Lightning Applications on Unfairly Cheap Hardware

Elaine Ou
Chaincode Lightning Residency
October 23 2018

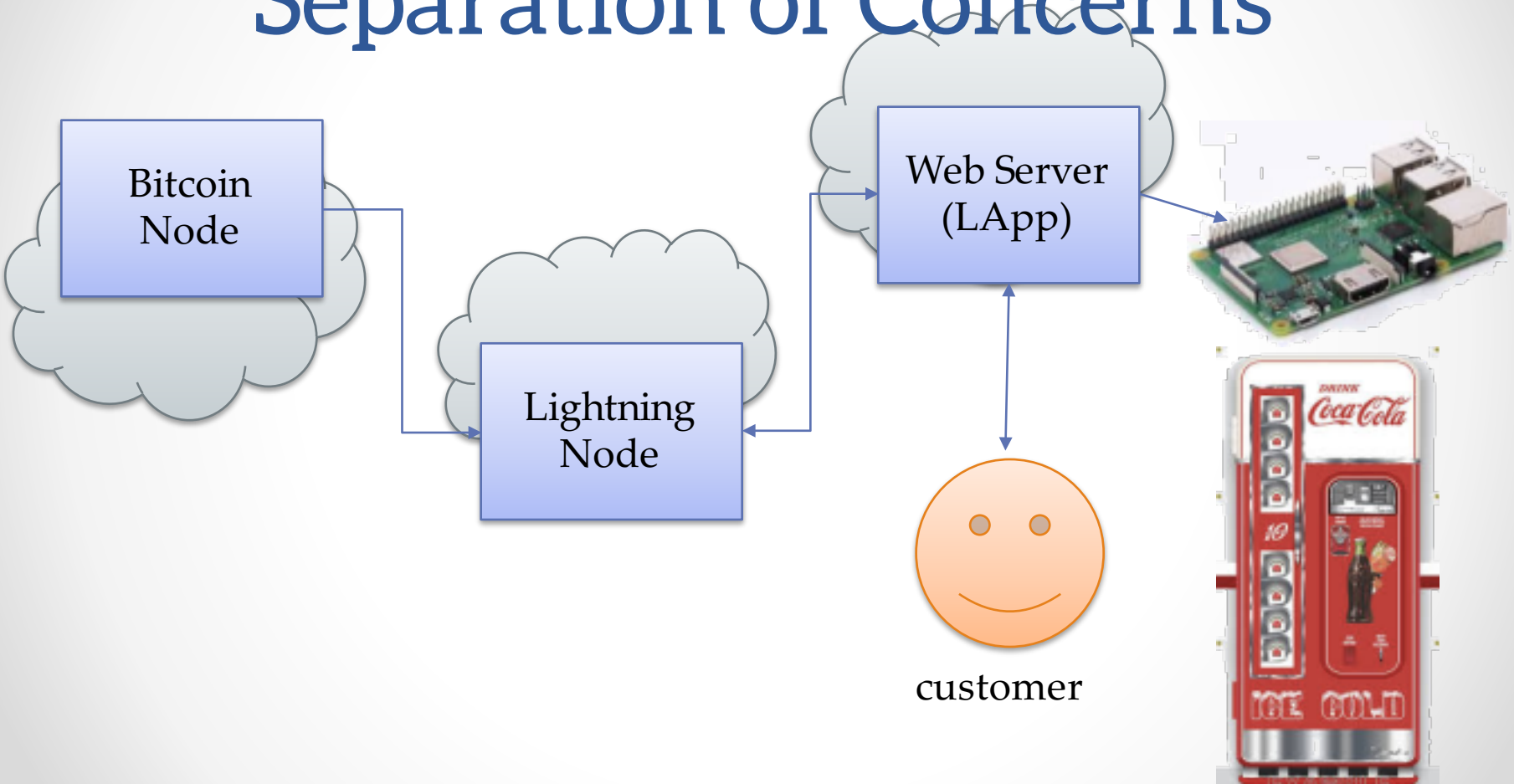
Building for Light Clients

- Constraints of Unfairly Cheap Hardware
- Optimizing for Minimal Hardware
- Raspberry Pi Example: Lightning Peep Show

Unfairly Cheap Hardware

- Potential Challenges
  - Unreliable connection
  - Non-static IP address
  - Low-bandwidth connection
  - Computational resource constraints

Separation of Concerns



Put it all on one machine

- Full Bitcoin node + LN node + LApp
  - Least hardware

- Tradeoffs
  - Storage requirements limits mobility
  - Network challenges remain: unstable connection, low bandwidth, need access to router
  - Unreliable LN channels

Basic Setup: Raspberry Pi

- OS: Ubuntu MATE (64-bit) or Raspbian
- External HDD: 200+ GB (less with pruning)
  - Power adapter needed
  - Can use microSD, but will suffer flash degradation over time
  - Bitcoin Core will need enlarged swap
- Port Forwarding (how will users access your LApp?)

- https://github.com/Stadicus/guides/tree/master/raspibolt
  - LND
- https://medium.com/@meeDamian/c-lightning-node-on-rbp3-b950660fb835
  - C-Lightning

Lightning in Light Client Mode

- Hardware constraints relaxed
- Don’t need a full Bitcoin node to run a Lightning node
  - both parties have to sign off on each transaction
  - both can stay aware of all relevant channel txs without monitoring the blockchain

sPRUNED: Bitcoin Lightweight Pseudonode

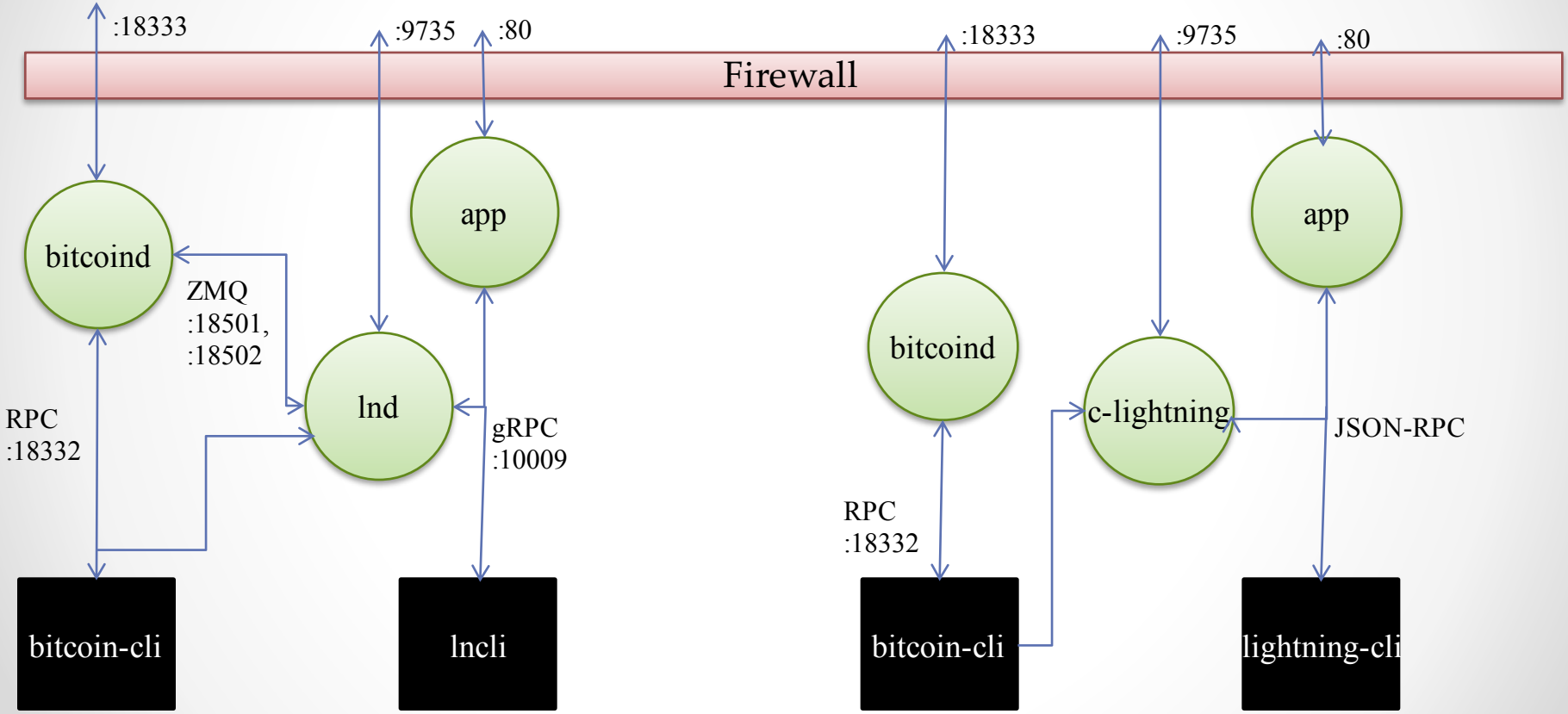
- Fetches block headers and transactions from Electrum network, blocks from bitcoin p2p network
  - Only needs ~300 mb storage
- Can be used with c-Lightning and bitcoin-cli
- ~~Suboptimal: Can compromise privacy~~
- Suboptimal: Need enough bandwidth to download all blocks

https://github.com/gdassori/spruned

Neutrino: Privacy-Preserving Light Client

- BIP-157: Full nodes generate filters on block data
  - Client downloads and validates chain of block headers
  - Client downloads and validates chain of filter headers
  - Client determines whether a block contains relevant info
  - Can download block from any source
- Less likely to miss relevant transactions
  - Filter-matching done by client, not the remote node
- Lightweight
  - Filters don't need to be stored
  - Filter bandwidth ~70MB/month

Network Configuration



Network Configuration

- LApp and Lightning node need open ports
- ufw firewall

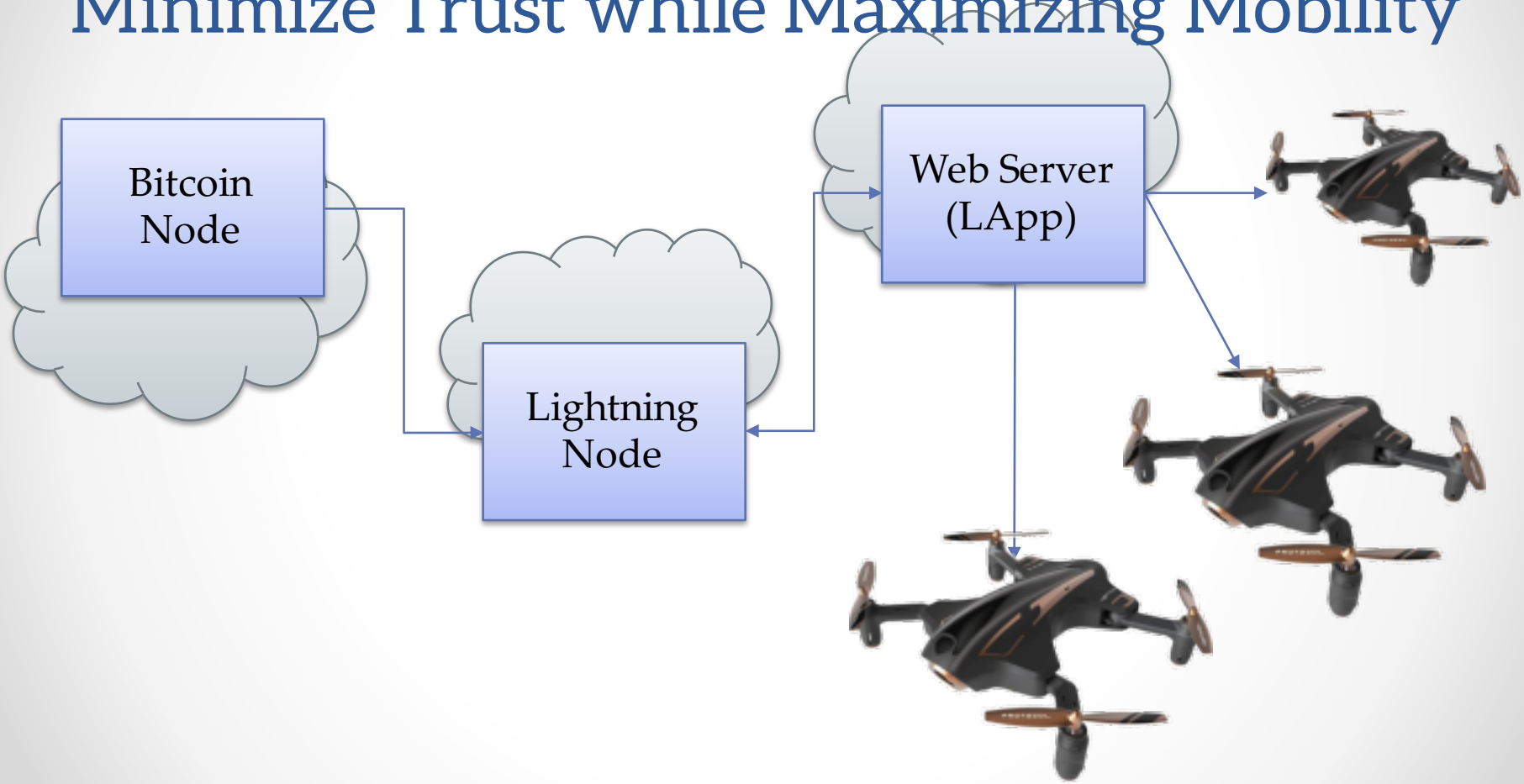
```
$ ufw default deny incoming
$ ufw default allow outgoing
$ ufw allow from <ip address> to any port 22 comment 'allow SSH from my IP'
$ ufw allow 80  comment 'allow LApp'
$ ufw allow 9735  comment 'allow Lightning'
$ ufw allow 18333 comment 'allow Bitcoin testnet'
$ ufw enable
$ systemctl enable ufw
```

- https://www.digitalocean.com/community/tutorials/ufw-essentials-common-firewall-rules-and-commands

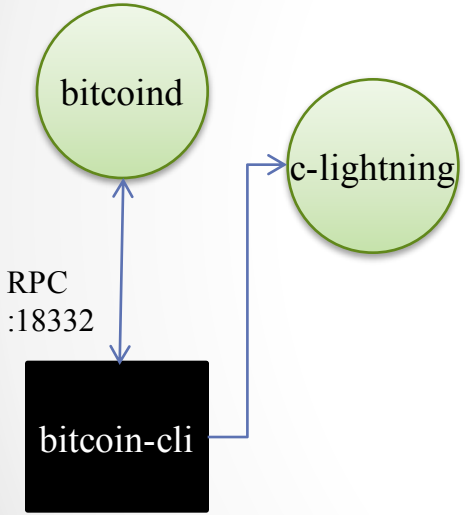
Configuring DHCP

- https://bitcoin.org/en/full-node#enabling-connections
- Limitations still exist
  - what if you don’t have access to the router?

Minimize Trust while Maximizing Mobility



Remote Bitcoin Node (c-Lightning)




```
#bitcoin.conf
server=1
rpcport=18332
rpcuser=<user>
rpcpassword=<password>
rpcallowip=<lnd_node_ip>
```

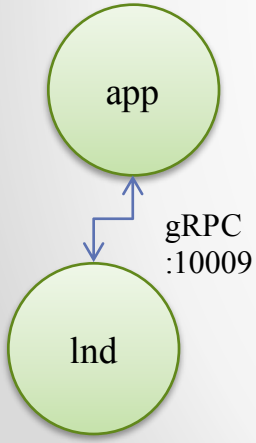
```
#~/.lightning/config
bitcoin-rpcconnect=<remoteip>
bitcoin-rpcport=18332
bitcoin-rpcuser=<user>
bitcoin-rpcpassword=<password>
```

- https://github.com/bitcoin/bitcoin/tree/master/share/rpcauth

Remote Lightning Node (LND)

- LND will generate tls certificate for you

```
;lnd.conf
[Application Options]
tlsextraip=<lnd_node_ip>
tlsextradomain=<lnd_node_domain>
rpclisten=0.0.0.0:10009
```



- Copy ~/.lnd/tls.cert to remote client

```
import rpc_pb2 as ln
import rpc_pb2_grpc as lnrpc
import grpc

cert = open(<CERT/PATH/tls.cert>, 'rb').read()
creds = grpc.ssl_channel_credentials(cert)
channel = grpc.secure_channel(<lnd_node_ip:10009>, creds)
```

Example:
Lightning Peep Show

The Hardware

Web Server (LApp)

- Run an app on the pi
  - This will serve a camera stream on port :8080

```
pi$ raspistill --nopreview -w 640 -h 480 -tl 100 -o /tmp/stream/pic.jpg

pi$ mjpg_streamer -i "input_file.so -f /tmp/stream -n pic.jpg" -o "output_http.so -w
/usr/local/www"
```

- Create a reverse ssh tunnel from pi to web server

```
pi$ autossh -R <webserver.address>:<proxy_port>:127.0.0.1:8080 <user@webserver.address>
```

- Now the webserver's proxy_port is a tunnel to the pi on :8080

- https://github.com/elaineo/peepshow

The Webserver

- Run a relay Lapp that re-broadcasts the camera stream conditional on lightning invoice payment

```
pi$ autossh -R lapp.elaineou.com:1234:127.0.0.1:8080  user@lapp.elaineou.com
```

localhost:1234

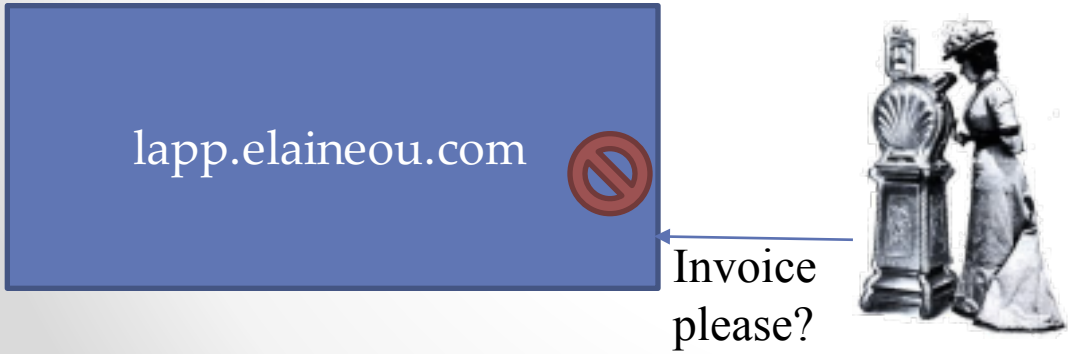
lapp.elaineou.com

:80

The LApp

```
def get_invoice(self, memo="Peepshow"):
    try:
        request = ln.Invoice(
            memo=memo,
            value=self.DEFAULT_PRICE,
            expiry=self.DEFAULT_EXPIRY
        )
        response = self.stub.AddInvoice(request)
        return MessageToJson(response)
    except grpc.RpcError as e:
        logging.error(e)
        return e.details()
```

- https://api.lightning.community/#



The LApp


```
{
  "r_hash":
"mnQqp1ipg7iEX8ZpMN9EYiI2i0RaeVIR9wwYAxkPkSo
=",
  "payment_request":
"lntb9990n1pduarzxpp5nf6z4f6c4xpm3pzlce5nph6
yvg3rdz6ytfu4yy0hpsvqxxg0jy4qdqd2pjk2urndphh
wcqzysxqyz5vqv30gahgrtgt58vjq9xmwledkfynckdq
urdgcf6ulpsts9y2gxcnsgd74jyflpevywqpxdacy4h6
cx2hdhxkqr9cxmdguvflmvftgfnqpc2qq47”
}
```


lapp.elaineou.com

invoice

The LApp

```python
def subscribe_invoices(self):
    try:
        request = ln.InvoiceSubscription()
        invoices = self.stub.SubscribeInvoices(request)
        for invoice in invoices:
            self.broadcast.updateClients(MessageToJson(invoice))
    except grpc.RpcError as e:
        logging.error(e)
        return e.details()
```

lapp.elaineou.com

wss://

```javascript
success: function(response) {
    var websocketAddress = "wss://lapp.elaineou.com:8675/?r_hash=";
    var websocket = new WebSocket(websocketAddress +
        encodeURIComponent(response.r_hash));

    websocket.onmessage = function (event) {
        renderImage(event.data);
    };
```

The LApp

```
def updateClients(self, invoice):
    for client in self.clients:
        if client.r_hash == invoice["r_hash"]:
            if invoice["settled"]:
                client.paid = invoice["settled"]
                client.timeConnected = datetime.now()
```

lapp.elaineou.com

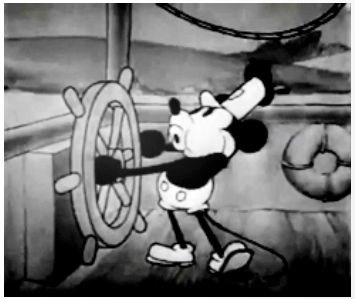
wss://

The LApp

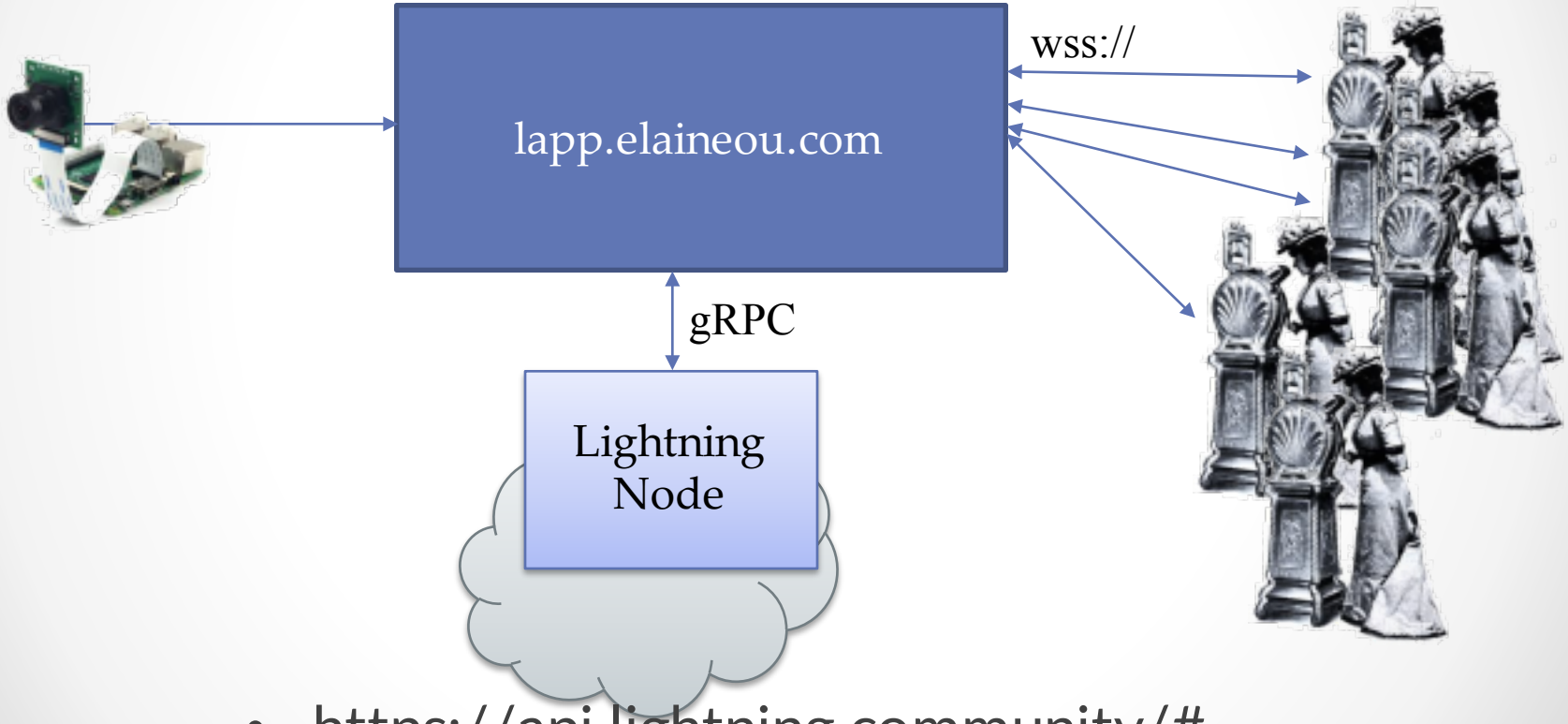
```
#
def broadcastToStreamingClients(self, clients, data):
    for client in self.clients:
        if client.paid:
            if datetime.now() - client.timeConnected < timedelta(seconds=60):
                client.bufferStreamData(data)
            else:
                client.kill
```

lapp.elaineou.com

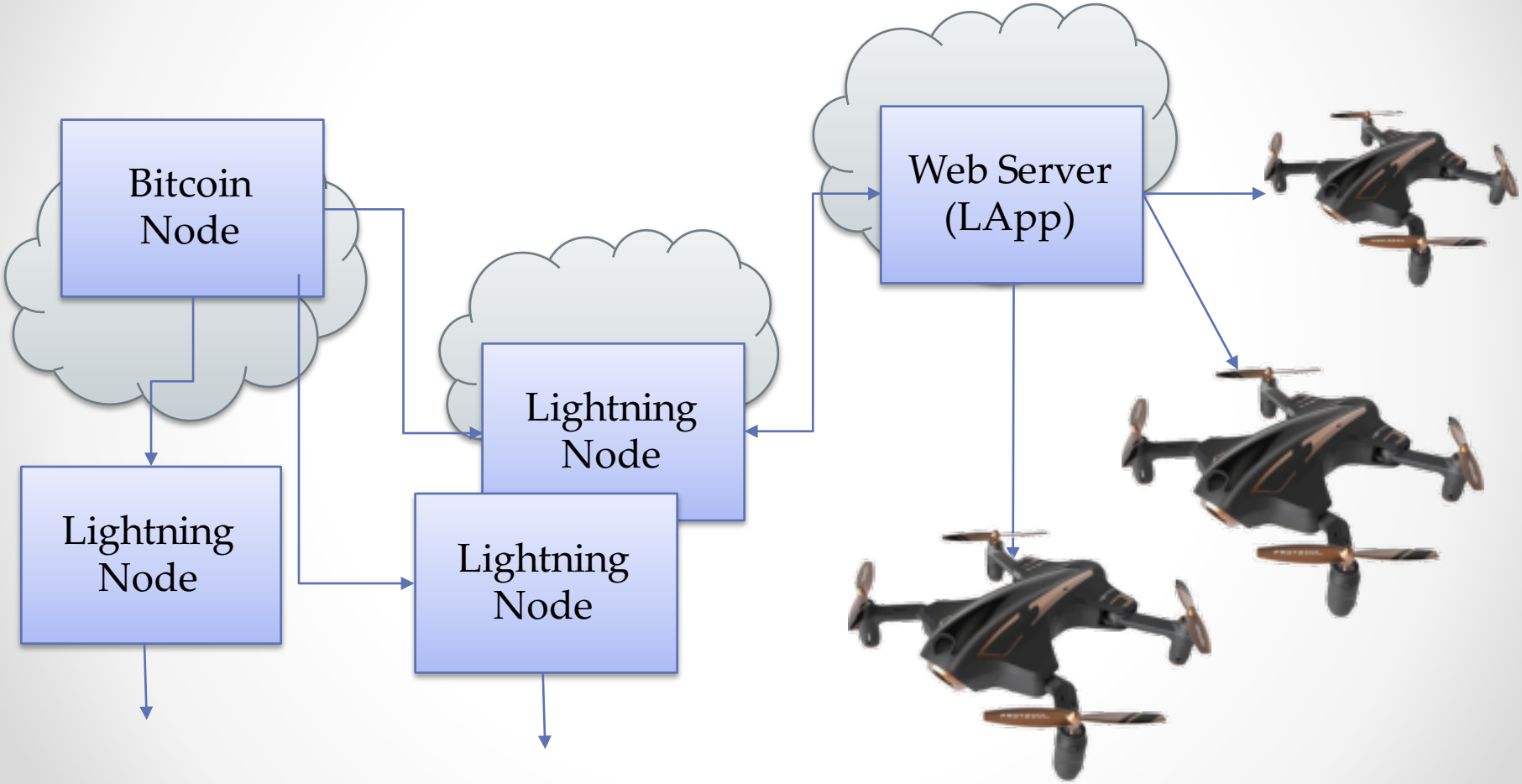
wss://



The Node



- https://api.lightning.community/#


Bitcoin Node
Lightning Node
Lightning Node
Lightning Node
Web Server (LApp)