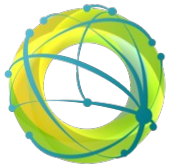


FTS
File Transfer Service

FTS Perspective

WLCG/HSF Workshop 2024 at DESY Hamburg, May 13-17 2024

Presenter: Steven Murray

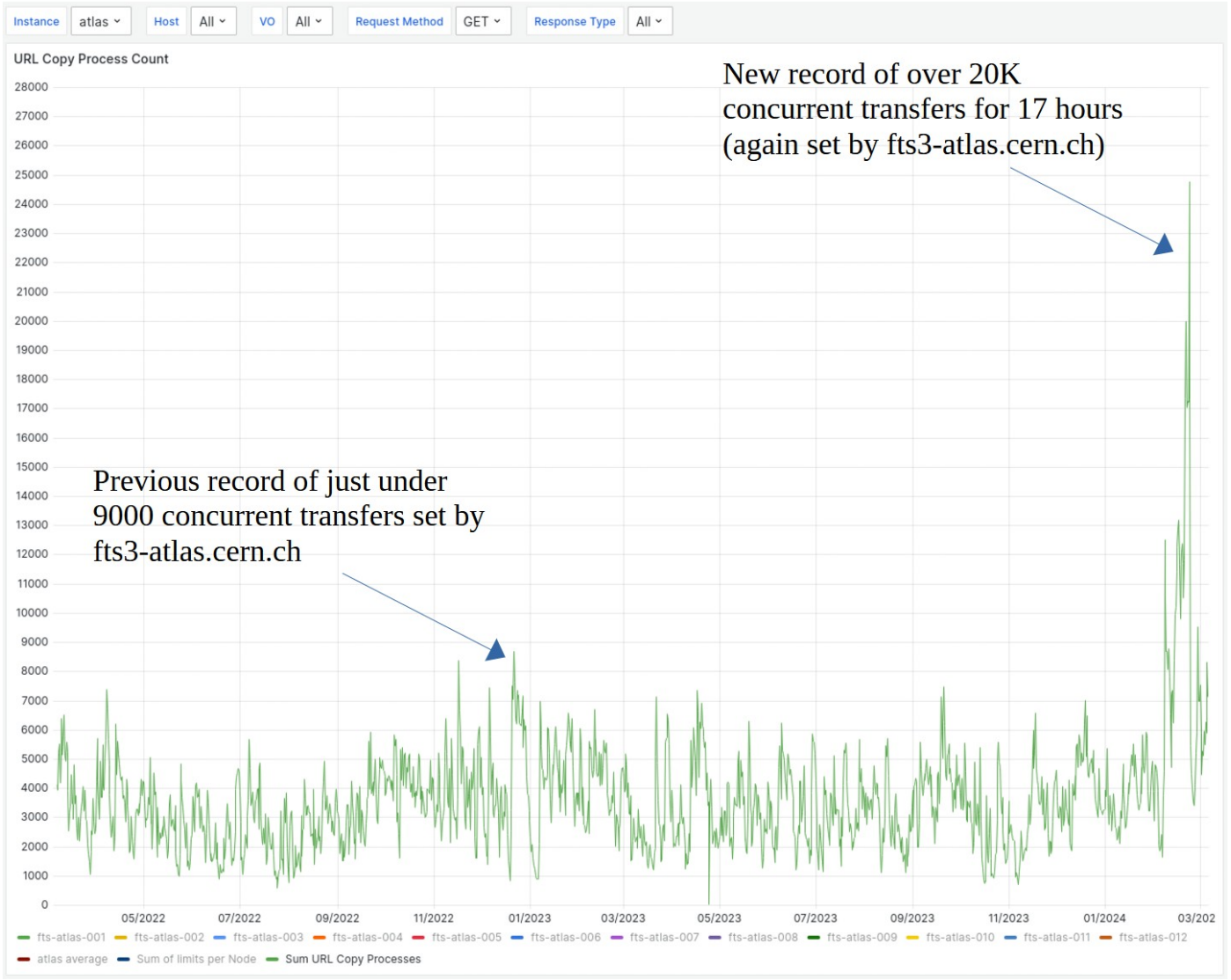
Authors: Joao Pedro Lopes, Shubhangi Misra, Steven Murray, Mihai Patrascoiu and Luca Mascetti

Tuesday 14th May 2024

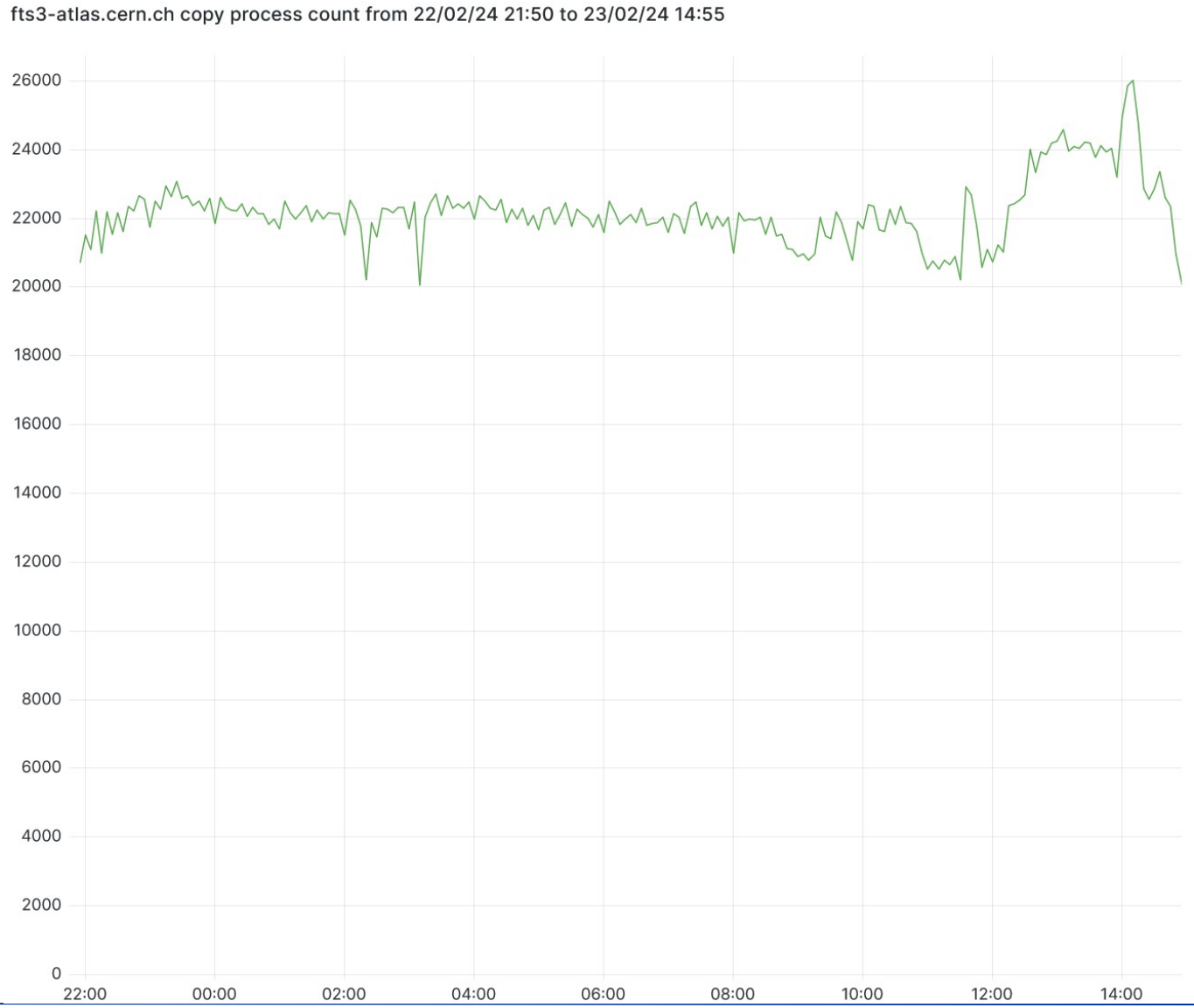
General overview and impressions

- The data challenge was a success for FTS
  - The sum of all FTS instances: ATLAS, CMS, LHCb and Pilot
    - 33 million file-transfers and 249 PB over a two week period
  - Best transfer concurrency provided by a single instance:
    - FTS ATLAS broke the previous record (also held by FTS ATLAS)
    - 20K concurrent transfers for 17 hours (previous record peak was 9K)
  - Half of the FTS transfers used token authentication
- There were challenges
  - Too much fire fighting behind the scenes with respect to **`fts3-atlas.cern.ch`**
    - Many thanks to the database-on-demand team for quickly increasing the DB RAM
    - Defragmentation of the **`fts3-atlas.cern.ch`** DB was not completed
  - FTS maximises concurrent-transfers per link whereas data challenges maximise data throughput

Previous concurrency record vs DC24

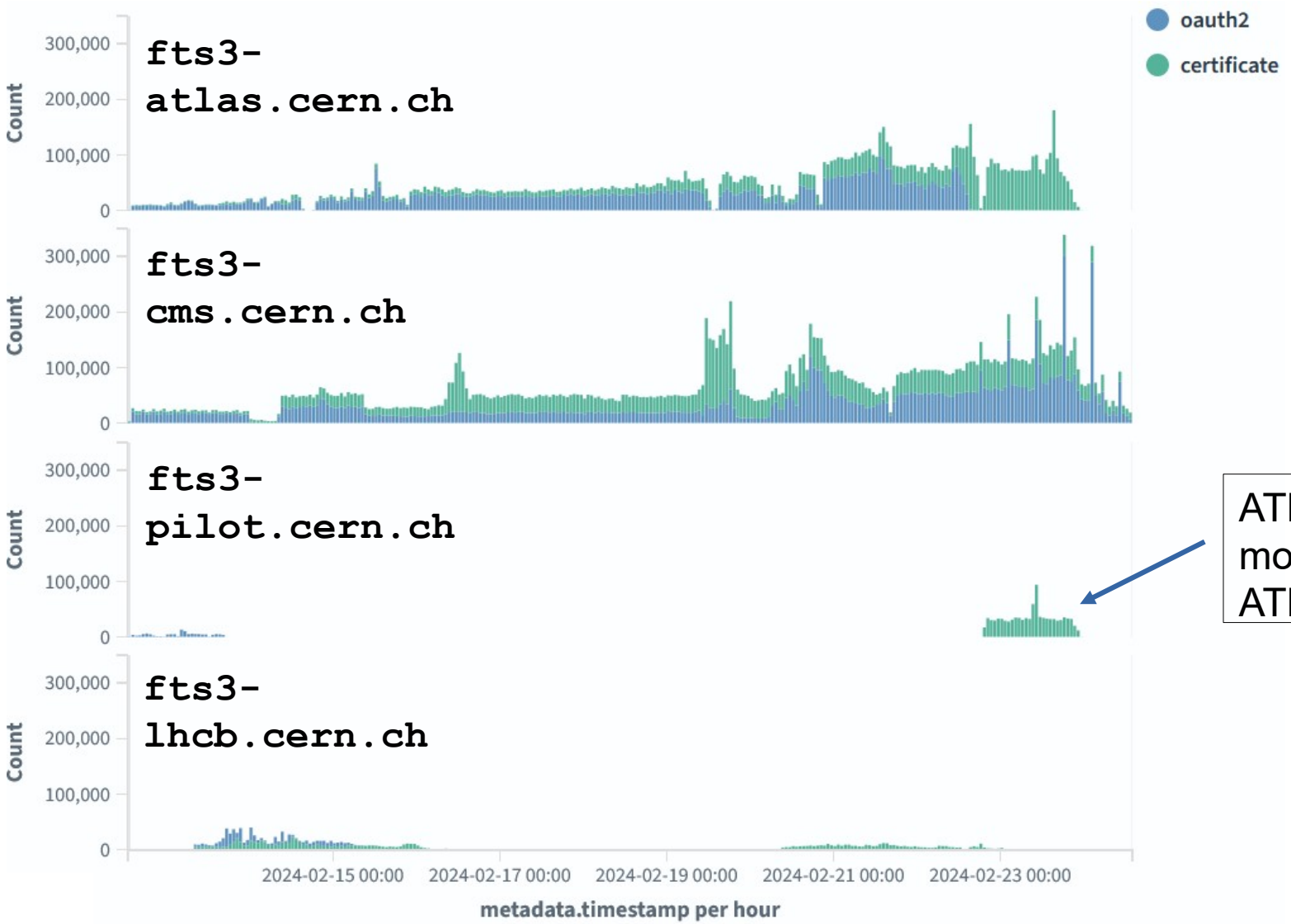


FTS ATLAS - 20K concurrent transfers for 17 hours



Concurrent transfers of FTS CMS and LHCb

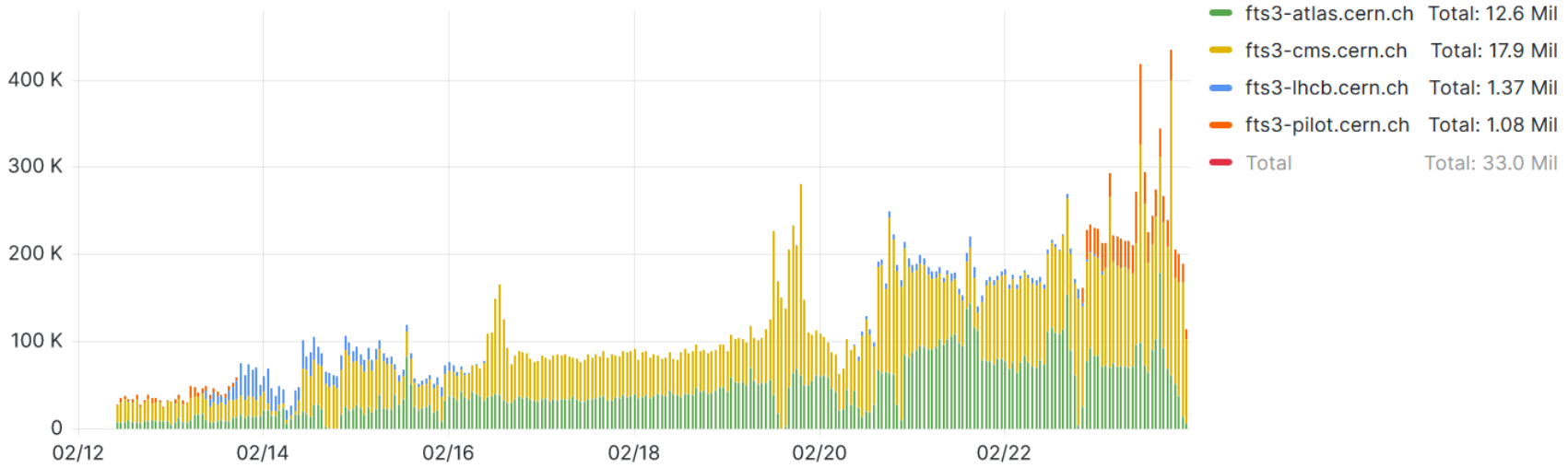
Tokens vs X509 certificates



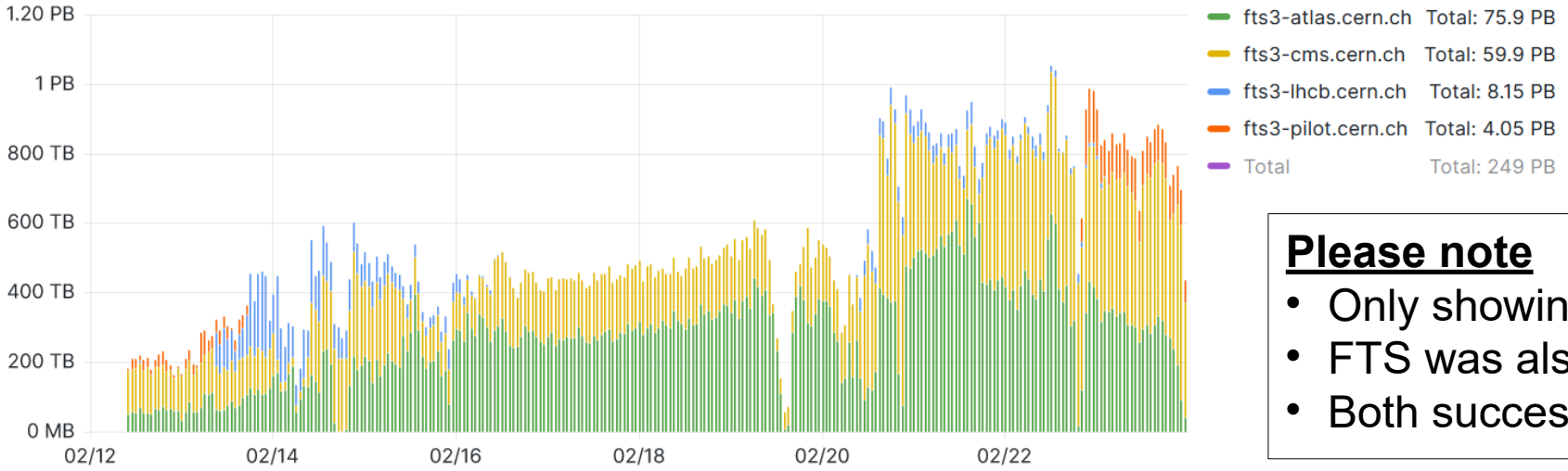
ATLAS Tier-2 traffic moved from FTS ATLAS to FTS Pilot

33 million transfers – 249 PB

DC24 file transfers per FTS instance per hour



DC24 data volume transfered per hour



Please note

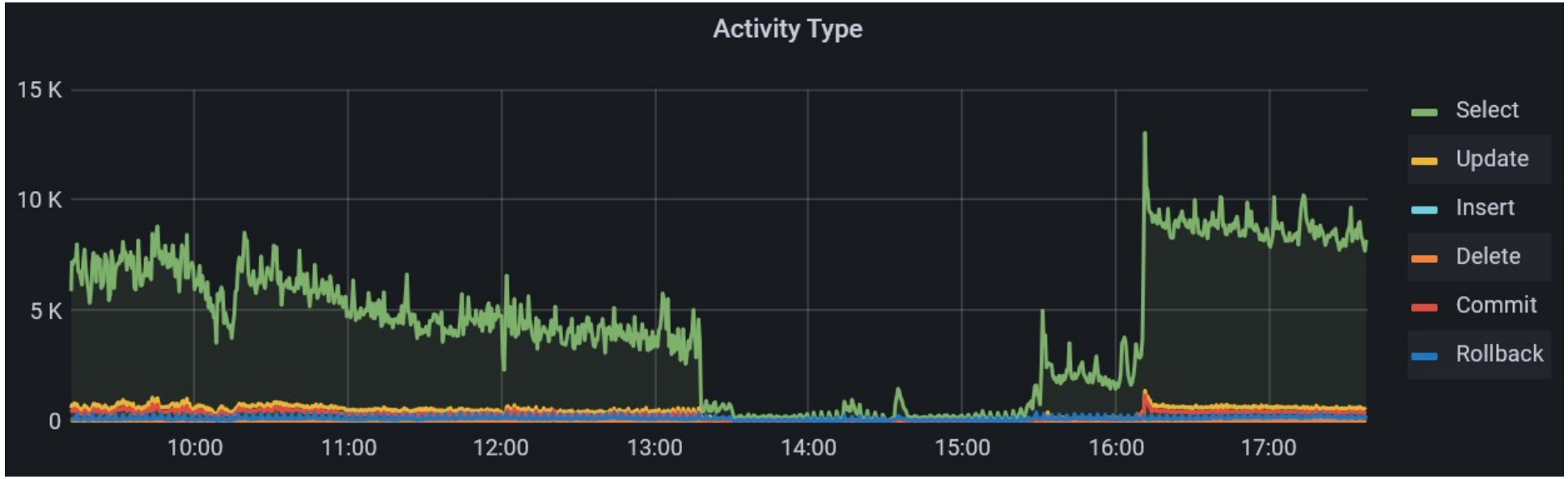
- Only showing “Data Challenge” activity
- FTS was also running production transfers
- Both successful and failed transfers are shown

FTS ATLAS database was overloaded

- It was believed queuing more transfers would increase data throughput
- DB slowed and practically stopped during weekly defrag (every Monday at 10:00am)
- FTS token-refreshing was the main cause of the overload
  - FTS continually polled DB for near-to-expire tokens
  - SQL queries had not been optimised
- Urgent fix required because FTS ATLAS also handled non-DC24 transfers
- Database-on-demand team quickly increased DB RAM from 80 to 120 GB
- FTS team migrated a lower-priority FTS instance out of a high-performance DB server
- FTS team migrated FTS Pilot into the newly vacated DB server
- ATLAS split DC24 load across FTS ATLAS and Pilot

FTS ATLAS database was overloaded

- FTS ATLAS DB blocked at 13:00 on Monday 19th February
- Outage was ~2.5 hours
- Bad interaction between high DB activity and regular defragmentation operation started at 10:00am
- Defragmentation reduces time to warm RAM cache after DB-server restart



Slow FTS ATLAS optimizer

- FTS optimizer increases or decreases the amount of transfers on a link based on the link's current performance
- Usually takes 6 to 12 minutes for the optimizer to run
- 3 hour optimizer runs were observed during DC24
- Slow down was a linear function of file-transfer queue-length
- Hourly restart of daemons meant a full optimizer run could not complete
- Slowdown and restarts "froze" the majority of concurrency decisions
- **Data manager changes were effectively ignored**

Optimizing concurrency vs throughput

- Main reason for not sustaining DC24 target for 48 hours
  - FTS tries to maximize concurrency within its configured boundaries
  - Data challenges try to maximize data throughput
- FTS is configured to enforce limits on links and storage endpoints
- If a link or storage endpoint is not configured then the default is used

Link configuration

First | Previous | 1 | Next | Last

Symbolic name	Source	Destination	Optimizer mode	TCP buffersize	Number of streams	Disable delegation	Min. Active	Max. Active
+ *	*	*	3	0	0	false	2	200
+ RALTest	*	davs://ceph-svc20.gridpp.rl.ac	0	0	0	false	200	200

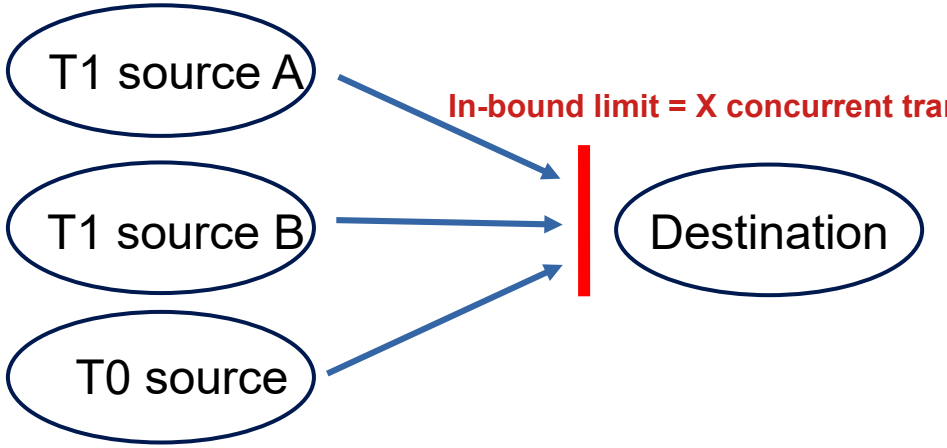
Storage configuration

First | Previous | 1 | 2 | Next | Last

Storage	Site	Debug level	IPv6	UDT	Skip Eviction	Active limit*	Throughput limit*	TPC support
*		0				300 / 300	/	test
davs://antares.stfc.ac.uk		0				1000 / 500	/	test

Optimizing concurrency vs throughput

- FTS CANNOT reach maximum throughput for the following configuration:



All concurrent transfers are treated equally even though those out of T0 may have been faster

- Operational experience shows storage endpoint limits are the system brakes

Lessons learned about tokens

- Incorrectly used tokens are **NOT** secure:
  - Tokens were and **WILL** be leaked (not by FTS)
  - FTS filter added just before DC24
- Too much time spent “discovering” tokens
  - No agreed FTS configuration within multiple IAM instances
  - Single-use refresh-tokens discovered on the fly - Fixed by IAM configuration change
  - 10 hour tokens were refreshed into 1 hour tokens - Fixed by IAM configuration change
- FTS had to deal with “hard” token tests on the fly:
  - Replaced token-refreshing cron-jobs with daemons to prevent overlap when IAM was slow
  - Separated “heavy” housekeeping tasks for tokens from their refresh logic to reduce DB load
- FTS did not know its limits:
  - DC24 helped understand them but FTS has no concept of back pressure
- FTS is now fully responsible for refreshing credentials which is not the case for X509

Work done after DC24

- Continued to provide support for tokens on the following instances:
  - FTS ATLAS
  - FTS CMS
  - FTS LHCb
  - FTS Pilot
- Continued to work with modify-tokens with “relaxed” but “risky” modify-tokens
  - Arguably wide scopes with long durations
- Increased the parallelism of the token-refresher daemon
- Optimised the SQL used by the token-refresher and token-housekeeper daemons

Work done after DC24: Optimized token-refresher

- Token-enabled DB schema uses index to implement efficient “work” queue for token-refresher daemon

```
CREATE TABLE `t_token` (
  ...
  `access_token_refresh_after` timestamp NOT NULL,
  ...
  `retired` tinyint(1) NOT NULL DEFAULT 0,
  ...
  KEY `idx_retired_access_token_refresh_after`
(`retired`,`access_token_refresh_after`),
```

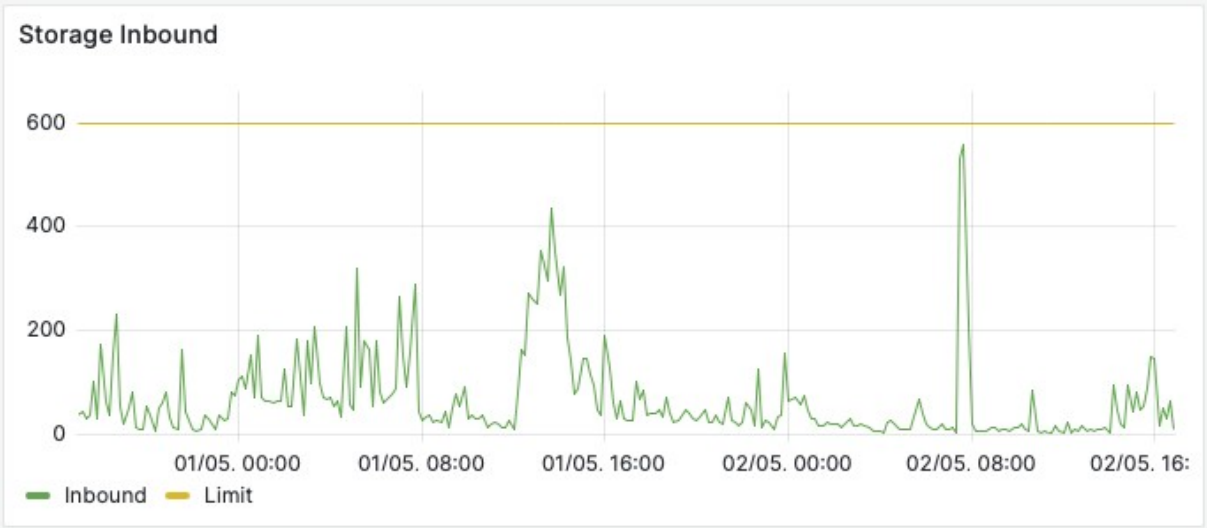
- FTS REST interface calculates and INSERTs **`access_token_refresh_after`** for each token

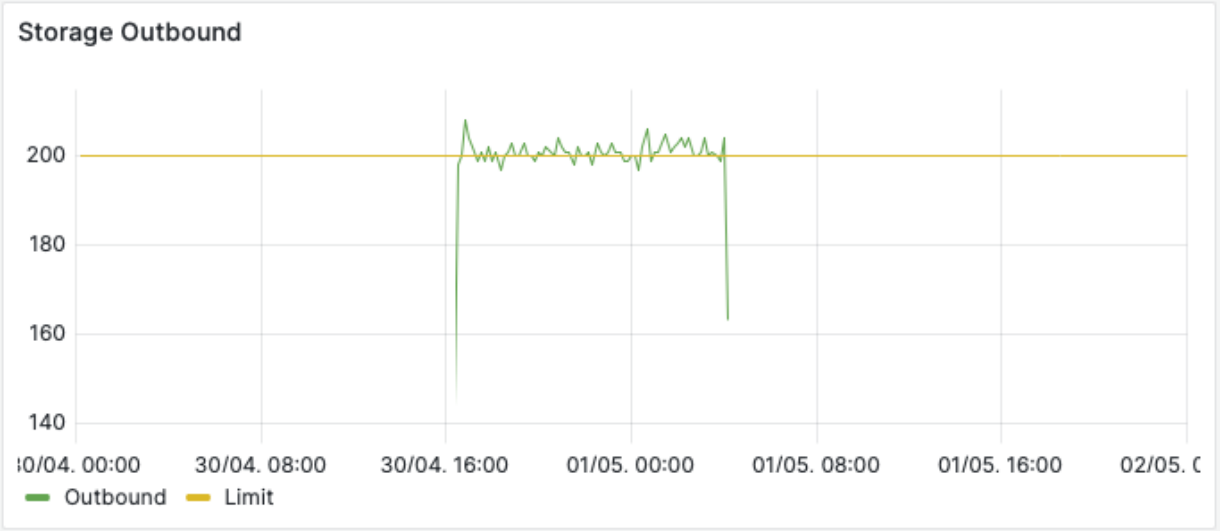
```
access_token_refresh_after = token_dict["nbf"] + lifetime_sec * 0.5

INSERT INTO t_token
  ...
  access_token_refresh_after,
  ...
```

- FTS token-refresher daemon efficiently reads “work” queue using database index

```
SELECT
  ...
FROM
  t_token
WHERE
  retired = 0
AND
  access_token_refresh_after >= NOW()
ORDER BY
  retired ASC,
  access_token_refresh_after ASC
```

Work done after DC24: Visualize storage saturation





Future work and investigations

- Add a back-pressure mechanism
  - RUCIO have kindly offered to switch on their FTS back-pressure
- Improve the performance of the optimiser
  - Allow the optimiser to be switched off
  - Parallelize the optimiser algorithm
- Provide a better way to show the saturation of destination storage-endpoints
- Provide token support for the Tape REST API
- Introduce a new FTS scheduler
  - Reduce amount of required DB RAM
  - Add priorities between links

Food for thought

- **We need a single contact person for tokens**
- Can single-shot refresh-tokens be banned from the WLCG token lifecycle?
- Can dynamic IAM-client registration be banned to reduce the attack surface?
- Should FTS automatically refresh access-tokens?
  - Why can't fresh tokens be pushed into FTS like X509 proxy certificates are today?
- Can we agree on how to put the VO in tokens?
  - FTS had to invent a way to map tokens to VOs
  - VO values must be the same for tokens and certificates
- We learnt from ATLAS that not all tokens are equal – what optimisations can be made?
  - Read and create tokens can have wide scopes and long durations
  - Modify tokens should have narrow scopes and preferably short durations
- We learnt from CMS that they use the same file paths on all storage endpoints:
  - Can we ban the `https://wlcg.cern.ch/jwt/v1/any` wildcard audience from modify-tokens?
- Can all storages ensure they have integrated themselves with the `dteam` token provider?