode,      // "LN"
    label: 'TX ' + Date.now(),
    hdIndex: tx.hdIndex,
    cryptoAtoms: tx.cryptoAtoms,    // Amount in satoshis
    isLightning: tx.isLightning     // true
  }
  return wallet.newAddress(settings, info, tx)  // Calls Galoy plugin
}
```

Galoy Lightning Plugin **File**: packages/server/lib/plugins/wallet/galoy/galoy.js:310-320

```
function newAddress(account, info, tx) {
  const { cryptoAtoms, cryptoCode } = tx
  return checkCryptoCode(cryptoCode).then(() =>
    newInvoice(
      account.walletId,
      cryptoAtoms,           // Amount in satoshis
      account.apiSecret,
      account.endpoint,
    ),
  )
}
```

4. Galoy GraphQL Lightning Invoice Creation

Invoice Creation Function **File**: packages/server/lib/plugins/wallet/galoy/galoy.js:279-298

```
function newInvoice(walletId, cryptoAtoms, token, endpoint) {
  const createInvoice = {
    operationName: 'lnInvoiceCreate',
    query: `mutation lnInvoiceCreate($input: LnInvoiceCreateInput!) {
      lnInvoiceCreate(input: $input) {
        errors {
          message
          path
        }
        invoice {
          paymentRequest    // Lightning invoice string
        }
      }
    }`,
    variables: {
      input: {
        walletId,
        amount: cryptoAtoms.toString()
      }
    }
  }

  return request(createInvoice, token, endpoint).then(result => {
    return result.data.lnInvoiceCreate.invoice.paymentRequest  // Returns "lnbc..."
  })
}
```

GraphQL Request Handler **File**: packages/server/lib/plugins/wallet/galoy/galoy.js:10-28

```
function request(graphqlQuery, token, endpoint) {
  const headers = {
    'content-type': 'application/json',
    'X-API-KEY': token,
  }
  return axios({
    method: 'post',
    url: endpoint,
    headers: headers,
    data: graphqlQuery,
  })
  .then(r => {
    if (r.error) throw r.error
    return r.data
  })
}
```

5. Response to Machine

The server returns the Lightning invoice to the ATM machine:

```
{
  "id": "tx-uuid-12345",
  "toAddress": "lnbc5464480n1pn2s...",  // Lightning invoice
  "layer2Address": null,
  "cryptoCode": "LN",
```

```
  "direction": "cashIn",
  "fiat": 10000,
  "cryptoAtoms": "546448",
  "status": "authorized"
}
```

Complete Machine-Server Interaction Flow

```
[ATM Machine]                    [Lamassu Server]              [Galoy API]

      1. POST /tx
      {direction:"cashIn", LN, $100}

                                   2. Process transaction
                                     - Route to cash-in
                                     - Extract amount & type

                                   3. GraphQL newInvoice

                                     lnInvoiceCreate mutation

                                   4. Lightning invoice

                                     "lnbc5464480n1..."

      5. Transaction response
        {toAddress: "lnbc..."}


      6. Display QR code to user
         Show Lightning invoice

      7. Poll for payment status
      GET /tx/:id?status=confirmed

                                   8. Check invoice status

                                     lnInvoicePaymentStatus

                                   9. Payment status

      10. Confirmation response      "PAID" or "PENDING"
         {status: "confirmed"}


      11. Dispense cash to user
```

Lightning Payment Detection and Processing

Address Type Detection

File: packages/server/lib/plugins/wallet/galoy/galoy.js:94-100

```
function isLnInvoice(address) {
  return address.toLowerCase().startsWith('lnbc')  // Lightning mainnet invoices
```

```
}

function isLnurl(address) {
  return address.toLowerCase().startsWith('lnurl')  // Lightning URL format
}
```

Payment Status Monitoring

File: `packages/server/lib/plugins/wallet/galoy/galoy.js:322-339`

```
function getInvoiceStatus(token, endpoint, address) {
  const query = {
    operationName: 'lnInvoicePaymentStatus',
    query: `query lnInvoicePaymentStatus($input: LnInvoicePaymentStatusInput!) {
      lnInvoicePaymentStatus(input: $input) {
        status    // Returns "PENDING" or "PAID"
      }
    }`,
    variables: { input: { paymentRequest: address } }
  }

  return request(query, token, endpoint)
    .then(r => r?.data?.lnInvoicePaymentStatus?.status)
    .catch(err => {
      throw new Error(err)
    })
}
```

Status Polling Implementation

The machine continuously polls the server to check if the Lightning payment has been received:

Route: `/tx/:id?status=confirmed` **Handler**: `packages/server/lib/routes/txRoutes.js:51-65`

```
function getTx(req, res, next) {
  if (req.query.status) {
    return helpers
      .fetchStatusTx(req.params.id, req.query.status)
      .then(r => res.json(r))
      .catch(err => {
        if (err.name === 'HTTPError') {
          return res.status(err.code).send(err.message)
        }
        next(err)
      })
  }
  return next(httpError('Not Found', 404))
}
```

Lightning Transaction Capabilities

Supported Operations

1. **Invoice Creation** (`newAddress`)
   - Generates Lightning invoices for incoming payments
   - Specifies exact amount in satoshis
   - Returns BOLT11 payment request string

2. **Payment Sending** (`sendCoins`)
   - Pays existing Lightning invoices
   - Handles both amount-specified and no-amount invoices
   - Supports LNURL payments
3. **Status Checking** (`getStatus`)
   - Monitors payment confirmation status
   - Provides real-time payment updates
   - Handles payment timeouts and failures
4. **Balance Queries** (`balance`)
   - Checks Lightning wallet balance
   - Returns available satoshi amounts
   - Accounts for pending transactions
5. **Payment Probing** (`probeLN`)
   - Tests Lightning payment routes
   - Validates payment feasibility
   - Determines supported amounts

Payment Route Probing

File: `packages/server/lib/plugins/wallet/galoy/galoy.js:244-254`

```
function probeLN(account, cryptoCode, invoice) {
  const probeHardLimits = [200000, 1000000, 2000000]  // Test limits in satoshis
  const promises = probeHardLimits.map(limit => {
    return sendProbeRequest(
      account.walletId,
      invoice,
      limit,
      account.apiSecret,
      account.endpoint,
    ).then(r => _.isEmpty(r.errors))  // Check if amount is routable
  })
  return Promise.all(promises)  // Return array of supported limits
}
```

Probe Request Implementation:

```
function sendProbeRequest(walletId, invoice, cryptoAtoms, token, endpoint) {
  const sendProbeNoAmount = {
    operationName: 'lnNoAmountInvoiceFeeProbe',
    query: `mutation lnNoAmountInvoiceFeeProbe($input: LnNoAmountInvoiceFeeProbeInput!) {
      lnNoAmountInvoiceFeeProbe(input: $input) {
        amount
        errors {
          message
          path
        }
      }
    }`,
    variables: {
      input: {
        paymentRequest: invoice,
        walletId,
        amount: cryptoAtoms.toString(),
      },
    },
```

```
  }
  return request(sendProbeNoAmount, token, endpoint)
}
```

Lightning Cash-Out (Payment) Flow

For outgoing Lightning payments (when user wants to send Lightning Bitcoin):

Payment Processing

File: packages/server/lib/plugins/wallet/galoy/galoy.js:196-242

```
function sendCoins(account, tx) {
  const { toAddress, cryptoAtoms, cryptoCode } = tx
  return checkCryptoCode(cryptoCode)
    .then(() => {
      if (isLnInvoice(toAddress)) {
        return sendFundsLN(
          account.walletId,
          toAddress,            // Lightning invoice to pay
          cryptoAtoms,
          account.apiSecret,
          account.endpoint,
        )
      }

      if (isLnurl(toAddress)) {
        return sendFundsLNURL(
          account.walletId,
          toAddress,            // LNURL to pay
          cryptoAtoms,
          account.apiSecret,
          account.endpoint,
        )
      }

      // Handle on-chain addresses
      return sendFundsOnChain(...)
    })
}
```

Lightning Payment Execution

```
function sendFundsLN(walletId, invoice, cryptoAtoms, token, endpoint) {
  const sendLnNoAmount = {
    operationName: 'lnNoAmountInvoicePaymentSend',
    query: `mutation lnNoAmountInvoicePaymentSend($input: LnNoAmountInvoicePaymentInput!) {
      lnNoAmountInvoicePaymentSend(input: $input) {
        errors {
          message
          path
        }
        status  // Payment result status
      }
    }`,
```

```
    variables: {
      input: {
        paymentRequest: invoice,
        walletId,
        amount: cryptoAtoms.toString(),
      },
    },
  }
  return request(sendLnNoAmount, token, endpoint)
}
```

Configuration and Environment

Environment Variables

File: `packages/server/.sample.env:38`

```
LIGHTNING_NETWORK_DAEMON=
```

Note: This variable exists but is not used. Lightning functionality is provided through Galoy's API configuration in the admin interface.

Galoy Account Configuration

Required configuration parameters for Galoy integration: - `walletId`: Galoy wallet identifier - `apiSecret`: Authentication token for Galoy API - `endpoint`: Galoy GraphQL API endpoint URL

Supported Cryptocurrency Codes

File: `packages/server/lib/plugins/wallet/galoy/galoy.js:5-6`

```
const NAME = 'LN'
const SUPPORTED_COINS = ['LN']
```

Error Handling and Edge Cases

Lightning-Specific Error Handling

1. **Invoice Validation**
   - Validates Lightning invoice format (starts with "lnbc")
   - Checks invoice expiration
   - Verifies payment request integrity
2. **Payment Routing Failures**
   - Handles insufficient Lightning liquidity
   - Manages routing failures
   - Provides fallback mechanisms
3. **Network Connectivity Issues**
   - Implements retry logic for Galoy API calls
   - Handles temporary network failures
   - Maintains transaction state during outages

Transaction State Management

Lightning transactions follow these states: - `pending`: Invoice created, awaiting payment - `authorized`: Payment detected but not confirmed - `confirmed`: Payment fully confirmed - `expired`: Invoice expired without payment - `failed`: Payment attempt failed

Security Considerations

API Security

- All Galoy API requests use authenticated connections
- API keys are securely stored and transmitted
- GraphQL queries are parameterized to prevent injection

Lightning Network Security

- Invoices have built-in expiration times
- Payment amounts are validated before processing
- Route probing prevents overpayment attempts

Transaction Integrity

- Database transactions ensure atomicity
- Serializable isolation prevents race conditions
- Transaction versioning prevents double-processing

Performance Characteristics

Lightning Advantages

- **Instant Payments**: Lightning payments confirm in seconds
- **Low Fees**: Minimal network fees compared to on-chain
- **Scalability**: High transaction throughput capability

System Performance

- **GraphQL Efficiency**: Single API calls for complex operations
- **Caching**: Invoice status results cached for performance
- **Batching**: Multiple probe requests processed in parallel

Monitoring and Observability

- Transaction logs include Lightning-specific metadata
- Payment routing information tracked for debugging
- API response times monitored for performance optimization

Conclusion

The Lamassu Lightning Network integration provides a complete solution for instant Bitcoin transactions through ATMs. By leveraging Galoy's hosted Lightning infrastructure, the system achieves the benefits of Lightning Network payments without the operational complexity of running local Lightning nodes.

Key strengths of this implementation: - **Reliable Infrastructure**: Galoy provides enterprise-grade Lightning services - **Simple Integration**: GraphQL API provides clean, documented interfaces
- **Real-time Processing**: Instant payment detection and confirmation - **Comprehensive Features**: Supports full Lightning payment lifecycle - **Robust Error Handling**: Graceful failure modes and recovery mechanisms

This architecture enables Lamassu ATMs to offer Lightning Network functionality while maintaining the reliability and security required for financial hardware deployment.