EZIO:
Predictable, Fast, Scalable BitTorrent-Based Bare Metal Provisioning

Date (Yu-Chiang) Huang <tjjh89017 [at] hotmail.com>
@Open Source Summit Korea, 2026-Aug-12

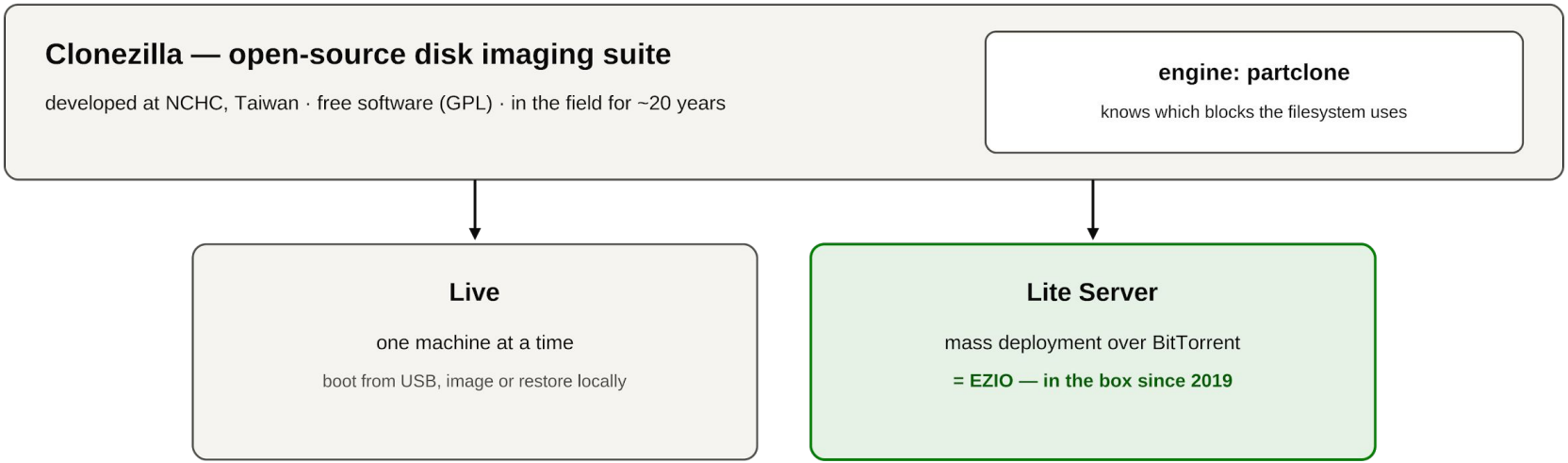
Who am I: Date (Yu-Chiang) Huang

- Cloud and Network Solution Architect with 7+ years of experience
- Creator of vRouter-Operator, STUNMESH-go and EZIO Project
- Expertise in major public cloud networking services and on-prem datacenter networking design
- Specialized in Cloud Network, OpenStack, Kubernetes, SD-WAN, and open-source
- Extensive speaking experience at international conferences

What Is EZIO?

- An open-source tool for **Mass Deployment**: clone one disk image onto many machines at once.
- The transport is **BitTorrent**; the target is the **raw disk**
  - no filesystem in between.
- Ships inside **Clonezilla** since 2019
  - you may have run it without knowing.
- The rest of this talk: why it exists, the one trick inside different from others, and the numbers.

Clonezilla in Thirty Seconds



same partclone engine underneath — Lite Server swaps the transport for a swarm

The Problem

Why deploying one image to many machines is still hard.

One Image, Many Machines

A cluster room. A lab. A data center.

- Tens or hundreds of machines, all needing the same OS image.
- Images are large:
  - tens to hundreds of gigabytes.

One of the oldest admin jobs there is.

What Good Deployment Needs

- [ ] **A finish time you can plan around**

  schedule the maintenance window — and hit it

- [ ] **A high success rate, even at scale**

  hundreds of nodes, no babysitting

- [ ] **Support for very large images**

  tens to hundreds of gigabytes

- [ ] **No expensive provisioning server**

  no beefy central box just for deployment

miss any one of these, and someone stays late at work

Why Now: The AI Cluster Wave

Hardware failures are constant

1 / ~3 hours

Llama 3: 419 unexpected interruptions
in 54 days on 16,384 GPUs

The Llama 3 Herd of Models — arXiv, 2024

Build-outs are measured in days

122 days

xAI Colossus: 0 -> 100,000 H100 GPUs,
every node imaged and provisioned

NVIDIA Newsroom, 2024

Provisioning is the hard part

30% → 98.5%

Crusoe: first-pass success rate,
before and after a provisioner rebuild

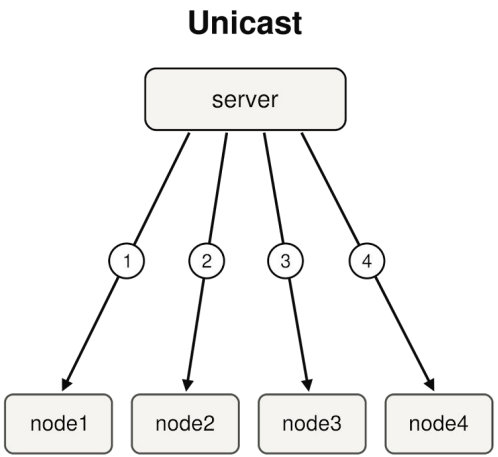
Crusoe engineering blog, 2026

every failed node must be wiped, re-imaged, and back in the job — fast re-imaging is real money

Re-Imaging Speed Is Now Real Money

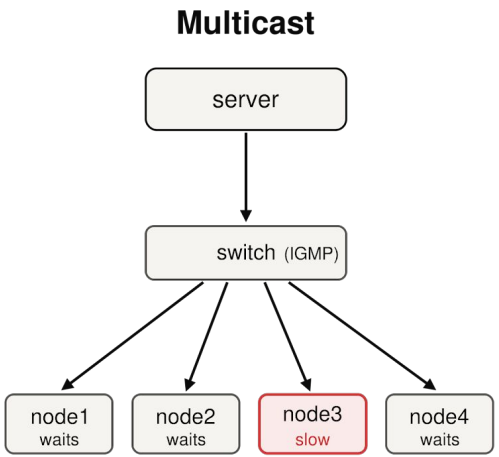
- Most AI starts running on bare metal: **no VM layer, direct GPU access**.
- A bare-metal node **cannot be "handed over"** like a VM or Container
- Re-assigning it means wiping and re-imaging the whole machine.
- Every re-imaging window is dead time on the most expensive hardware in the building.

Three Ways to Deploy



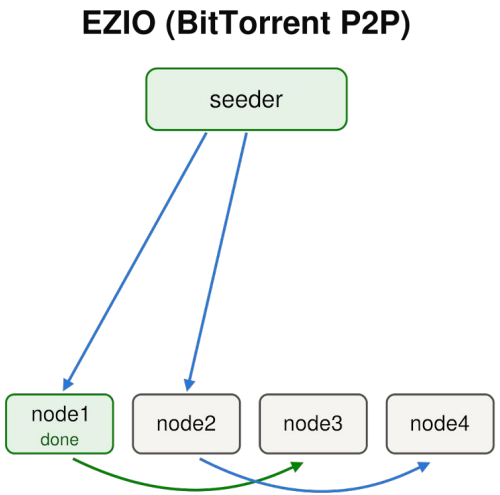
One full copy per node, in turn

T ≈ N × (image / bandwidth)



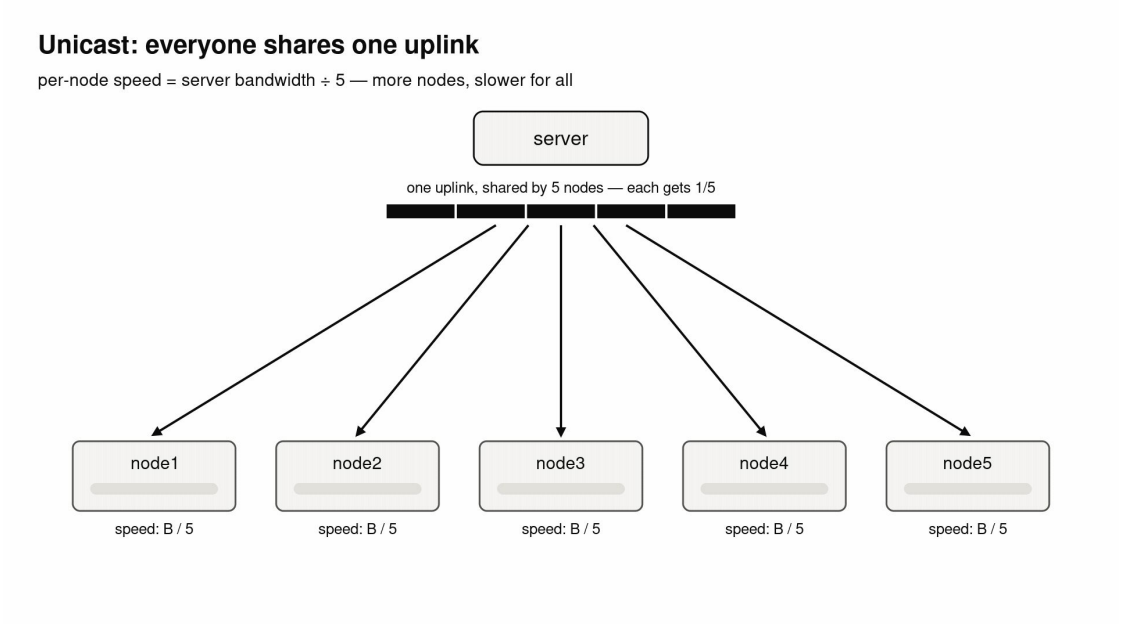
One stream, one group pace

The slowest node stalls everyone



Every node uploads too; finished nodes reseed the rest

Unicast: Everyone Shares One Uplink

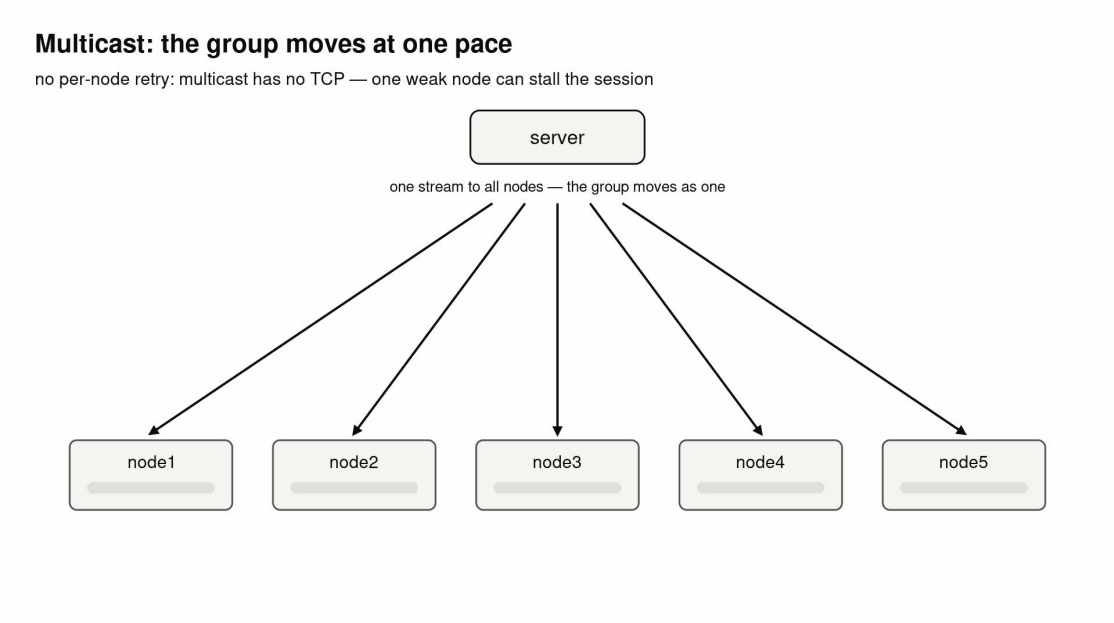


Unicast: The Math Is Against You

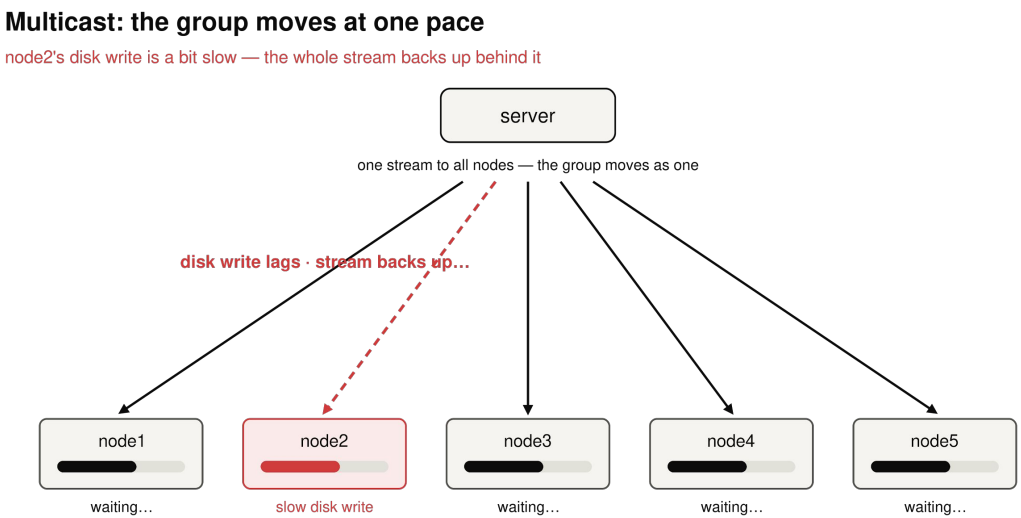
```
T = N x (image size / server bandwidth)
```

- The server sends the same bytes again for every node.
- 50 GB, 1 Gbps, 32 nodes -> about 3.5 hours.
- Every new machine makes it worse.

Multicast: The Group Moves at One Pace



Not Only Packet Loss



Multicast: Hard to Predict

Multicast sends the data only once, but the finish time swings with:

- the slowest node in the group,
- every packet loss event,
- every recovery pause.

A stuck node is a hard choice

- stall the whole group forever, or drop it on a timeout.
- Either the finish time slips, or nodes fail and need a second run.

Peer-to-Peer: The Right Idea, People Tried Before

BitTorrent fixes the unicast problem by design:

- every node that has data also uploads it,
- the server stops being the only source.

People tried this for OS deployment years ago.

It did not take over.

BitTorrent in Thirty Seconds

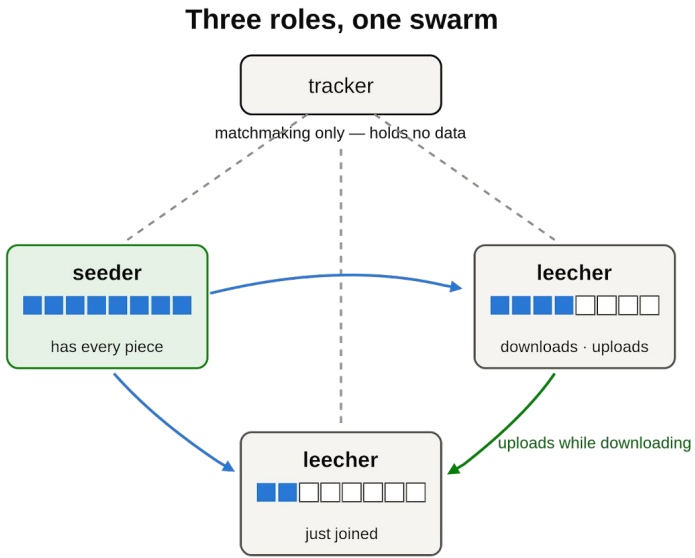
- A **torrent file**: metadata, piece size, hashes, file names.
- A **tracker**: helps peers find each other. Holds no data.
- **Peers**: a seeder has all the data; a leecher is still downloading - but uploads what it already has.

Every piece is checked against a SHA1 hash. Bad data never spreads.

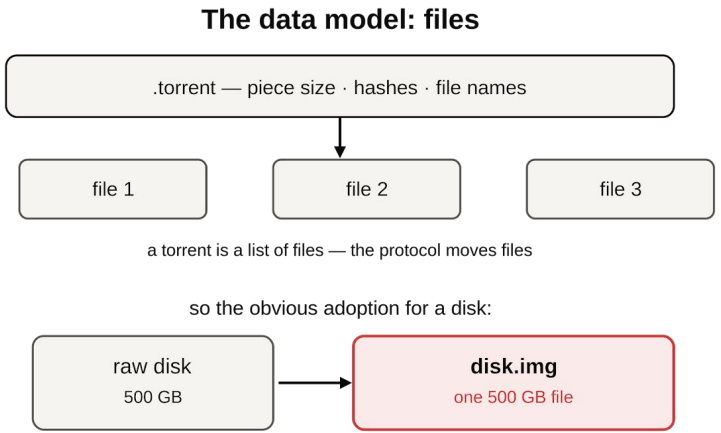
BitTorrent moves **files**. So the obvious way to deploy disks with it:

- make the disk a file: a dd image, a qcow2, a VM disk.

A Swarm in One Picture



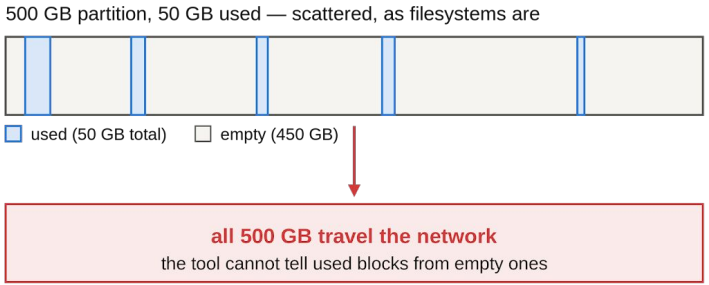
every piece is checked against its SHA1 hash — bad data never spreads



every byte of the disk becomes payload — the two walls follow from this move

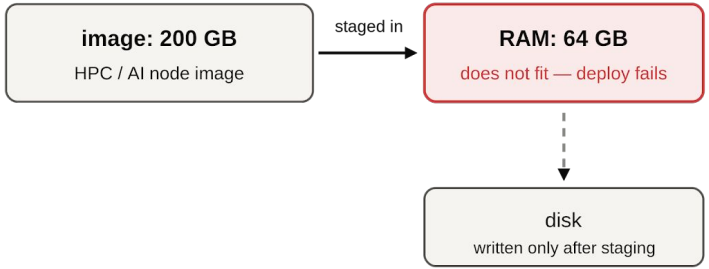
Two Walls Stopped Them

Wall 1: they send the whole partition



10x more traffic, 10x more time — for nothing

Wall 2: the image must fit in RAM



RAM sets a hard limit on image size — big images simply cannot deploy

BitTorrent was the right idea — these two walls are what EZIO had to remove

What We Actually Need

BitTorrent's swarm speed, plus three things:

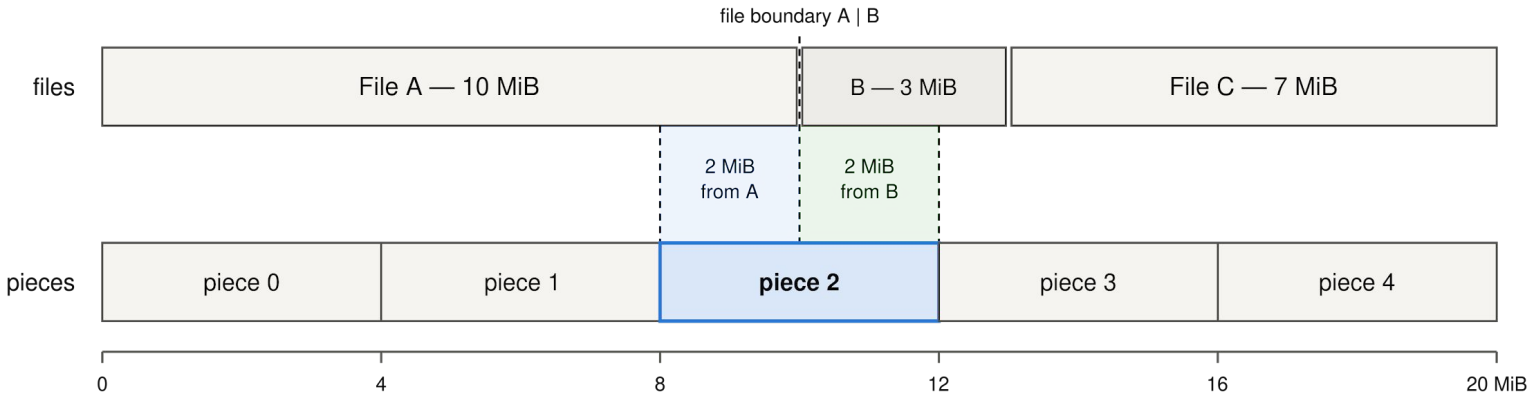
1. Transfer only **used blocks**, not empty space.
2. **No RAM limit** on image size.
3. **Write straight** to the raw disk.

Part 2 shows how EZIO gets all three.

The Idea

Everyone before us wrapped the disk in a file.
What if the torrent could *be* the disk?

Pieces Cut One Byte Stream - Not Files



piece 2 = last 2 MiB of File A + first 2 MiB of File B — one piece, two files

pieces cut the joined byte stream, so piece edges and file edges do not line up

What a Normal Client Must Do

For every piece, a regular BT client:

1. finds which file (or files) the piece touches,
2. splits the piece into per-file parts,
3. opens each file, seeks, and goes through the **filesystem**.

The data model, as we saw, is **file-based** - the protocol insists on files.

But for disk imaging, we want exact bytes at exact disk locations.

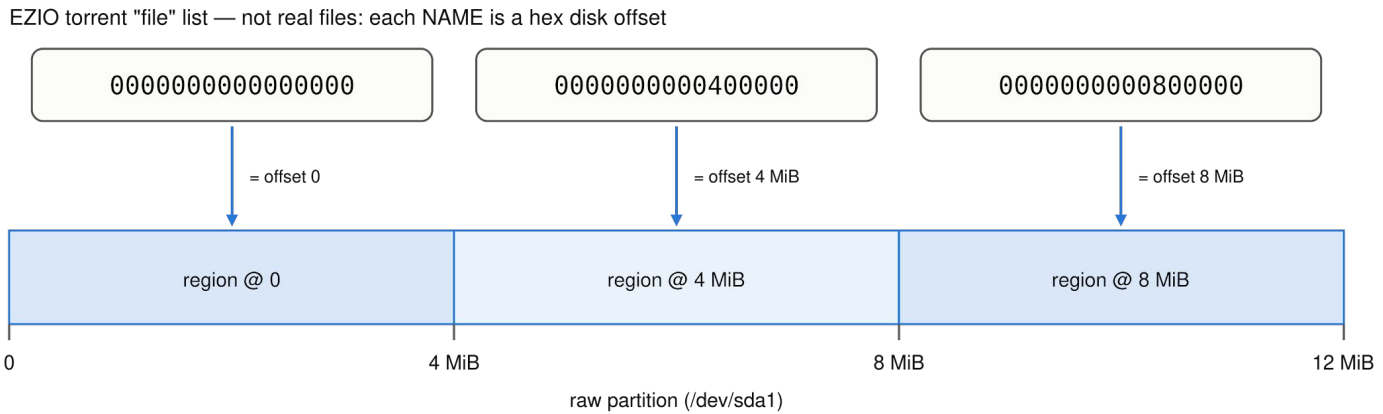
The Reframe

We did not fork the protocol - on purpose.
A stock BT client must still read our torrent.

The one field we could use freely: the **file name**.

What if each "file" NAME simply told us **where its data lives on the disk?**

The Trick: A "File" Name Is a Disk Offset




```
for each slice: disk_offset = hex(file_name) + offset_in_file
```

a piece can span several "files" (used-block runs); libtorrent splits it into slices — each is one hex parse + one add, no filesystem

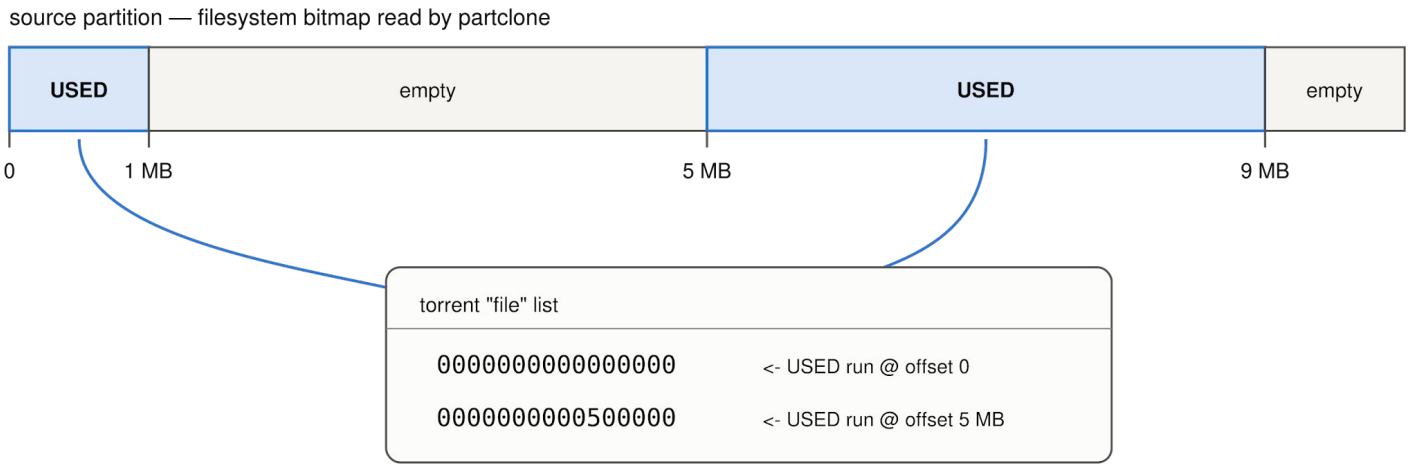
The Math, Per Slice

```
for each slice:
    disk_offset = hex(file_name) + offset_in_file
```

- One hex parse. One add. One pread()/pwrite().
- No filesystem. No lookup table. No real files.

The torrent is a map of the disk.

Which Blocks Go In? Ask the Filesystem



empty space never enters the torrent — only used blocks travel the network

partclone: The Expert We Reuse - and Extended

Clonezilla's imaging engine. It knows the on-disk layout of ext4, XFS, btrfs, NTFS, FAT, and a dozen more.

Our upstream contribution (flag -T): while partclone scans the used blocks, it **also computes the piece hashes.**

- One pass over the disk: image, block list, hashes.
- A small script wraps them into the hex-named torrent.

We reuse their filesystem knowledge. They ship our torrent support. Open source, both ways.

Wall 1 Falls: Send Only What Matters

Remember the 500 GB partition with 50 GB of data?

- Old BT tools sent the full 500 GB.
- EZIO transfers the 50 GB that partclone marks used.

Works for ext4, XFS, btrfs, NTFS - anything partclone understands. One tool, no per-FS code.

Wall 2 Falls: Straight to Raw Disk

```
network -> EZIO -> pwrite() -> /dev/sda1
```

- Each block goes from the network to the disk.
- No RAM sized to the image. No conversion step. No size limit.
- A 200 GB image deploys fine on a 64 GB RAM node.

Seeding is the same in reverse: pread() from raw disk, send.

No Staging, No Size Limit

before: download, stage, convert

the target disk is being wiped — the image can only stage in RAM

so RAM caps the image size; then the convert is a second full pass

EZIO: network straight to raw disk

each block: network -> disk — no staging, no convert, no size limit

a 200 GB image deploys fine on a 64 GB node

seeding is the same in reverse: pread() from disk -> send

wall 2 falls: RAM no longer limits image size

The Full Pipeline

used blocks only, straight to raw disk — no filesystem layer, no RAM staging, no size limit

Built on libtorrent-rasterbar

- libtorrent handles peers, pieces, and the protocol.
- It lets an application replace its **disk layer**.
- EZIO replaces it with a custom module: `raw_disk_io`.

We reuse twenty years of battle-tested BitTorrent code and only own the part that is special: the disk.

Re-write the Cache with Lock-Free model

- Same test, before -> after: **270 -> 766 MB/s (+184%).**
- Cache hit rate: **98-100%** - peers keep asking for the same hot pieces; the disk is asked only once.
- Best recorded run, image in RAM, same code path: **over 1 GB/s.**

Current Status

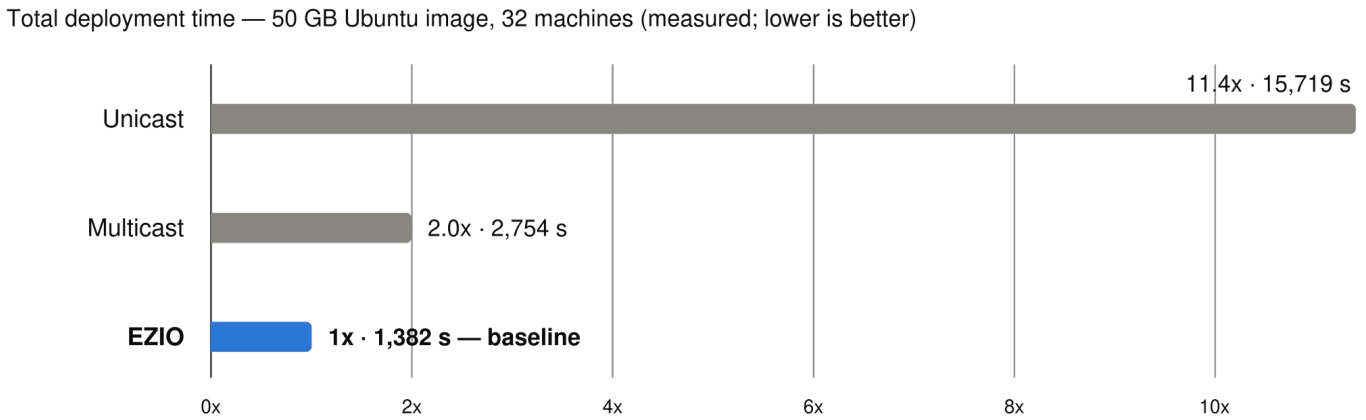
Shipped, measured, and how it behaves at scale.

In Production Since 2019

Clonezilla ships EZIO as **Lite Server Mode**.

- In the standard downloadable release. Not a fork, not a demo.
- Thousands of users worldwide, labs to datacenters.
- The exact pipeline you just saw, end to end.
- Over thousands time of HPC deployment in our center

Measured: 50 GB Image, 32 Machines



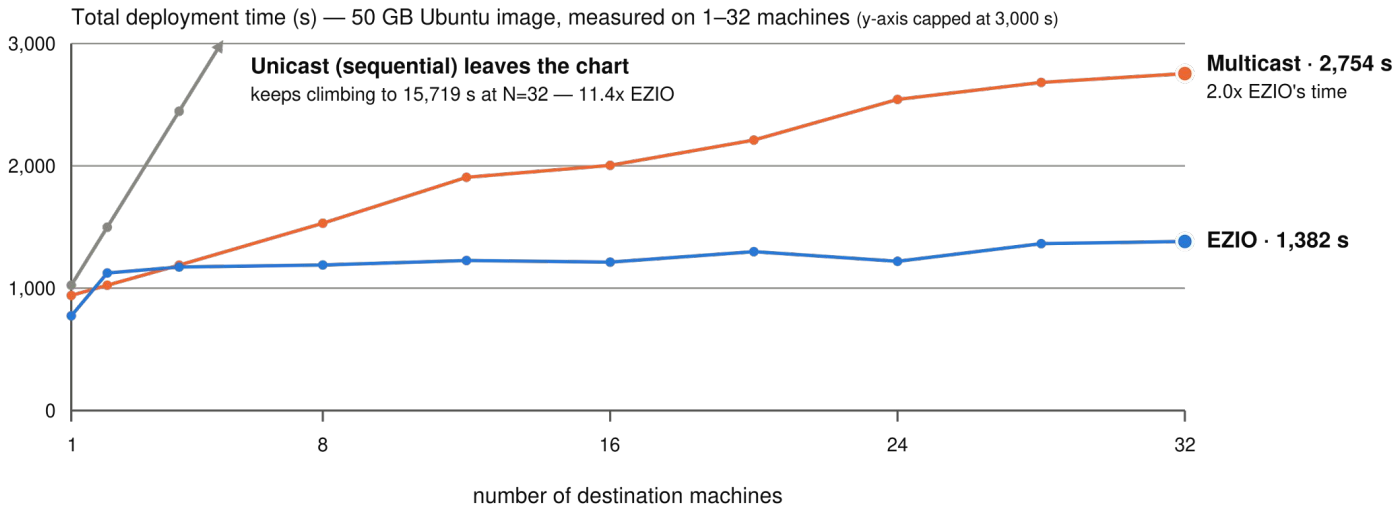
Data: IEEE Access, 2021 (Table 4; 32-machine testbed, HDD + GbE). EZIO deploys in 1,382 s — 11.4x faster than sequential unicast, 2x faster than multicast.

And on Modern Hardware?

- NCHC production cluster: NVMe SSD, 10GbE, 32 nodes.
- Measured: **~500 MB/s** provisioning across all nodes.
- A 50 GB image: under 2 minutes.
- After the cache work: **~840 MB/s** on raw NVMe in lab tests.

The ratios are structural. The speed scales with the hardware.

The Curve That Matters



Data: A BitTorrent Mechanism-Based Solution for Massive System Deployment — IEEE Access, 2021 (Table 4; BDMfRD = EZIO)

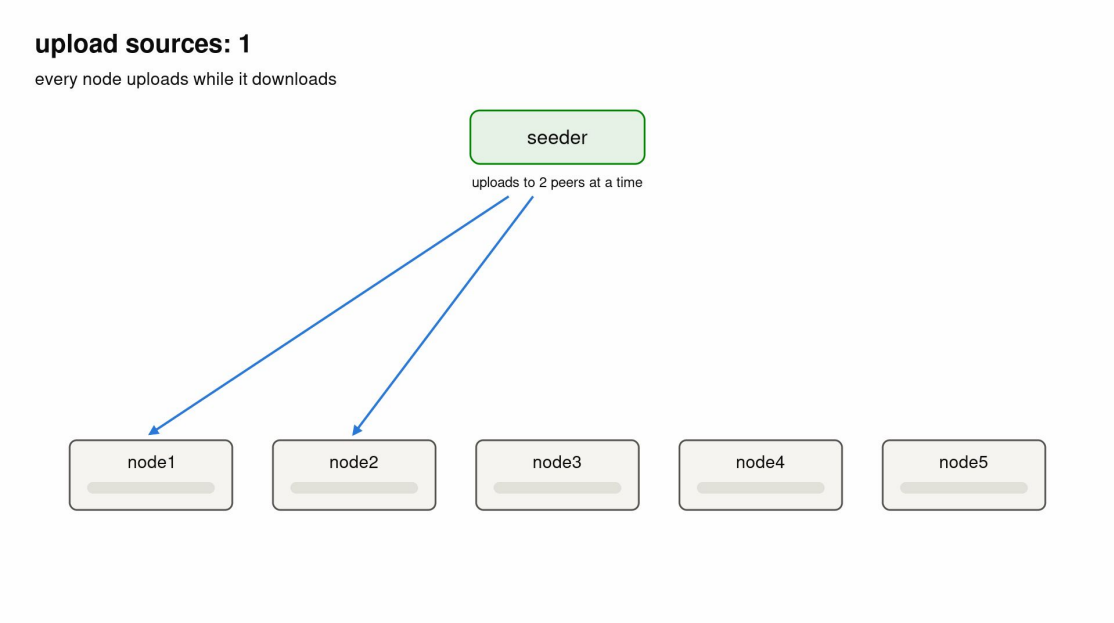
What "Predictable" Means

Provisioning time became a property of the **image**, not of the cluster.

- time ~ image size / bandwidth
- 32 nodes or 320: plan the same window.

That is the word "predictable" in the talk title.

The Reseed Model



A Broken Node Blocks Nobody

- A node crashes mid-download? It just drops out.
- Everyone else keeps going - no single point of failure.
- After repair it reannounces to the tracker and re-images itself from nodes that already finished.

Remember: AI clusters lose a node every ~3 hours.
This failure model is exactly the one they live with.

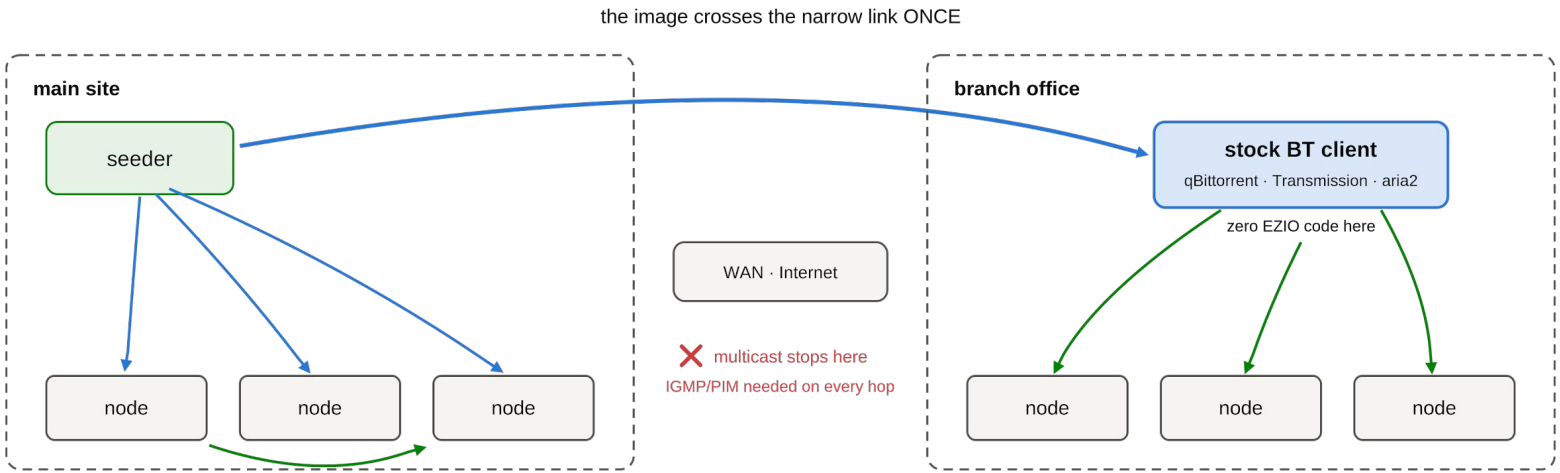
Free Relays: Any Stock BT Client

The torrent is 100% standard - so qBittorrent, Transmission, or aria2 can join the swarm as a **relay**.

- One stock client at a remote site pulls the image once over the narrow uplink; local nodes swarm from it.
- Works across switches, across regions - even over the Internet.

Multicast ends where your control of the network ends.
Plain BitTorrent over TCP does not.

One Swarm Across Sites



one swarm, two sites: local nodes fill at LAN speed from a relay that is just a stock client

plain BitTorrent over TCP crosses any network you can route — multicast ends where your switches end

Adopting EZIO: Four Steps

1. Boot a small Linux on each node (PXE - Clonezilla does this).
2. Prepare once: partclone + build the torrent.
3. Run a tracker and one initial seeder.
4. Drive every node over gRPC:
    a. `AddTorrent, GetTorrentStatus, Pause/Resume, Shutdown.`

Any language can call gRPC. You are not locked into C++

(We have a Python script for those operations)

What You Do NOT Need

- No RAM sized to the image.
- No temporary storage on nodes.
- No powerful central server.
- No per-filesystem integration code.
- No new transfer protocol to invent.

The Scorecard, Settled

A finish time you can plan around

time ≈ image size / bandwidth — flat from 1 to 32+ nodes

A high success rate, even at scale

a broken node blocks nobody; it re-images itself after repair

Support for very large images

network -> raw disk, no RAM staging — 200 GB on a 64 GB node

No expensive provisioning server

every node uploads — the swarm itself is the server

all four ticked — measured on 32 machines, shipping in Clonezilla since 2019

Conclusion

The Evidence Base

- Applied Sciences (2019) - peer-reviewed.
- IEEE Access (2021) - peer-reviewed; the 32-machine numbers.
- Years of Clonezilla production use on top.

Try it, run your own numbers, tell us what you see.

https://github.com/tjjh89017/ezio

EZIO in Five Lines

- Torrent "file" names are disk offsets - placing data is pure math.
- Only used blocks travel. Empty space never enters the swarm.
- Provisioning time depends on image size, not node count.
- A broken node never blocks the rest - it rejoins on its own.
- In Clonezilla since 2019; 11.4x vs unicast on 32 real machines.

Who Should Pick This Up

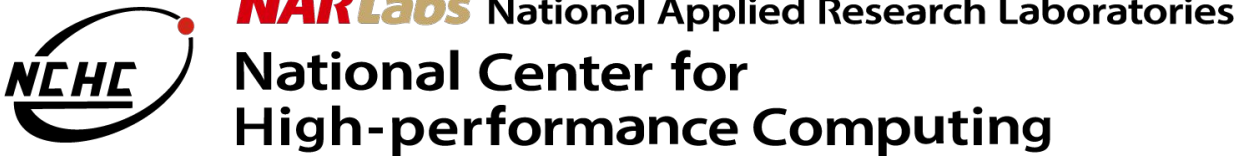
- Provisioning platforms: OpenStack Ironic, MAAS, Metal3.
- GPU clouds that must turn broken nodes around in minutes.
- Edge and factory fleets that re-image far from home.

EZIO is a free building block.

We cannot integrate it everywhere alone.

Special Thanks

- National Center for High-performance Computing, NIAR, Taiwan
  - We deployed the clusters with Clonezilla Bittorrent mode for several years



Thank You - Questions?

EZIO: Predictable, Fast, Scalable BitTorrent-Based Bare Metal Provisioning

Date Huang

Find me after the session, or contact me via Linkedin
I would love to hear about your cluster.

Also, please help me to start my project.

https://github.com/tjjh89017/ezio

Linkedin