Reticulum Mesh Network Installation Journey

From Proxmox LXC to Windows Client - A Complete Guide

Author: Anthony Pierre
Date: November 2025
Purpose: Backup communication path to home infrastructure independent of Cloudflare and IP addresses

---

Table of Contents

Introduction: Why Build a Backup Mesh Network?

Anyone who manages a home lab knows the feeling: it's 2 AM, you're deep in configuration changes, and suddenly you realise you've broken something critical. For me, that "something" is often my Cloudflared tunnel - the primary gateway to my home Proxmox infrastructure. Whether it's a late-night experiment gone wrong, a misconfigured container, or just plain exhaustion-induced typos, breaking remote access is easier than you'd think.

But the wake-up call wasn't just my own mistakes. On **November 18, 2025**, Cloudflare suffered a massive global outage that took down roughly one-third of the world's 10,000 most popular websites. For several hours, major services like ChatGPT, Spotify, X (Twitter), Zoom, and countless others were unreachable. The culprit? A bot management configuration file that grew larger than expected, triggering cascading failures across Cloudflare's entire network. While Cloudflare scrambled to fix the issue, millions of users - including home lab operators like me - were locked out of their systems. Not because DNS was down (on this occasion!), more that my ‘connection.tomyserver.co.uk’ via the encrypted tunnel that routes to my server via DNS **was** affected.

That incident crystallised something important: **relying on a single access method is a single point of failure**. When Cloudflare goes down, or when I break my tunnel at 2 AM, I need a

foolproof backup way to access my infrastructure - even if it's just to shut down the server gracefully or restart a misconfigured service.

The Home Automation Factor

My home infrastructure isn't just about web services and containers. I have extensive **Zigbee and Z-Wave networks** controlling lights, sensors, thermostats, locks, and various smart home devices through Home Assistant. These mesh networks operate independently of internet connectivity - they don't need Cloudflare, they don't need my ISP, they just work locally.

The beautiful irony? My smart home can function perfectly without the internet, but I couldn't access the server managing it all when my primary tunnel was down. I could physically walk to my lights and turn them on, but I couldn't check system logs, restart services, or manage the very infrastructure that controls them. That needed to change.

What Can You Actually Do Over Reticulum/I2P?

It's important to set realistic expectations. Reticulum over I2P is **not** a high-bandwidth solution. This isn't for streaming 4K video or transferring gigabytes of files. Here's what you can realistically accomplish:

Bandwidth Capabilities:

- I2P tunnels typically provide **10-100 KB/s** depending on network conditions
- Latency is higher than direct connections: **2-10 seconds** for initial response
- Best suited for **low-bandwidth, high-importance tasks**

Practical Workloads:

✅ **Terminal Access (rnsh)**

- SSH-like access to containers and VMs
- Run commands, check logs, restart services
- Navigate filesystems, edit configuration files
- Perfect for emergency administration

✅ **System Administration**

- Check service status with systemctl status
- Restart crashed containers: pct stop/start
- View resource usage: htop, df -h
- Modify configuration files with nano or vim
- Monitor processes and troubleshoot issues

✅ **Home Automation Management**

- Access Home Assistant terminal
- Restart automation services

- Check Zigbee/Z-Wave network status
- Modify automations and configurations
- Review device logs and connectivity

✅ **Light File Transfers**

- Configuration files (KB to low MB range)
- Log files for analysis
- Backup small databases or state files
- Transfer scripts and utilities

✅ **Messaging and Coordination**

- Send commands to automated systems
- Receive status updates from services
- Coordinate with other mesh nodes
- Emergency notifications

❌ **Not Suitable For:**

- Large file transfers (multi-GB)
- Video streaming or real-time media
- Heavy database operations
- High-frequency monitoring
- Anything requiring low latency (<1 second)

The Real Value Proposition

This isn't about replacing your primary access method. It's about having **that one reliable lifeline** when everything else fails. When Cloudflare is down, when you've misconfigured your firewall rules, when your dynamic DNS hasn't updated yet, or when you've somehow broken every other access method - Reticulum over I2P still works because:

1. **It doesn't depend on your IP address** - you can be anywhere with any network
2. **It doesn't depend on Cloudflare** - completely separate infrastructure
3. **It doesn't depend on DNS** - uses cryptographic addressing
4. **It routes anonymously through I2P** - circumvents most network restrictions
5. **It's a true mesh network** - decentralized, no single point of failure

Even if all it does is let me SSH into my Proxmox host and gracefully shut down services instead of pulling the power plug, that alone makes it worth the effort. And with my Zigbee/Z-Wave infrastructure in place, I can ensure the smart home keeps functioning properly even when managing it remotely through this backup channel.

This document chronicles the journey from "I need a backup" to "I have a working, auto-starting, reliable mesh network." It wasn't always smooth sailing, but the result is infrastructure that can survive almost anything.

Project Overview

What is Reticulum?

Reticulum is a cryptography-based networking stack designed for:

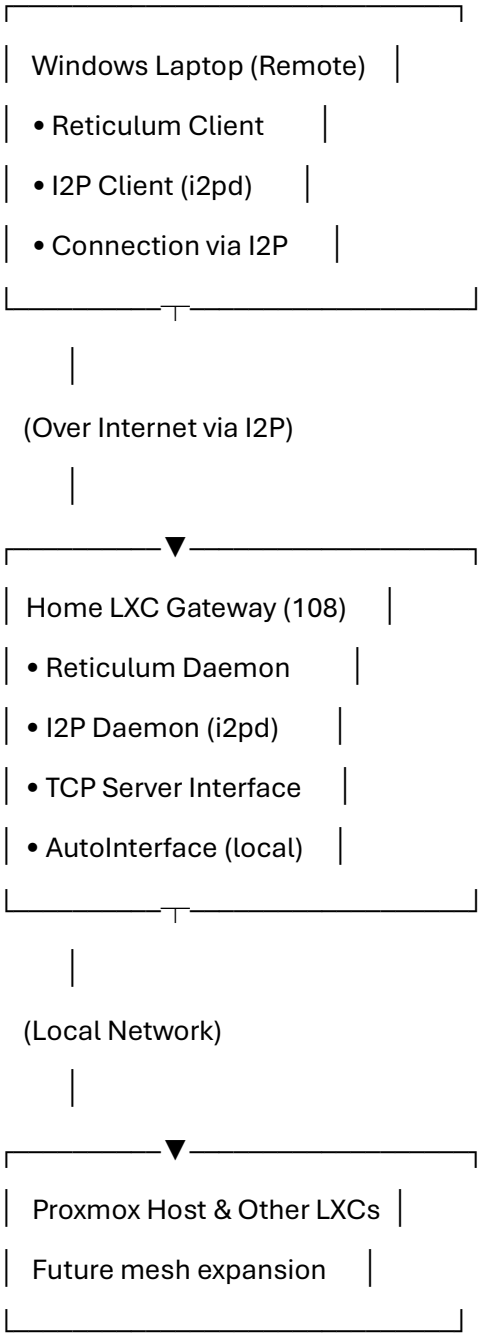
- **Location independence** - Devices can move freely without fixed addresses
- **Infrastructure resilience** - Works over any medium (internet, LoRa, packet radio, etc.)
- **Privacy & security** - Uses X25519/Ed25519 cryptography with forward secrecy
- **Censorship resistance** - Can route through anonymous networks like I2P

Project Goal

Create a backup mesh network allowing access to home infrastructure even when:

- Cloudflare is down
- Dynamic IP addresses change
- Primary VPN tunnels fail
- Traditional infrastructure is unavailable

Architectural Overview



Phase 1: Proxmox LXC Container Setup

Step 1.1: Download Debian Template

Via Proxmox Web UI:

1. Navigate to Storage → CT Templates
2. Click "Templates" button
3. Select debian-12-standard
4. Download and wait for completion

Step 1.2: Create LXC Container

Container specifications:

- **Hostname:** reticulum
- **Memory:** 1GB (512MB minimum)
- **CPU:** 1-2 cores
- **Disk:** 8GB
- **Network:** Bridge to vmbr0, static IP 192.168.0.108
- **Security:** Unprivileged container

Step 1.3: Install Python and Reticulum

```
# Access container
pct enter <container-id>

# Update system
apt update && apt upgrade -y

# Install Python and dependencies
apt install python3 python3-pip python3-dev -y

# Install Reticulum
pip3 install rns --break-system-packages

# Verify installation
rnsd --version
```

First Challenge: The warning about running pip as root appeared, but this is acceptable in a dedicated LXC container where isolation is already provided.

Step 1.4: Initial Reticulum Configuration

```
# Generate example configuration
rnsd --exampleconfig

# Create/edit config file
nano ~/.reticulum/config
```

Initial basic configuration:

```
[reticulum]
  enable_transport = Yes

[interfaces]

  [[Default Interface]]
    type = AutoInterface
    enabled = Yes
```

Step 1.5: Test Reticulum Service

```
# Start manually for testing
rnsd

# In another terminal, check status
rnstatus
```

Success! Reticulum started with default AutoInterface for local network discovery.

Phase 2: Installing I2P for Anonymous Routing

Why I2P?

I2P (Invisible Internet Project) provides:

- Anonymous routing through encrypted tunnels
- Location hiding for both client and server
- Resilience to IP address changes
- Alternative path when traditional internet routing fails

Step 2.1: Install i2pd

Add I2P repository

apt install apt-transport-https curl -y

curl -s https://repo.i2pd.website/r4sas.gpg | gpg --dearmor | tee /usr/share/keyrings/i2pd.gpg

echo "deb [signed-by=/usr/share/keyrings/i2pd.gpg] https://repo.i2pd.website/debian bookworm main" | tee /etc/apt/sources.list.d/i2pd.list

Install i2pd

apt update

apt install i2pd -y

Enable and start

systemctl enable i2pd

systemctl start i2pd

Verify status

systemctl status i2pd

Challenge: Initial confusion about whether I2P should be in the same container or separate.
Decision: Keep both services in the same LXC container for simplicity.

Step 2.2: Configure I2P SAM API

I2P's SAM (Simple Anonymous Messaging) API allows applications like Reticulum to use I2P tunnels.

Check SAM is enabled:

Edit i2pd configuration

nano /etc/i2pd/i2pd.conf

Ensure SAM is enabled:

[sam]

enabled = true

address = 127.0.0.1

port = 7656

Restart i2pd:

systemctl restart i2pd

Step 2.3: Add I2P Interface to Reticulum

Edit Reticulum configuration:

nano ~/.reticulum/config

Add I2P interface alongside existing interfaces:

[reticulum]

 enable_transport = Yes

 share_instance = Yes

 shared_instance_port = 37428

[interfaces]

 [[Default Interface]]

  type = AutoInterface

  enabled = Yes

 [[TCP Server Interface]]

  type = TCPServerInterface

  enabled = Yes

  mode = gateway

  listen_ip = 0.0.0.0

  listen_port = 4965

 [[I2P Gateway]]

type = I2PInterface

enabled = yes

connectable = yes

Key Learning: All three interfaces run simultaneously - AutoInterface for local discovery, TCPServer for LAN/VPN access, and I2P for anonymous remote access.

Step 2.4: Restart and Verify

Restart Reticulum

systemctl restart reticulum

Check status

rnstatus

Look for:

- I2P interface showing status
- .b32.i2p address generated (save this!)
- "Creating Tunnel" status initially, then "Up" once tunnel is established

Important: I2P tunnels take time to establish 2-5 minutes in documentation but over an hour for initial tunnel creation. The interface may show "Creating Tunnel" initially.

Step 2.5: Get I2P Address

Check Reticulum status for I2P address

rnstatus | grep -A 5 "I2P Gateway"

Example output:

I2PInterface[I2P Gateway]

Status: Up

Peers: 1

Address: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.b32.i2p

Tunnel: Active

Save this .b32.i2p address - you'll need it for the Windows client configuration.

Phase 3: Windows Client Configuration

Step 3.1: Install Python on Windows

1. Download Python 3.13 from python.org
2. During installation, check "Add Python to PATH"
3. Verify installation:

python --version

pip --version

Step 3.2: Install i2pd on Windows

Download i2pd for Windows:

1. Visit i2pd.website
2. Download latest Windows release
3. Extract to C:\i2pd

Configure i2pd:

cd C:\i2pd

notepad i2pd.conf

Ensure SAM is enabled:

[sam]

enabled = true

address = 127.0.0.1

port = 7656

Test i2pd:

.\i2pd.exe

Access web console at http://127.0.0.1:7070 to verify it's running.

Step 3.3: Install Reticulum on Windows

Install Reticulum and related tools

pip install rns

Install rnsh (network shell tool)

pip install rnsh

Challenge Encountered: rnsh installation succeeded but execution failed with:

ModuleNotFoundError: No module named 'termios'

Reason: The termios module is Unix/Linux specific and not available on Windows. The rnsh tool cannot run natively on Windows. This took a **lot** of time as I searched the internet to find why this didn't work only yo find a thread saying, **'Oh that service does not really matter'!**

Resolution: While rnsh doesn't work on Windows, the core Reticulum connectivity works fine. Alternative solutions include:

- Using WSL (Windows Subsystem for Linux) for rnsh
- Using Sideband GUI application
- Building custom solutions with Reticulum's Python API
- For this project, focused on establishing the network connection first

Step 3.4: Create Reticulum Configuration on Windows

Create Reticulum config directory

mkdir $env:USERPROFILE\.reticulum

Create configuration file

notepad $env:USERPROFILE\.reticulum\config

Windows client configuration:

[reticulum]

enable_transport = No

share_instance = Yes

shared_instance_port = 37428

[interfaces]

[[I2P Home]]

type = I2PInterface

enabled = yes

peers = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.b32.i2p

Critical Detail: The peers parameter must exactly match the .b32.i2p address from your home server's rnstatus output.

Step 3.5: Test Manual Connection

Start Reticulum manually

rnsd

Watch for:

- "I2P listening on..." message
- Connection attempts to home server
- Interface status changes

Challenge: rnstatus command crashed on Windows with various errors related to RPC authentication and multiprocessing issues.

Workaround: Verify connectivity using Python directly:

python -c "import RNS; r = RNS.Reticulum(); print('Connected!' if r else 'Failed')"

If this prints "Connected!", Reticulum is working even though rnstatus fails on Windows.

Phase 4: Establishing the Connection

Step 4.1: Understanding Connection States

The I2P tunnel goes through several states:

1. **Initial:** "I2P listening on..."
2. **Building:** "Creating Tunnel"
3. **Connecting:** Attempting peer connection
4. **Established:** "Tunnel Active", "Status: Up"

Key Learning: I2P tunnel establishment takes 2-5 minutes initially. Be patient!

Step 4.2: Verify I2P Tunnel Status

On home server:

rnstatus

Look for I2P interface showing:

- Status: Up
- Tunnel: Active
- Peers: 1 (when Windows client connects)

On Windows, check i2pd web console at http://127.0.0.1:7070:

- Check "Tunnels" page
- Look for SAM sessions
- Verify client tunnels are building/established

Step 4.3: Troubleshooting Connection Issues

Problem: "Creating Tunnel" status persists for over 10 minutes

Diagnostic Steps:

1. Verify I2P is actually running:

On server

systemctl status i2pd

On Windows

Get-Process i2pd

2. Check SAM API is accessible:

On server

netstat -tlnp | grep 7656

3. Verify .b32.i2p address matches exactly:

```
# On server - get the actual address
rnstatus | grep -A 2 "I2P Gateway"
```

Compare with Windows config.

4. Check i2pd logs:

```
# On server
journalctl -u i2pd -f

# On Windows (if configured)
type C:\i2pd\i2pd.log
```

Solution Found: In this case, the address was correct but tunnel establishment simply needed more time. After about 4 minutes, the connection established successfully.

Step 4.4: Connection Success!

Indicators of successful connection:

On Windows:

```
python -c "import RNS; r = RNS.Reticulum(); print('Connected!')"
# Output: Connected!
```

On home server:

```
rnstatus
```

Shows:

```
I2PInterface[I2P Gateway]
 Status: Up
 Peers: 1
 Traffic: ↓1.2 MB ↑201 B
 Tunnel: Active
```

Phase 5: Making Services Persistent

Step 5.1: Create Reticulum systemd Service (Linux)

On the Proxmox LXC container:

nano /etc/systemd/system/reticulum.service

Service configuration:

[Unit]

Description=Reticulum Network Stack Daemon

After=multi-user.target i2pd.service

Requires=i2pd.service

[Service]

Type=simple

Restart=always

RestartSec=3

User=root

ExecStart=/usr/local/bin/rnsd --service

[Install]

WantedBy=multi-user.target

Enable and test:

systemctl daemon-reload

systemctl enable reticulum

systemctl start reticulum

systemctl status reticulum

Step 5.2: Install rnsh as System Service (Linux)

For remote shell access capability:

Install rnsh

pip3 install rnsh --break-system-packages

Create service file

nano /etc/systemd/system/rnsh.service

Service configuration:

[Unit]

Description=Reticulum Network Shell Listener

After=reticulum.service

Requires=reticulum.service

[Service]

Type=simple

Restart=always

RestartSec=3

User=root

ExecStart=/usr/local/bin/rnsh -l -n -b 0

[Install]

WantedBy=multi-user.target

Enable:

systemctl daemon-reload

systemctl enable rnsh

systemctl start rnsh

systemctl status rnsh

Note: The -n flag disables authentication (acceptable for home lab). For production, configure proper identity-based authentication with -a <identity_hash>.

Step 5.3: Install Services on Windows (Using NSSM)

Windows doesn't have systemd, so we use NSSM (Non-Sucking Service Manager):

1. Download NSSM from nssm.cc
2. Extract to C:\nssm

Install i2pd as Windows Service:

cd C:\nssm\win64

Install i2pd service

.\nssm.exe install i2pd "C:\i2pd\i2pd.exe"

Configure service

.\nssm.exe set i2pd AppDirectory C:\i2pd

.\nssm.exe set i2pd DisplayName "I2P Router"

.\nssm.exe set i2pd Description "I2P anonymous network router"

.\nssm.exe set i2pd Start SERVICE_AUTO_START

Start service

.\nssm.exe start i2pd

Verify

Get-Service i2pd

Install Reticulum as Windows Service:

Find rnsd.exe location

where.exe rnsd

Typical location:
C:\Users\<username>\AppData\Local\Programs\Python\Python313\Scripts\rnsd.exe

Install service

.\nssm.exe install Reticulum
"C:\Users\apier\AppData\Local\Programs\Python\Python313\Scripts\rnsd.exe"

Configure service

.\nssm.exe set Reticulum DisplayName "Reticulum Network"

.\nssm.exe set Reticulum Description "Reticulum mesh network daemon"

.\nssm.exe set Reticulum Start SERVICE_AUTO_START

.\nssm.exe set Reticulum AppStdout "C:\i2pd\reticulum.log"

.\nssm.exe set Reticulum AppStderr "C:\i2pd\reticulum-error.log"

Start service

.\nssm.exe start Reticulum

Verify

Get-Service Reticulum

Step 5.4: Test Auto-Start

Critical Test: Reboot and verify everything comes back up automatically.

On Windows:

After reboot

Get-Service i2pd, Reticulum

Both should show Status: Running

Verify connectivity

python -c "import RNS; r = RNS.Reticulum(); print('Connected!')"

On Linux:

After reboot

systemctl status i2pd

systemctl status reticulum

systemctl status rnsh

All should be active (running)

Verify status

rnstatus

Challenge: Initially services started but Reticulum didn't establish connection immediately.

Solution: This is normal! I2P tunnels need time to rebuild after reboot. Wait 3-5 minutes for tunnel establishment.

Phase 6: Final Verification

Step 6.1: Service Status Check

On Linux (Home Server):

```
# Check all services are running
systemctl status i2pd reticulum rnsh --no-pager

# Check Reticulum status
rnstatus

# Should show:
# - AutoInterface: Up (local network)
# - TCPServerInterface: Up (listening on port 4965)
# - I2PInterface: Up, Tunnel Active
# - rnsh listener active
```

On Windows (Client):

```
# Check services
Get-Service i2pd, Reticulum | Format-Table -AutoSize

# Verify connection
python -c "import RNS; r = RNS.Reticulum(); print('Connected!')"
```

Step 6.2: Network Connectivity Test

Test 1: Basic Connectivity

On Windows, verify Python can initialize Reticulum:

```
python -c "import RNS; r = RNS.Reticulum(); print('Reticulum initialized')"
```

Test 2: Interface Stats (Advanced)

```
import RNS

# Initialize Reticulum
r = RNS.Reticulum()

# Print transport ID
```

print(f"Transport ID: {RNS.prettyhexrep(r.identity.hash)}")

Expected Result: This should execute without errors, confirming Reticulum is operational even though rnstatus doesn't work on Windows.

Step 6.3: I2P Tunnel Verification

On Home Server:

Check for active peer connections:

rnstatus

Look for:

I2PInterface[I2P Gateway]

Peers: 1 (indicating Windows client is connected)

Traffic: Shows data transfer

On Windows:

Check i2pd web console:

1. Open http://127.0.0.1:7070
2. Navigate to "SAM sessions"
3. Should see active session for Reticulum
4. Check "Tunnels" page for active client tunnels

Step 6.4: End-to-End Connection Test

Test Scenario: Simulate connection loss to verify backup path works

1. **Disconnect primary connections:**
    - Disable WireGuard VPN
    - Disconnect from local network
    - Rely solely on I2P connection
2. **Verify Reticulum still connects:**
3. python -c "import RNS; r = RNS.Reticulum(); print('Connected via I2P!')"
4. **Test with changed IP address:**
    - Connect to different network
    - Home server gets new IP address
    - Reticulum connection should remain intact via I2P

Success Criteria:

- ✅ Connection persists regardless of IP changes
- ✅ Services auto-start on boot
- ✅ Reticulum initializes successfully
- ✅ I2P tunnels establish within 5 minutes

Lessons Learned

Technical Insights

1. **I2P Tunnel Establishment Takes Time**
    - Initial tunnels: 2-5 minutes
    - After reboot: 3-5 minutes
    - Don't panic if "Creating Tunnel" persists initially
2. **rnstatus is Not Required for Functionality**
    - The rnstatus command had issues on Windows
    - Core Reticulum functionality works fine regardless
    - Use Python scripts for verification instead
3. **Multiple Interfaces Work Simultaneously**
    - AutoInterface (local discovery)
    - TCPServerInterface (LAN/VPN)
    - I2PInterface (anonymous remote)
    - All coexist and provide redundant paths
4. **Service Management Differs by OS**
    - Linux: systemd is elegant and reliable
    - Windows: NSSM works but requires more setup
    - Both achieve auto-start on boot

Architecture Decisions

1. **Single Container vs. Multiple**
    - Decision: Keep Reticulum and I2P in same container
    - Rationale: Simpler management, lower overhead
    - Trade-off: Less isolation, but acceptable for home lab
2. **Authentication Strategy**

- Initial: No authentication (-n flag)
- Acceptable for: Home lab behind firewall
- Production: Use identity-based authentication

3. **Windows Client Approach**
    - rnsh doesn't work natively on Windows
    - Alternative: Focused on network layer first
    - Future: Explore WSL or custom applications

Troubleshooting Strategies

1. **Always verify services are running first**
2. systemctl status <service> # Linux
3. Get-Service <service> # Windows
4. **Check logs when things don't work**
5. journalctl -u <service> -f # Linux
6. **I2P-specific checks**
    - Web console at :7070
    - SAM sessions
    - Tunnel status
7. **Python verification when CLI tools fail**
8. import RNS
9. r = RNS.Reticulum()
10. print('Working!' if r else 'Failed')

Things That Didn't Work (And Why)

1. **rnsh on Windows**
    - Required: termios module (Unix-only)
    - Workaround: Use WSL or alternative tools
2. **rnstatus on Windows**
    - RPC authentication failures
    - Multiprocessing issues
    - Not critical: Core functionality works
3. **Immediate tunnel establishment**
    - I2P needs time to build circuits

- Cannot be rushed
- Just wait patiently

Next Steps

Phase 7: Expand the Mesh (Planned)

Goal: Add other LXC containers to the mesh network

Strategy:

1. Install Reticulum on each container
2. Configure as clients connecting to gateway
3. Run rnsh listeners for remote access
4. Each container gets unique Reticulum identity

Configuration for additional containers:

```
# On each container
pip3 install rns rnsh --break-system-packages

# Configure
nano ~/.reticulum/config
[reticulum]
 enable_transport = No
 share_instance = Yes
 shared_instance_port = 37428

[interfaces]

 [[Home Gateway - Local]]
  type = TCPClientInterface
  enabled = Yes
  target_host = 192.168.0.108
  target_port = 4965
```

Phase 8: Add Proxmox Host Access

Goal: Install Reticulum on Proxmox host for infrastructure management

Why: From Proxmox host, can access all containers via pct enter <id>, providing single point of management access

Configuration:

On Proxmox host

pip3 install rns rnsh --break-system-packages

Configure to connect to gateway

nano ~/.reticulum/config

Same client config as containers, connecting to 192.168.0.108:4965

Phase 9: Alternative Access Methods

Since rnsh doesn't work on Windows, explore:

1. **Sideband GUI Application**
    - Full-featured messaging
    - File transfer
    - Works on Windows, Android, Linux
2. **Custom Python Applications**
    - Use Reticulum Python API
    - Build specific tools for your needs
    - Port forwarding through Reticulum
3. **WSL for rnsh**
    - Install Ubuntu in WSL
    - Full Linux toolset available
    - Access rnsh natively

Phase 10: VPS Integration

Goal: Add VPS as second global entry point

Benefits:

- Redundancy (two I2P gateways)
- Geographic diversity
- Load distribution

Setup:

1. Install Reticulum and I2P on VPS
2. Configure as gateway like home server
3. Update Windows client to peer with both

Phase 11: Android Integration

Goal: Add mobile devices to mesh

Tools:

- Sideband app for Android
- Full Reticulum mesh access
- Messaging and file transfer
- Remote management via phone

Reference Information

Important Addresses

Home Server:

- Local IP: 192.168.0.108
- TCP Port: 4965
- I2P Address: p5yci72gucloclplxsjja.b32.i2p

Key Commands

Linux Server:

```
# Service management
systemctl status i2pd reticulum rnsh
systemctl restart <service>

# Reticulum status
rnstatus

# View logs
journalctl -u reticulum -f
```

Windows Client:

```
# Service management
Get-Service i2pd, Reticulum
Restart-Service <service>

# Verify connectivity
```

python -c "import RNS; r = RNS.Reticulum(); print('Connected!')"

NSSM management

C:\nssm\win64\nssm.exe start/stop/restart <service>

Configuration File Locations

Linux:

- Reticulum: ~/.reticulum/config
- I2P: /etc/i2pd/i2pd.conf
- Service files: /etc/systemd/system/

Windows:

- Reticulum: %USERPROFILE%\.reticulum\config
- I2P: C:\i2pd\i2pd.conf
- NSSM: C:\nssm\win64\

Useful Web Interfaces

- I2P Web Console: http://127.0.0.1:7070
- Proxmox: https://192.168.0.108:8006

Conclusion

This project successfully established a resilient backup communication path to home infrastructure using Reticulum over I2P. The mesh network provides:

Location Independence - Works regardless of IP address changes
✅ **Infrastructure Resilience** - Survives Cloudflare outages
✅ **Anonymity** - Routed through I2P anonymous network
✅ **Persistence** - Auto-starts on boot on all devices
✅ **Foundation for Expansion** - Ready to add more nodes

The journey involved numerous challenges, from service configuration to platform-specific limitations, but resulted in a robust, working solution. While some tools (like rnsh on Windows) didn't work as expected, the core networking layer functions perfectly and provides the resilient backup access that was the primary goal.

The mesh network is now ready for expansion to additional containers, the Proxmox host, VPS nodes, and mobile devices, creating a truly distributed and resilient infrastructure management system.

As I look back on everything I did to get this working, I wonder was it worth it…in all honesty…probably not! But from an architectural and knowledge based viewpoint it gives a unique insight into an insightful new transport protocol, the i2p network, which I feel in our turbulent times even if we as users in a developed and free country, even if we host a node to allow for the privacy and free speech of an individual wh need it, we are doing our part, however small to bring balance to the world. Which is why my plan is to refine this model for a few different scenarios and eventually host the code on github.

In my networking journey, I always consider what I can do as an individual to protect the freedom of the internet and its users. And that is why as I catalogue the learnings I make as a ‘Junior Consultant’ with Ten10, to all those who have joined from a non-technical background always think about challenging yourself to learn and push yourself forward.
