
CERN

Michal Simon

Powered by XRootD

Outline

- Short introduction to XRootD5
- Secure root/xroot, why and how?
- Is there anything else?
- Tailor made for EOS
- Plans & Summary

XRootD5 in few words

- **Major release**, with the most important new feature being **encryption**
- Protocol and API level backwards compatibility, it is not ABI compatible – plugins will require recompilation
- Released in July, followed by 3 bugfix and 1 feature release
  - Release in OSG and EPEL repos, and pushed into Debian distribution

Secure root/xroot protocol

- roots/xroots is the old good **root/xroot protocol plus TLS**
- Based on OpenSSL
  - Version 1.0.0 and above
- Encrypted and unencrypted version of root/xroot protocol run on the same port (by default 1094)

Why do we need encryption?

- Allows for authorization token handling (e.g. SciToken)
  - Prerequisite for **replacing proxy delegation with access tokens** in WLCG
- Encrypt confidential data
  - Encryption ‘in transit’ especially for **CERNBox**
- Encrypt possibly destructive metadata operations (could replace in the future request signing)
- Improves data integrity and allows for further evolution of Third-Party-Copy

What triggers encryption?

- On the client side the **roots/xroots** protocol;
  - --**notlsok** options allows to proceed without encryption if the server is too old to support it
  - --**tlsmetalink** option allows to apply encryption to all URLs in a metalink file

- On the server side the **xrootd.tls** configuration directive, with few compatibility options:
  - by default it is **off**
  - enforce encryption only for clients that support it (**capable**)
  - do encryption only at client discretion (**none**)

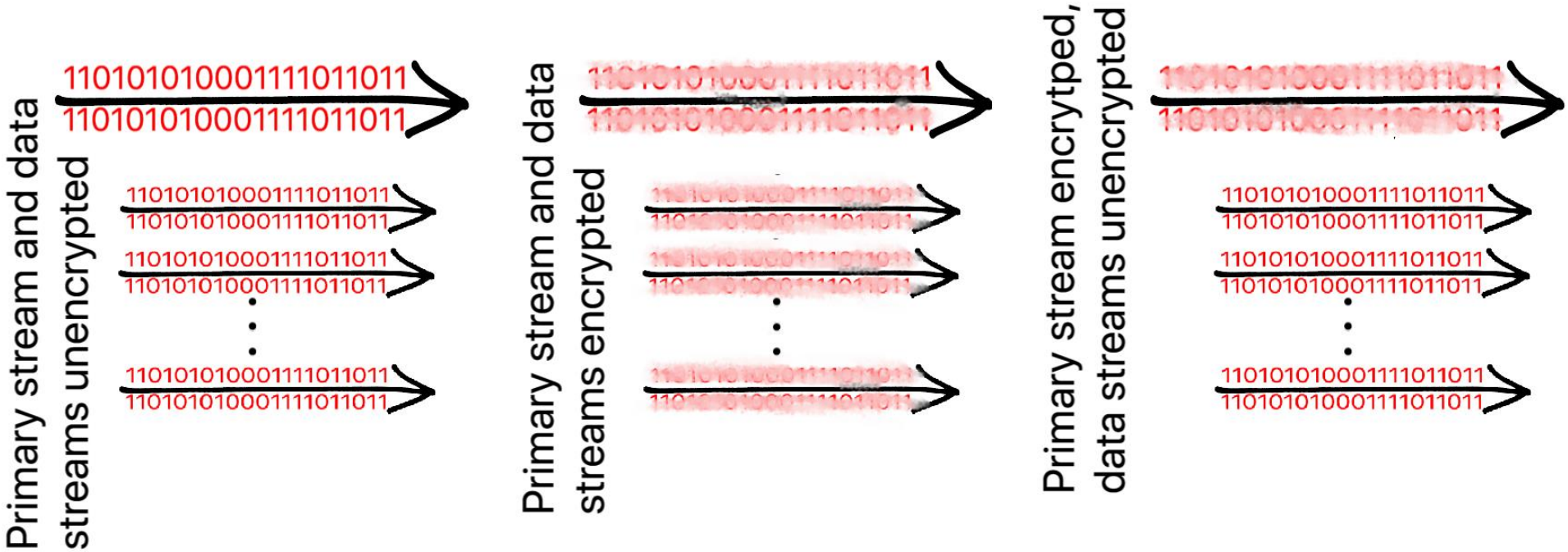
How flexible is it?

- It is pretty flexible ;-) – not everything needs encryption and (at the beginning) not everyone will support encryption

- One can configure the server to encrypt:
  - only the **third-party-copy** orchestration
  - **control channel** after login (handy for GSI auth)
  - control channel before login
  - **data streams**
  - everything

- On the client side:
  - **--tlsnodata** allows to apply roots/xroots only to the control stream

How flexible is it?



What lies beneath the flexibility?

- Handshake negotiation
  - All connections are **initially non-encrypted**
  - The connection is being **upgraded to TLS on client or server request**

- If only control channel should be encrypted we open a second (or multiple) physical connection for the raw data

- **Encrypted and unencrypted traffic uses the same port** number (not like http vs https) to ease operators lives

Certificates, certificates, …

- XRootD server needs a host certificate in order to enable encryption
  - configurable with **xrd.tls** directive

- If roots/xroots is being used client will **enforce host verification**
  - the hostname must match the one in the host certificate (or one of the SAN extensions)

Certificates, certificates, …

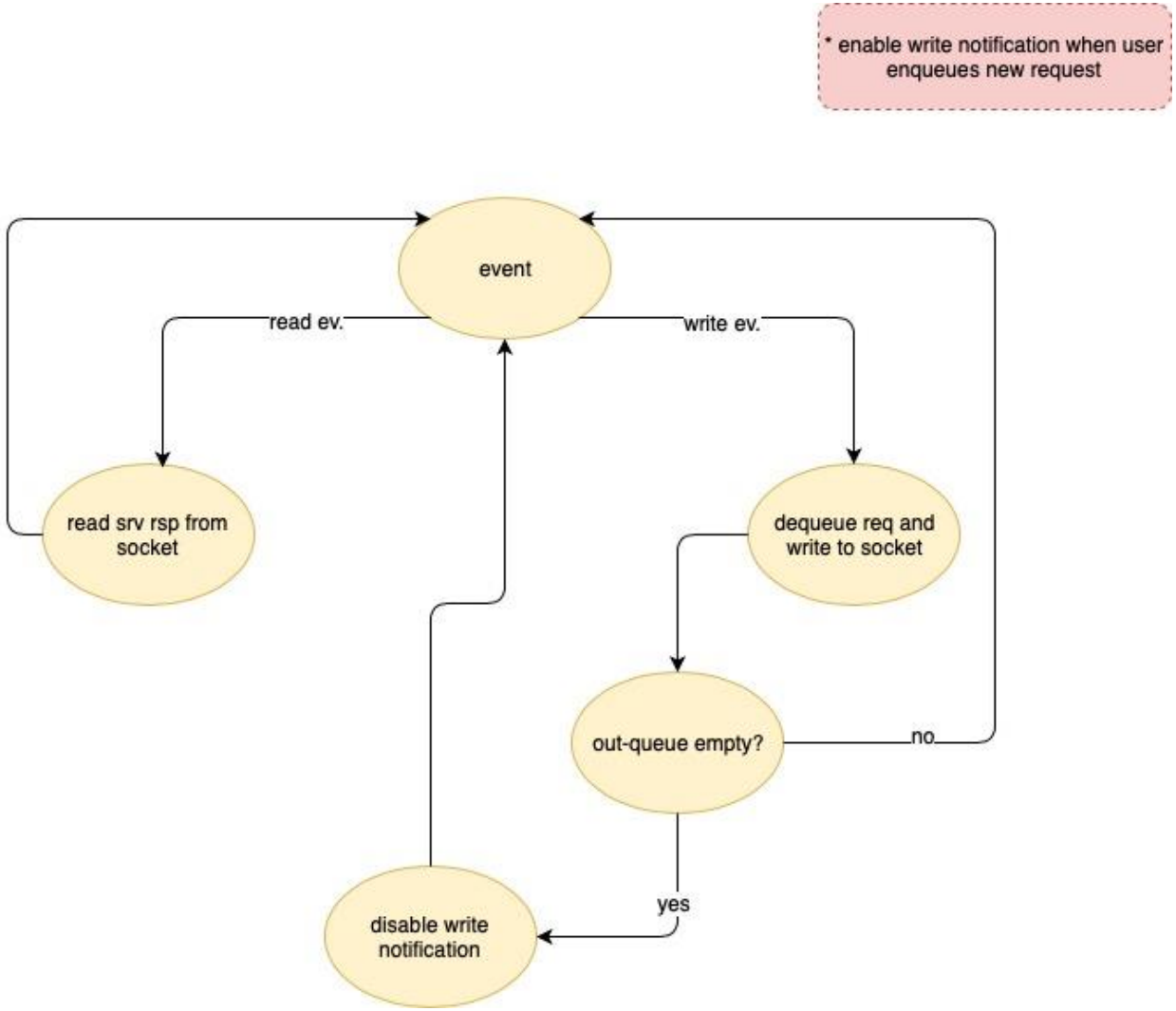
- The client does not need to have a certificate
  - the user **may use his proxy certificate** in order to establish a TLS connection
  - server can be configured to enforce client certificate verification with: **xrd.tlsca**

- Allowing the client to establish the TLS connection based on user X509 proxy certificate opens door to a new **less complex implementation of gsi authentication** in the future

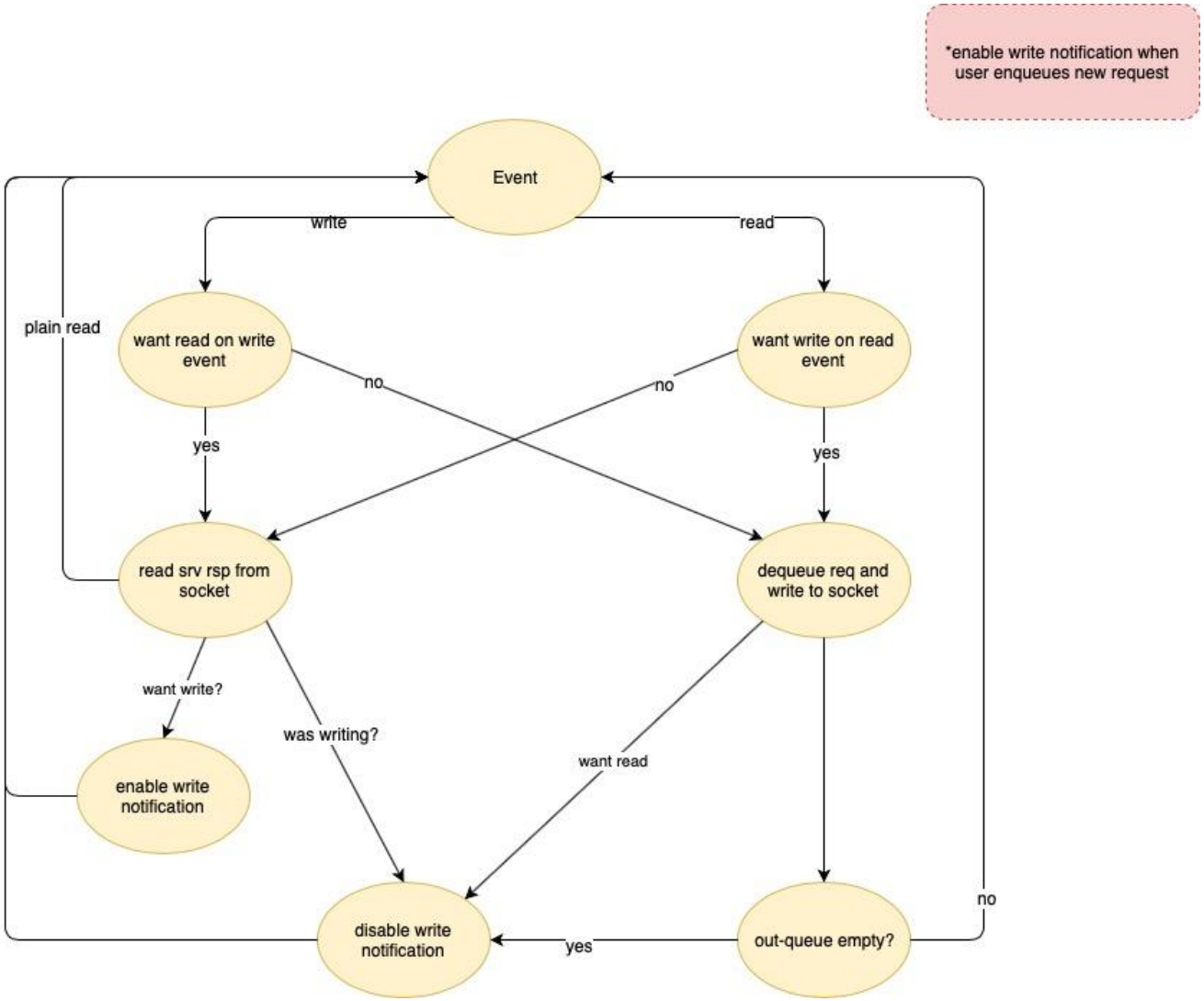
Implementation

- roots/xroots implementations is based on OpenSSL
  - for better performance, **asynchronous APIs and socket BIOs** were used
- **All TLS actions are logged** (e.g. when connection is upgraded to TLS, what version of TLS is being used)
- We are aiming at **isolating OpenSSL in the XrdTls** component
  - should facilitate migration from OpenSSL in the future (e.g. to NSS)

Implementation: event loop (no TLS)



Implementation: event loop (with TLS support)



One last word on TLS …

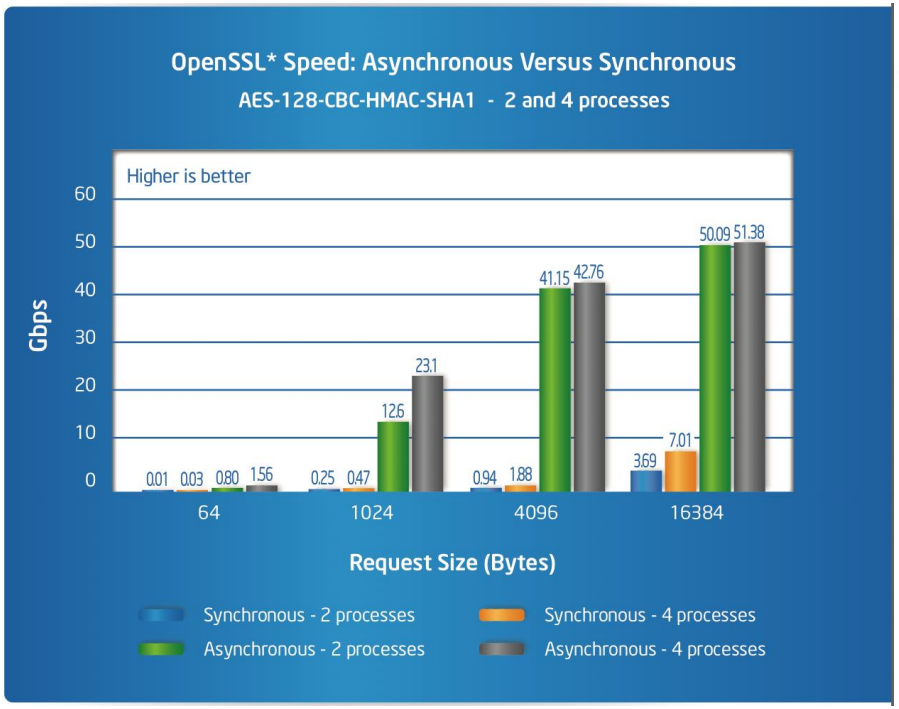
Design choices (OpenSSL)

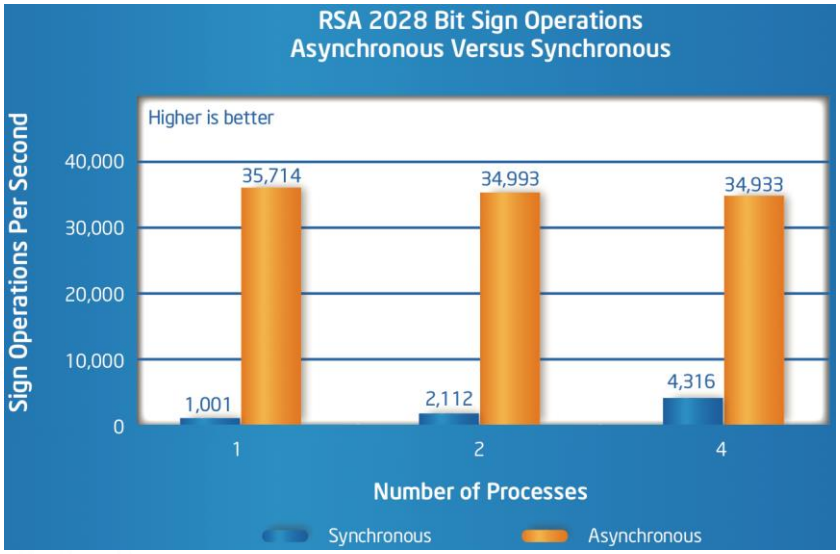
- Memory BIO vs **socket BIO**
  - avoid **copying the data in memory**
  - safes time, memory and CPU cycles
- Synchronous vs **asynchronous** interface
  - reducing **context management overhead** (single-threaded way to handle multiple TLS connections)
  - cryptographic transformations can be more easily **processed on dedicated h/w** (Intel QuickAssist)
    - allows for more optimization
    - parallel processing at crypto-level

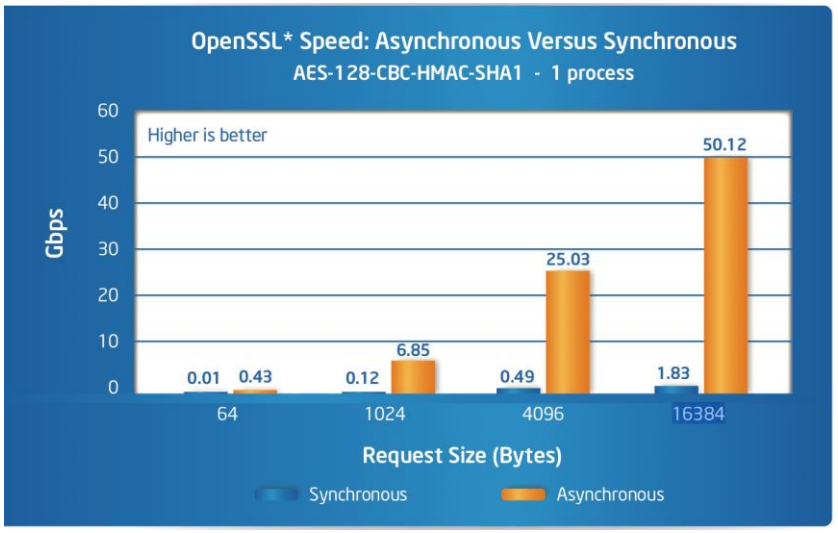
Async vs Sync

Intel preliminary test (2015)

https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/accelerating-openssl-brief.pdf



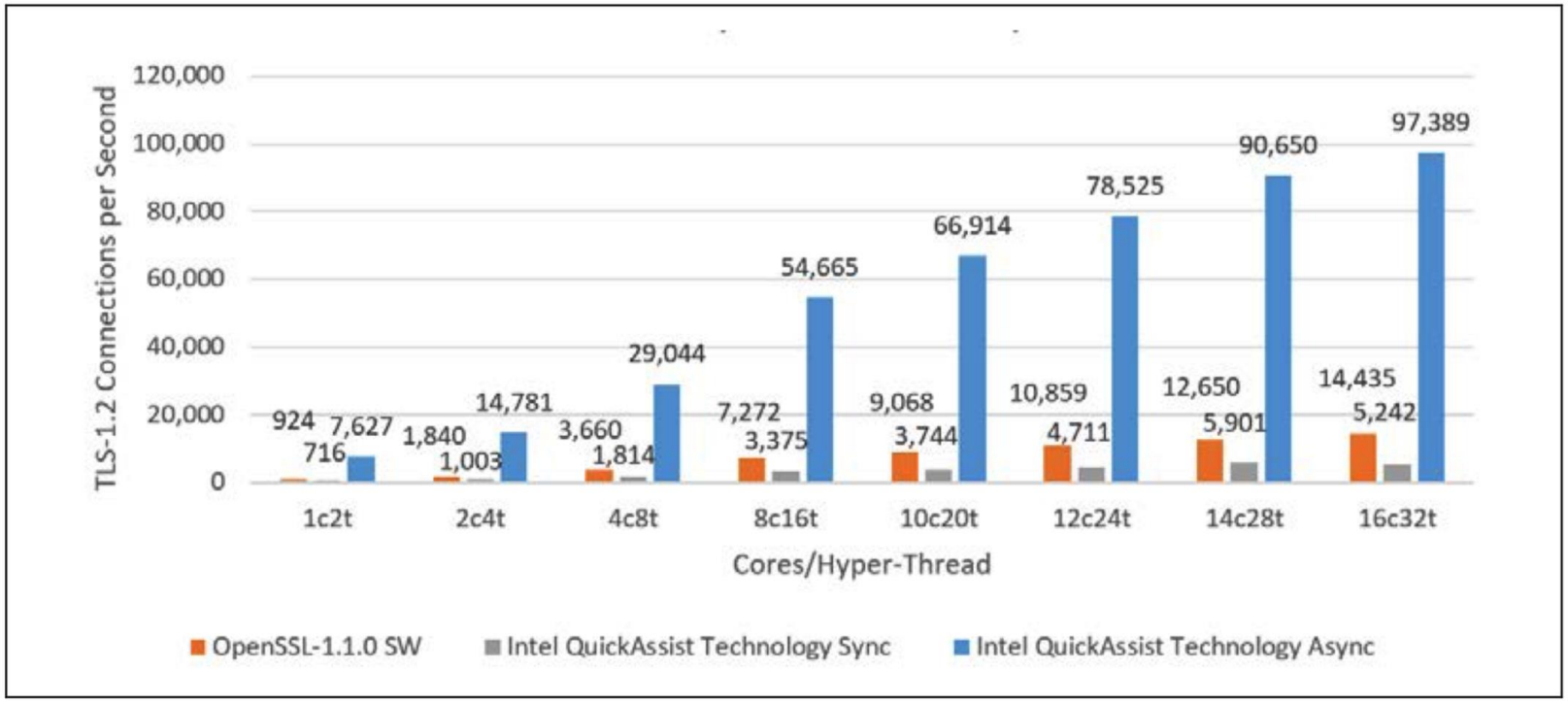




Async vs Sync

Intel NGINX test (2018)

- async + QuickAssits is **significantly faster** than sync
- https://01.org/sites/default/files/downloads/intelr-quickassist-technology/337003-001-intelquickassisttechnologyandopenssl-110.pdf



Some tests with XRootD

- CC7, openssl 1.02k-21
- Overhead of upgrading the connection to TLS
  - ~2.3 msec to carry out the plain XRootD HS
  - ~7.6 msec to carry out the XRootD HS including upgrade to TLS (host & client cert verification)
  - ~37.6 msec to carry out the XRootD HS including GSI authentication
- No throughput penalty for encrypting data observed for test transfers, about +5% increase in CPU usage per connection (server-receiver +15%, to be investigated)

Anything else?

SecEntity re-mastered:

- X509 capabilities, key-value attributes
- **Credential forwarding**, Multi-VO credentials
- Easily **extensible without breaking API**

Boost data integrity (in XCache) significantly reduce transfer failures due to checksum errors

- **Paged read**: read request with **CRC32C** (hardware assisted) **per 4KB block**

Intel ISAL based erasure coding library

- For the AliceO2 use case

Anything else?

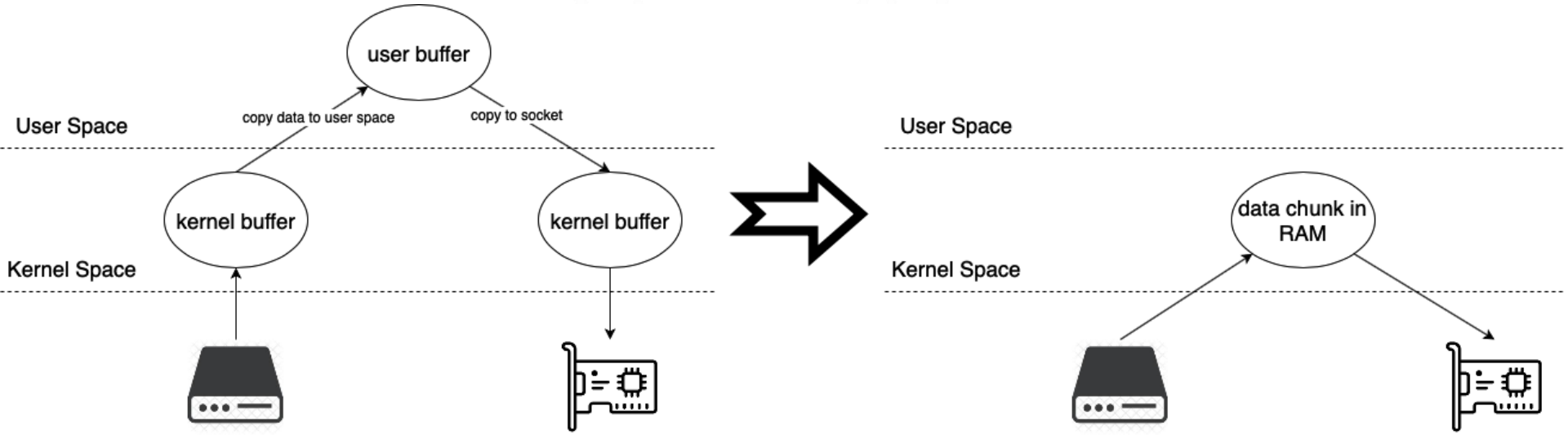
General purpose new features

- **Extended file attributes**
- Extended stat (sets stages for proper **uid/gid tracking**)
- Hardware assisted **CRC32C**
- gstream (monitoring stream optimized to deliver periodic medium-level info)
- Server side **plug-in stacking** with `++` directive
  - User plugin gets a pointer to the level-up plugin so it can call it’s implementation
- **SciTokens** plug-in
- Dirlist with checksums

Tailor-made for EOS

Simplify buffer management & avoid copying data between kernel and user space

- Using **splice/vmsplice** syscalls
- Speed up data transfer: for **slow medium ~3-5%**, for **fast (like ramdisk) ~40%**; reduces CPU usage **by a factor of 3-4**



Tailor-made for EOS

Write recovery at MGM (allows to **recover 99% of I/O errors** for *xrdcp* transfers to EOS)

- EOS does not support standard Write/Read recovery at FST
  - Write is a stateful operation
  - We cannot reopen the file with a valid CGI token from MGM
- MGM (redirector) needs to **set the *kXR_recoverWrts* flag in redirect response** (this way *xrdcp* knows it can recover writes at MGM)
- On an error **xrdcp will fallback to MGM and restart the transfer** at a different FST provided by the MGM

Tailor-made for EOS

Collapse redirect from passive to active MGM in xrootd client

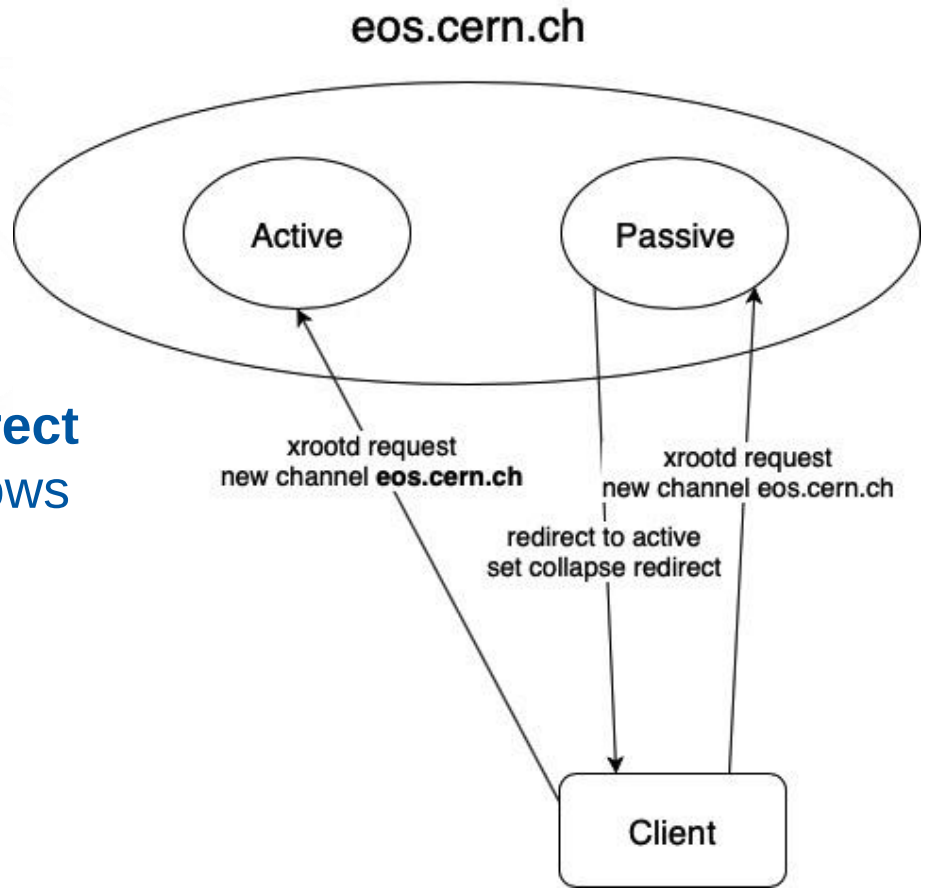
- Use case: facilitate FUSE integration with **passive-active MGM deployment**

- The problem:
  - Select at random one of the hosts behind a common DNS alias (say eos.cern.ch)
  - Say we choose a passive MGM, then the channel labeled with eos.cern.ch is for ever associated with the passive MGM
  - From now on **every request for eos.cern.ch will go through the passive MGM** (which will always redirect to the active one)

Tailor-made for EOS

Collapse redirect from passive to active MGM in xrootd client

- Passive MGM needs to **set the kXR_collapseRedir flag in redirect response** (this way the client knows it is an active-passive deploymen



XRootD4 status

- We are planning to support **XRootD4 for as long as needed for EOS** to move to XRootD5 (though we hope it doesn't mean for ever ;-)

- We had 5 bugfix releases in 4.12.x series after releasing XRootD5
  - Mostly to backport **important fixes for the third-party-copy with HTTP**

2021 plans

- Follow up and support **XRootD5 deployment**; **backporting critical bugfixes** to XRootD4
- High priority new developments
  - **Further extend client EC library and integrate it with EOS**
  - **ZIP append** (initial work done by a summer student, needs checkpoint support on server side)
  - **Paged Write** (Further boost XCache data integrity)
  - Better local **async I/O** support (both in client and server)
- Other new developments
  - **uid/gid** tracking; connect **control and data streams on different interface**; recursive delete (driven by webdav semantics)
  - Get/put file (new TPC); channel level plug-ins; RDMA support; Extending testing infrastructure (mock event-loop)

Questions?