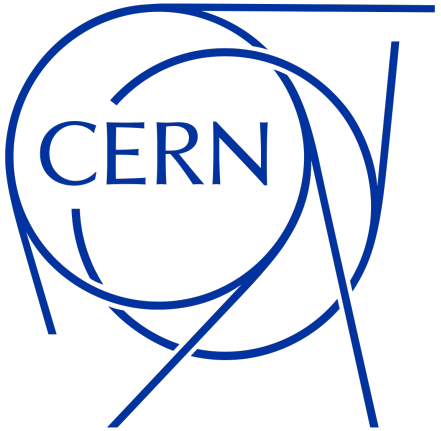


Mirage – File-Level Simulation on an XRootD Server

Diogo Mattioli

September 2026

Introduction

It lets you fake exabytes of storage with zero disk space

XrdOssMirage

Is a server-side plugin that simulates files on an XRootD server. It requires no storage: any uploaded file content is discarded, while only the file size is retained.

Motivation

- A tool to help with testing, stressing and benchmarks
- Dynamic “fake” filesystem
- No additional layers (disk, drivers, etc.)
- No storage, with deterministic reproducible content
- Maximum possible throughput

Features

Since 6.1

Oss lib

- Create, delete and rename files
- Read (download) and write (upload)
- Truncate size

Xattr lib

- Custom return code when opening, reading or writing
  - optionally at specific offset
- Content data patterns *(single char or string - performance impact)*

Features

On next release 6.3

- Checksums (deterministic or custom)
- Option to work under a specific path only (stacking)
- Custom return code when closing
- Data pattern (string) performance enhancing

Features

Not supported (yet?)

Oss lib

- Directory support
- File properties (date, permissions, ownership)

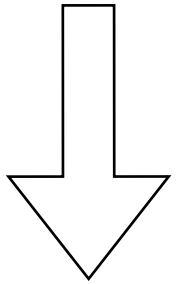
How to enable

Object Storage System (mandatory)

```
ofs.osslib libXrdOssMirage.so
```

Extended attributes (optional)

```
ofs.xattrlib libXrdOssMirage.so
```



One-liner

```
ofs.osslib +xattr libXrdOssMirage.so
```

```
set name = mirage
set port = 6543
set run  = $PWD

all.export /
all.adminpath $run/$name
all.pidpath   $run/$name
oss.localroot $run/$name

xrootd.trace all

all.sitename $name

xrd.port $port

ofs.osslib +xattr libXrdOssMirage.so

all.trace    all
ofs.trace    all
oss.trace    all
```

How to enable (with stacking from 6.3 onwards)

Object Storage System (mandatory)

```
ofs.osslib ++ libXrdOssMirage.so /path/to/mirage
```

Or

```
ofs.osslib ++ libXrdOssMirage.so
```

Extended attributes (optional)

```
ofs.xattrlib ++ libXrdOssMirage.so
```

Or

```
ofs.xattrlib libXrdOssMirage.so
```

One-liner (not possible)

~~`ofs.osslib +xattr libXrdOssMirage.so`~~

```
set name = mirage
set port = 6543
set run  = $PWD

all.export /
all.adminpath $run/$name
all.pidpath   $run/$name
oss.localroot $run/$name

xrootd.trace all

all.sitename $name

xrd.port $port

ofs.osslib ++ libXrdOssMirage.so /path
ofs.xattrlib ++ libXrdOssMirage.so

all.trace    all
ofs.trace    all
oss.trace    all
```

Could it help with DC27? 💡

Usage

Creating an empty file and truncating its size

Creating the file from standard input

```
head -c 0 /dev/zero | xrdcp - root://localhost//remotefile
```

Truncating file size to 1GB

```
xrdfs root://localhost/ truncate /remotefile 1073741824
```

```
$ head -c 0 /dev/zero | xrdcp -f - root://localhost//remotefile
[0B/0B][100%][==================================================][0B/s]
$ xrdfs root://localhost/ truncate /remotefile 1073741824
$ xrdfs root://localhost/ stat /remotefile

Path:   /remotefile
Id:     25769803871
Size:   1073741824
MTime:  1970-01-01 00:00:00
Flags:  4 (Other)
```

Return when reading a file at a specific position

Set a custom return code for the reading of a file

```
xrdfs root://localhost/ xattr /remotefile set read.return_code=CODE
```

Set at what position it should be returned (optional)

```
xrdfs root://localhost/ xattr /remotefile set read.return_position=POSITION
```

```
$ xrdfs root://localhost/ xattr /remotefile set read.return_code=12
$ xrdfs root://localhost/ xattr /remotefile set read.return_position=1000000000
$ xrdcp root://localhost//remotefile localfile
[568MB/1024MB][ 55%][==========================>                        ][568MB/s]
Run: [ERROR] Server responded with an error: [3008] Unable to read /remotefile; cannot
allocate memory (source)
```

Opening, reading, writing or closing with return code

- **Resumable file transfers**
  - Simulate a connection dropping after 500 MB of a 2 GB upload
  - Verify that the client resumes from the correct byte offset rather than restarting
- **Backup and restore testing**
  - Create partially corrupted backup archives
  - Test whether restore software detects corruption and reports which files/blocks are affected
- **Distributed storage**
  - Test systems such as object stores, replication services, or distributed filesystems against incomplete writes
  - Useful for validating consistency and recovery mechanisms
- **Crash-recovery testing**
  - Check whether the application leaves behind a recoverable temporary file, rolls back, or cleans up correctly

Custom generated pattern for file contents

Set a custom pattern to be generated when reading a file

```
xrdfs root://localhost/ xattr /remotefile set pattern=PATTERN
```

Throughput decrease
- *20% single char*
- *94% string*

Garbage: 4.278GB/s
Single char: 3.45GB/s
String: 244.6MB/s

40 logical CPUs 2.8MHz
256GB RAM

```
$ head -c 100 /dev/zero | xrdcp -f - root://localhost//remotefile
[100B/0B][100%][==================================================][100B/s]
$ xrdfs root://localhost/ xattr /remotefile set pattern=abcde123
$ xrdfs root://localhost/ cat /remotefile
abcde123abcde123abcde123abcde123abcde123abcde123abcde123abcde123abcde123abcde123abcde123ab
cde123abcd
$ xrdfs root://localhost/ xattr /remotefile set pattern=1
$ xrdfs root://localhost/ cat /remotefile
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111
```

Calculating checksum (from 6.3 onwards)

Retrieving checksum from server

```
xrdcp --cksum adler32:source root://localhost//remotefile localfile
```

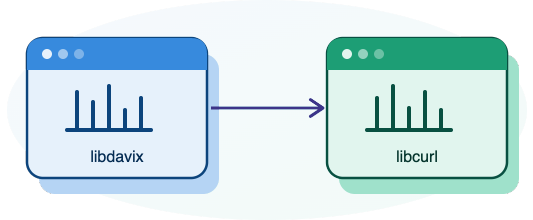
Set a custom checksum to be retrieved

```
xrdfs root://localhost/ xattr /remotefile set checksum.adler32=a1b2c3d4
```

```
$ xrdfs root://localhost/ xattr /remotefile set pattern=abcde
$ xrdcp --cksum adler32:source root://localhost//remotefile localfile
[100B/100B][100%][==================================================][100B/s]
adler32: a0f326ad root://localhost/file 100

$ xrdfs root://localhost/ xattr /remotefile set checksum.adler32=a1b2c3d4
$ xrdcp --cksum adler32:source root://localhost//remotefile localfile
[100B/100B][100%][==================================================][100B/s]
adler32: a1b2c3d4 root://localhost/file 100
```

Current use cases

- **Benchmarks for XrdClHttp command line plug-in**
  Comparison between libcurl and libdavix
  Comparison among xrdcp 5.9, xrdcp 6, davix and curl

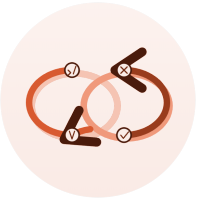


- **Development of XrdClHttp TPC support**
  Debug, simulate and test the copy of big files between servers locally

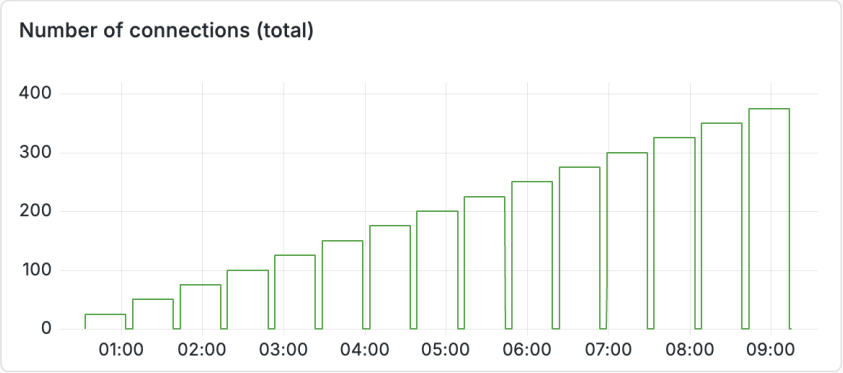
- **Everpure FlashBlade//S500 network, CPU and memory throughput**

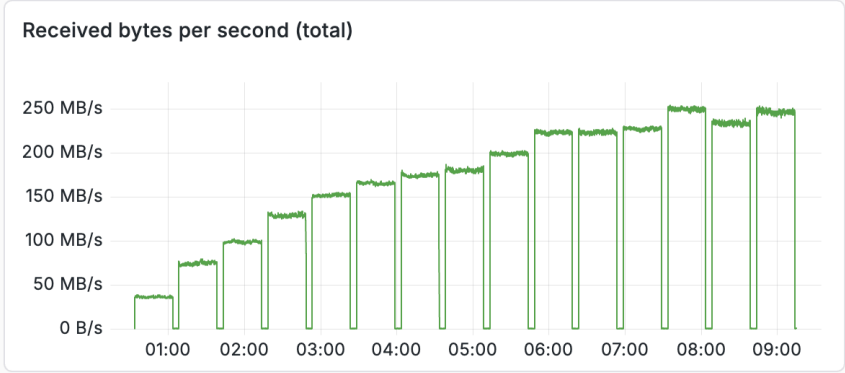
- **400-gigabit network interface testing**
  Understand limitations, bottlenecks and possible improvements on XRootD

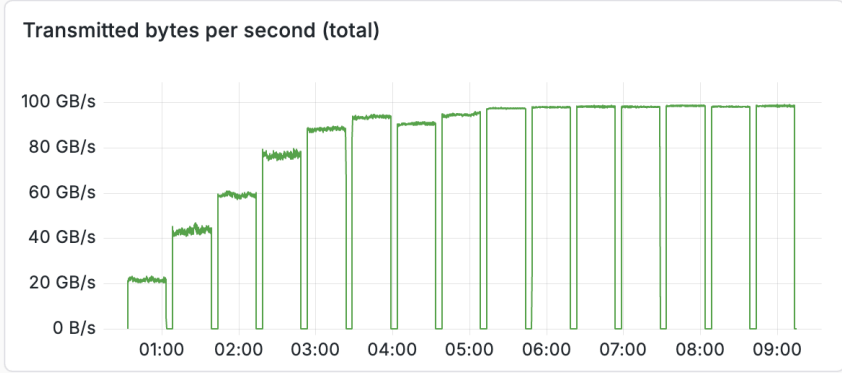
- **CI/CD testing with big files**
  On a container 10GB ~1s, 25GB ~3s and 100GB ~10s *(when downloading)*

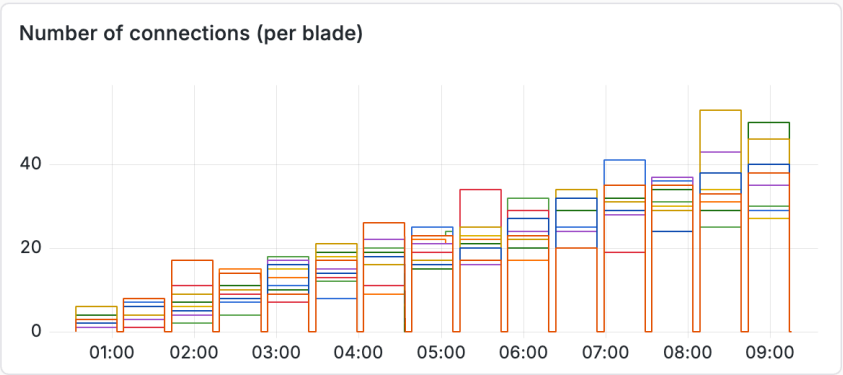


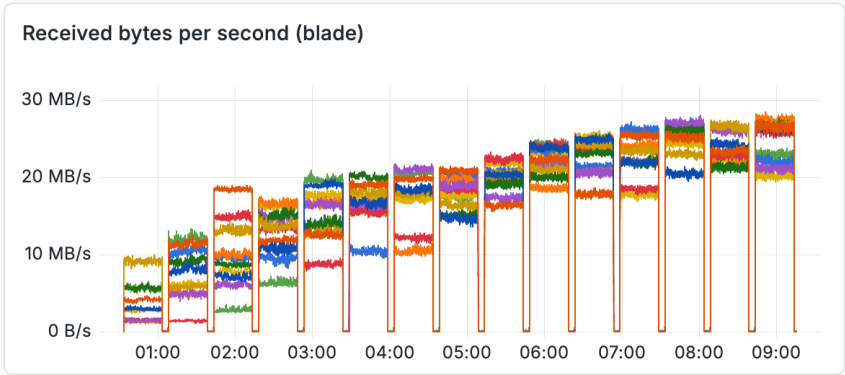
Current use cases (Everpure Metrics - roots)

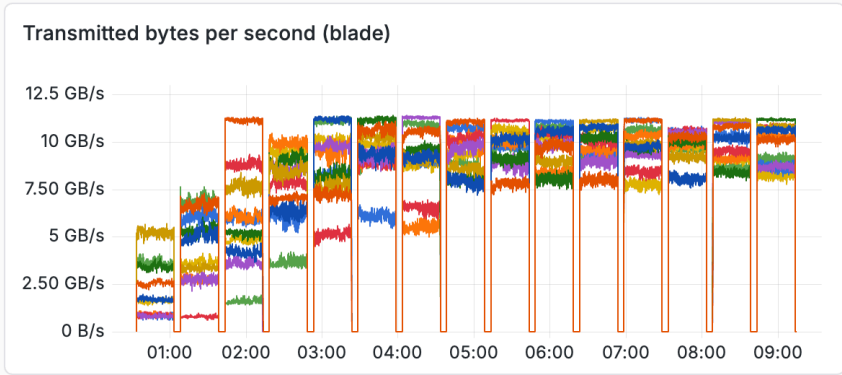


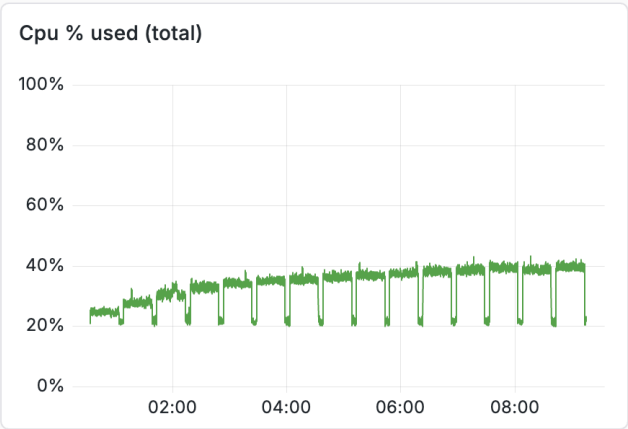


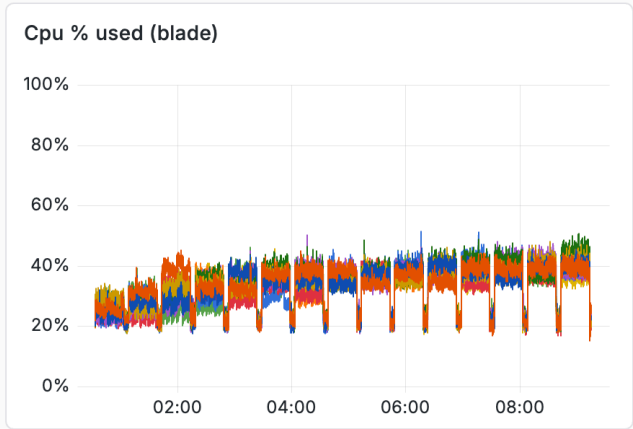


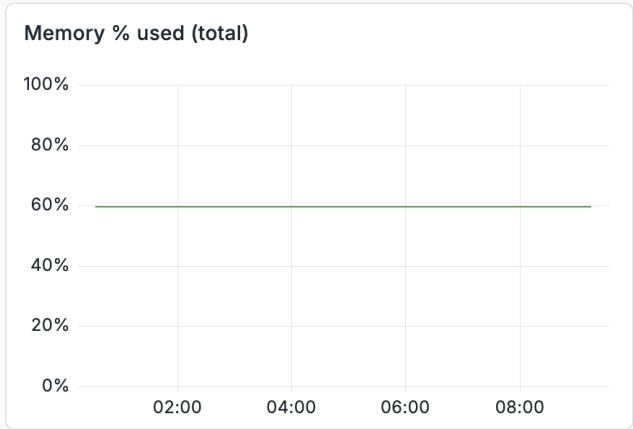


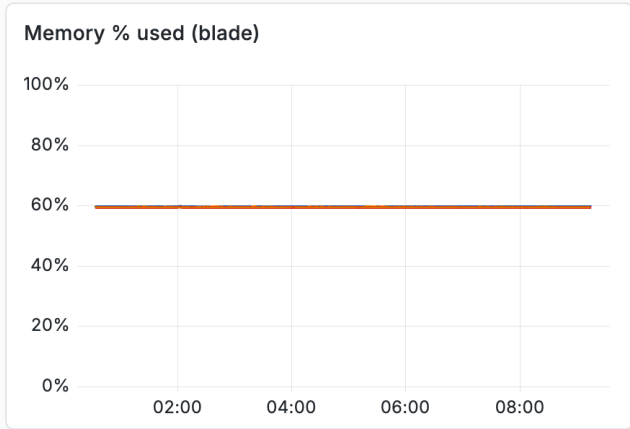




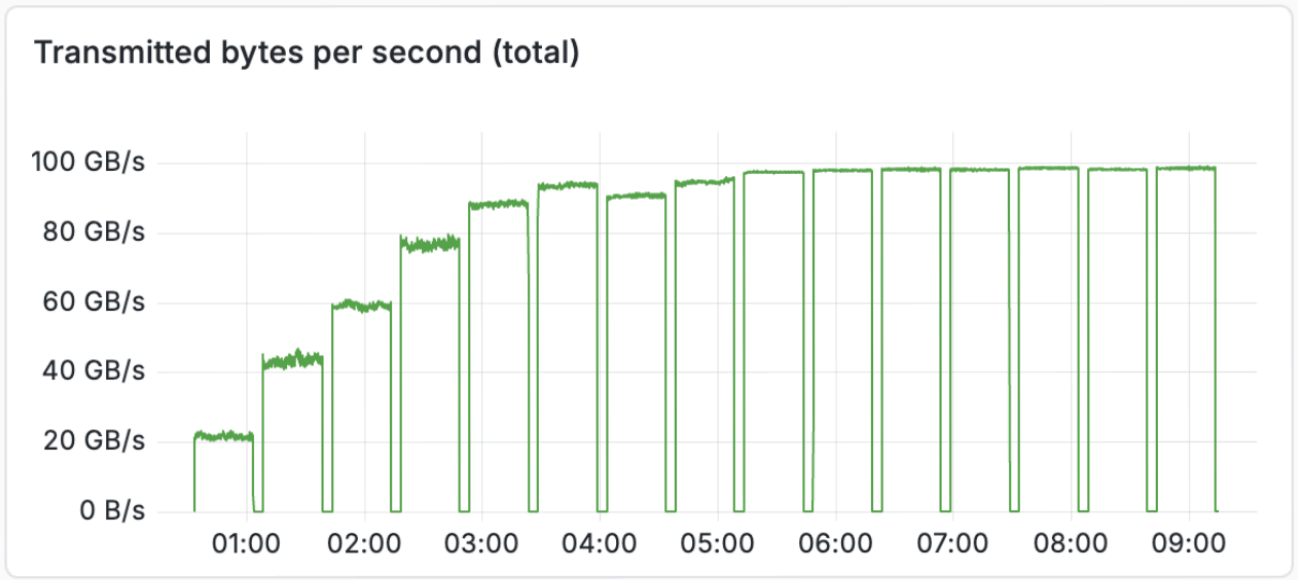


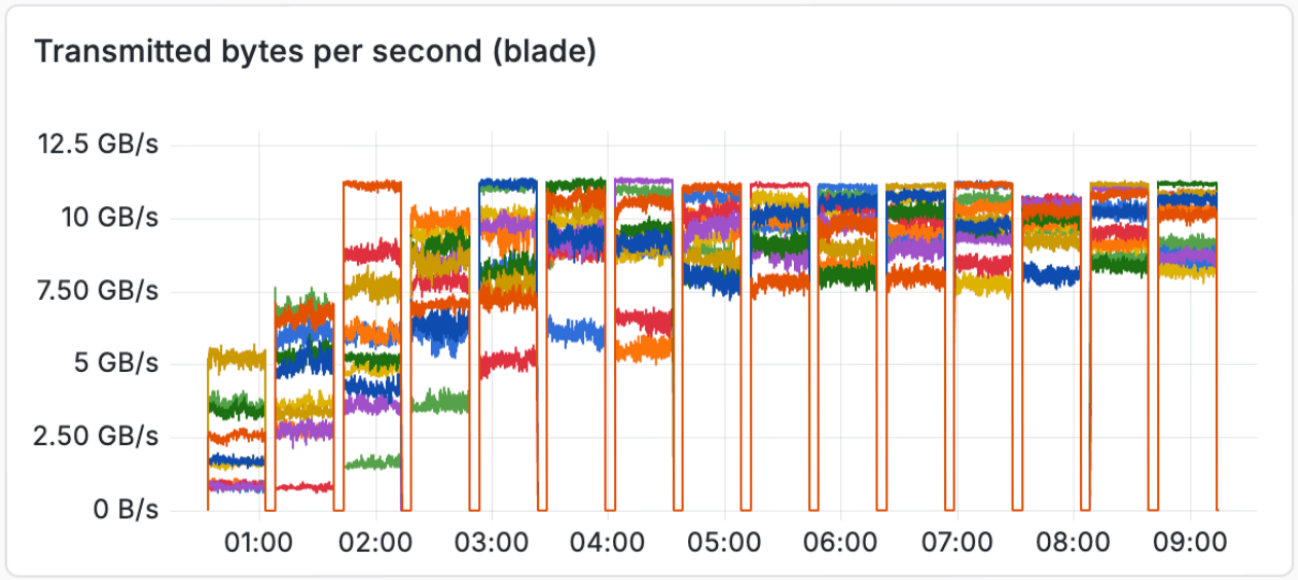


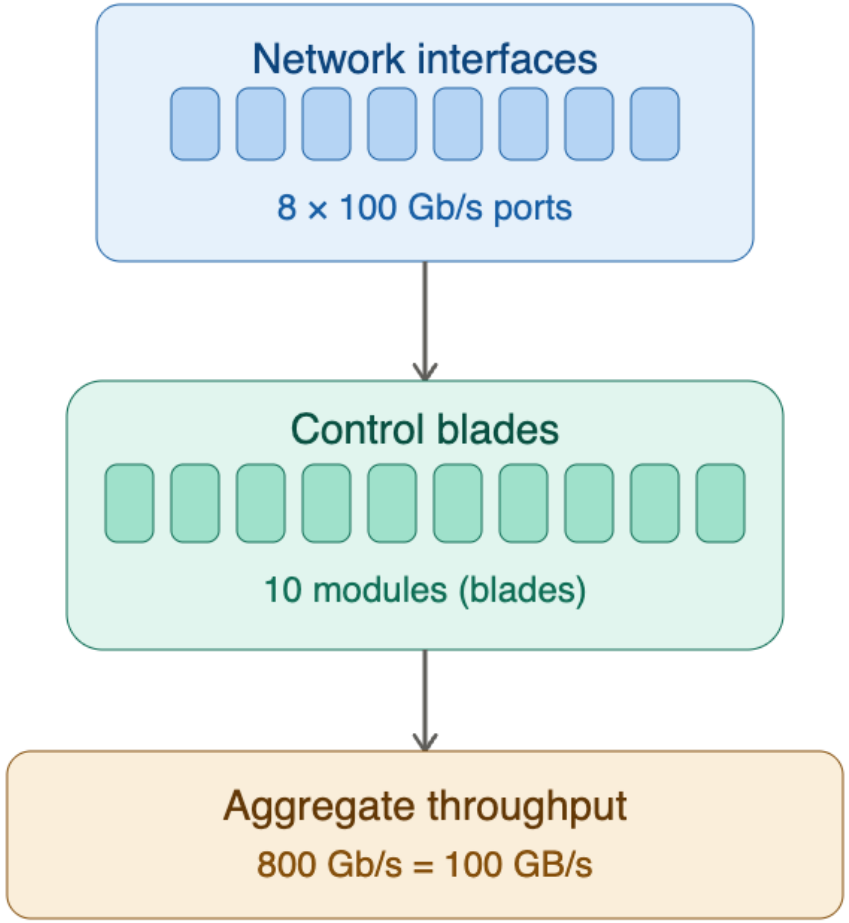




Current use cases (Everpure Metrics - roots)

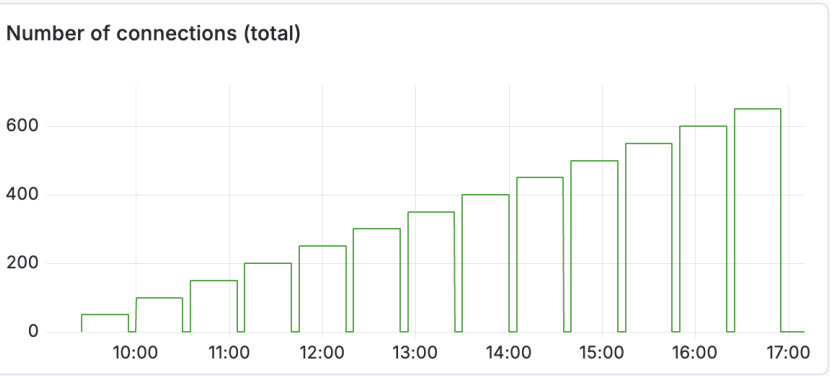


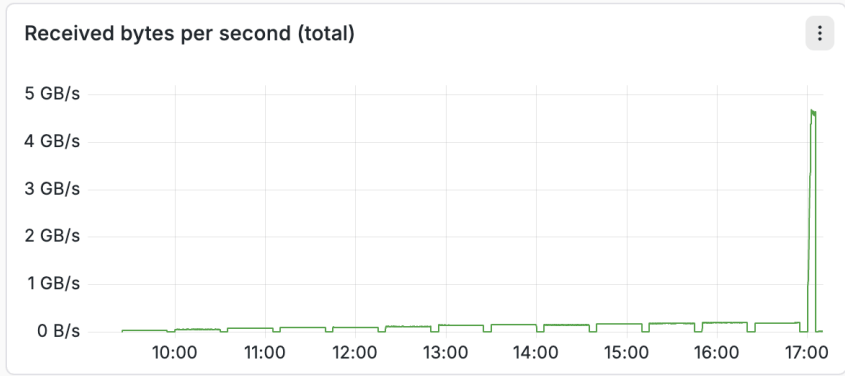


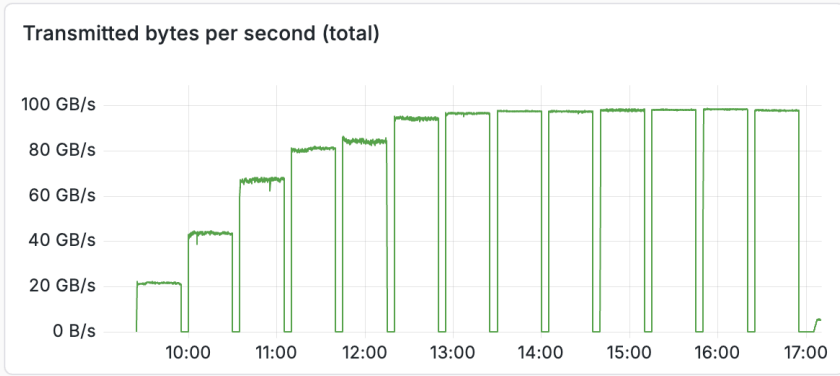


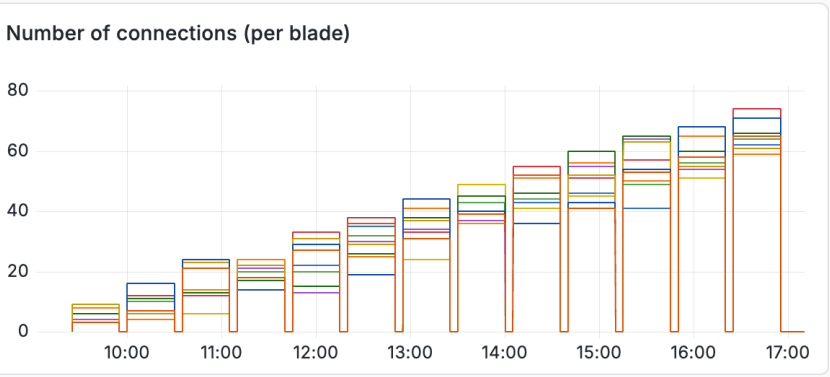
100GB/s x 5 hours = 1.8 petabytes
1.8PB / ~380 conn = 4.7 terabytes
4.7TB x 10 blades = 47 terabytes

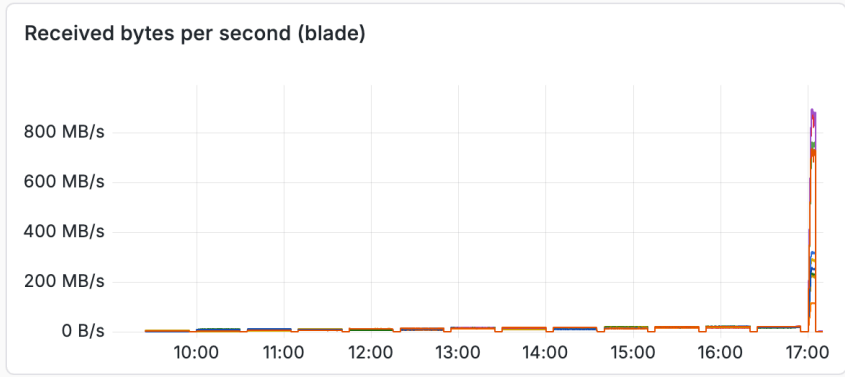
Current use cases (Everpure Metrics - https)

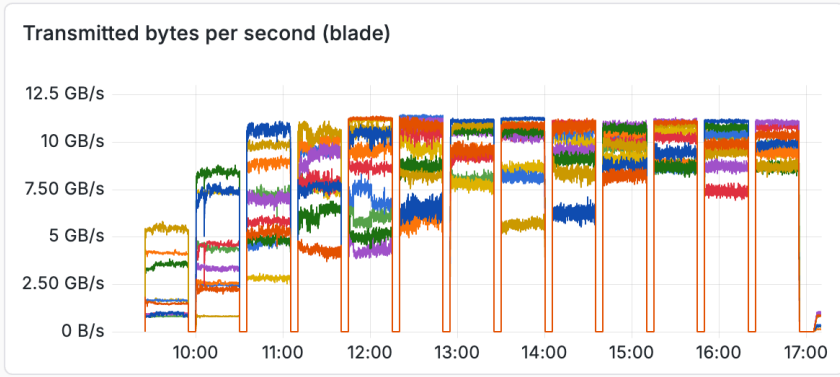


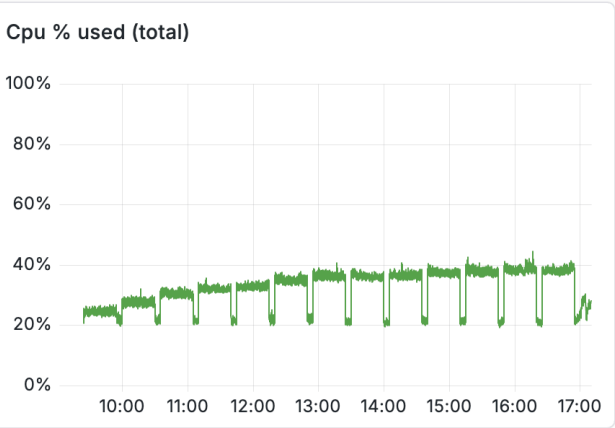


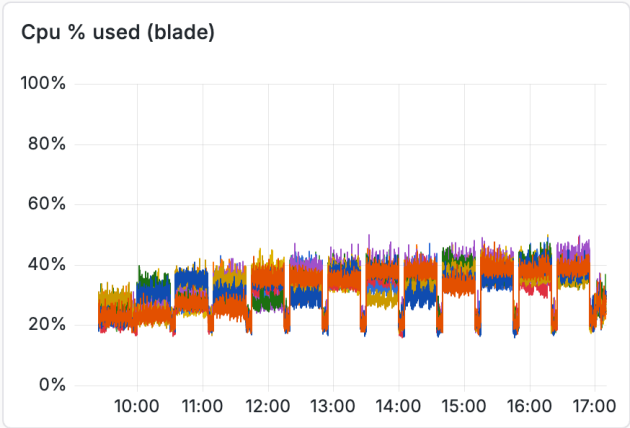


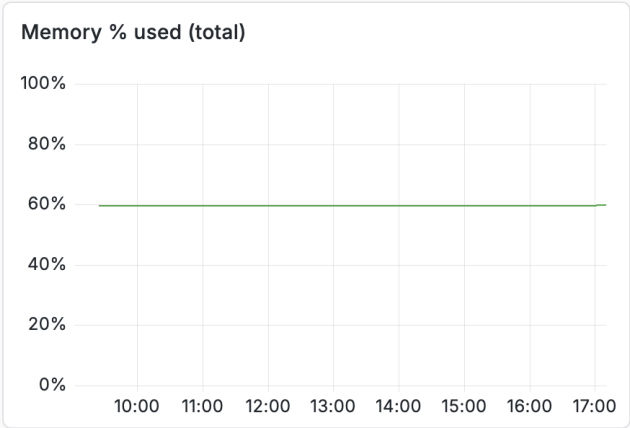


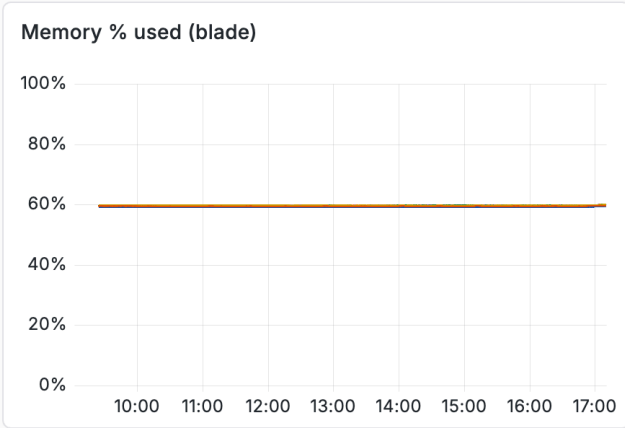






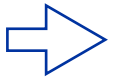






Current use cases (CI/CD testing with big files)

```
set run  = $BATS_TEST_TMPDIR
set name = $NAME
set port = $PORT

all.export /
all.adminpath $run/$name
all.pidpath   $run/$name
oss.localroot $run/$name

xrootd.trace all

tpc.allow local

xrd.protocol XrdHttp:$port libXrdHttp.so
http.exthandler xrdtpc +notls libXrdHttpTPC.so

ofs.osslib +xattr libXrdOssMirage.so

all.sitename $name

xrd.port $port
```



```
setup() {
    xrdfs root://localhost:7094/ truncate /file_src 26843545600
}

@test "pull copy transfers a 25GB file" {
    run -0 xrdcp -T only http://localhost:7094//file_src http://localhost:7095//file_dst
}

@test "pull copy with 10 streams transfers a 25GB file" {
   run -0 xrdcp -S 10 -T only http://localhost:7094//file_src http://localhost:7095//file_dst
}

@test "push copy transfers a 25GB file" {
   run -0 xrdcp -T push only http://localhost:7094//file_src http://localhost:7095//file_dst
}

@test "push copy with 10 streams transfers a 25GB file" {
   run -0 xrdcp -S 10 -T push only http://localhost:7094//file_src http://localhost:7095//file_dst
}
```

```
$ bats -T tests/end-to-end/tpc/http/bigfiles.bats -f '25GB'
bigfiles.bats
 ✓ pull copy transfers a 25GB file [5751]
 ✓ pull copy with 10 streams transfers a 25GB file [12736]
 ✓ push copy transfers a 25GB file [4354]
 ✓ push copy with 10 streams transfers a 25GB file [4072]

4 tests, 0 failures in 27 seconds
```

Future improvements

- **Bandwidth simulation**
  To simulate HDD and SSD speed
- **Random or deterministic delays**
  To introduce out-of-order reads and writes
- **Static configuration**
  To allow configuration of predefined fake files via XRootD configuration
- **Anyone interested in a specific feature?** 🤔

Thank you!
Questions?

home.cern