OSG Xrootd Monitoring

Derek Weitzel & Diego Davila

Introduction

The OSG’s goal is to **retire** the GLED XRootD collector, replacing it with the new OSG collector (python).

And, provide **trust** in the collector’s measurements through a series of verifications.

Introduction

With the objective of providing trust in the XRootD transfer accounting we carried out 2 validations of the pipeline of the monitoring data:

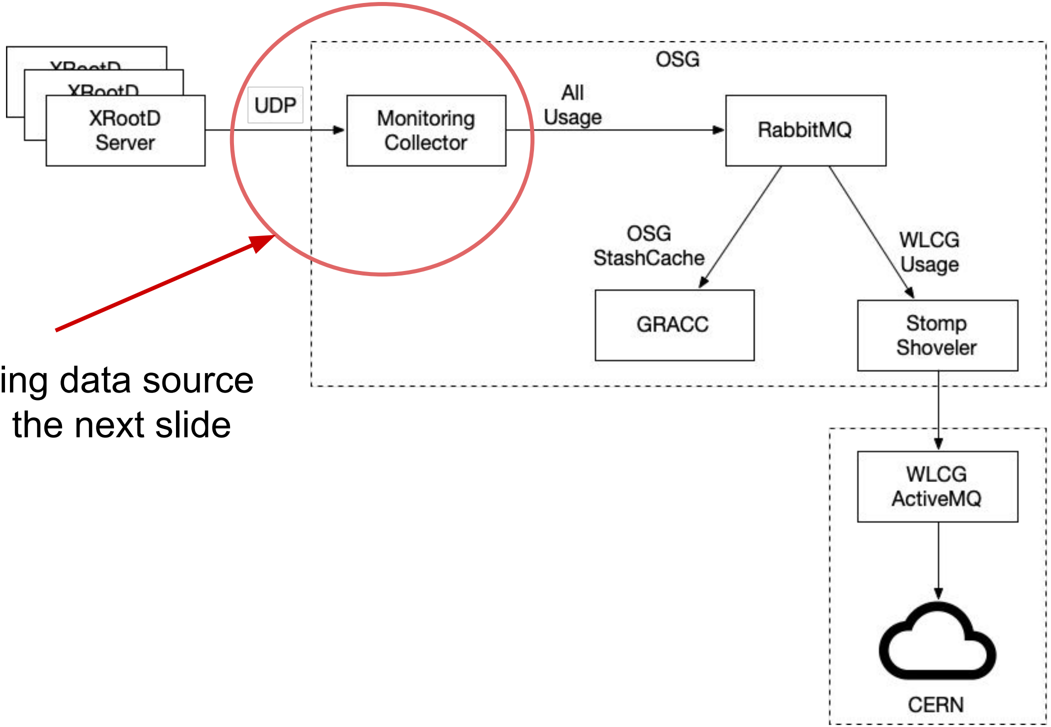
1. **XrootD Monitoring Validation**. Verify the correctness of the components along the pipeline

   https://zenodo.org/record/3981359

2. **XRootD Monitoring Scale Validation**. Find scale related issues

   https://zenodo.org/record/4688624#.YIBS1UhKi3c

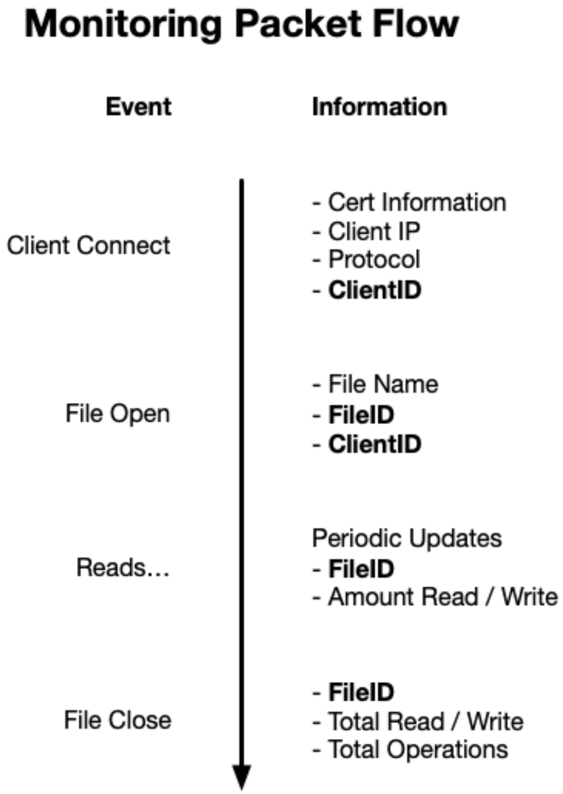
XRootD monitoring data pipeline



Details of monitoring data source and processing in the next slide

Why XRootD Detailed Monitoring is Hard - Format

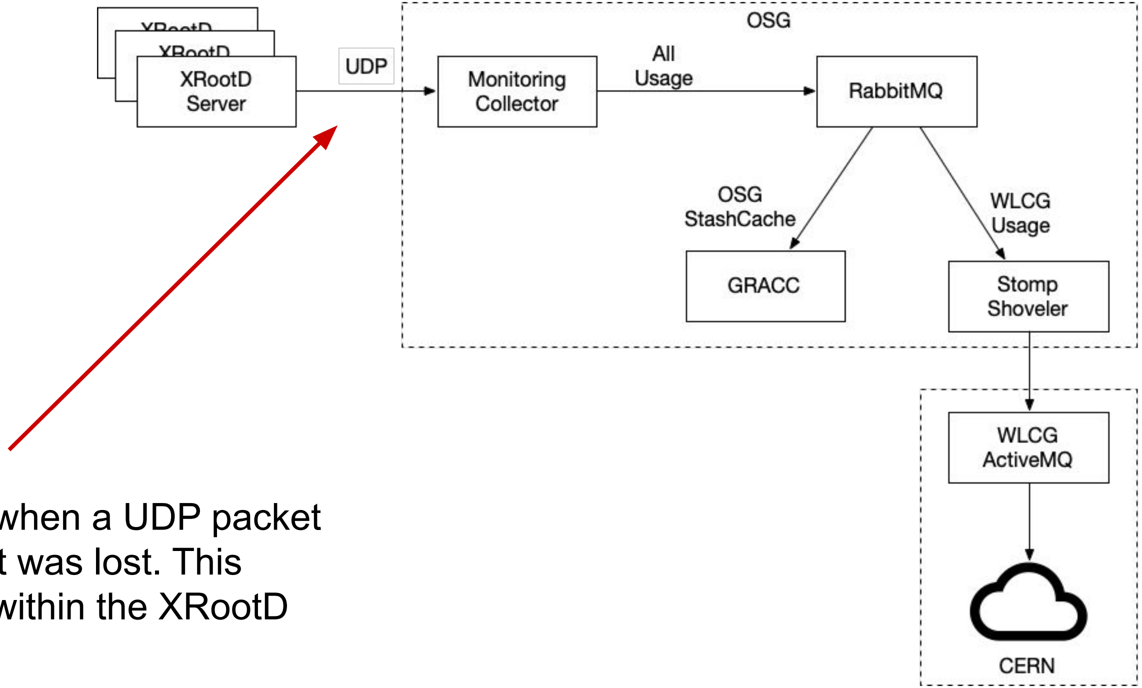
- Collector has to keep a lot of state
- Potential for packet loss means we have to place TTL on state
- Time between client connect and file close can be **hours**
- Must “join” different messages, but may lose packets
- For example, if you get a file close without the corresponding file open, then no idea what file was read.



Monitoring Packet Flow

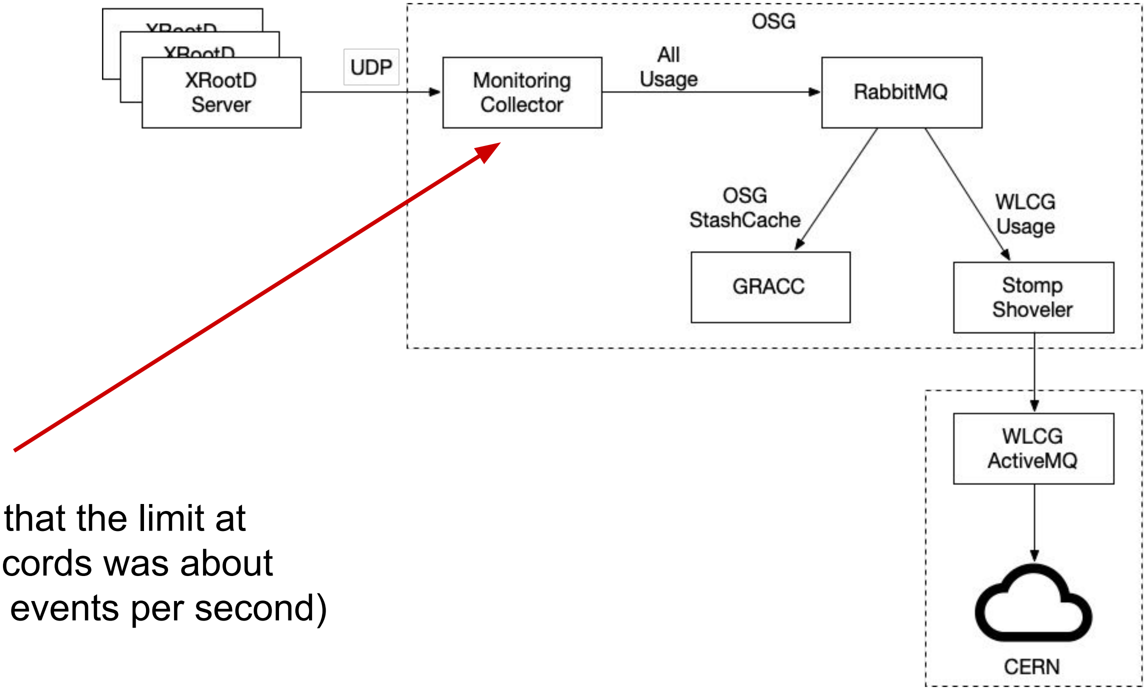
Event	Information
Client Connect	- Cert Information - Client IP - Protocol - **ClientID**
File Open	- File Name - **FileID** - **ClientID**
Reads...	Periodic Updates - **FileID** - Amount Read / Write
File Close	- **FileID** - Total Read / Write - Total Operations

XRootD monitoring data pipeline - first validation



In the first validation, we found that when a UDP packet was larger than the MTU, the packet was lost. This happens when many events occur, within the XRootD server, within a flush window.

XRootD monitoring data pipeline - second validation



On the second validation, we found that the limit at which the collector could process records was about 100 per second, (i.e. 100 *File Close* events per second)

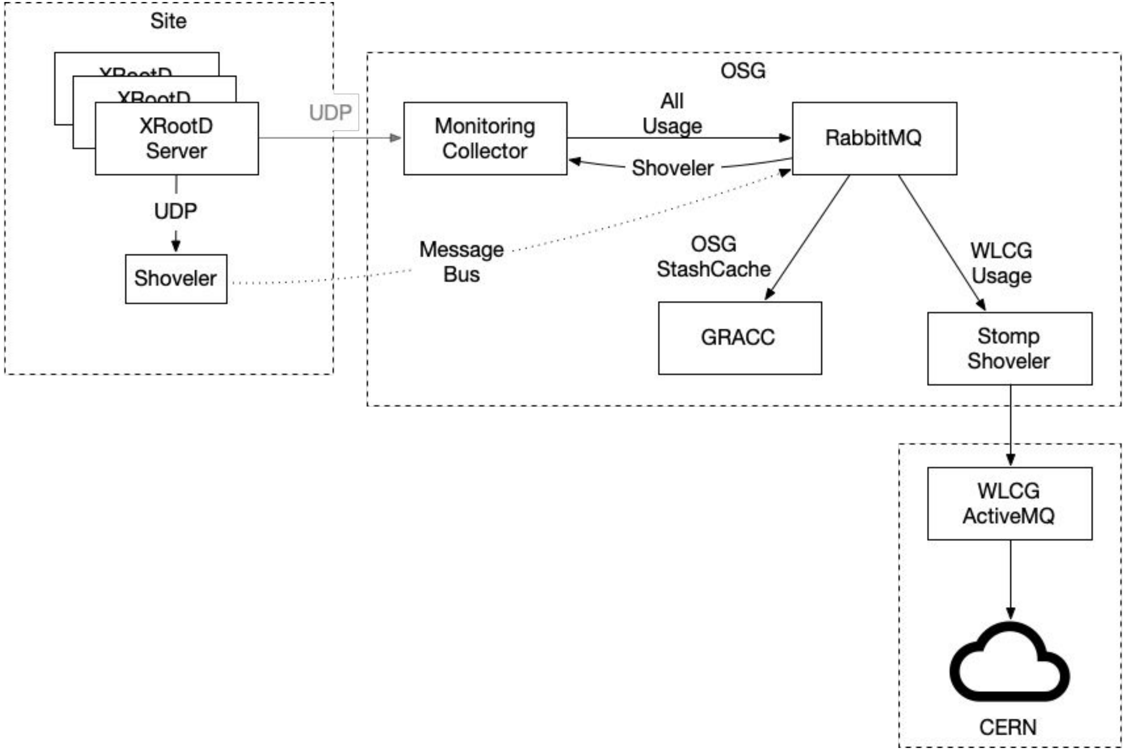
Next steps

- Design and develop a “shoveler” from the UDP format to a resilient transport mechanism (Message Bus)
  - Message bus is preferred since it allows the client and server to be independent
- Redesign the collector to increase the scale
  - State only needs to be kept for a single server.
  - Route messages from servers between processes

- Our goal is to complete these changes by Q3 of this year

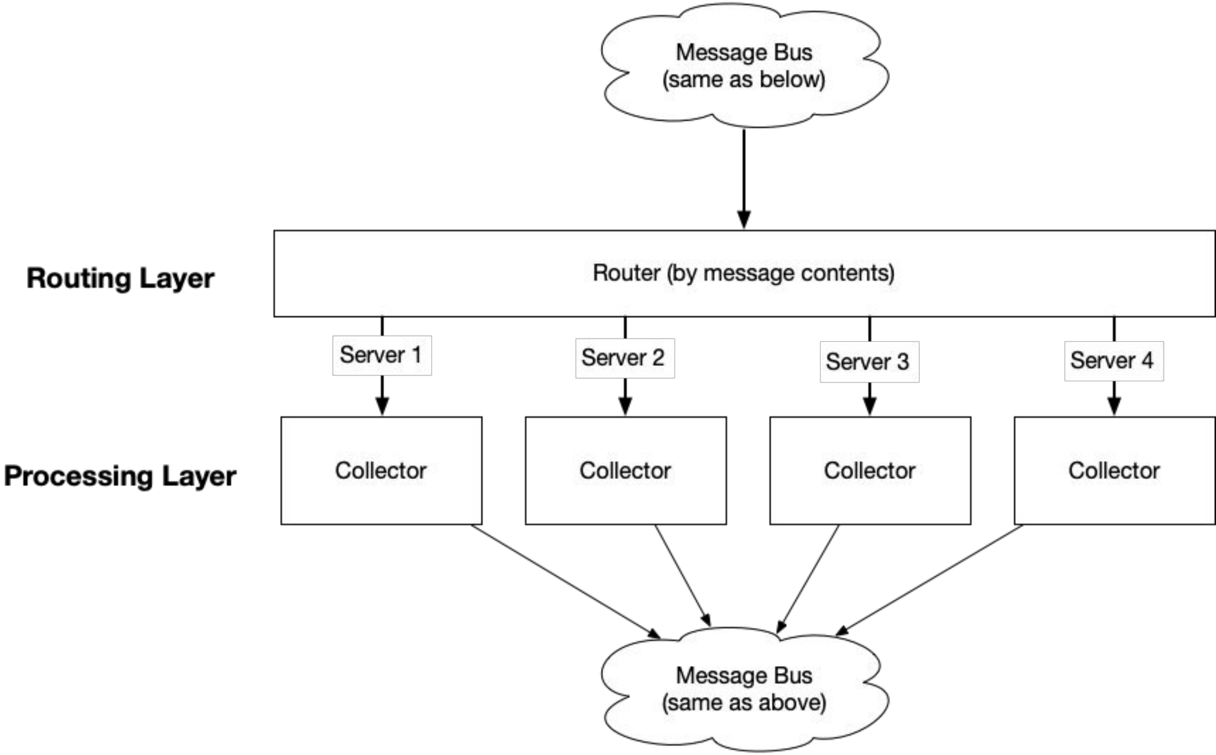
Shoveler

- A lightweight shoveler from UDP to a resilient transfer method
- Connection to RabbitMQ
- Shoveler messages are routed back to the collector for parsing
- Shoveler will use JWT to authenticate to message bus.



Collector Scaling

- Collector scaled by splitting processing
- Each server (or group of servers) has its own collector



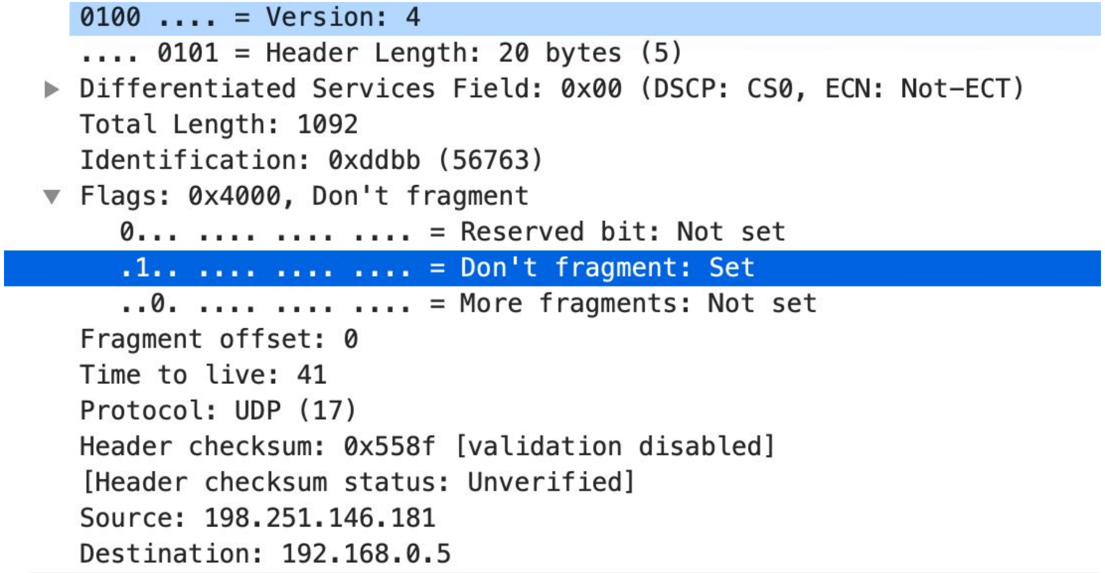
Acknowledgments

This project is supported by the National Science Foundation under Cooperative Agreement OAC-1836650. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Backup slides

UDP Fragmentation

- UDP Fragmentation is a known problem: https://blog.cloudflare.com/ip-fragmentation-is-broken/
- The very Zoom meeting you are on uses UDP packets:



```
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1092
  Identification: 0xddbb (56763)
▼ Flags: 0x4000, Don't fragment
    0... .... .... .... = Reserved bit: Not set
    .1.. .... .... .... = Don't fragment: Set
    ..0. .... .... .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 41
  Protocol: UDP (17)
  Header checksum: 0x558f [validation disabled]
  [Header checksum status: Unverified]
  Source: 198.251.146.181
  Destination: 192.168.0.5
```