# Resilient by Design

**A Complete Guide to Decentralized, Private & Censorship-Resistant Technology From First Principles to Running Infrastructure**

The Techno Anarchist

# Contents

**Preface: Who This Book Is For and How to Use It** **8**

**Part I — The Threat & The Strategy** **9**

**Chapter 1: How Authoritarian Regimes Silence People** **11**
Learning Objectives . . . 11
1.1 The Anatomy of a Digital Crackdown . . . 11
1.2 The Infrastructure They Control . . . 12

**Chapter 2: The Strategy — Decentralize Everything** **14**
Learning Objectives . . . 14
2.1 The Three Pillars . . . 14
2.2 Layered Defense — Defense in Depth . . . 14

**Part II — Foundations: How Computers and Networks Work** **16**

**Chapter 3: Linux — Your Trustworthy Foundation** **18**
Learning Objectives . . . 18
Theory: What Linux Is and Why It Matters . . . 18
Theory: The Linux Architecture . . . 18
Theory: Processes — Every Running Program . . . 19
Theory: systemd — The Service Manager . . . 19
Theory: Namespaces and cgroups — How Containers Work . . . 19
Application: Essential Linux Commands for Beginners . . . 20
Navigating the Filesystem . . . 20
Managing Services with systemd . . . 20
Security Essentials . . . 20
Creating a Service Unit File (Example: a simple service) . . . 21
Chapter 3 Summary . . . 21

**Chapter 4: How Networks Work — From Cables to the Web** **22**
Learning Objectives . . . 22
Theory: The OSI Model — Seven Layers of Networking . . . 22
Theory: IP Addresses — The Postal System of the Internet . . . 23
Theory: DNS — The Phone Book . . . 24
Theory: Routing — How Data Finds Its Way . . . 24
Theory: VLANs — Dividing One Network into Many . . . 25

Application: Setting Up DNS-over-HTTPS . . . 25
Install and configure systemd-resolved with DoH (Ubuntu/Debian) . . . 25
Chapter 4 Summary . . . 26

**Chapter 5: Virtualization — Running Many Computers Inside One** **27**
Learning Objectives . . . 27
Theory: What Virtualization Is . . . 27
Theory: VMs vs. Containers — Two Isolation Models . . . 28
Theory: Proxmox VE . . . 28
Application: Installing Proxmox VE . . . 28
Step 1: Prepare installation media . . . 28
Step 2: Install Proxmox . . . 29
Step 3: Access the management interface . . . 29
Step 4: Create your first VM (Ubuntu Server) . . . 29

**Part III — Private Networking** **30**

**Chapter 6: WireGuard — Building Your Encrypted Backbone** **32**
Learning Objectives . . . 32
Theory: What a VPN Is and How WireGuard Works . . . 32
Theory: Public and Private Keys . . . 32
Theory: Hub-and-Spoke vs. Full Mesh . . . 33
Application: Setting Up WireGuard Full Mesh . . . 33
Step 1: Install WireGuard on each node . . . 33
Step 2: Create the WireGuard configuration (Node A — 10.2.0.1) . . . 33
Step 3: Enable and start WireGuard . . . 34
Step 4: Open the firewall . . . 34

**Chapter 7: Mesh Networks — When There Is No Infrastructure** **36**
Learning Objectives . . . 36
Theory: What Makes a Mesh Network Different . . . 36
Theory: BATMAN-adv — Better Approach To Mobile Adhoc Networking . . . 37
Application: Setting Up a BATMAN-adv Mesh . . . 37
Step 1: Install BATMAN-adv . . . 37
Step 2: Configure the wireless interface for mesh mode . . . 37
Step 3: Verify the mesh is working . . . 38

**Chapter 8: Yggdrasil — A Self-Organizing Encrypted Internet** **39**
Theory: What Yggdrasil Is . . . 39
Application: Installing and Connecting Yggdrasil . . . 39
Install and configure Yggdrasil . . . 39

**Part IV — Decentralized Architecture** **41**

**Chapter 9: Peer-to-Peer Networking — Theory and Practice** **43**
Theory: The Architecture of P2P Systems . . . . . 43
Theory: Overlay Networks in Detail . . . . . 43
Theory: Distributed Service Discovery Without a Center . . . . . 44

**Chapter 10: Distributed Hash Tables and Kademlia** **45**
Theory: What a Hash Is . . . . . 45
Theory: How a Distributed Hash Table Works . . . . . 45
Theory: Kademlia in Detail . . . . . 46
Application: IPFS and DHT in Practice . . . . . 46
Using IPFS (which uses Kademlia DHT internally) . . . . . 46

**Chapter 11: Content-Addressed Storage — Files That Cannot Be Deleted** **48**
Theory: Location-Addressed vs. Content-Addressed . . . . . 48
Theory: Merkle Trees — How Git and IPFS Structure Data . . . . . 48
Application: Publishing a Document on IPFS . . . . . 49
Publishing and sharing content that cannot be censored . . . . . 49

**Chapter 12: Federation, Distributed Identity, and Consensus** **51**
Theory: Federation — The Email Model for Everything . . . . . 51
Theory: Distributed Identity — Who Are You Without a Central Authority? . . . . . 51
Theory: Eventual Consistency vs. Strong Consistency . . . . . 52
Theory: Consensus — How Distributed Systems Make Decisions . . . . . 52

**Part V — Anonymous & Censorship-Resistant Communications** **54**

**Chapter 13: Tor — Anonymity Through the Crowd** **56**
Theory: How Tor Works . . . . . 56
Theory: Onion Services — Hidden Servers . . . . . 56
Application: Running Your Matrix Server as a Tor Onion Service . . . . . 57
Step 1: Install Tor . . . . . 57
Step 2: Configure a hidden service . . . . . 57
Step 3: Configure Matrix to use the onion address . . . . . 57

**Chapter 14: I2P — The Hidden Services Network** **58**
Theory: I2P vs. Tor . . . . . 58
Application: Running an I2P Service . . . . . 58
Install I2P (Java version) . . . . . 58

**Chapter 15: Matrix — Federated Encrypted Messaging** **60**
Learning Objectives . . . . . 60
Theory: The Matrix Protocol Architecture . . . . . 60

Application: Installing Matrix Synapse . . . 60
Step 1: Install Synapse . . . 60
Step 2: Configure Synapse . . . 61
Step 3: Create your first user . . . 61
Step 4: Enable end-to-end encryption verification . . . 61

**Chapter 16: XMPP — The Battle-Tested Protocol** **63**
Theory: What XMPP Is . . . 63
Application: Installing Prosody XMPP Server . . . 63
Install and configure Prosody . . . 63

**Chapter 17: IPFS — The Censorship-Resistant File System** **65**
Application: Running a Private IPFS Cluster . . . 65
Configure a private IPFS network . . . 65

**Chapter 18: Hyphanet — Publishing Without a Trace** **66**
Theory: How Hyphanet Achieves Censorship Resistance . . . 66
Application: Installing Hyphanet . . . 66
Install Hyphanet (requires Java) . . . 66

**Chapter 19: Voice — Asterisk and Encrypted Calls** **68**
Theory: How VoIP Works . . . 68
Application: Installing Asterisk . . . 68
Basic Asterisk installation and extension setup . . . 68

**Part VI — Storage, Databases & Infrastructure** **70**

**Chapter 20: Storage — ZFS, Ceph, and Backups** **72**
Theory: What ZFS Does That Normal Filesystems Don't . . . 72
Theory: Ceph — Storage That Scales Across Many Machines . . . 72
Application: Setting Up ZFS on Proxmox . . . 72
Create a ZFS pool in Proxmox . . . 72
Application: The 3-2-1 Backup System . . . 73
Automated backup script using ZFS send/receive . . . 73

**Chapter 21: Databases — PostgreSQL and Redis** **75**
Theory: Why Your Applications Need Databases . . . 75
Theory: PostgreSQL — The Production-Grade Database . . . 75
Theory: Redis — Fast In-Memory Storage . . . 75
Application: PostgreSQL High Availability with Patroni . . . 76
Setting up PostgreSQL with automatic failover . . . 76

**Chapter 22: Containers — Docker, Podman, and LXC** **78**
Theory: Containers Are Not Virtual Machines . . . 78

Application: Running Matrix in Docker . . . 78
Docker Compose file for Matrix + PostgreSQL . . . 78

**Part VII — Automation & Observability** **80**

**Chapter 23: Infrastructure as Code — Ansible and OpenTofu** **82**
Learning Objectives . . . 82
Theory: Why Infrastructure as Code Matters for Resistance . . . 82
Application: Writing an Ansible Playbook for WireGuard . . . 82
Ansible playbook: configure WireGuard on all nodes . . . 82
Application: OpenTofu to Provision VMs on Proxmox . . . 84
OpenTofu configuration for a Proxmox VM . . . 84

**Chapter 24: Monitoring — Seeing What Is Happening** **86**
Theory: The Observability Stack . . . 86
Application: Deploying the Monitoring Stack . . . 87
Deploy Prometheus + Grafana + Loki with Docker Compose . . . 87
Prometheus configuration: scrape targets . . . 88
Alerting rules: critical alerts for resistance infrastructure . . . 88

**Part VIII — Security** **90**

**Chapter 25: Threat Modeling — Know Your Adversary** **92**
Theory: The STRIDE Framework . . . 92
Application: Building Your Threat Model . . . 92
Four questions to answer before building anything . . . 92

**Chapter 26: Cryptography From First Principles** **94**
Theory: Symmetric Encryption . . . 94
Theory: Asymmetric (Public-Key) Encryption . . . 94
Theory: The Double Ratchet Algorithm (Signal, Matrix OMEMO) . . . 94
Application: Practical Cryptography Tools . . . 95
GPG: encrypting files and email . . . 95
LUKS: full disk encryption . . . 95

**Chapter 27: Operational Security — The Human Layer** **97**
Theory: Why Technology Alone Is Not Enough . . . 97
Theory: The Five-Step OpSec Process . . . 97
Application: OpSec Checklist . . . 98

**Part IX — Radio** **99**

**Chapter 28: Radio Fundamentals and the Spectrum** **101**
Learning Objectives . . . 101
Theory: What Radio Waves Are . . . 101
Theory: How Different Frequencies Travel . . . 102
Theory: Modulation — Encoding Information onto Radio . . . 102

**Chapter 29: Software Defined Radio — Hardware and Software** **104**
Theory: What "Software Defined" Means . . . 104
Theory: IQ Sampling in Plain Language . . . 104
Application: SDR Hardware Comparison . . . 105
Application: Setting Up SDR++ for Monitoring . . . 105
Install and use SDR++ (Linux) . . . 105
Decode ADS-B aircraft positions in real time . . . 106

**Chapter 30: Building a Radio Communication Node** **107**
The Complete Radio Stack for a Resistance Node . . . 107
Application: APRS — Radio Text Messaging Without Internet . . . 107
Set up an APRS digipeater and message gateway . . . 107

**Part X — Putting It All Together** **109**

**Chapter 31: Reliability Engineering — When Things Break** **111**
Theory: Failure Modes and Their Probabilities . . . 111
Theory: RPO and RTO . . . 111
Application: Chaos Engineering Exercises . . . 112
Monthly resilience drills . . . 112

**Chapter 32: Progressive Lab Design — Small to Regional** **114**
Scale 1: Individual or Pair — The "Go Bag" Lab . . . 114
Scale 2: Small Group — The Community Hub . . . 114
Scale 3: Regional Network — The Distributed Collective . . . 115
Network Topology Templates . . . 116

**Chapter 33: Quick Reference and Appendices** **117**
Priority Order: What to Build First . . . 117
Complete Software Toolkit . . . 119
Troubleshooting: The Most Common Problems . . . 121
Glossary . . . 123

**Closing: The Philosophy of Resilience** **124**

# Preface: Who This Book Is For and How to Use It

This book was written for two types of readers who share the same goal: building communication infrastructure that cannot be silenced.

**If you are a complete beginner** — you have never set up a server, you do not know what a VPN really is inside, and phrases like "DHT" or "P2P overlay" mean nothing to you — this book starts from first principles. Every concept is explained before it is used. Look for the blue *Beginner's Corner* boxes; they are written specifically for you.

**If you are intermediate** — you know your way around Linux, you have maybe set up a home server — this book gives you the deeper theory and the real configuration examples to build production-grade distributed infrastructure. Look for the purple *Going Deeper* boxes for extra nuance.

**How this book is organized:** Each chapter follows the same structure. First, *Theory* — the why and how of the concept from first principles. Then, *Application* — the practical steps, configuration examples, and troubleshooting. You can read theory-only on a first pass, then return to application when you are ready to build.

All software described in this book is free, open-source, and used by journalists, researchers, human rights workers, and emergency responders worldwide. Knowledge of how technology works is not a crime anywhere in the world.

# Part I — The Threat & The Strategy

Understanding why this technology matters and what philosophy guides the entire guide

# Chapter 1: How Authoritarian Regimes Silence People

### Learning Objectives

- Understand the seven-step playbook that governments use to silence digital communications
- Know which technologies counter each step
- Understand that each suppression technique has a technical counter-measure

## 1.1 The Anatomy of a Digital Crackdown

When a government decides to suppress its population digitally, it does not flip one switch. It works through a predictable sequence of escalating measures, each targeting a different layer of the communications stack. Understanding this sequence tells you exactly which technologies you need — and in what order of priority.

| Step | What They Do | Technical Method | Counter-measure |
|---|---|---|---|
| 1 | Block social media | DNS blocking, IP blocking, deep packet inspection (DPI) | VPNs, Tor, DNS-over-HTTPS, self-hosted alternatives |
| 2 | Block commercial VPNs | Block known VPN server IPs; detect VPN traffic signatures | WireGuard (obfuscated), Tor bridges, Shadowsocks |
| 3 | Take down news sites | DNS removal, hosting provider pressure, IP blacklists | IPFS, Hyphanet, Tor onion services, mirrors in other countries |
| 4 | Throttle or cut mobile internet | ISP-level bandwidth caps, mobile network throttling | Wi-Fi mesh (BATMAN-adv), radio communications (APRS, LoRa, HF) |
| 5 | Intercept and read messages | ISP-level traffic capture, SSL inspection at the ISP, malware on devices | End-to-end encryption (Matrix/XMPP with OMEMO), Signal, device hardening |
| 6 | Identify and locate dissidents | IP address tracking, metadata analysis, cell tower triangulation | Tor, Yggdrasil, no-log policies, operational security |
| 7 | Full internet shutdown ("kill switch") | Order ISPs to stop routing traffic; BGP route withdrawal | Radio (HF shortwave, APRS), local mesh networks, satellite terminals |

**Beginner's Corner: What is "deep packet inspection"?**

Imagine postal workers who not only check the address on your envelope, but open it, read the letter, and decide whether to deliver it based on the content. DPI does the same thing with internet traffic — it inspects the content of data packets, not just their destination. This is how governments block VPNs (they recognize the "shape" of VPN traffic even if they cannot read the encrypted content). The counter-measure is to disguise VPN traffic so it looks like ordinary web browsing.

## 1.2 The Infrastructure They Control

To understand what can be blocked, you must understand what authoritarian governments control:

- **Internet Service Providers (ISPs)** — In most countries, ISPs must obtain a license. The government can revoke that license or issue legal orders to filter traffic. All internet traffic in a country passes through licensed ISPs.
- **DNS Resolvers** — Most people's computers use ISP-provided DNS resolvers. The ISP controls these and can make `bbc.com` resolve to nowhere, or to a government warning page.
- **Border Routers** — Every country has a small number of connection points to the global internet (Internet Exchange Points). A government can order these to filter or block traffic.
- **App Stores and Hosting Providers** — Apple and Google operate app stores in every country. A government can pressure them to remove apps. Similarly, hosting providers operating in a country can be ordered to take down content.
- **Physical Infrastructure** — Fiber cables, cell towers, and exchange buildings are physical locations that can be raided, damaged, or controlled.

**The key insight:** Everything on this list is centralized. There are specific chokepoints — specific companies, specific buildings, specific people — that a government can pressure. The answer to all of it is *decentralization* — making the system have no chokepoints, no single company, no single building that controls it.

**The Optimistic Reality**

Complete internet shutdowns are rare and costly — they damage the economy, disrupt government services, and create international backlash. Most authoritarian regimes aim for selective suppression: block specific sites, monitor specific people, silence specific voices. That is what the tools in this book are primarily designed to counter.

# Chapter 2: The Strategy — Decentralize Everything

### Learning Objectives

- Understand the core philosophy: eliminate every single point of failure and control
- Learn the three pillars: distribute, encrypt, and diversify
- Understand the layered defense model

## 2.1 The Three Pillars

**Pillar 1: Distribute.** Never let any single machine, company, or location be essential to your communications. If the loss of any one component silences your network, that component is a liability. Build so that the loss of any component is automatically compensated for by the others.

**Pillar 2: Encrypt.** Assume all traffic is observed. Make the contents of that traffic meaningless to observers. End-to-end encryption means the only people who can read a message are the people it is addressed to — not the ISP, not the platform, not the government.

**Pillar 3: Diversify.** Do not depend on a single communication channel. If Matrix is blocked, use XMPP. If XMPP is blocked, use Tor. If the internet is down, use radio. Diversity means redundancy; redundancy means resilience.

## 2.2 Layered Defense — Defense in Depth

Security and resilience professionals use a concept called "defense in depth" — multiple independent layers of protection, so that breaking through one layer does not compromise everything. Think of a medieval castle: a moat, then walls, then an inner keep, then a fortified room. An attacker must breach all layers to reach the treasure.

```
Layer 1: Physical - Hardware you own, encrypted disks, UPS power backup
         ↓
Layer 2: Network - Firewall, VPN, network segmentation (VLANs)
         ↓
Layer 3: Transport - TLS encryption on all connections
         ↓
Layer 4: Application - End-to-end encrypted messaging (Matrix/XMPP)
         ↓
Layer 5: Anonymity - Tor or I2P to hide who is talking to whom
```

```
         ↓
Layer 6: Content - IPFS/Hyphanet so content exists without a location
         ↓
Layer 7: Radio - HF/LoRa/APRS for when all internet is down
```

You do not need all seven layers for every situation. Match your layers to your threat model (which Chapter 25 covers in depth). But understanding all seven tells you where your current gaps are.

**Beginner's Corner: An Analogy for the Whole Book**

Imagine your group needs to communicate secretly during a crackdown. You have a plan:

1. **Normally:** Use your private Matrix chat server, encrypted, over a VPN
2. **If Matrix is blocked:** Switch to your private XMPP server on a Tor onion address
3. **If internet is throttled too badly:** Use your local Wi-Fi mesh (BATMAN-adv) to communicate within the city
4. **If everything is cut:** Use handheld radios on prearranged frequencies with Morse code or digital modes

This book teaches you how to build all four of those fallback layers. By the end, losing any one — or even two — of them does not stop you.

# Part II — Foundations: How Computers and Networks Work

The essential theory you need before building anything

# Chapter 3: Linux — Your Trustworthy Foundation

### Learning Objectives

- Understand why Linux is the foundation for all the infrastructure in this book
- Learn the key Linux concepts: processes, systemd, namespaces, cgroups, logging
- Know how to install and navigate a Linux system for the first time

## Theory: What Linux Is and Why It Matters

Linux is a free, open-source operating system kernel — the core software that manages your computer's hardware and allows other software to run. Unlike Windows (owned by Microsoft) or macOS (owned by Apple), Linux is owned by nobody and everybody. Its source code is publicly available for anyone to read, audit, and modify.

This is not just a philosophical point — it has practical consequences for security and trust:

- **No hidden backdoors:** Any backdoor in Linux's source code would be visible to the millions of people who read and audit it. Proprietary operating systems make this kind of independent verification impossible.
- **No company to pressure:** Microsoft has responded to government requests for data. Apple has removed apps at the request of authoritarian governments. The Linux Foundation has no products to sell in any country, no app store to police.
- **Complete control:** On Linux, you decide exactly what software runs. Nothing installs itself, nothing phones home, nothing updates without your consent.

**Beginner's Corner: Which Linux should I use?**

There are hundreds of Linux "distributions" (distros) — versions of Linux packaged with different software. For beginners, use **Ubuntu Server 24.04 LTS** or **Debian 12 (Bookworm)**. Both are stable, widely used, and have excellent documentation. "LTS" means "Long Term Support" — you will receive security updates for years without needing to upgrade.

## Theory: The Linux Architecture

```
User Applications (Matrix, IPFS, Docker, your scripts)
         
```

```
System Libraries (libc, OpenSSL - shared code apps use)
         
System Calls (the API between programs and the kernel)
         
Linux Kernel (manages hardware, memory, processes, network)
         
Hardware (CPU, RAM, disks, network cards)
```

## Theory: Processes — Every Running Program

When a program runs on Linux, it becomes a "process" — an isolated instance of a running program with its own memory space. Every process has:

- A Process ID (PID) — a unique number
- A user it runs as (e.g. `root` or a limited service account)
- Permissions — what files and resources it can access
- A parent process — most processes are started by another process

The key security principle: **run every service as a non-root user with the minimum permissions it needs.** If your Matrix server is compromised, it should only have access to the Matrix data directory — not to your WireGuard keys, your backup encryption keys, or your monitoring configuration.

## Theory: systemd — The Service Manager

When Linux boots, the first process that starts is `systemd` (PID 1). It is responsible for starting all other services in the correct order and keeping them running. Think of it as the city manager who makes sure all city services start up in the morning and restarts anything that crashes.

Every service on your system is described by a "unit file" — a configuration file that tells systemd how to start the service, what user to run it as, what to do if it crashes, and what other services it depends on.

## Theory: Namespaces and cgroups — How Containers Work

Two Linux kernel features enable containers:

**Namespaces** create isolated views of the system. A process inside a namespace sees only the resources in its namespace — its own network interfaces, its own process list, its own filesystem. From inside a container, it looks like you have your own computer.

**cgroups** (control groups) limit how much of each resource (CPU, memory, disk I/O, network) a process or group of processes can use. This prevents one misbehaving service from consuming all resources and crashing everything else.

## Application: Essential Linux Commands for Beginners

### Navigating the Filesystem

```
pwd                  # Print Working Directory - where am I?
ls -la               # List files (l = detailed, a = include hidden files)
cd /etc              # Change to the /etc directory
cd ~                 # Go to your home directory
cat /etc/hostname    # Display the contents of a file
less /var/log/syslog # View a file one page at a time (q to quit)
```

### Managing Services with systemd

```
systemctl status matrix-synapse  # Is the service running?
systemctl start matrix-synapse   # Start it
systemctl stop matrix-synapse    # Stop it
systemctl restart matrix-synapse # Restart it
systemctl enable matrix-synapse  # Start it automatically on boot
journalctl -u matrix-synapse -f  # Watch the service's logs in real time
                                  # (-f means "follow" - keeps updating)
```

### Security Essentials

```
# Update all software (do this regularly!)
apt update && apt upgrade -y

# Check who is logged in
who

# Check what ports are listening (what is exposed to the network)
ss -tlnp

# Check firewall rules
nft list ruleset

# View recent login attempts (look for brute-force attacks)
journalctl _SYSTEMD_UNIT=sshd.service | grep "Failed password" | tail -20
```

### Creating a Service Unit File (Example: a simple service)

```
# /etc/systemd/system/myservice.service
[Unit]
Description=My Custom Service
After=network.target          # Start after networking is up
Requires=postgresql.service   # Start after PostgreSQL is ready

[Service]
Type=simple
User=myserviceuser            # Run as a non-root user
WorkingDirectory=/opt/myservice
ExecStart=/opt/myservice/bin/start
Restart=on-failure            # Restart if it crashes
RestartSec=5                  # Wait 5 seconds before restarting

[Install]
WantedBy=multi-user.target    # Start in normal (multi-user) mode

# After creating/editing: reload systemd, then enable and start
systemctl daemon-reload
systemctl enable --now myservice
```

**Going Deeper: Linux Security Modules**

AppArmor (used on Ubuntu) and SELinux (used on RHEL/Fedora) are Mandatory Access Control systems. Where normal Linux permissions say "this user can read this file," AppArmor adds "this specific program can only access these specific files and network ports, regardless of what user it runs as." Even if an attacker exploits a vulnerability in your Matrix server, AppArmor can prevent them from reading files outside the Matrix data directory. Enable and configure AppArmor profiles for all sensitive services.

## Chapter 3 Summary

- Linux is the only OS suitable for privacy-critical infrastructure because its code is fully auditable
- Every service should run as a non-root user with minimal permissions
- systemd manages services — use it to start services automatically and restart them on crash
- Namespaces and cgroups are the building blocks of containers
- Update software regularly; monitor logs for signs of intrusion

# Chapter 4: How Networks Work — From Cables to the Web

### Learning Objectives

- Understand the OSI model and why layering matters for building resistance networks
- Know how IP addresses, DNS, and routing work — and how each can be attacked and defended
- Understand VLANs, network segmentation, and why they matter for security

## Theory: The OSI Model — Seven Layers of Networking

All networking is built in layers. Each layer provides services to the layer above it and relies on the layer below it. This is one of the most important concepts in all of computing — understanding it lets you identify exactly where a problem or a block is occurring and what the appropriate counter-measure is.

| Layer | Name | What it does | Real examples | How it can be attacked |
|---|---|---|---|---|
| 7 | Application | The actual program the user interacts with | HTTP (web), SMTP (email), Matrix, XMPP | Block specific protocols or platforms |
| 6 | Presentation | Encoding, encryption, compression | TLS/SSL, JSON encoding | Break TLS, intercept before encryption |
| 5 | Session | Opening, maintaining, closing connections | TLS sessions, TCP sessions | Session hijacking |
| 4 | Transport | End-to-end delivery, error correction | TCP (reliable), UDP (fast) | Block specific ports |
| 3 | Network | Logical addressing and routing | IPv4, IPv6, ICMP | Block IP ranges; BGP route withdrawal |
| 2 | Data Link | Device-to-device delivery on the same network | Ethernet, Wi-Fi, MAC addresses | ARP spoofing, MAC flooding |
| 1 | Physical | The actual medium: electricity, light, radio waves | Fiber optic, Ethernet cable, Wi-Fi radio waves | Cut cables, jam radio frequencies |

Think of sending a letter internationally. Layer 7 is the actual letter content. Layer 6 is the language it is written in. Layer 5 is the ongoing correspondence between you and the recipient. Layer 4 is the envelope. Layer 3 is the address on the envelope. Layer 2 is the postal truck carrying it within a city. Layer 1 is the road the truck drives on.

**Why this matters for resistance:** When a regime blocks access to a website (Layer 7), you can tunnel through that block at Layer 3 using a VPN. If they block VPNs at Layer 3, you can obfuscate your VPN traffic to look like normal Layer 7 HTTPS traffic. If they block all internet (Layer 3), you can communicate at Layer 1 with radio. Understanding the layers lets you always find a counter-move.

## Theory: IP Addresses — The Postal System of the Internet

Every device on a network needs an address so that data can be routed to it. IP addresses are these addresses.

**IPv4** addresses look like `192.168.1.5` — four numbers from 0 to 255, separated by dots. There are about 4.3 billion possible IPv4 addresses. This seemed like a lot in the 1970s but is now running out.

**IPv6** addresses look like `2001:0db8:85a3:0000:0000:8a2e:0370:7334` — eight groups of four hexadecimal digits. There are 340 undecillion possible IPv6 addresses (that is 340 followed by 36 zeros). Every device on Earth could have billions of addresses.

**Private vs. public addresses:** Some IP address ranges are reserved for private networks (inside homes and offices). These include `10.0.0.0/8`, `172.16.0.0/12`, and `192.168.0.0/16`. They cannot be routed over the public internet — your router translates them to your single public IP address (this is called NAT — Network Address Translation).

**Beginner's Corner: What is a subnet mask (/24, /16 etc.)?**

When you see `192.168.1.0/24`, the `/24` is a "prefix length" that tells you how many devices can be in this network. `/24` means the first 24 bits are the "network" part and the remaining 8 bits are for device addresses — giving you 254 usable addresses (192.168.1.1 to 192.168.1.254). `/16` gives you 65,534 addresses. `/8` gives you 16,777,214 addresses.

## Theory: DNS — The Phone Book

When you type `matrix.example.com`, your computer does not know the IP address of that server. It asks a DNS (Domain Name System) resolver: "What is the IP address for matrix.example.com?" The resolver looks it up and returns the answer.

**How DNS censorship works:** The government orders ISPs to return wrong answers for certain domain names. Your computer asks "What is the IP of bbc.com?" and the ISP's DNS resolver returns either the wrong address (redirecting you to a government page) or nothing at all (making the site unreachable).

**DNS counter-measures:**

- **DNS-over-HTTPS (DoH)** — Send DNS queries over an encrypted HTTPS connection to a resolver outside the country (e.g., `1.1.1.1` or `9.9.9.9`). The ISP cannot intercept and manipulate the query.
- **Run your own DNS resolver** — Your network queries authoritative servers directly, bypassing ISP resolvers entirely.
- **Tor** — All DNS resolution happens at the Tor exit node, bypassing local DNS completely.

## Theory: Routing — How Data Finds Its Way

When your computer sends data to a server in another country, that data passes through many "routers" — devices that look at the destination IP address and forward the packet toward its destination. Routers share information about which networks they can reach using routing protocols like BGP and OSPF.

**BGP (Border Gateway Protocol)** is used between organizations — between ISPs, between countries. It is how the global internet is "stitched together." Each organization announces which IP ranges it owns, and other organizations learn those announcements and route traffic accordingly.

**Internet shutdown mechanism:** A government can order ISPs to withdraw their BGP announcements. Suddenly the rest of the world's routers do not know how to reach IP addresses in

that country. The country disappears from the internet. This is what happened in Egypt in 2011, Myanmar in 2021, and Sudan in 2023.

## Theory: VLANs — Dividing One Network into Many

A VLAN (Virtual LAN) is a way of creating logically separate networks on the same physical switch infrastructure. A packet tagged with VLAN 10 cannot reach devices on VLAN 20 without explicitly crossing a router with rules permitting it.

For a resistance network, VLANs enforce compartmentalization:

```
Physical switch (one device, multiple VLANs):

VLAN 10 (Management):  Admin laptops, Proxmox management UI
VLAN 20 (Services):    Matrix, XMPP, IPFS containers
VLAN 30 (Monitoring):  Prometheus, Grafana
VLAN 40 (External):    Servers that face the internet (with Tor)

Rules:
  VLAN 40 → VLAN 20: allowed (external services can reach internal)
  VLAN 20 → VLAN 10: DENIED (services cannot reach management)
  Any VLAN → VLAN 10: only from specific admin IPs
```

## Application: Setting Up DNS-over-HTTPS

### Install and configure systemd-resolved with DoH (Ubuntu/Debian)

```
# Edit the resolved configuration
nano /etc/systemd/resolved.conf

# Add these lines:
[Resolve]
DNS=1.1.1.1#cloudflare-dns.com 9.9.9.9#dns.quad9.net
DNSOverTLS=yes
DNSSEC=yes

# Restart the resolver
systemctl restart systemd-resolved

# Verify it is working (should show "DNSSEC: yes" and "DNS over TLS: yes")
resolvectl status
```

### Chapter 4 Summary

- The OSI model's 7 layers each represent a different aspect of networking — and a different attack surface
- IP addresses are the routing addresses of the internet; understanding private vs. public addresses matters for network design
- DNS censorship is countered by DNS-over-HTTPS or running your own resolver
- BGP route withdrawal is how governments cause complete internet blackouts
- VLANs enforce compartmentalization between different security zones in your network

# Chapter 5: Virtualization — Running Many Computers Inside One

### Learning Objectives

- Understand what a hypervisor is and how virtualization works
- Know the difference between a Virtual Machine (VM) and a Container
- Learn to install Proxmox VE and create your first VM

## Theory: What Virtualization Is

A hypervisor is software that creates and manages "virtual machines" — simulated computers that run inside your real computer. Each virtual machine gets a portion of the real machine's CPU, RAM, and storage. From the software's point of view, it is running on its own dedicated computer.

**Type 1 hypervisors** (bare-metal) run directly on the hardware, with no underlying OS. They are faster and more secure. Proxmox VE is a Type 1 hypervisor.

**Type 2 hypervisors** run as an application inside an existing OS. VirtualBox and VMware Workstation are Type 2. They are convenient for personal desktops but slower and less suitable for servers.

Virtualization is like an apartment building. The building (physical server) has a finite amount of space (CPU, RAM, disk). The building manager (hypervisor) divides that space into apartments (virtual machines). Each apartment has its own lock, its own furniture, its own rules. A fire in apartment 3B does not spread to apartment 4A unless the building itself is compromised.

## Theory: VMs vs. Containers — Two Isolation Models

| Feature | Virtual Machine (VM) | Container |
|---|---|---|
| What is isolated | Full operating system, kernel, hardware emulation | Filesystem, network, process namespace — shares the host kernel |
| Isolation strength | Very strong — a compromised VM cannot easily escape to the host | Weaker — kernel vulnerabilities can allow container escape |
| Startup time | 30–120 seconds (full OS boot) | Less than 1 second |
| Resource use | High — each VM has its own OS using RAM and disk | Low — many containers share the host OS |
| Use for | Services requiring strong isolation (Tor relay, untrusted code) | Services you trust (Matrix, IPFS, Prometheus) |

**Rule of thumb:** Use VMs for anything internet-facing or security-sensitive (your Tor relay, your public-facing reverse proxy). Use containers for internal services that you control and trust.

## Theory: Proxmox VE

Proxmox VE (Virtual Environment) is a free, open-source Type 1 hypervisor based on Debian Linux. It supports both QEMU/KVM virtual machines and LXC containers. It includes a web-based management interface so you can manage everything from a browser.

For a resistance network, Proxmox is the foundation everything else runs on. It provides:

- Isolation between services (VMs and containers)
- Snapshots — instant backup of any VM's state
- Live migration — move a running VM from one physical host to another without downtime
- Cluster support — manage multiple physical machines as one pool of resources
- ZFS storage integration

## Application: Installing Proxmox VE

### Step 1: Prepare installation media

1. Download the Proxmox VE ISO from `proxmox.com/en/downloads`
2. Write it to a USB drive using Balena Etcher (Windows/macOS) or `dd` (Linux)
3. Boot your server from the USB drive (press F12, F2, or Del during startup to access boot menu)

### Step 2: Install Proxmox

1. Select "Install Proxmox VE (Graphical)"
2. Select your target disk — this will be erased. Choose a fast SSD if you have one.
3. Set your country, timezone, keyboard layout
4. Set a strong root password (save it somewhere safe)
5. Set a static IP address for the management interface. Example: `192.168.1.10/24`
6. After installation, remove the USB and reboot

### Step 3: Access the management interface

```
# From a browser on the same network:
https://192.168.1.10:8006

# Log in with: root / [password you set]
# Accept the self-signed certificate warning for now
# (we will set up proper TLS later)

# Remove the subscription warning (Proxmox is free without a subscription):
# From the shell on the Proxmox host:
sed -i.bak "s/data.status !== 'Active'/false/g" \
  /usr/share/javascript/proxmox-widget-toolkit/proxmoxlib.js
systemctl restart pveproxy
```

### Step 4: Create your first VM (Ubuntu Server)

1. Upload the Ubuntu Server 24.04 ISO: Datacenter → local → ISO Images → Upload
2. Click "Create VM" and follow the wizard: allocate 2 CPU cores, 4 GB RAM, 32 GB disk to start
3. Start the VM and click the Console button to complete the Ubuntu installation

**Hardware Recommendation**

A refurbished business desktop (Dell OptiPlex, HP EliteDesk, Lenovo ThinkCentre) with 16–32 GB RAM and a 500 GB SSD costs $100–$250 and runs Proxmox well. These machines are designed for 24/7 operation, have good Linux driver support, and consume less power than consumer gaming PCs. Used enterprise servers (like a Dell R720) give you more power but are louder, larger, and consume more electricity.

# Part III — Private Networking

Building encrypted tunnels and mesh networks that cannot be easily blocked

# Chapter 6: WireGuard — Building Your Encrypted Backbone

### Learning Objectives

- Understand how WireGuard works from first principles
- Know the difference between a hub-and-spoke and a full-mesh VPN topology
- Set up a WireGuard full-mesh VPN between multiple nodes

## Theory: What a VPN Is and How WireGuard Works

A VPN (Virtual Private Network) creates an encrypted "tunnel" between two or more computers. All traffic that passes through this tunnel is encrypted before it leaves your machine and decrypted only when it reaches the other end. To anyone watching the network in between — an ISP, a surveillance device, a government — the traffic appears as random, meaningless noise.

WireGuard is a modern VPN protocol that is:

- **Simpler:** About 4,000 lines of code, compared to OpenVPN's ~100,000 lines. Fewer lines = smaller attack surface = easier to audit for security flaws.
- **Faster:** Uses state-of-the-art cryptography (ChaCha20 for encryption, Curve25519 for key exchange, BLAKE2 for hashing) that is extremely fast on modern hardware.
- **Stealthier:** WireGuard uses UDP and does not respond to connection attempts from unauthorized peers — the port appears "closed" or non-existent to outsiders, making the server harder to detect.

## Theory: Public and Private Keys

WireGuard uses public-key cryptography for authentication. Every WireGuard peer has:

- **A private key:** Generated once, kept secret, never shared with anyone. This is your identity — your proof that you are you. If someone gets your private key, they can impersonate you on the network.
- **A public key:** Mathematically derived from the private key. You share this with everyone you want to connect to. Knowing someone's public key allows you to encrypt messages that only their private key can decrypt.

Your private key is like the key to your house. Your public key is like your home address. You share your address with anyone who wants to send you letters. But only you have the key that opens the lock on your mailbox. Even if someone knows your address perfectly, they cannot open your mailbox without your physical key.

## Theory: Hub-and-Spoke vs. Full Mesh

```
Hub-and-Spoke (centralized, fragile):   Full Mesh (decentralized, resilient):

        [HUB]                           [A]——[B]
       / | \                              \   /
      /  |  \                              [C]
    [A] [B] [C]

Pros: Simple to configure               Pros: No single point of failure
Cons: HUB failure = network down        Cons: More configuration
      HUB can see all traffic                 (but Ansible handles this)
```

For a resistance network, always use full-mesh topology. Every node connects directly to every other node. Losing any one node does not interrupt connectivity between the survivors.

## Application: Setting Up WireGuard Full Mesh

### Step 1: Install WireGuard on each node

```
# On Debian/Ubuntu:
apt update && apt install wireguard -y

# Generate a key pair on each node:
wg genkey | tee /etc/wireguard/private.key | wg pubkey > /etc/wireguard/public.key

# Set correct permissions (private key must not be readable by other users)
chmod 600 /etc/wireguard/private.key

# Display your public key (share this with peers):
cat /etc/wireguard/public.key
```

### Step 2: Create the WireGuard configuration (Node A — 10.2.0.1)

```
# /etc/wireguard/wg0.conf on Node A
```

```
[Interface]
Address = 10.2.0.1/24            # This node's VPN IP address
ListenPort = 51820               # UDP port to listen on
PrivateKey = [Node A private key]

# Post-up/down: enable IP forwarding for routing
PostUp = sysctl -w net.ipv4.ip_forward=1
PostDown = sysctl -w net.ipv4.ip_forward=0

# Peer: Node B
[Peer]
PublicKey = [Node B public key]
AllowedIPs = 10.2.0.2/32        # Traffic to 10.2.0.2 goes through this peer
Endpoint = [Node B public IP]:51820   # Where to find Node B on the internet
PersistentKeepalive = 25        # Send a keepalive every 25 seconds (maintains NAT)

# Peer: Node C
[Peer]
PublicKey = [Node C public key]
AllowedIPs = 10.2.0.3/32
Endpoint = [Node C public IP]:51820
PersistentKeepalive = 25
```

### Step 3: Enable and start WireGuard

```
# Enable WireGuard to start on boot and start it now:
systemctl enable --now wg-quick@wg0

# Verify it is running:
wg show

# Test connectivity between nodes:
ping 10.2.0.2   # From Node A, ping Node B's VPN address
ping 10.2.0.3   # From Node A, ping Node C's VPN address
```

### Step 4: Open the firewall

```
# Allow WireGuard UDP traffic through the firewall (nftables):
nft add rule inet filter input udp dport 51820 accept comment "WireGuard"

# Save and persist:
```

```
nft list ruleset > /etc/nftables.conf
systemctl enable --now nftables
```

**If a peer is behind NAT (home router)**

If a node does not have a public IP address (it is behind a home router), it cannot easily receive incoming WireGuard connections. Solutions: (1) Forward UDP port 51820 on the router to the node. (2) Have the node behind NAT be the one that initiates the connection to a node with a public IP (the `PersistentKeepalive` setting handles this automatically). (3) Use a cheap VPS ($5/month) with a public IP as a relay.

**Going Deeper: Obfuscating WireGuard Traffic**

Some firewalls can detect and block WireGuard traffic by its UDP packet patterns. Tools like **Masque**, **AmneziaWG**, or wrapping WireGuard in a WebSocket tunnel using **wstunnel** make the traffic look like ordinary HTTPS to DPI systems. This adds complexity but is essential if WireGuard itself is being blocked.

# Chapter 7: Mesh Networks — When There Is No Infrastructure

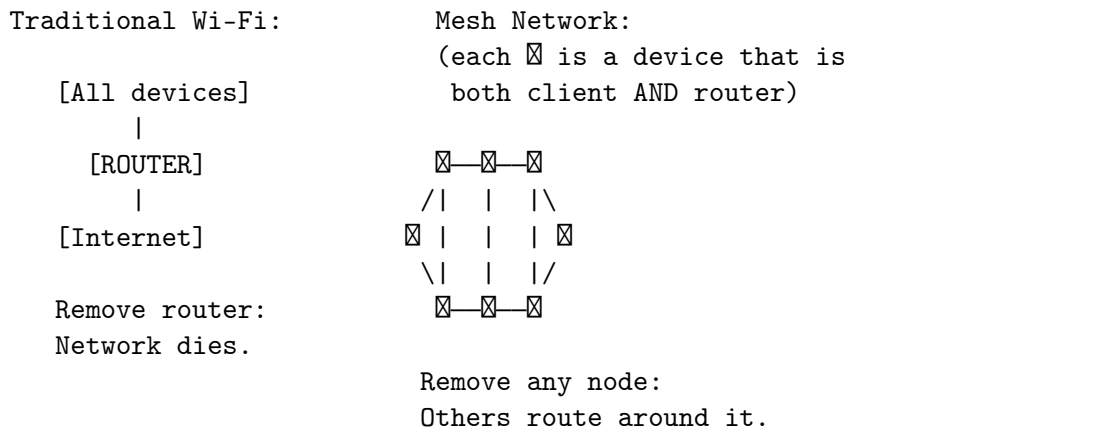
### Learning Objectives

- Understand how mesh networking works without any central infrastructure
- Learn about BATMAN-adv and how to build a local Wi-Fi mesh
- Know when and why to use a mesh network vs. a VPN

## Theory: What Makes a Mesh Network Different

In a normal Wi-Fi network, all devices connect to a single router. The router provides internet access and routes traffic between devices. Remove the router, and the network dies.

In a **mesh network**, every device is both a client and a router. Data "hops" from device to device until it reaches its destination. There is no central router to remove. Every device that joins makes the network larger and more capable. Every device that leaves causes the network to route around the gap.



## Theory: BATMAN-adv — Better Approach To Mobile Adhoc Networking

BATMAN-adv (B.A.T.M.A.N. advanced) is a mesh routing protocol that runs at Layer 2 (Data Link layer) of the OSI model. This means it works at the Ethernet/Wi-Fi level, making it transparent to all Layer 3 protocols (TCP/IP) and applications running above it.

How BATMAN-adv finds routes:

1. Each node periodically broadcasts "Originator Messages" (OGMs) — small packets that say "I am here, this is my address."
2. Neighboring nodes receive these OGMs, add themselves, and rebroadcast them.
3. By analyzing the OGMs it receives, each node learns the best path to every other node on the mesh.
4. When a node disappears, its OGMs stop circulating. Other nodes automatically recalculate routes around it.

## Application: Setting Up a BATMAN-adv Mesh

### Step 1: Install BATMAN-adv

```
# On each device that will be a mesh node:
apt update && apt install batctl -y

# Load the batman-adv kernel module:
modprobe batman-adv

# Make it load on boot:
echo "batman-adv" >> /etc/modules
```

### Step 2: Configure the wireless interface for mesh mode

```
# Bring down the wireless interface:
ip link set wlan0 down

# Set it to mesh (ad-hoc or 802.11s) mode:
# For 802.11s (Wi-Fi Mesh):
iw dev wlan0 set type mesh

# Bring it back up:
ip link set wlan0 up
```

```
# Join a mesh network (all nodes must use the same mesh ID):
iw dev wlan0 mesh join "resilience-mesh" freq 2412

# Add the interface to batman-adv:
batctl if add wlan0

# Bring up the batman-adv interface (bat0):
ip link set bat0 up
ip addr add 192.168.100.1/24 dev bat0  # Unique address per node
```

### Step 3: Verify the mesh is working

```
# See all nodes in the mesh and their link quality:
batctl n        # Neighbors
batctl o        # Originator table (all reachable nodes)

# Test connectivity:
ping 192.168.100.2   # Ping another mesh node

# See routing path to a destination:
batctl traceroute 192.168.100.5
```

**Practical Scenario: City-Wide Mesh During Internet Shutdown**

During a complete internet shutdown, a group of 30 people across a city each run BATMAN-adv on a laptop or Raspberry Pi with a Wi-Fi adapter. As long as enough nodes are close enough to each other (ideally within 100–200 meters in urban environments), the mesh forms automatically. Users on the mesh can run a local Matrix server and communicate as if they have internet — without using internet at all.

# Chapter 8: Yggdrasil — A Self-Organizing Encrypted Internet

## Theory: What Yggdrasil Is

Yggdrasil is a proof-of-concept for a new type of internet — one that is end-to-end encrypted, fully decentralized, and self-organizing. Every node generates a public/private key pair. Its IPv6 address on the Yggdrasil network is derived from its public key. This means:

- Addresses are self-certifying — the address proves you are talking to the right node
- No central authority assigns addresses — you generate your own
- Routing is DHT-based — no central routing tables
- All traffic is encrypted end-to-end at the network layer

## Application: Installing and Connecting Yggdrasil

### Install and configure Yggdrasil

```
# Download and install (Debian/Ubuntu):
curl -o /tmp/yggdrasil.deb \
  https://github.com/yggdrasil-network/yggdrasil-go/releases/latest/download/yggdrasil-
dpkg -i /tmp/yggdrasil.deb

# Generate a default config:
yggdrasil -genconf > /etc/yggdrasil/yggdrasil.conf

# Edit the config - add peers (public peers available at:
# https://github.com/yggdrasil-network/public-peers)
nano /etc/yggdrasil/yggdrasil.conf

# Key section - add public peers:
# Peers:
#   - tls://[IP]:port   # Public Yggdrasil peer

# Enable and start:
systemctl enable --now yggdrasil
```

```
# Check your Yggdrasil IPv6 address:
yggdrasilctl getSelf | grep -i address

# Ping another Yggdrasil node (use their Yggdrasil IPv6 address):
ping6 200:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx
```

**For your group's private network:** Instead of using public peers, configure your group's nodes as each other's peers. Exchange peer addresses out-of-band (in person or over an already-trusted channel). Your group then has a fully private, encrypted network using Yggdrasil addresses.

# Part IV — Decentralized Architecture

The theory behind why distributed systems are resilient — from P2P to consensus

# Chapter 9: Peer-to-Peer Networking — Theory and Practice

## Theory: The Architecture of P2P Systems

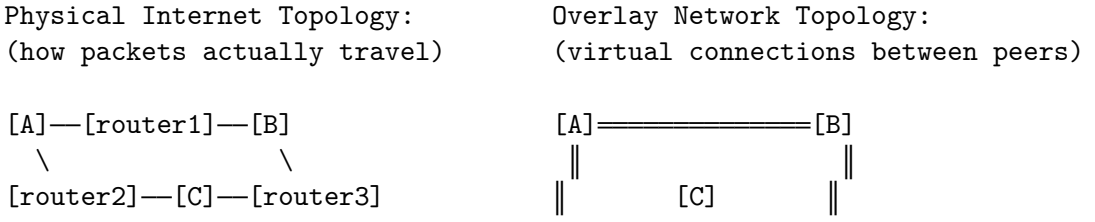
In a peer-to-peer (P2P) network, every participant is simultaneously a client (consuming resources) and a server (providing resources). There is no privileged central node. Every peer is equal in architecture, even if some are more "connected" than others in practice.

P2P systems can be categorized by how structured their routing is:

| Type | How It Routes | Example | Tradeoff |
|---|---|---|---|
| **Unstructured P2P** | Flood queries to all neighbors; search spreads like a rumor | Early Gnutella, BitTorrent search | Simple but doesn't scale — too many messages for large networks |
| **Structured P2P (DHT)** | Mathematical routing table (DHT); queries go directly to responsible nodes | IPFS, BitTorrent DHT, Hyphanet | Efficient and scalable; slightly more complex |
| **Hybrid P2P** | Some centralized components (e.g. a tracker) for bootstrapping; rest is P2P | BitTorrent with trackers, Tor with directory servers | More reliable initially; central components are potential targets |

## Theory: Overlay Networks in Detail

An overlay network is a virtual network layered on top of an existing network. The overlay defines its own addressing, routing, and topology — independent of how the underlying network routes packets.




Physical path A→B goes through router1.
Physical path A→D goes through many hops.

Overlay: A has a direct encrypted tunnel to B, C, and D. The overlay can route differently than the physical network.

This is exactly what Tor, I2P, and Yggdrasil do — they create a virtual topology that is independent of, and often contrary to, the physical internet topology. This is why they can defeat routing-based blocks: your traffic might physically travel through the blocking ISP, but it is encrypted and does not look like the traffic being blocked.

## Theory: Distributed Service Discovery Without a Center

How do peers find each other in a P2P network without a central directory? There are several mechanisms:

**Bootstrap nodes:** A small, well-known set of initial entry points. You contact a bootstrap node to get your initial list of peers. After that, discovery is fully decentralized. Bootstrap nodes are not essential for ongoing operation — they only help you join for the first time.

**Gossip protocols:** Each node periodically tells a random selection of its neighbors what it knows — peers it has seen, services it has heard about. Information "gossips" through the network. Within a few rounds of gossip, every node knows about every service. This is how Matrix federation finds other servers.

**mDNS (Multicast DNS):** Devices on the same local network announce services by broadcasting to a multicast address. Anyone on the network who is interested listens. Used by IPFS to find local peers automatically.

**Practical Advice: Bootstrapping a Private Network**

For a private group network, distribute a list of bootstrap addresses out-of-band — printed on paper, shared in person. Include at least 3 bootstrap addresses in case some nodes are down. Once a new node connects to even one bootstrap peer, it discovers the rest automatically through DHT or gossip. Update the bootstrap list whenever your infrastructure changes significantly.

# Chapter 10: Distributed Hash Tables and Kademlia

## Theory: What a Hash Is

A **cryptographic hash function** takes any input (a file, a string, a document) and produces a fixed-length output (the "hash" or "digest") with two critical properties:

1. **Deterministic:** The same input always produces the same output
2. **One-way and avalanche:** You cannot reverse the hash to find the input; changing even one character of the input completely changes the output

Example (SHA-256 hash):

```
Input: "Hello, world!"
Hash:  315f5bdb76d078c43b8ac0064e4a0164612b1fce77c869345bfc94c75894edd3

Input: "Hello, world?" (only one character changed - ! to ?)
Hash:  61c9b1a5d07b9e4acea9a7b3c8a34f15e2c6e4d9f7a2b1c3e5d7f9a1b3c5d7f9
       (completely different output)
```

## Theory: How a Distributed Hash Table Works

A Distributed Hash Table (DHT) is a decentralized key-value store. You can think of it as a dictionary — a lookup table where you provide a key and get back a value — except that this dictionary is spread across millions of nodes with no central server.

The core idea:

1. Both nodes and stored items are assigned IDs in the same ID space (e.g., 160-bit numbers)
2. Each node is responsible for storing items whose IDs are "close" to its own ID
3. To find an item, you route a query toward nodes with IDs closest to the item's ID
4. The routing converges in O(log N) hops — logarithmic scaling means even huge networks require very few hops

## Theory: Kademlia in Detail

Kademlia is the DHT algorithm used by IPFS, the BitTorrent DHT, and many other systems. Its key innovation is using XOR (exclusive-or) as its distance metric.

**Why XOR?** XOR has a useful mathematical property: the XOR distance between two IDs is symmetric (distance A→B = distance B→A), and there is exactly one "closest point" between any two IDs. This makes routing tables unambiguous and efficient.

**k-buckets:** Each node maintains a routing table organized as "k-buckets" — lists of contacts at different XOR distances from itself. Nodes that are "XOR-close" go in one bucket; nodes that are "XOR-far" go in another. Each bucket holds at most k contacts (usually k=20).

**Self-healing:** Kademlia nodes regularly check that their k-bucket contacts are still alive (by pinging them). Dead contacts are replaced with live ones. When a node joins after being offline, it re-announces itself to its responsible section of the DHT. The network heals itself without any administrator action.

```
Kademlia lookup example (finding node or content with ID=42):

Your node ID: 5 (binary: 00000101)
Target ID: 42  (binary: 00101010)
XOR distance: 00101111 = 47

Step 1: Check your k-bucket for IDs with XOR distance ~47
        Closest known node: ID=30 (XOR 5⊠30=27, closer to 42 than we are)

Step 2: Ask node ID=30 "Who do you know closest to ID=42?"
        It replies: ID=38 (XOR 30⊠38=8, even closer)

Step 3: Ask node ID=38 "Who do you know closest to ID=42?"
        It replies: ID=41 (XOR 38⊠41=11, very close)

Step 4: Ask node ID=41 "Who do you know closest to ID=42?"
        It replies: ID=43 (stores data for ID=42)

Step 5: Ask node ID=43 for the content. Retrieved!

Total: 4 hops to find one item in a network of any size.
In a network of 1 million nodes, this takes ~20 hops.
```

## Application: IPFS and DHT in Practice

### Using IPFS (which uses Kademlia DHT internally)

```
# Install IPFS:
wget https://dist.ipfs.tech/kubo/v0.28.0/kubo_v0.28.0_linux-amd64.tar.gz
tar -xzf kubo_v0.28.0_linux-amd64.tar.gz
cd kubo && ./install.sh

# Initialize a node:
ipfs init

# Start the IPFS daemon:
ipfs daemon &

# Add a file to IPFS (returns the content ID - the hash):
ipfs add important-document.pdf
# Output: added QmXxxxx... important-document.pdf

# Retrieve the same file from any other IPFS node:
ipfs get QmXxxxx...

# Pin a file (keep a permanent local copy, don't garbage-collect it):
ipfs pin add QmXxxxx...

# See your peer connections (these are all DHT neighbors):
ipfs swarm peers | head -20
```

# Chapter 11: Content-Addressed Storage — Files That Cannot Be Deleted

## Theory: Location-Addressed vs. Content-Addressed

The traditional web uses **location-addressed storage**: you find a file by where it is (its URL). Change the server, move the file, or shut down the hosting provider, and the file is gone — even if a million copies exist elsewhere.

**Content-addressed storage** names files by what they contain (their cryptographic hash). The content hash is a permanent, unforgeable identifier derived from the file itself. This has profound implications:

| Property | Explanation | Resistance Application |
|---|---|---|
| **Immutability** | The hash is a permanent name — you cannot "update" a file without creating a new hash | Published documents cannot be secretly altered; share the hash, and anyone can verify the document is unchanged |
| **Location independence** | The same file can be served from any node that has it — the hash doesn't care where it comes from | No URL to block; as long as one copy exists anywhere, the file is retrievable |
| **Deduplication** | Identical files always have the same hash — stored only once even if "uploaded" many times | Efficient spreading of documents across many nodes without wasted storage |
| **Cryptographic verification** | After downloading, hash the file — if it matches the expected hash, the content is authentic | Detect tampering even if the transmission channel is compromised |

## Theory: Merkle Trees — How Git and IPFS Structure Data

A Merkle tree is a tree structure where every node's label is the hash of its children. The root of the tree (the "Merkle root") summarizes the hash of all data beneath it. Changing any piece of data changes its hash, which changes its parent's hash, which propagates all the way to the root.

This is how IPFS represents directories: a directory is a Merkle tree node containing the hashes of all its files and subdirectories. You only need the root hash to verify the integrity of the entire directory tree.

This is also how Git works: every commit contains the hash of the file tree plus the hash of the previous commit. You cannot alter any past commit without changing all subsequent commits — tampering is immediately detectable.

A Merkle tree is like a family tree where every person's name is derived from their children's names. If you change a grandchild's name, it changes the child's name, which changes the grandparent's name. Anyone who knows the grandparent's "name" can instantly detect that something has changed downstream.

## Application: Publishing a Document on IPFS

### Publishing and sharing content that cannot be censored

```
# Add a document to IPFS:
ipfs add -r /path/to/document-or-folder/

# The output gives you the CID (Content Identifier - the hash):
# added QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG document.pdf

# Share the CID with your group.
# Anyone who runs IPFS can now retrieve it:
ipfs get QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG

# To ensure the document is always available, pin it on multiple nodes:
# (Run this on each node in your network)
ipfs pin add QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG

# Make IPFS start on boot:
# Create a systemd service for the IPFS daemon
cat > /etc/systemd/system/ipfs.service << 'EOF'
[Unit]
Description=IPFS Daemon
After=network.target

[Service]
Type=simple
User=ipfs
ExecStart=/usr/local/bin/ipfs daemon
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOF

useradd -r -s /bin/false ipfs
```

```
mkdir -p /home/ipfs && chown ipfs:ipfs /home/ipfs
systemctl enable --now ipfs
```

# Chapter 12: Federation, Distributed Identity, and Consensus

## Theory: Federation — The Email Model for Everything

Email is the oldest and most successful federated system. You have a Gmail address; your friend has a Protonmail address. You can email each other. Why? Because both Gmail and Protonmail implement the SMTP standard. Neither company controls the whole email network.

Matrix and XMPP apply this model to instant messaging. Anyone can run a server, and servers interoperate automatically. This is fundamentally different from WhatsApp or Telegram, where a single company controls all servers and can:

- Read all messages (even if they claim not to)
- Delete accounts
- Block accounts from specific regions
- Hand over user data to governments
- Shut down the service entirely

## Theory: Distributed Identity — Who Are You Without a Central Authority?

In centralized systems, identity is delegated to a trusted authority. Google says "this is Alice's account." If Google is pressured, Alice's identity can be revoked. In decentralized systems, identity is cryptographic.

**How it works:** Alice generates a key pair (private key + public key). Her public key becomes her identity — it is her name in the decentralized network. When she sends a message, she signs it with her private key. Anyone can verify the signature using her public key, confirming the message came from Alice (the holder of the matching private key).

No central authority needed. No government can revoke her key. The only way to "take away" her identity is to obtain her private key — which only she possesses.

**The Key Exchange Problem**

The hardest problem with cryptographic identity: how do you initially get and verify someone's public key? If an adversary can intercept the exchange and substitute their own public key, they can impersonate the legitimate party (a "man-in-the-middle" attack). The solution: verify key fingerprints out-of-band — in person, or over a channel you already trust independently. Matrix's "cross-signing" feature handles this with QR codes.

## Theory: Eventual Consistency vs. Strong Consistency

When data is replicated across multiple nodes, those nodes may temporarily disagree about the current state. There are two approaches:

**Eventual consistency:** Nodes may be temporarily out of sync, but the system guarantees that given enough time and no new updates, all nodes will converge to the same state. Good enough for most communication systems (messages delivered in seconds rather than immediately).

**Strong consistency:** All reads see the most recent write. No node ever sees stale data. Requires coordination between nodes for every write — more complex and slower. Required for financial ledgers, voting systems.

For resistance networks, eventual consistency is almost always sufficient and much simpler to implement.

## Theory: Consensus — How Distributed Systems Make Decisions

When multiple nodes store data and any node can receive writes, they need a way to agree on which write "wins" when there are conflicts. This is the consensus problem.

**The Raft algorithm** (used by many modern distributed systems):

1. Nodes elect a **leader** by vote — the node that gets votes from a majority of nodes becomes leader
2. All writes go to the leader first
3. The leader replicates the write to followers and waits for a majority to acknowledge
4. Only after a majority acknowledges does the leader commit the write
5. If the leader dies, remaining nodes automatically elect a new leader

**Why "majority" matters:** With 3 nodes, a majority is 2. You can lose 1 node and still make progress. With 5 nodes, a majority is 3 — you can lose 2 nodes. With 2 nodes, a majority requires both — losing one node stops all writes. Always use an odd number of nodes.

| Cluster Size | Quorum (majority) | Can lose N nodes | Recommendation |
|---|---|---|---|
| 1 | 1 | 0 | Never use for critical data |
| 2 | 2 | 0 | Worse than useless — same as 1 but costs more |
| 3 | 2 | 1 | Minimum for high availability |
| 5 | 3 | 2 | Good for higher resilience |
| 7 | 4 | 3 | For maximum resilience across geographic locations |

# Part V — Anonymous & Censorship-Resistant Communications

The actual tools for private, uncensorable communication

# Chapter 13: Tor — Anonymity Through the Crowd

## Theory: How Tor Works

Tor (The Onion Router) anonymizes internet traffic by routing it through a series of relays, with each relay knowing only the previous and next hop — never both the origin and destination simultaneously. This is called "onion routing" because messages are wrapped in multiple layers of encryption, like the layers of an onion.

```
Without Tor:
[You]──────────────────────────────[Website]
     ← ISP sees: You visited Website →

With Tor:
[You]→[Guard relay]→[Middle relay]→[Exit relay]→[Website]
  ↑         ↑              ↑              ↑
  Knows     Knows          Knows          Knows
  you're    Guard+Middle   Middle+Exit    Exit→Web
  online    but not you    but not you    but not you

No single relay knows both who you are AND where you're going.
```

**The three hop circuit:** Every Tor connection uses at least 3 relays: a Guard (entry), a Middle, and an Exit. Your client negotiates encryption with each relay separately. Messages are triple-encrypted before leaving your machine. Each relay decrypts one layer (like peeling an onion) and forwards the remainder to the next relay.

## Theory: Onion Services — Hidden Servers

An "onion service" (formerly called a "hidden service") is a server that is only accessible through Tor. Its real IP address is never revealed. It has an address like `duskgytldkxiuqc6.onion` — a hash of the service's public key.

How onion services work:

1. The server generates a key pair and derives its .onion address from the public key
2. The server selects "introduction points" (Tor relays) and announces them to the Tor DHT

3. A client that wants to connect selects a "rendezvous point" relay and tells the server (via an introduction point)
4. The server connects to the rendezvous point; now the client and server have a circuit through the rendezvous — neither knows the other's IP address

This means: the server's location is hidden from the client, the client's location is hidden from the server, and neither the ISP nor any network observer can determine where either party is.

## Application: Running Your Matrix Server as a Tor Onion Service

### Step 1: Install Tor

```
apt update && apt install tor -y
```

### Step 2: Configure a hidden service

```
# Edit /etc/tor/torrc and add:
HiddenServiceDir /var/lib/tor/matrix-onion/
HiddenServicePort 8448 127.0.0.1:8448   # Matrix federation port
HiddenServicePort 443 127.0.0.1:8448    # HTTPS port

# Restart Tor:
systemctl restart tor

# Get your .onion address (generated automatically):
cat /var/lib/tor/matrix-onion/hostname
# Output: something like duskgytldkxiuqc6ztxy.onion
```

### Step 3: Configure Matrix to use the onion address

```
# In your Matrix (Synapse) homeserver.yaml:
server_name: "duskgytldkxiuqc6ztxy.onion"
public_baseurl: "http://duskgytldkxiuqc6ztxy.onion"
```

**Tor Bridges: When Tor Itself is Blocked**

Some countries block access to Tor's public relays. Tor bridges are unlisted relays that are not published in Tor's directory. Get bridges from `bridges.torproject.org` or by emailing `bridges@torproject.org`. For the most difficult blocking environments, use Pluggable Transports like **obfs4** (disguises Tor traffic as random data) or **meek** (disguises it as HTTPS traffic to major cloud providers).

# Chapter 14: I2P — The Hidden Services Network

## Theory: I2P vs. Tor

I2P (Invisible Internet Project) and Tor are both anonymity overlay networks, but they are designed for different use cases:

| Feature | Tor | I2P |
|---|---|---|
| Primary use | Accessing the regular internet anonymously (outbound proxy) | Hosting services within the I2P network (internal services) |
| Architecture | Centralized directory servers; sequential circuit | Fully distributed; "garlic routing" bundles multiple messages |
| Exit to regular internet | Yes, via exit nodes | Limited (outproxies exist but not recommended) |
| Hidden services | .onion addresses | .i2p addresses ("eepsites") |
| DHT | No — uses centralized directory | Yes — fully distributed |
| Best for | Browsing the web, accessing external sites anonymously | Hosting internal services (chat, file sharing, forums) with strong anonymity |

## Application: Running an I2P Service

### Install I2P (Java version)

```
# Add the I2P repository (Debian/Ubuntu):
apt-add-repository ppa:i2p-maintainers/i2p
apt update && apt install i2p -y

# Start I2P:
systemctl enable --now i2p

# Access the I2P router console from a browser:
# http://127.0.0.1:7657
```

```
# The console shows your tunnel build status, bandwidth, and
# lets you create new "eepsite" (hidden service) tunnels.
# Allow 10-20 minutes for tunnels to build and the network to integrate.
```

# Chapter 15: Matrix — Federated Encrypted Messaging

### Learning Objectives

- Understand how Matrix federation works and why it is superior to centralized chat
- Install and configure a Matrix homeserver (Synapse)
- Set up end-to-end encryption and verify device keys

## Theory: The Matrix Protocol Architecture

Matrix is an open standard for real-time communication. The architecture has three parts:

- **Homeservers:** Servers that store messages and user accounts for their users. Anyone can run one. Synapse is the reference implementation.
- **Clients:** Apps that users interact with. Element is the most common. Clients connect to their homeserver.
- **Rooms:** The fundamental unit of communication. A room can have participants from any homeserver. Each homeserver that has a member in a room stores a replicated copy of all messages in that room.

**End-to-end encryption (E2EE):** Matrix supports E2EE via the Megolm protocol (based on Double Ratchet, the same protocol used by Signal). When E2EE is enabled in a room, messages are encrypted by the sender's device before being sent to the homeserver. The homeserver stores and forwards only ciphertext — it cannot read messages even if subpoenaed.

## Application: Installing Matrix Synapse

### Step 1: Install Synapse

```
# Add the Matrix repository:
apt install -y lsb-release wget apt-transport-https
wget -O /usr/share/keyrings/matrix-org-archive-keyring.gpg \
  https://packages.matrix.org/debian/matrix-org-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/matrix-org-archive-keyring.gpg] \
  https://packages.matrix.org/debian/ $(lsb_release -cs) main" \
```

```
  > /etc/apt/sources.list.d/matrix-org.list
apt update && apt install matrix-synapse-py3 -y
```

### Step 2: Configure Synapse

```
# Generate the initial config (replace with your server name):
python3 -m synapse.app.homeserver \
  --server-name matrix.yourdomain.onion \
  --config-path /etc/matrix-synapse/homeserver.yaml \
  --generate-config \
  --report-stats=no

# Key settings to review in homeserver.yaml:
# server_name: "matrix.yourdomain.onion"   ← your server's identity
# enable_registration: false                ← disable open registration
# registration_shared_secret: "CHANGE_ME"  ← for adding users via admin
# database:                                 ← configure PostgreSQL here

# Set up PostgreSQL as the database backend (much better than SQLite):
apt install postgresql -y
sudo -u postgres psql -c "CREATE USER synapse WITH PASSWORD 'STRONGPASS';"
sudo -u postgres psql -c "CREATE DATABASE synapse OWNER synapse \
  ENCODING 'UTF8' LC_COLLATE='C' LC_CTYPE='C' template=template0;"
```

### Step 3: Create your first user

```
# Start Synapse:
systemctl enable --now matrix-synapse

# Create an admin user (run as root, uses registration_shared_secret):
register_new_matrix_user -c /etc/matrix-synapse/homeserver.yaml \
  -u admin -p 'STRONGPASSWORD' -a http://localhost:8008

# Check Synapse logs:
journalctl -u matrix-synapse -f
```

### Step 4: Enable end-to-end encryption verification

```
# In Element (the client app):
# Settings → Security → Cross-signing
```

```
# Click "Set up" and follow the prompts
# This creates a master key that certifies all your device keys

# Share your room and enable encryption:
# In any room → Settings → Security & Privacy → Encrypted → Enable

# Verify another user's identity:
# Click their name → Verify
# Use QR code scan (in-person) or emoji comparison (remote)
```

**Going Deeper: Matrix Replication**

Every homeserver that has at least one user in a room stores a full replicated copy of all messages in that room. This means if your homeserver goes down, messages are still stored on other homeservers. This built-in replication is a major resilience advantage. For maximum resilience, ensure your users are in rooms with members from multiple homeservers — the room's history is then replicated across all of them.

# Chapter 16: XMPP — The Battle-Tested Protocol

## Theory: What XMPP Is

XMPP (Extensible Messaging and Presence Protocol) was designed in 1999 as an open, federated alternative to proprietary instant messaging. It has been used by activists, journalists, and privacy-conscious users for decades and is widely considered one of the most battle-tested messaging protocols available.

XMPP's key strengths for resistance:

- **OMEMO encryption:** An extension providing double-ratchet end-to-end encryption (the same type Signal uses)
- **Lightweight:** Runs on very old or low-power hardware
- **Mature:** Decades of security research and real-world deployment
- **Works over Tor:** Connect your XMPP client to your server via Tor for anonymity

## Application: Installing Prosody XMPP Server

### Install and configure Prosody

```
# Install Prosody:
apt install prosody -y

# Edit /etc/prosody/prosody.cfg.lua:
VirtualHost "chat.yourdomain.onion"

# Enable modules:
modules_enabled = {
  "roster"; "saslauth"; "tls"; "dialback";
  "disco"; "carbons"; "pep"; "private";
  "blocklist"; "vcard4"; "vcard-legacy";
  "version"; "uptime"; "time"; "ping";
  "register"; "admin_adhoc";
  "http_upload"; -- for file sharing
  "smacks"; -- for mobile reconnection
```

```
}

# Disable open registration (invite-only network):
allow_registration = false

# Enable HTTPS for web-based file upload:
https_ssl = { certificate = "/etc/ssl/..."; key = "/etc/ssl/..."; }

# Start Prosody:
systemctl enable --now prosody

# Add a user:
prosodyctl adduser alice@chat.yourdomain.onion
```

# Chapter 17: IPFS — The Censorship-Resistant File System

(See Chapter 11 for theory. This chapter focuses on operational use.)

## Application: Running a Private IPFS Cluster

A public IPFS node connects to the global IPFS network. For a private network, you can configure IPFS to connect only to trusted peers.

### Configure a private IPFS network

```
# Generate a shared swarm key (all private nodes must have this exact key):
go install github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen@latest
ipfs-swarm-key-gen > /var/ipfs/.ipfs/swarm.key
# Copy /var/ipfs/.ipfs/swarm.key to every node in your private network

# Remove the default public bootstrap nodes:
ipfs bootstrap rm all

# Add your own nodes as bootstrap peers:
ipfs bootstrap add /ip4/10.2.0.1/tcp/4001/p2p/[PeerID of node 1]
ipfs bootstrap add /ip4/10.2.0.2/tcp/4001/p2p/[PeerID of node 2]

# Get a node's PeerID:
ipfs id | grep '"ID"'

# Verify only your peers are connected:
ipfs swarm peers
# Should only show your private nodes
```

# Chapter 18: Hyphanet — Publishing Without a Trace

## Theory: How Hyphanet Achieves Censorship Resistance

Hyphanet (formerly Freenet) achieves censorship resistance through two mechanisms:

**1. Plausible deniability through fragment distribution:** Files are split into small encrypted chunks that are distributed across participating nodes. No single node stores a complete file. No single node knows what data it is storing (the chunks are encrypted). A node cannot be held legally responsible for content it cannot read and does not know it has.

**2. Opennet vs. Darknet mode:** In opennet mode, Hyphanet connects to any available node. In darknet mode, it connects only to a list of nodes you explicitly trust. Darknet mode is far more resistant to infiltration and monitoring — an adversary would need to personally befriend all of a node's trusted contacts to analyze its network position.

## Application: Installing Hyphanet

### Install Hyphanet (requires Java)

```
# Install Java:
apt install default-jre -y

# Download the Hyphanet installer:
wget https://github.com/hyphanet/fred/releases/download/build01497/new_installer_offlin

# Run the installer:
java -jar new_installer_offline_1497.jar

# Hyphanet runs a local web interface at:
# http://127.0.0.1:8888

# First run: the wizard guides you through:
# 1. Data store size (how much disk space to contribute)
# 2. Connectivity (opennet or darknet)
# 3. If darknet: add trusted friend nodes by exchanging node references

# Publishing a document (Freesites):
# Use the Freesite Inserter in the web interface
```

```
# After insertion, you receive a permanent URI (key) to share
```

# Chapter 19: Voice — Asterisk and Encrypted Calls

## Theory: How VoIP Works

VoIP (Voice over IP) converts voice audio into digital data packets that travel over a network. The SIP (Session Initiation Protocol) is used to set up and tear down calls. RTP (Real-time Transport Protocol) carries the actual audio.

Asterisk is a free, open-source PBX (Private Branch Exchange) — a phone system. It can route calls between extensions, connect to the public telephone network via SIP trunks, manage voicemail, and run conference bridges.

For a resistance network, Asterisk running on your private infrastructure means:

- Voice calls that do not pass through any telecommunications company
- No call records at any telco (they cannot be subpoenaed)
- SRTP (Secure RTP) for encrypted audio
- Accessible over your WireGuard VPN or as a Tor onion service

## Application: Installing Asterisk

### Basic Asterisk installation and extension setup

```
# Install Asterisk:
apt install asterisk -y

# Configure extensions (/etc/asterisk/sip.conf):
[general]
context=default
udpbindaddr=0.0.0.0:5060
transport=tls    ; Use TLS for signaling
tlscertfile=/etc/ssl/certs/asterisk.crt
tlsprivatekey=/etc/ssl/private/asterisk.key
tlscafile=/etc/ssl/certs/ca.crt

; Extension 100 (Alice):
[alice]
```

```
type=friend
secret=STRONGPASSWORD
host=dynamic     ; Alice's IP is dynamic
context=default
encrypt=yes      ; Require SRTP encryption
qualify=yes      ; Ping regularly to detect availability

; Extension 101 (Bob):
[bob]
type=friend
secret=STRONGPASSWORD2
host=dynamic
context=default
encrypt=yes
qualify=yes

# Configure dialplan (/etc/asterisk/extensions.conf):
[default]
; Dial extension 100:
exten => 100,1,Dial(SIP/alice,30)  ; Ring for 30 seconds
; Dial extension 101:
exten => 101,1,Dial(SIP/bob,30)

# Restart Asterisk:
systemctl enable --now asterisk

# Clients: Install "Linphone" or "Zoiper" on a phone/laptop
# Connect to your Asterisk server's IP over the VPN with SIP+TLS
```

# Part VI — Storage, Databases & Infrastructure

Storing data reliably so nothing is ever lost

# Chapter 20: Storage — ZFS, Ceph, and Backups

## Theory: What ZFS Does That Normal Filesystems Don't

Traditional filesystems (ext4, NTFS, FAT32) are vulnerable to "silent corruption" — data on disk is silently changed by a failing drive, cosmic ray, or software bug, and the filesystem reports the corrupted data as if it were correct. ZFS prevents this with:

- **Checksums on all data and metadata:** Every block on disk has a checksum. On every read, ZFS verifies the checksum. If it fails, ZFS knows the data is corrupt.
- **Copy-on-write:** ZFS never overwrites existing data. When you update a file, ZFS writes the new data to a new location, then atomically updates the pointer. If the system crashes mid-write, the old data is still intact.
- **Snapshots:** Because of copy-on-write, snapshots are instant (zero copy, zero space initially) and cheap. A snapshot captures the exact state of a filesystem at a moment in time. Snapshots only use space as the original data changes.
- **RAID-Z:** ZFS has built-in redundancy levels equivalent to RAID 5/6/7 but without the "RAID-5 write hole" bug that can corrupt data on traditional RAID systems.

## Theory: Ceph — Storage That Scales Across Many Machines

ZFS is excellent for a single server's storage. Ceph is for when you need storage that spans multiple physical machines. Ceph creates a single virtual storage pool from the disks of many servers. Data is automatically distributed and replicated across those disks.

Ceph's key component: the CRUSH algorithm (Controlled Replication Under Scalable Hashing) determines where each piece of data is stored without any centralized lookup table. Every Ceph client can calculate the location of any data independently, enabling massive scaling without bottlenecks.

## Application: Setting Up ZFS on Proxmox

### Create a ZFS pool in Proxmox

```
# Check available disks:
lsblk
```

```
# Create a mirrored ZFS pool (RAID-1 equivalent - requires 2 disks):
# The pool will survive the failure of either disk.
zpool create tank mirror /dev/sdb /dev/sdc

# Or a RAID-Z1 pool (RAID-5 equivalent - requires 3+ disks, can lose 1):
zpool create tank raidz /dev/sdb /dev/sdc /dev/sdd

# Check pool status:
zpool status

# Create a dataset for VM storage:
zfs create tank/vms
zfs set compression=on tank/vms    # Enable transparent compression
zfs set atime=off tank/vms         # Disable access time updates (performance)

# Create a dataset for backups:
zfs create tank/backups
zfs set compression=on tank/backups

# Take a snapshot (instant, no downtime):
zfs snapshot tank/vms@2024-01-15-14h30

# List snapshots:
zfs list -t snapshot

# Rollback to a snapshot (if something went wrong):
zfs rollback tank/vms@2024-01-15-14h30

# Add ZFS pool to Proxmox:
# In Proxmox web UI: Datacenter → Storage → Add → ZFS
# Select your pool (tank) and name it (e.g. "local-zfs")
```

## Application: The 3-2-1 Backup System

### Automated backup script using ZFS send/receive

```
#!/bin/bash
# /opt/scripts/backup.sh
# Sends ZFS snapshots to a remote backup location

DATASET="tank/vms"
REMOTE_HOST="backup.yourdomain"        # Remote server over WireGuard VPN
```

```
REMOTE_USER="backup"
REMOTE_DATASET="backup-pool/vms"
SNAPSHOT_NAME="auto-$(date +%Y%m%d-%H%M)"

# Create a new snapshot:
zfs snapshot "${DATASET}@${SNAPSHOT_NAME}"

# Get the most recent snapshot that was sent to the remote:
LAST_REMOTE=$(ssh "${REMOTE_USER}@${REMOTE_HOST}" \
  "zfs list -t snapshot -H -o name ${REMOTE_DATASET} | tail -1")

if [ -z "$LAST_REMOTE" ]; then
  # First backup - send full snapshot:
  zfs send "${DATASET}@${SNAPSHOT_NAME}" | \
    ssh "${REMOTE_USER}@${REMOTE_HOST}" \
    "zfs receive ${REMOTE_DATASET}"
else
  # Incremental backup - only send what changed:
  LAST_LOCAL=$(echo "$LAST_REMOTE" | sed "s|${REMOTE_DATASET}|${DATASET}|")
  zfs send -i "${LAST_LOCAL}" "${DATASET}@${SNAPSHOT_NAME}" | \
    ssh "${REMOTE_USER}@${REMOTE_HOST}" \
    "zfs receive ${REMOTE_DATASET}"
fi

# Delete snapshots older than 30 days locally:
zfs list -t snapshot -H -o name "${DATASET}" | \
  head -n -30 | xargs -I{} zfs destroy {}

echo "Backup completed: ${SNAPSHOT_NAME}"

# Install as a daily cron job:
# echo "0 3 * * * root /opt/scripts/backup.sh >> /var/log/backup.log 2>&1" \
#   >> /etc/crontab
```

# Chapter 21: Databases — PostgreSQL and Redis

## Theory: Why Your Applications Need Databases

Every stateful service you run needs a place to persistently store data. Matrix stores messages and room state. XMPP stores roster lists and message history. Monitoring systems store metric time series. Databases provide:

- **Persistence:** Data survives crashes and restarts
- **ACID properties:** Atomicity, Consistency, Isolation, Durability — guarantees that operations complete correctly or not at all
- **Querying:** Retrieve exactly the data you need efficiently
- **Replication:** Copy data to standby servers for high availability

## Theory: PostgreSQL — The Production-Grade Database

PostgreSQL is a full-featured relational database. It is what Matrix Synapse uses as its backend (in production — the default SQLite is only for testing). PostgreSQL supports:

- Streaming replication to standby servers (hot standby)
- Logical replication for selective data replication
- Row-level security (further limit what queries can access)
- Automatic failover with Patroni

## Theory: Redis — Fast In-Memory Storage

Redis stores data in memory (RAM) rather than on disk, making it extremely fast. It is used as a cache (storing frequently-accessed data), a message queue (tasks waiting to be processed), and a session store (user login sessions). Matrix Synapse uses Redis for worker coordination.

## Application: PostgreSQL High Availability with Patroni

### Setting up PostgreSQL with automatic failover

```
# Install PostgreSQL on all nodes:
apt install postgresql-16 -y

# Install Patroni (manages automatic failover):
apt install patroni -y

# /etc/patroni/config.yml (example for 3-node cluster):
scope: postgres-ha-cluster
name: node1  # Change to node2, node3 on other nodes

restapi:
  listen: 0.0.0.0:8008
  connect_address: 10.2.0.1:8008  # This node's IP

etcd:
  hosts:
    - 10.2.0.1:2379  # etcd provides distributed state for Patroni
    - 10.2.0.2:2379
    - 10.2.0.3:2379

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576  # 1MB - max allowed replication lag
  initdb:
    - encoding: UTF8
    - data-checksums           # Enable data checksums

postgresql:
  listen: 0.0.0.0:5432
  connect_address: 10.2.0.1:5432
  data_dir: /var/lib/postgresql/16/main
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicator
      password: STRONGPASSWORD
    superuser:
```

```
      username: postgres
      password: STRONGPASSWORD

# Start Patroni:
systemctl enable --now patroni

# Check cluster status:
patronictl -c /etc/patroni/config.yml list
# Shows which node is primary, which are replicas, and replication lag
```

# Chapter 22: Containers — Docker, Podman, and LXC

## Theory: Containers Are Not Virtual Machines

Containers share the host OS kernel but isolate the process using Linux namespaces and cgroups (see Chapter 3). This makes them much lighter than VMs but with slightly weaker isolation. For most internal services (Matrix, IPFS, monitoring), containers are the right choice. For internet-facing services or untrusted code, use full VMs.

**Docker vs. Podman:** Docker requires a background daemon running as root — a security concern. Podman is rootless by design (containers run as a normal user, not root). For security-sensitive infrastructure, prefer Podman. For ease of use and Docker Compose compatibility, Docker is still widely used.

## Application: Running Matrix in Docker

### Docker Compose file for Matrix + PostgreSQL

```
# /opt/matrix/docker-compose.yml
version: '3.8'

services:
  postgres:
    image: postgres:16-alpine
    restart: unless-stopped
    environment:
      POSTGRES_USER: synapse
      POSTGRES_PASSWORD: STRONGPASSWORD    # Change this!
      POSTGRES_DB: synapse
      POSTGRES_INITDB_ARGS: "--encoding=UTF-8 --lc-collate=C --lc-ctype=C"
    volumes:
      - ./postgres-data:/var/lib/postgresql/data
    networks:
      - matrix-internal

  synapse:
    image: matrixdotorg/synapse:latest
    restart: unless-stopped
```

```
    depends_on:
      - postgres
    volumes:
      - ./synapse-data:/data
    ports:
      - "8448:8448"    # Federation port
      - "8008:8008"    # Client API port
    networks:
      - matrix-internal
    environment:
      SYNAPSE_SERVER_NAME: "matrix.yourdomain.onion"
      SYNAPSE_REPORT_STATS: "no"

networks:
  matrix-internal:
    driver: bridge
    internal: true   # No external internet access for this network

# Start:
docker compose up -d

# Check logs:
docker compose logs -f synapse
```

**LXC Containers in Proxmox**

Proxmox includes excellent support for LXC (Linux Containers) through its web interface. LXC containers in Proxmox are easier to manage than Docker for full services, support snapshots, and can be cloned and migrated like VMs. For each major service (Matrix, IPFS, XMPP, monitoring), consider creating a separate LXC container in Proxmox — it gives you clean isolation and easy management from the Proxmox web UI.

# Part VII — Automation & Observability

Making your infrastructure self-managing and self-reporting

# Chapter 23: Infrastructure as Code — Ansible and OpenTofu

### Learning Objectives

- Understand the principle of Infrastructure as Code (IaC)
- Write basic Ansible playbooks to configure servers automatically
- Know when to use Ansible (configuration) vs. OpenTofu (provisioning)

## Theory: Why Infrastructure as Code Matters for Resistance

Consider what happens when a key member of your infrastructure team is suddenly unavailable (arrested, forced to flee, ill). If configuration knowledge exists only in their head, your infrastructure cannot be maintained or rebuilt. Infrastructure as Code solves this: all configuration is in text files in a Git repository. Anyone with access to the repository and the encryption keys can reproduce the entire infrastructure.

**Ansible:** Agentless configuration management. You run it from your laptop; it SSHs into remote servers and configures them. You write "playbooks" — YAML files describing the desired state of a server.

**OpenTofu:** Infrastructure provisioning. It creates the servers, networks, and storage from text descriptions. Works with cloud providers (Hetzner, DigitalOcean, Linode) and local hypervisors (Proxmox).

OpenTofu builds the house (creates the VM, allocates the IP address, attaches the storage). Ansible furnishes and decorates the house (installs the software, writes the configuration files, starts the services). You need both for a complete infrastructure.

## Application: Writing an Ansible Playbook for WireGuard

### Ansible playbook: configure WireGuard on all nodes

```
# inventory.yml - list of servers to manage:
all:
  hosts:
    node-a:
      ansible_host: 10.0.0.1    # Management IP of each node
      wireguard_ip: 10.2.0.1    # WireGuard IP for this node
    node-b:
```

```
      ansible_host: 10.0.0.2
      wireguard_ip: 10.2.0.2
    node-c:
      ansible_host: 10.0.0.3
      wireguard_ip: 10.2.0.3

---
# playbooks/wireguard.yml - the playbook:
- name: Configure WireGuard mesh VPN
  hosts: all
  become: true    # Run as root

  tasks:
    - name: Install WireGuard
      apt:
        name: wireguard
        state: present
        update_cache: yes

    - name: Generate private key if not present
      shell: |
        wg genkey | tee /etc/wireguard/private.key | \
        wg pubkey > /etc/wireguard/public.key
        chmod 600 /etc/wireguard/private.key
      args:
        creates: /etc/wireguard/private.key  # Only run if file doesn't exist

    - name: Read private key
      slurp:
        src: /etc/wireguard/private.key
      register: private_key

    - name: Write WireGuard interface config
      template:
        src: templates/wg0.conf.j2    # Jinja2 template
        dest: /etc/wireguard/wg0.conf
        mode: '0600'

    - name: Enable and start WireGuard
      systemd:
        name: wg-quick@wg0
        enabled: yes
        state: started

# Run the playbook:
```

```
# ansible-playbook -i inventory.yml playbooks/wireguard.yml

# Run only for one host:
# ansible-playbook -i inventory.yml --limit node-a playbooks/wireguard.yml

# Check what would change without making changes (dry run):
# ansible-playbook -i inventory.yml playbooks/wireguard.yml --check
```

## Application: OpenTofu to Provision VMs on Proxmox

### OpenTofu configuration for a Proxmox VM

```
# main.tf
terraform {
  required_providers {
    proxmox = {
      source  = "bpg/proxmox"
      version = "~> 0.50"
    }
  }
}

provider "proxmox" {
  endpoint  = "https://192.168.1.10:8006"
  api_token = "root@pam!terraform=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  insecure  = true    # For self-signed cert (replace with real cert)
}

# Create a VM from a cloud-init template:
resource "proxmox_virtual_environment_vm" "matrix_server" {
  name      = "matrix-server"
  node_name = "proxmox-node1"
  vm_id     = 101

  cpu {
    cores = 4
    type  = "x86-64-v2-AES"
  }

  memory {
    dedicated = 8192    # 8 GB RAM
  }
```

```
  disk {
    datastore_id = "local-zfs"
    file_id      = "local:iso/ubuntu-24.04-server.img"
    size         = 50   # 50 GB
  }

  network_device {
    bridge = "vmbr0"
  }

  # Cloud-init configuration:
  initialization {
    ip_config {
      ipv4 {
        address = "10.1.0.10/24"
        gateway = "10.1.0.1"
      }
    }
    user_account {
      username = "ansible"
      keys     = [file("~/.ssh/id_ed25519.pub")]  # Your SSH public key
    }
  }
}

# Deploy:
# tofu init
# tofu plan    # Preview what will be created
# tofu apply   # Create the VM
```

# Chapter 24: Monitoring — Seeing What Is Happening

## Theory: The Observability Stack

Observability is the ability to understand the internal state of a system from its external outputs. It has three pillars:

- **Metrics:** Numerical measurements over time (CPU %, requests/second, queue depth)
- **Logs:** Timestamped records of events ("user logged in", "error: database connection failed")
- **Traces:** Records of how a single request flows through multiple services

For a resistance network, you primarily need metrics and logs. The monitoring stack:

| Tool | Pillar | Job |
|---|---|---|
| **Node Exporter** | Metrics | Exports Linux system metrics (CPU, RAM, disk, network) for Prometheus to collect |
| **cAdvisor** | Metrics | Exports container metrics (Docker/Podman container resource usage) |
| **Prometheus** | Metrics | Scrapes (collects) metrics from all exporters every 15 seconds; stores them in a time-series database |
| **Alertmanager** | Metrics | Receives alert rules from Prometheus; routes and deduplicates alerts; sends to Matrix/email |
| **Loki** | Logs | Collects and indexes logs from all services; searchable like Prometheus metrics |
| **Grafana** | Both | Visualization: dashboards for metrics (from Prometheus) and logs (from Loki) |

## Application: Deploying the Monitoring Stack

### Deploy Prometheus + Grafana + Loki with Docker Compose

```
# /opt/monitoring/docker-compose.yml
version: '3.8'

services:
  prometheus:
    image: prom/prometheus:latest
    restart: unless-stopped
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    ports:
      - "127.0.0.1:9090:9090"   # Only accessible from localhost
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.retention.time=90d'  # Keep 90 days of data

  grafana:
    image: grafana/grafana:latest
    restart: unless-stopped
    volumes:
      - grafana-data:/var/lib/grafana
    ports:
      - "127.0.0.1:3000:3000"   # Access via SSH tunnel or VPN only
    environment:
      GF_AUTH_ANONYMOUS_ENABLED: "false"
      GF_SECURITY_ADMIN_PASSWORD: "STRONGPASSWORD"

  loki:
    image: grafana/loki:latest
    restart: unless-stopped
    ports:
      - "127.0.0.1:3100:3100"
    command: -config.file=/etc/loki/local-config.yaml

  alertmanager:
    image: prom/alertmanager:latest
    restart: unless-stopped
    volumes:
      - ./alertmanager.yml:/etc/alertmanager/alertmanager.yml
    ports:
```

```
      - "127.0.0.1:9093:9093"

volumes:
  prometheus-data:
  grafana-data:
```

### Prometheus configuration: scrape targets

```
# /opt/monitoring/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'node-exporters'
    static_configs:
      - targets:
        - '10.2.0.1:9100'   # Node A - Node Exporter
        - '10.2.0.2:9100'   # Node B - Node Exporter
        - '10.2.0.3:9100'   # Node C - Node Exporter

  - job_name: 'cadvisor'
    static_configs:
      - targets:
        - '10.2.0.1:8080'   # Container metrics from Node A

  - job_name: 'matrix-synapse'
    metrics_path: '/_synapse/metrics'
    static_configs:
      - targets:
        - '10.2.0.2:8008'

rule_files:
  - 'alerts.yml'

alerting:
  alertmanagers:
    - static_configs:
      - targets: ['alertmanager:9093']
```

### Alerting rules: critical alerts for resistance infrastructure

```
# /opt/monitoring/alerts.yml
groups:
  - name: critical
    rules:
      - alert: ServiceDown
        expr: up == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "Service {{ $labels.job }} on {{ $labels.instance }} is down"

      - alert: DiskSpaceLow
        expr: (node_filesystem_avail_bytes / node_filesystem_size_bytes) < 0.15
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Disk space below 15% on {{ $labels.instance }}"

      - alert: HighMemoryUsage
        expr: (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) > 0.9
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Memory usage above 90% on {{ $labels.instance }}"

      - alert: WireGuardPeerDown
        expr: wireguard_peer_last_handshake_seconds > 180
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "WireGuard peer has not handshaked in 3 minutes"
```

# Part VIII — Security

# Chapter 25: Threat Modeling — Know Your Adversary

## Theory: The STRIDE Framework

Threat modeling is the practice of systematically identifying what can go wrong before it does. The STRIDE framework is a structured way to think through threats:

| Letter | Threat | Example in our context |
|---|---|---|
| S | Spoofing (pretending to be someone else) | Adversary impersonates a trusted peer's WireGuard identity |
| T | Tampering (modifying data) | Adversary alters a published document in transit |
| R | Repudiation (denying you did something) | Member denies sending a message — or falsely accused |
| I | Information Disclosure (data leaks) | Unencrypted logs containing IP addresses or usernames |
| D | Denial of Service (making service unavailable) | DDoS attack against your Matrix server |
| E | Elevation of Privilege (gaining unauthorized access) | Attacker exploits a Synapse bug to gain root access |

## Application: Building Your Threat Model

### Four questions to answer before building anything

1. **What assets am I protecting?** List: message content, member identities, server locations, operational plans, document cache.
2. **Who are my adversaries?** Are they a local police department? A national intelligence agency? A criminal group? Each has different capabilities.
3. **What are my adversary's capabilities?** Can they do passive monitoring only? Can they conduct raids? Can they compromise global internet infrastructure?
4. **What is my acceptable risk?** What threats am I willing to accept vs. what must I defend against at all costs?

| Adversary | Capabilities | Primary defenses |
|---|---|---|
| Local law enforcement | Seize devices; access ISP records; limited technical capability | Full disk encryption; traffic encryption; no logs |
| National intelligence | Mass surveillance; legal demands to providers; limited hacking | Tor; self-hosted infrastructure; no third-party providers |
| Nation-state intelligence | Zero-day exploits; compromising update infrastructure; human infiltration | Air-gapped systems; open hardware; need-to-know principle; physical security |

# Chapter 26: Cryptography From First Principles

## Theory: Symmetric Encryption

Symmetric encryption uses the same key to encrypt and decrypt. It is fast — modern CPUs can encrypt gigabytes per second using hardware instructions. The challenge: how do you securely share the key with the other party in the first place?

Used for: encrypting data at rest (disk encryption with LUKS), bulk data transfer inside an established VPN tunnel.

**AES-256-GCM** is the modern standard. The "256" means a 256-bit key — there are $2^{256}$ possible keys, a number larger than the number of atoms in the observable universe. Brute-forcing it is computationally impossible.

## Theory: Asymmetric (Public-Key) Encryption

Asymmetric encryption uses a *pair* of mathematically related keys. What one key encrypts, only the other can decrypt. It solves the key exchange problem: you can publish your public key openly; anyone can encrypt a message with it that only your private key can decrypt.

**Key pair uses:**

- **Encryption:** Encrypt with recipient's public key → only recipient's private key can decrypt
- **Signing:** Sign with your own private key → anyone with your public key can verify the signature is genuine

**Diffie-Hellman Key Exchange:** How two parties agree on a shared secret key over an insecure channel — without ever sending the key itself. This is what enables the initial handshake in WireGuard, TLS, and Signal's Double Ratchet protocol.

## Theory: The Double Ratchet Algorithm (Signal, Matrix OMEMO)

The Double Ratchet is what makes Signal, Matrix (E2EE rooms), and XMPP (OMEMO) resistant even to key compromise. It has two ratchets:

- **Diffie-Hellman ratchet:** New Diffie-Hellman keys are exchanged with every message, ensuring that even if a key is compromised, only recent messages are at risk (perfect forward secrecy)

- **Symmetric ratchet:** Each message derives a new key from the previous one, so past and future messages remain secure even if one message key is compromised

Each message uses a different key. Keys are derived sequentially from a master key, like pages torn from a pad — once used and destroyed, they are gone. Capturing message 15's key tells you nothing about messages 1–14 (past) or 16+ (future). This property is called "forward secrecy" and "break-in recovery."

## Application: Practical Cryptography Tools

### GPG: encrypting files and email

```
# Generate a GPG key pair:
gpg --full-gen-key
# Choose: RSA and RSA, key size 4096, no expiry (or set an expiry for security)
# Provide: your pseudonym (NOT your real name), a pseudonymous email

# Export your public key (share this with people who want to send you encrypted files):
gpg --armor --export yourpseudo@example.com > yourpseudo-public.asc

# Import someone else's public key:
gpg --import theirpseudo-public.asc

# Encrypt a file for a recipient (they use their private key to decrypt):
gpg --encrypt --armor --recipient theirpseudo@example.com secret-document.pdf

# Decrypt a file encrypted for you:
gpg --decrypt secret-document.pdf.asc

# Sign a file (proves it came from you, without encrypting it):
gpg --detach-sign --armor document.pdf
# Creates document.pdf.asc (the signature file)

# Verify a signature:
gpg --verify document.pdf.asc document.pdf
```

### LUKS: full disk encryption

```
# Encrypt a new disk/partition with LUKS:
# WARNING: This destroys all data on the disk. Back up first!
cryptsetup luksFormat /dev/sdb
```

```
# Open (unlock) an encrypted disk:
cryptsetup open /dev/sdb encrypted-disk

# Now format and use it:
mkfs.ext4 /dev/mapper/encrypted-disk
mount /dev/mapper/encrypted-disk /mnt/secure

# Close (lock) when done:
umount /mnt/secure
cryptsetup close encrypted-disk

# For automatic unlock at boot (e.g., using a key file stored on a trusted USB):
# /etc/crypttab:
# encrypted-disk UUID=xxxx /path/to/keyfile luks
```

# Chapter 27: Operational Security — The Human Layer

## Theory: Why Technology Alone Is Not Enough

The most common cause of compromise in real-world resistance operations is not technical — it is human. Someone tells the wrong person. Someone logs in from home when they should use Tor. Someone uses the same username on two different platforms. Someone loses a device with unencrypted data.

OpSec (Operational Security) is the discipline of minimizing information leakage through human behavior. It originated in military intelligence and has been adapted by journalists, activists, and security researchers.

## Theory: The Five-Step OpSec Process

1. **Identify critical information:** What do you need to protect? Member identities, server locations, operational plans, communications content, financial resources.

2. **Analyze threats:** Who wants this information? What are they willing and able to do to get it?

3. **Analyze vulnerabilities:** Where are the gaps between what you need to protect and how well you are protecting it?

4. **Assess risk:** For each vulnerability, how likely is exploitation and how damaging would it be?

5. **Apply counter-measures:** Implement the measures that address the highest-risk vulnerabilities.

## Application: OpSec Checklist

| Practice | Why | How |
|---|---|---|
| Compartmentalize | Limit damage from any single person being compromised | Divide operations into cells; no member knows members of other cells; need-to-know only |
| Separate identities | Linking a pseudonym to a real identity compromises both | Different devices, different usernames, different email addresses for different roles |
| Use Tor for sensitive browsing | Your ISP can see what IP addresses you connect to even if traffic is encrypted | Always use Tor Browser for anything related to your work |
| No phone in secure locations | Phones track location via cell towers even without GPS; microphones can be exploited | Leave phones at home or in a Faraday bag during sensitive meetings |
| Verify before trusting | Infiltrators are a real threat; anyone can claim to be trustworthy | Cryptographic key verification; vouching chains; in-person introduction |
| Minimize data retention | Data that does not exist cannot be seized or leaked | Delete messages after reading; use self-destructing message settings; minimal logging |
| Encrypt everything at rest | Seized hardware reveals nothing if encrypted | LUKS on all drives; VeraCrypt for portable encrypted volumes |
| Strong unique passwords | Reused passwords compromise multiple accounts when one is breached | Use Bitwarden (self-hosted) or KeePassXC (local); generate 20+ character random passwords |
| Hardware MFA | Software TOTP codes can be phished; hardware keys cannot | YubiKey or SoloKey for all critical accounts |

**Metadata: The Information You Forget to Hide**

Even with perfect message encryption, metadata reveals a great deal: who communicated with whom, when, how often, and for how long. An intelligence agency does not need to read your messages if they can infer from metadata that you called the journalist in the hour before the article was published. Counter-measure: Tor hides destination metadata from your ISP. Matrix over a private server reduces the metadata exposed to third parties. But only reducing contact frequency and using dead-drops truly minimizes metadata.

# Part IX — Radio

Communicating when the internet is completely gone

# Chapter 28: Radio Fundamentals and the Spectrum

### Learning Objectives

- Understand what radio waves are and how they travel
- Know which frequency bands are useful for different scenarios
- Understand modulation — how information is encoded onto radio waves

## Theory: What Radio Waves Are

Radio waves are electromagnetic radiation — the same physical phenomenon as visible light, X-rays, and microwaves, just at different frequencies. They propagate (travel) at the speed of light and can travel through air, vacuum, and some materials without any infrastructure.

Key properties:

- **Frequency:** How many times per second the wave oscillates. Measured in Hertz (Hz). Higher frequency = more oscillations per second.
- **Wavelength:** The physical distance between two wave peaks. Wavelength = speed of light / frequency. Higher frequency = shorter wavelength.
- **Power:** How strong the signal is. Measured in Watts. More power = farther range (but also more detectability).

## Theory: How Different Frequencies Travel

| Frequency | Name | How It Travels | Range | Use |
|---|---|---|---|---|
| 3–30 MHz | HF (High Frequency) / Shortwave | Bounces off the ionosphere (sky wave) — can travel around the world | Global | International shortwave radio; best for long-range resistance comms |
| 30–300 MHz | VHF (Very High Frequency) | Line-of-sight; some ducting (atmospheric bounce) | City/regional (30–300 km with good antennas) | FM radio; aviation comms; emergency services |
| 300 MHz–3 GHz | UHF (Ultra High Frequency) | Line-of-sight; penetrates walls | Local (1–50 km) | Wi-Fi; mobile phones; walkie-talkies |
| 865–928 MHz | LoRa ISM band | Line-of-sight with penetration | 2–15 km urban; up to 40 km rural | Low-power digital messages without internet |

## Theory: Modulation — Encoding Information onto Radio

A radio wave by itself (a "carrier wave") carries no information. Modulation changes some property of the carrier wave to encode information:

| Modulation | What Changes | Characteristics | Use |
|---|---|---|---|
| **AM (Amplitude Modulation)** | The strength (amplitude) of the wave | Simple; susceptible to interference; travels far on HF | Standard AM radio broadcasting |
| **FM (Frequency Modulation)** | The frequency of the wave | Better audio quality; more resistant to static | FM radio broadcasting; walkie-talkies |
| **SSB (Single Sideband)** | A single sideband of an AM signal | Very efficient — 3x the range of AM for the same power; standard for HF voice | Amateur radio HF voice; maritime radio |
| **CW (Morse Code)** | Turning the carrier on and off | Extremely narrow bandwidth; cuts through interference; works when voice cannot | Emergency communications; long-distance HF |
| **FSK (Frequency Shift Keying)** | Alternates between two frequencies to represent 0 and 1 | Robust digital mode | APRS; packet radio; low-rate data |
| **LoRa (Long Range)** | Chirp spread spectrum (frequency sweeps) | Extremely long range at very low power; digital | IoT sensors; message networks without internet |

**The Most Important Radio Skill: Listening**

Before you transmit anything, spend time listening. An RTL-SDR receiver costs $25 and lets you listen to virtually everything across a huge frequency range. Listen to emergency services, aviation communications, weather satellites, and shortwave broadcasts in your area. Understanding what frequencies are used for what, and where transmissions originate, is invaluable intelligence for any operation — and requires no license to do.

# Chapter 29: Software Defined Radio — Hardware and Software

## Theory: What "Software Defined" Means

A traditional radio is hardware-defined: its circuitry determines what it can receive and transmit, at what frequencies, and using what modulation. To change what it does, you need different hardware.

A Software Defined Radio (SDR) replaces most of that hardware with a general-purpose analog-to-digital converter (ADC) and a computer. The ADC samples raw radio signals at very high speed. The computer then performs all the signal processing in software — filtering, demodulation, decoding. To receive a different mode, you load different software — or write your own.

## Theory: IQ Sampling in Plain Language

To fully capture a radio signal, you need to know two things at each moment: its amplitude (strength) and its phase (where it is in its cycle). A single measurement at one point in time tells you only amplitude. IQ sampling captures both by measuring the signal twice simultaneously, 90 degrees apart in phase.

These two measurements are called the "I" (In-phase) component and the "Q" (Quadrature — 90 degrees offset) component. Together, they fully describe the signal, enabling any type of demodulation in software.

## Application: SDR Hardware Comparison

| Device | Frequency Range | TX/RX | Cost | Best For |
|---|---|---|---|---|
| **RTL-SDR Blog V4** | 500 kHz – 1.7 GHz | RX only | $35 | First device; listening, monitoring, ADS-B, weather sats |
| **HackRF One** | 1 MHz – 6 GHz | TX+RX (half-duplex) | $300 | Versatile experimentation; transmitting; signal analysis |
| **LimeSDR Mini 2** | 10 MHz – 3.5 GHz | TX+RX (full-duplex) | $250 | Two-way communications; higher performance than HackRF |
| **PlutoSDR (ADALM-PLUTO)** | 325 MHz – 3.8 GHz | TX+RX | $150 | Learning; good documentation; medium performance |

## Application: Setting Up SDR++ for Monitoring

### Install and use SDR++ (Linux)

```
# Download SDR++ AppImage from sdrpp.org/downloads:
wget https://github.com/AlexandreRouma/SDRPlusPlus/releases/download/nightly/sdrpp_linu
chmod +x sdrpp_linux_amd64.AppImage

# Plug in your RTL-SDR, run SDR++:
./sdrpp_linux_amd64.AppImage

# In SDR++:
# 1. Source → Select "RTL-SDR"
# 2. Click "Play" button to start receiving
# 3. Click on the frequency display and type a frequency
# 4. Select demodulation mode (AM/FM/SSB/CW) from the dropdown
# 5. Adjust the waterfall contrast to see signals clearly

# Useful frequencies to monitor (vary by country):
# - 121.5 MHz: International aviation emergency frequency
```

```
# - 156.8 MHz: Marine Channel 16 (international distress)
# - 162.4-162.55 MHz: NOAA Weather Radio (US)
# - 406-406.1 MHz: Emergency Position Indicating Radio Beacons (EPIRBs)
# - 1090 MHz: ADS-B aircraft transponders (decode with dump1090)
```

### Decode ADS-B aircraft positions in real time

```
# Install dump1090 (ADS-B decoder):
apt install dump1090-fa -y

# Run with your RTL-SDR connected:
dump1090 --net --interactive

# View aircraft positions at:
# http://localhost:8080

# This shows every aircraft within range broadcasting its
# position, altitude, speed, and flight number - no internet needed.
# Useful for tracking movements in your area.
```

# Chapter 30: Building a Radio Communication Node

## The Complete Radio Stack for a Resistance Node

A fully equipped radio node provides communication capability at every scale — from local (mesh/Wi-Fi) to continental (HF shortwave) — without depending on any internet infrastructure.

| Layer | Equipment | Range | Purpose |
|---|---|---|---|
| Local | Wi-Fi router running BATMAN-adv, Raspberry Pi | 100–500m | Local mesh when internet is down |
| City | RTL-SDR or HackRF + directional antenna | 5–30 km | Monitoring; APRS messaging |
| Regional | LoRa gateway (e.g., RAK WisGate) or HF transceiver | 40–200 km | Regional coordination without internet |
| Continental | HF SSB transceiver (e.g., Icom IC-7300) + dipole antenna | World | Long-distance coordination; news reception from free broadcasters |

## Application: APRS — Radio Text Messaging Without Internet

### Set up an APRS digipeater and message gateway

```
# Install Dire Wolf (APRS software modem/digipeater):
apt install direwolf -y

# /etc/direwolf.conf:
ADEVICE default             # Use default sound card (or RTL-SDR via rtl_fm)
CHANNEL 0
MYCALL CALLSIGN-9           # Your APRS callsign (requires amateur radio license)
MODEM 1200                  # 1200 baud AFSK (standard APRS)

# Digipeater (repeat received packets to extend range):
DIGIPEAT 0 0 ^WIDE[3-7]-[3-7]$ ^WIDE[12]-[12]$
```

```
# I-Gate (connect APRS to internet when available):
# IGSERVER rotate.aprs2.net
# IGLOGIN CALLSIGN-9 PASSCODE

# Run Dire Wolf:
direwolf -c /etc/direwolf.conf

# Send a message from another APRS device:
# Use any APRS app (APRSDroid on Android) - your gateway
# will receive and relay it.

# Without a license (listen-only mode):
# Remove MYCALL and DIGIPEAT lines; just monitor
rtl_fm -f 144.39M -s 22050 | direwolf -c /dev/stdin -r 22050 -n 1
```

**LoRa Mesh Networks: No License Required**

LoRa operates in license-free ISM (Industrial, Scientific, Medical) bands in most countries. The Meshtastic project (meshtastic.org) provides open-source firmware for LoRa devices that creates a text message mesh network. Each node repeats messages from others, extending range. A $30 LoRa module running Meshtastic can send text messages to another node 10+ km away — completely without internet, cellular, or any infrastructure.

# Part X — Putting It All Together

# Chapter 31: Reliability Engineering — When Things Break

## Theory: Failure Modes and Their Probabilities

In reliability engineering, we think in terms of failure modes — the specific ways a system can fail — and design against each one. For a resistance network, failure modes include:

| Failure Mode | Probability | Impact | Counter-measure |
|---|---|---|---|
| Hard drive failure | High (1–5% per drive per year) | Data loss; service outage | ZFS mirror; regular backups |
| Power outage | Medium (depends on location) | Service outage | UPS battery backup; generator |
| ISP outage or throttling | High (in an authoritarian context) | Loss of internet connectivity | Second ISP; 4G backup; radio |
| Physical raid/seizure | Low-Medium | Total loss of one site | Geographic distribution; remote backups |
| Software vulnerability exploit | Medium | Service compromise; data breach | Regular updates; minimal attack surface; network segmentation |
| Operator error | High | Varies widely | Infrastructure as Code; dry-run testing; change management |
| Member arrest/departure | Medium | Loss of institutional knowledge | Documentation; multiple operators per system |

## Theory: RPO and RTO

**Recovery Point Objective (RPO):** The maximum acceptable amount of data loss, measured in time. If you back up every 4 hours, your RPO is 4 hours — in the worst case, you could lose 4 hours of messages. For critical communications data, target RPO of 1 hour or less.

**Recovery Time Objective (RTO):** The maximum acceptable time for a service to be unavailable after a failure. For critical communications (Matrix, XMPP), target RTO of 30 minutes or less. For non-critical services (monitoring, file archival), RTO of several hours may be acceptable.

## Application: Chaos Engineering Exercises

### Monthly resilience drills

```
# Exercise 1: Node failure
# Shut down Node A completely (without warning the team):
proxmox: Right-click VM → Stop

# Observe:
# - Does Matrix automatically fail over to Node B? (Yes if HA is configured)
# - Do WireGuard peers reconnect automatically? (Yes - persistent keepalive)
# - Do Prometheus alerts fire within 2 minutes? (Check Alertmanager)
# - Can users still send messages? (Check on client apps)

# Exercise 2: Disk failure simulation
# On a running ZFS mirror, offline one disk:
zpool offline tank sdb

# Observe:
# - Does `zpool status` show the pool as DEGRADED? (It should)
# - Do services continue running? (Yes - mirror still has one disk)
# - Does Prometheus alert on the degraded pool? (Configure this alert)

# Replace the "failed" disk and restore the mirror:
zpool online tank sdb   # Re-online the test disk

# Exercise 3: Backup restore drill
# Take a real backup and restore it to a fresh VM:
# 1. Create a new VM in Proxmox
# 2. Restore your latest Matrix backup
# 3. Start the service
# 4. Verify messages are accessible
# 5. Measure how long this took - target: under 30 minutes

# Exercise 4: Internet cutoff
# Unplug the WAN connection (or block it with nftables):
nft add rule inet filter output oif eth0 drop comment "Simulate WAN cutoff"

# Observe:
# - Does local mesh (BATMAN-adv) still work? (It should)
# - Can users communicate via the local Matrix server? (If it is on the local mesh, yes
# - Do Tor onion services still work? (Yes - Tor is already on the mesh)

# Restore:
```

```
nft delete rule inet filter output oif eth0 drop
```

# Chapter 32: Progressive Lab Design — Small to Regional

## Scale 1: Individual or Pair — The "Go Bag" Lab

A single machine that can be set up quickly at any location. Prioritizes portability and speed of deployment over redundancy.

```
Hardware:
  - 1x refurbished laptop or mini PC
  - 1x 1 TB SSD (external, encrypted)
  - 1x RTL-SDR dongle + antenna
  - 1x 4G USB modem (backup internet)
  - 1x TP-Link TL-MR3020 (travel router for local network)

Software stack:
  - Proxmox VE on the laptop
  - WireGuard (connect to other trusted peers)
  - Matrix Synapse (personal homeserver)
  - Tor (for browsing and onion services)
  - IPFS (for document sharing)

Setup time: 2 hours (first time); 30 minutes with Ansible playbook
Cost: $200-400 (depending on what you have)
```

## Scale 2: Small Group — The Community Hub

```
Hardware (per location - aim for 2 locations):
  - 3x mini PCs or refurbished desktops (e.g., Intel NUC, HP EliteDesk)
  - 16 GB RAM each, 500 GB SSD each
  - 1x 8-port managed switch (VLAN-capable)
  - 1x UPS battery backup
  - 1x RTL-SDR + outdoor antenna (on a tripod or window mount)
  - 1x 4G router for backup internet

Network diagram:
```

```
  Location A:                Location B:
  [Node A1]─[Node A2]       [Node B1]
      |    ×    |               |
  [Switch A]               [Switch B]
      |                        |
  [Router A]               [Router B]
      |                        |
  [Internet A]            [Internet B]
      └──────────WireGuard VPN────────┘

Services:
  Matrix: Primary on A1, replica on A2, cold standby on B1
  PostgreSQL: 3-node cluster (A1 + A2 + B1) with Patroni
  IPFS: All 3 nodes pin critical content
  Monitoring: Prometheus on B1 (separate from what it monitors)

Monthly cost: ~$0 (hardware is one-time; electricity is low for mini PCs)
```

## Scale 3: Regional Network — The Distributed Collective

```
Multiple cities, each with 2-3 nodes:

City A ─── City B ─── City C
  │           │          │
  └───────────┴──────────┘
        Yggdrasil + WireGuard mesh overlay

Within each city: BATMAN-adv Wi-Fi mesh for local comms without WAN
Between cities: WireGuard tunnels over whatever internet is available
Fallback: HF radio stations in each city for when all internet is cut

Services architecture:
  Each city: runs own Matrix homeserver (federated with other cities)
  Each city: runs own XMPP server (federated)
  Each city: runs own IPFS pinning cluster
  Each city: runs own Tor bridge relay
  One location: runs Asterisk for voice coordination
  All locations: contribute to Prometheus federation for global monitoring

Capacity planning per city (serves ~50 active users):
  Storage: 2 TB (messages + files) → grows ~100 GB/year at moderate usage
  RAM: 32 GB (Matrix + PostgreSQL + IPFS + monitoring)
```

```
Bandwidth: 50 Mbps (adequate for 50 users)
Backup: 6 TB offsite (3x 2 TB rotating)
```

## Network Topology Templates

**IP addressing scheme (consistent across all scales):**

| Subnet | Purpose | Notes |
|---|---|---|
| 10.0.0.0/24 | Management | Admin access, Proxmox UI, SSH — restrict to trusted admin IPs only |
| 10.1.0.0/24 | Services | Matrix, XMPP, IPFS, Asterisk containers |
| 10.2.0.0/24 | WireGuard VPN mesh | All nodes' WireGuard tunnel addresses |
| 10.3.0.0/24 | Monitoring | Prometheus, Grafana, Loki |
| 192.168.100.0/24 | BATMAN-adv mesh | Local physical mesh addresses |
| 200::/7 | Yggdrasil | Auto-assigned from public key; globally unique within Yggdrasil |

**VLAN plan:**

| VLAN ID | Name | Traffic | Access Policy |
|---|---|---|---|
| 10 | Management | SSH, Proxmox UI, Ansible | Admin IPs only; no internet access |
| 20 | Services | Matrix, XMPP, IPFS ports | VPN peers can reach; internet via Tor only |
| 30 | Monitoring | Prometheus scrape, Grafana | Management VLAN only; no external access |
| 40 | External | Public-facing (via Tor onion) | Strict egress; only Tor port 9050 outbound |

# Chapter 33: Quick Reference and Appendices

## Priority Order: What to Build First

| Priority | What | Why | Effort |
|---|---|---|---|
| 1 — Critical | Full disk encryption on all machines | Seized hardware reveals nothing | 1 hour |
| 2 — Critical | WireGuard mesh VPN between all nodes | Encrypted backbone for all services | 2–4 hours |
| 3 — Critical | Matrix homeserver on Tor onion | Primary encrypted group communication | 4–8 hours |
| 4 — High | Automated backups (ZFS + offsite) | Survive seizure of one location | 2–4 hours |
| 5 — High | Tor Browser + Tor bridges configured | Anonymous access when Tor is blocked | 30 minutes |
| 6 — High | IPFS node with critical documents pinned | Censorship-resistant publishing | 2 hours |
| 7 — Medium | RTL-SDR + monitoring setup | Situational awareness; radio fallback | 2 hours |
| 8 — Medium | Monitoring stack (Prometheus + Grafana) | Know when things fail | 4–6 hours |
| 9 — Medium | BATMAN-adv local mesh | Communicate if internet is cut | 4–8 hours |
| 10 — Medium | HF/LoRa radio capability | Ultimate fallback | Variable (requires equipment) |

## Complete Software Toolkit

| Category | Tool | Purpose | Where to Get |
|---|---|---|---|
| Hypervisor | Proxmox VE | Run VMs and containers | proxmox.com |
| VPN | WireGuard | Encrypted mesh VPN | wireguard.com |
| Mesh | BATMAN-adv | Local Wi-Fi mesh | open-mesh.org |
| Overlay | Yggdrasil | Self-organizing encrypted network | yggdrasil-network.github.io |
| Anonymity | Tor | Anonymous routing + onion services | torproject.org |
| Anonymity | I2P | Hidden internal services | geti2p.net |
| Messaging | Matrix (Synapse) | Federated encrypted group chat | matrix.org |
| Messaging | XMPP (Prosody) | Federated encrypted chat | prosody.im |
| Files | IPFS (Kubo) | Content-addressed file system | ipfs.tech |
| Publishing | Hyphanet | Censorship-resistant anonymous publishing | hyphanet.org |
| Voice | Asterisk | Private encrypted voice calls | asterisk.org |
| Storage | ZFS (via Proxmox) | Self-healing RAID storage | Built into Proxmox |
| Database | PostgreSQL | Reliable relational database | postgresql.org |
| Cache | Redis | Fast in-memory cache | redis.io |
| Automation | Ansible | Configure machines automatically | ansible.com |
| IaC | OpenTofu | Provision infrastructure | opentofu.org |
| Metrics | Prometheus + Grafana | Monitoring and dashboards | prometheus.io; grafana.com |
| Logs | Loki | Centralized log aggregation | grafana.com/oss/loki |
| SDR | SDR++ / GNU Radio | Software defined radio | sdrpp.org; gnuradio.org |
| Encryption | GPG / Age | File and email encryption | gnupg.org; age-encryption.org |
| Disk encryption | LUKS (via cryptsetup) | Full disk encryption | gitlab.com/cryptsetup/cryptsetup |
| Password manager | Bitwarden (self-hosted) / KeePassXC | Secure password storage | bitwarden.com; keepassxc.org |

## Troubleshooting: The Most Common Problems

| Problem | First Check | Second Check | Solution | | |
|---|---|---|---|---|---|
| WireGuard peers not connecting | `wg show` — is the interface up? | Firewall: <code>nft list ruleset | grep 51820</code> | Ensure UDP 51820 is open; verify public keys match; check endpoints | |
| Matrix messages not federating | Check port 8448 is reachable from outside | <code>journalctl -u matrix-synapse | grep federation</code> | Verify DNS SRV records; check firewall; verify TLS certificate | |
| Tor not connecting | <code>journalctl -u tor | grep -i error</code> | Is Tor blocked? Try bridges | Configure obfs4 bridge in `/etc/tor/torrc` | |
| IPFS content not found | `ipfs swarm peers` — any peers? | <code>ipfs pin ls | grep [CID]</code> | Ensure at least one node has the CID pinned; check firewall port 4001 | |
| Proxmox web UI unreachable | Can you SSH to the host? | `systemctl status pveproxy` | Restart pveproxy; check firewall allows port 8006 from admin IP | | |
| PostgreSQL replication lag | `patronictl list` | Check replica disk I/O: `iostat -x 1` | Increase wal_sender_timeout; ensure replica hardware is fast enough | | |
| Service crashes repeatedly | `journalctl -u [service] --since "1 hour ago"` | Check OOM killer: <code>journalctl -k | grep -i killed</code> | If OOM: allocate more RAM or add limits to other containers | |
| Disk full | `df -h` | <code>du -sh /* | sort -rh | head -20</code> | Identify large directories; clean old logs; ZFS: destroy old snapshots |

# Glossary

| Term | Plain-language definition |
|---|---|
| APRS | Automatic Packet Reporting System — a digital radio protocol for sending position and text messages without internet |
| BATMAN-adv | A mesh networking protocol where every device is both a client and a router |
| BGP | Border Gateway Protocol — the routing protocol that connects ISPs and countries to form the global internet |
| cgroups | Linux control groups — limit how much CPU/RAM a process or container can use |
| CID | Content Identifier — a hash-based address used in IPFS to name files by their content rather than their location |
| DHT | Distributed Hash Table — a decentralized lookup system spread across many nodes with no central directory |
| DPI | Deep Packet Inspection — inspection of network traffic content (not just the destination) to detect and block specific protocols |
| E2EE | End-to-End Encryption — messages are encrypted by the sender and can only be decrypted by the intended recipient |
| Federation | Multiple independently-operated servers speaking the same protocol — like email but for chat |
| HF | High Frequency (3–30 MHz) — shortwave radio bands that bounce off the ionosphere for global communication |
| IPFS | InterPlanetary File System — a P2P file system that identifies files by their content hash rather than their location |
| IQ sampling | A technique for capturing radio signals with two measurements 90° apart in phase, enabling full signal reconstruction in software |
| Kademlia | The DHT algorithm used by IPFS and BitTorrent; uses XOR for distance measurement; scales to millions of nodes |
| LoRa | Long Range — a radio modulation technique enabling low-power, long-range digital communications (2–40 km) |
| LUKS | Linux Unified Key Setup — the standard for full-disk encryption on Linux |
| Namespaces | Linux kernel feature that gives each container its own isolated view of the system (its own network, processes, filesystem) |
| Onion service | A Tor-hidden server whose real IP address is never revealed; only accessible via a .onion address |
| OpSec | Operational Security — practices to minimize information leakage through human behav- |

# Closing: The Philosophy of Resilience

The technology in this book is not magic. It requires time, patience, and deliberate practice. Systems will break during setup. Configurations will be wrong the first time. Commands will fail. This is normal and expected. Every expert was once a beginner who was confused by the same things.

The most important principle is simple: **decentralize everything.**

- Do not depend on a single chat application that a corporation can shut down — run your own.
- Do not depend on a single server that can be seized — distribute across many.
- Do not depend on the internet — build radio capability.
- Do not depend on any single person's knowledge — document everything and train others.
- Do not depend on location-addressed files — use content-addressed storage so documents survive.
- Do not depend on a central authority for identity — use cryptographic keys that only you control.

Authoritarian systems derive their power from control of information and communication. Every system described in this book reduces that control by moving infrastructure to the edges — into the hands of ordinary people with ordinary hardware, running free software that anyone can read, audit, and verify.

The goal is not to be undetectable. That is nearly impossible for sustained operations. The goal is to be **resilient** — so that when one piece is taken down, the rest continues working, and rebuilding what was lost takes hours, not months.

Decentralized networks have no single throat to choke. That is the point.

This book covers exclusively open-source, publicly documented technologies used by journalists, researchers, emergency responders, and privacy advocates worldwide. Nothing in this book is secret. Everything here can be found in public documentation, academic papers, textbooks, and the documentation of the software projects described. Knowledge of how technology works is not a crime.

Stay safe. Stay connected. Keep building.

The Techno Anarchist
Resilient by Design
A Complete Guide to Decentralized, Private & Censorship-Resistant Technology From First Principles to Running Infrastructure

Based on the Distributed Networking & SDR Homelab Handbook
and the Decentralized Networking & Peer-to-Peer Architecture Supplement
Open-source technologies only • No prior knowledge required • Complete with theory and practical application