Integrating Network-Awareness and Network-Management into PhEDEx

Vlad Lăpădătescu[*]
Caltech / USA
E-mail: vlad@cern.ch

Tony Wildish
Princeton / USA
E-mail: awildish@princeton.edu

ANSE Collaboration [†]

PhEDEx is the data placement and management tool for the CMS experiment at the LHC. It controls the large-scale data-flows on the WAN across the experiment, typically handling 1 PB of data per week. While robust, its architecture is now ten years old and has yet to fully adapt to today's production environment, an environment in which the network is the fastest and most reliable component.
The ANSE (Advanced Network Services for Experiments) project, in the context of CMS, aims to greatly improve PhEDEx' network awareness for smart source selection, as well as to integrate bandwidth provisioning capabilities in the data transfer management. Both parts require a good knowledge of the network status, topology and of course, access to useful and up-to-date performance metrics.
One of the first steps towards this goal involved the identification of a mechanism for informing PhEDEx about independent network performance metrics. Methods for providing these metrics have been prototyped and verified in a LAN testbed using fake data transfer requests. This mechanism is already directly usable by CMS in their production environment.
Currently, the ANSE-PhEDEx testbed is spread over many servers at a number of sites. It is composed of several machines dedicated to PhEDEx site agents, one server holding the PhEDEx central agents, a central database and one server which contains the PhEDEx website and data-service. Some of the site nodes have additional attached storage nodes.
In this paper, we present the work that has been done in ANSE for PhEDEx. This includes performance measurements using the Fast Data Transfer (FDT) tool and the extension of the PhEDEx agent that downloads files to a site to allow it to control the network via creation and use of dynamic circuits. We present the results of our tests using these new features, on high-speed WAN circuits ranging from a few Gbps to 40Gbps and detail the development done within PhEDEx itself.

Finally, the paper will also describe the future plans for the project.

International Symposium on Grids and Clouds (ISGC) 2014,
23-28 March 2014
Academia Sinica, Taipei, Taiwan

[*]Speaker.
[†]B. Ball, A. Barczyk, J. Batista, K. De, S. McKee, A. Melo, H. Newman, A. Petrosyan, P. Sheldon, R. Voicu

1. Introduction

PhEDEx[1] is the data-placement management tool for the CMS[2] experiment at the LHC. It manages the scheduling of all large-scale WAN transfers in CMS, ensuring reliable delivery of the data. It consists of several components:

- an Oracle database, hosted at CERN
- a website and data-service, which users (humans or machine) use to interact with and control PhEDEx
- a set of *central* agents that deal with routing, request-management, bookeeping and other activities. These agents are also hosted at CERN, though they could be run anywhere. The key point is that there is only one set of central agents per PhEDEx instance
- a set of *site-agents*, one set for every site that receives data

PhEDEx maintains knowledge and history of transfer performance, and the central agents use that information to choose among source replicas when a user makes a request (users specify the destination, PhEDEx chooses the source). The central agents then queue the transfer to be processed by the site agents. PhEDEx operates in a data-pull mode, the destination site pulls the data to itself when it is ready. This gives the sites more control over the activity at their site, so they can ensure that neither their network nor their storage are overloaded.

The key PhEDEx agents, for our purposes, are the *FileDownload* and *FileRouter* agents. The FileDownload agent is a site-agent, each site runs one or more copies of this agent. It is responsible for the actual execution of file-transfers; it retrieves the transfer queue from the database, organises the files in whatever way is suitable for the actual transfer tool that will be used, launches the transfer, monitors its progress, verifies the files have been delivered, and reports the transfer results via the database. The actual transfers are executed using lower-level tools such as the WLCG File Transfer Service (FTS[3]) or the Fast Data Transfer tool (FDT[4][5]).

The FileRouter agent is a central agent. This agent takes the global set of transfer requests and builds the work queues for each site to process with its FileDownload agents. The agent chooses a source-site for each destination, based on internal history and statistics. This works well in practice, but provides only a limited view of the network performance. The name *FileRouter* is perhaps misleading, since the agent doesn't decide the network path the data will take, only the source for each destination and the order in which files are transferred.

PhEDEx was originally conceived over ten years ago now, and the architecture still reflects design decisions made at that time. Then, the network was expected to be the weakest link in the developing Worldwide LHC Grid (WLCG)[6]. Networks were expected to have bandwidth of the order of 100 Mb/sec, to be unreliable, and to be poorly connected across the span of the CMS experiment. Accordingly, PhEDEx will back off fast and retry gently in the face of failed transfers, on the assumption that failures will take time to fix, and that there is other data that can be transferred in the meantime. This can lead to large latencies caused by transient errors, with subsequent delays in processing the data.

The data-transfer topology was designed with a strongly hierarchical structure. The Tier-0 (CERN) transferred data primarily to a set of 6-7 Tier-1 sites, and each Tier-1 site handled traffic between itself, the other Tier-1s. and it's local Tier-2 sites. Tier-2's wishing to exchange data would have to go via Tier-1 intermediaries. This kept the transfer-links (i.e. the set of (source,destination) pairs)

mostly in the realm of a single regional network operator, the only cross-region links were used by Tier-1s which were assumed to have the expertise to debug problems and keep the data flowing. It also kept the overall number of transfer links low, since the majority of sites (the Tier-2s) had only one link, to their associated Tier-1 site.
Today, the reality is very different. The network has emerged as the most reliable component of the WLCG; problems with transfers tend to be at the end-points rather than in the network itself. Bandwidths of 10 Gb/sec between Tier-2 sites is common in many areas, and 100 Gb/sec connectivity is starting to appear. Even where the bandwidth is still relatively low, connections are quite reliable, so data can be transferred effectively. This has led CMS to embrace a fully connected transfer mesh in which all sites are allowed to connect to any other site, so the number of transfer links has risen from about 100 to nearer 3000.
CMS has decided to address these limitations, and is considering a number of possible avenues for the future of PhEDEx[7]. The ANSE[1][8] project is addressing some of them, specifically how to make PhEDEx more aware of the network status, and how to provide PhEDEx with the means to control the network by way of virtual circuits and bandwidth-on-demand (BoD).

2. ANSE testbed

In the context of the ANSE project, the central agents, DB and frontend all run at CERN. The site agents are running on machines located in different geographical locations. (see Figure 1).

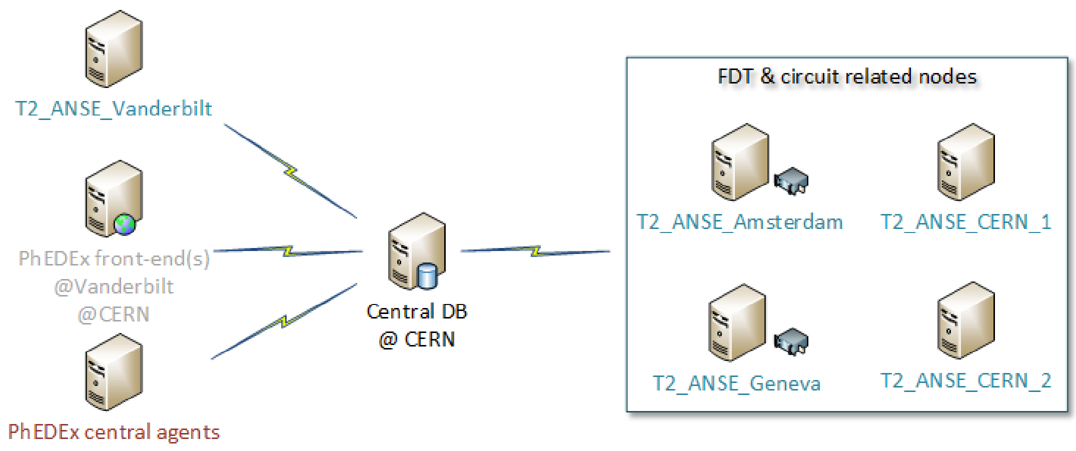

Figure 1: PhEDEx setup for ANSE

For development work and prototype tests we mainly use OpenStack virtual machines. These contain PhEDEx installations and also double as small storage nodes. These typically have 8GB of RAM, 4 VCPUs and 80GB disk. These VMs are able to have sustained disk-to-disk transfer rates of 1Gbps.
For more advanced tests we use four physical servers, two located in Geneva and two in Amsterdam. In each location, one of the two servers is used as a PhEDEx node, while the other is used as a storage node.

[1]A project funded by NSF CC-NIE program, initially for two years, which started in January 2013

Each storage node (sandy01-gva and sandy01-ams) has dual E5-2670 CPUs (32 logical cores) 64GB or RAM and 2 LSI disk controllers. Each of the LSI controllers manages 8 high speed SSDs. Two RAID-0 arrays are created on each LSI controller (4 SSD per RAID 0 array).
Between the two storage nodes we have a high capacity 40Gbps link and use Ciena routers to create dynamic links between them.
Since the storage nodes were designed for high speed rather than high capacity, we created a large number of soft links to large random-filled files (2->15GB) on the source disk.
On the destination side, we either used a crontab running every minute to remove transferred files or we relied on the PhEDEx post-validation script to remove files in bulk at the end of a transfer job.
Monitoring is performed with MonALISA[4][9] and PhEDEx.

3. Integrating external sources of information

3.1 Sources of information

Currently CERN runs three PhEDEx instances:
Production is used for production CMS data transfers
Debug is used for link monitoring and link commissioning tests
Development is used for testing software and DB schema upgrades
Although the instances run on the same infrastructure, each one is independent of the rest. The Production instance chooses transfer sources based on its internal monitoring data alone. The Debug instance, on the other hand, has monitoring information about many more links. Because of this, it makes sense that where links are shared, network statistics should be shared as well (Figure 2)

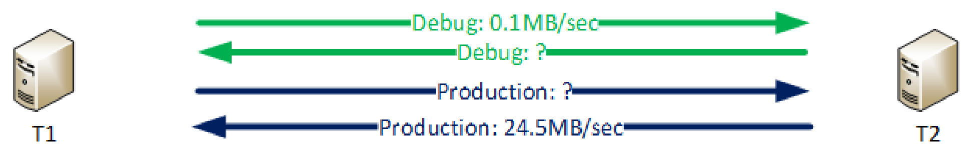

Figure 2: Example case where the Production and Debug instances contain complementary data

A tool has been developed which retrieves statistics from all instances and aggregates the data. This tool then produces a file that the FileRouter agent can use to supplement its internal statistics, giving it a more complete overview of the network performance.
The next step is to integrate it with perfSONAR[10], effectively gaining a more comprehensive picture of what's happening on the network. At the time of writing this paper, perfSONAR is yet to be deployed on all WLCG sites. We also lack a perfSONAR API which we can call to retrieve this data. Finally, PhEDEx will need some work to use this data, since it has its own internal naming convention for sites which corresponds to their logical identity, rather than to their internet hostnames.

4. Integrating virtual circuts into PhEDEx

There are several points in the PhEDEx software stack at which it is possible to integrate the control and use of virtual circuits; per transfer-job, at the level of the FileDownload agent, and at the FileRouter agent.

At the lowest level, a circuit can be requested per transfer-job, by the transfer tool called from the FileDownload agent. This level of integration was already achieved some time ago with the initial implementation of the FDT backend for the FileDownload agent. Each transfer-job would attempt to create its own virtual circuit, if it failed it would simply proceed to transfer over the general-purpose network.

The advantage of this approach is simplicity, there is nothing to modify in PhEDEx itself, all the work happens in the external transfer tool. There are, however, a few disadvantages:

- Transfer-jobs tend to be short-lived (tens of minutes rather than hours), so circuits are being created at a high rate. This is not currently sustainable.
- PhEDEx is typically configured to overlap transfer-jobs, to offset the overheads incurred between successive jobs. This means that concurrent transfer-jobs will be competing for circuits, with consequences that may be hard to understand. For example, if two jobs can share a circuit, what happens to one of them when the other finishes, and tears down the circuit?
- There is no overview of the use of network resources. Circuits are requested based on nothing more than the local knowledge of a single link, without regard for other traffic flows into the destination or out of the source.

The next level of integration is at the level of the FileDownload agent. This site-agent has an overview of the data-flows into the destination site at any time, so can make intelligent decisions about creating circuits. Apart from simply deciding that it should - or should not - create a circuit for a given flow, it can also create circuits that are longer-lived, serving a series of transfer-jobs lasting several hours. This will make for much more stable operation, both from the viewpoint of the network fabric and from the viewpoint of the transfer-jobs. This is the first level of integration at which virtual circuit control becomes really practical for production use by PhEDEx. The only significant disadvantage is that destination sites still make circuit-requests without regard for conditions at the source sites, so the risk of sub-optimal configurations still exists. For example, site A may book a circuit to site B to pull data from it at high speed, but that may commit most of site B's WAN bandwidth, leaving little for site B to download from sites C, D and E.

The highest level of integration is at the level of the FileRouter, or centrally, with a view of the entire transfer system. The FileRouter is essentially a high-level scheduler, deciding the set of source-destination pairs for fulfilling transfer requests and building the queues of transfers for each destination site. As such, it alone can have an overview of how busy each site is expected to be, so it could decide which transfers should use circuits and which should not. It could even base its choice of source-replica for a given transfer on availability of circuits and bandwidth, not just on current network performance between the sites. It can also decide that some transfers are low priority, so should not request a circuit even if one might be available, in order to leave that resource free for higher priority requests that might come along.

There are two disadvantages to integration at the FileRouter. The first is that the FileRouter deals

with transfers that *will* happen, in the near future. When something happens to invalidate its predictions (e.g. a power-cut at a site) this will invalidate those predictions to a greater or lesser degree. The more coupled the sites are in their scheduling, the wider the implications of a single site-outage.
The second disadvantage is the complexity of the problem. Developing efficient scheduling algorithms, including predicting the availability of circuits on a shared network, is not a trivial task. For example, queues may be re-organised to take advantage of circuits, instead of just using raw priority and age as the queueing criteria. The FileRouter agent will also have to inform FileDownload agents when they are expected to use a circuit and when not to, and not just which files to transfer. Despite these complications, having centrally managed knowledge of when and where to create circuits for PhEDEx is the ultimate goal.

5. Using a virtual circuit per file-transfer

5.1 Integrating FDT into PhEDEx

The FDT tool integrates InterDomain Controller Protocol (IDCP) On-Demand Secure Circuits and Advance Reservation System (OSCARS[11]) calls so integrating it in PhEDEx naturally gives us Bandwidth on Demand capabilities. In addition to this, FDT performs extremely well as a transfer tool in itself, as we will show in this paper.
As shown in Figure 3, the current architecture consists of four main components:

FDT tool - written in Java it's based on an asynchronous, flexible multithreaded system, using the capabilities of the Java NIO libraries. It's capable of reading and writing at disk speed over wide area networks and runs on all major platforms.

PhEDEx backend - written in Perl, this backend receives transfer jobs from the FileDownload site agent which will, in turn, invoke the fdtcp wrapper.

Fdtcp wrapper - written in Python this is the interface between PhEDEx and FDT. It prepares the file list as required by FDT, invokes the fdtd service and harvests reports to be propagated back to PhEDEx.

Fdtcp daemon - written in Python as well, it's a permanently running daemon, responsible for authentication and completion of requests from fdtcp. These requests are transmitted via PYthon Remote Objects (PYRO) calls and will either launch FDT in client mode on source sites or in server mode on destination sites.

Figure 3 also describes a normal scenario in which Site B needs to transfer data from Site A. In this example, the PhEDEx instance and the storage node are separate physical servers.
The FileDownload agent on the PhEDEx server on Site A gets a list of files to transfer from the PhEDEx database. It breaks the list (up to 10 PB of data) into manageable chunks (up to a few TB of data) and hands the chunks to the FDT backend. The FDT backend on that machine will then issue an fdtcp command for each group of files to copy those files from the storage node of Site A to the storage node of Site B.
Fdtcp will, in turn, invoke the *fdtd* daemons on the two storage nodes. Each fdtd daemon will first verify that the command comes from an approved server, after which it will launch the appropriate FDT tool. The fdtd daemon on Site B will start the FDT tool in server mode and the daemon on site B starts FDT in client mode.

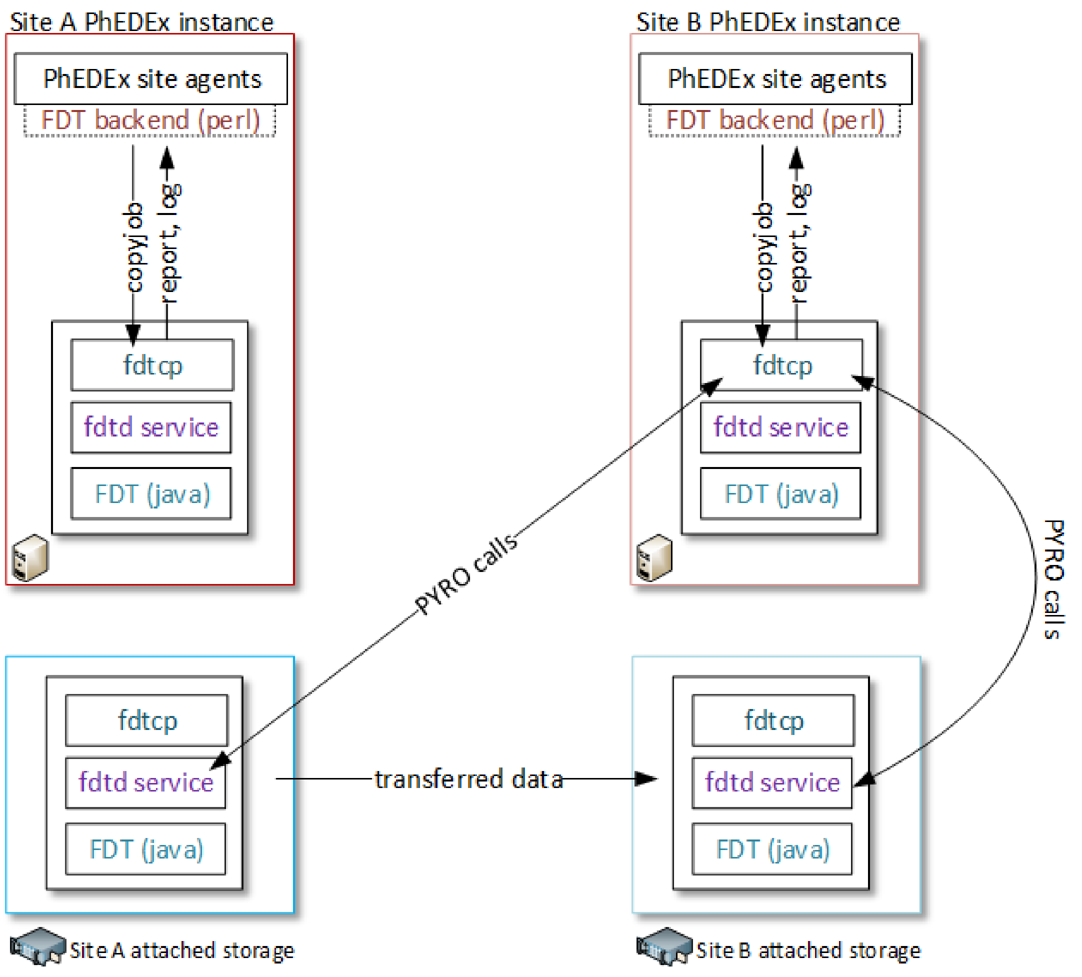

Figure 3: Diagram of various components needed to integrate FDT into PhEDEx

Once this is done, the transfers are handled by FDT. After the transfer finishes the reports are then propagated up the chain, eventually reaching PhEDEx.

5.2 Testbed configuration and transfer results with FDT

5.2.1 Testbed configuration

The tests were run on our testbed (Figure 1), using the Geneva (T2_ANSE_Geneva) and Amsterdam (T2_ANSE_Amsterdam) sites. All LSI controllers were used.
Each transfer job was 2.25TB (150x15TB files).

5.2.2 Transfer results with FDT

As seen in Figure 4 and Figure 5 FDT is able to achieve sustained rates of over 1500MB/sec. As PhEDEx shows, the average sits at a marginally slower 1360MB/sec. Figure 4, shows periodic drops in transfer rates. This is due to the fact that between each transfer job (which is composed of a limited number of files) there is a delay between the ending of one job and the launch of a new one. This is what accounts for the difference in average reported rates by PhEDEx and the average rate for each transfer job. This is an artifact of our configuration, which only allowed one transfer job to run at a time. In a production environment PhEDEx would run overlapping jobs, so this structure would not be apparent. Nonetheless, we chose to keep the simpler configuration for these first tests, so that results would be easy to interpret.

These high transfer rates are not only due to the storage system alone. FDT automatically starts the correct number of readers and writers if the files at the source and destinations sites are spread on different physical disks/controllers.

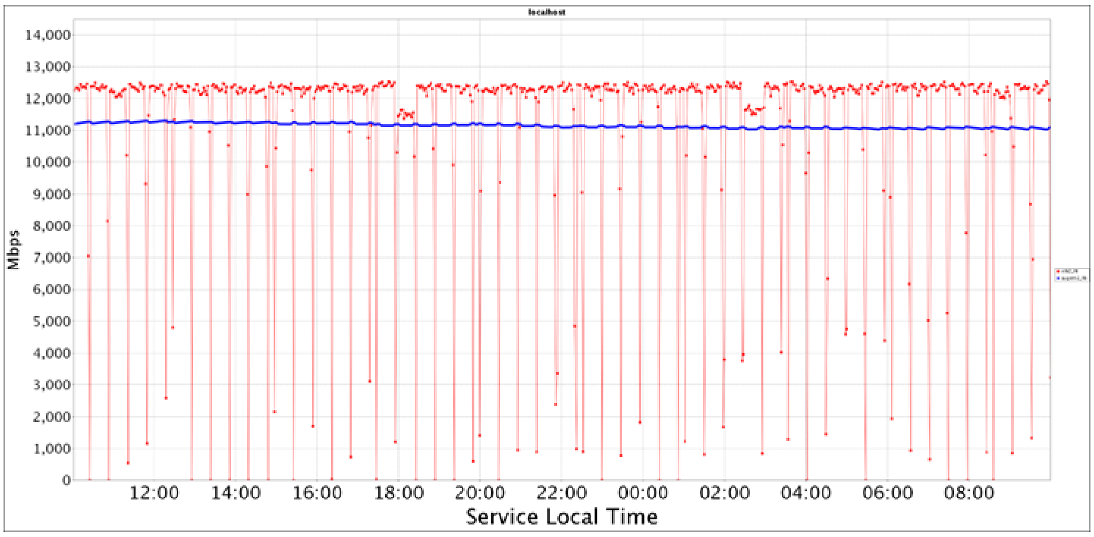

Figure 4: PhEDEx transfers over 24hrs using FDT as reported by MonALISA. The red line represents the network interface throughput in Mbit/sec, with data points each minute. The blue line represents a moving average over one hour of that throughput.

5.2.3 Limitations of the current testbed

At the moment this system does have some drawbacks:

- As noted, The PhEDEx agent configuration has been simplified, compared to the production environment, for ease of comprehension.
- FDT requires a POSIX compatible interface to function. At the moment not all PhEDEx sites provide such an interface to their storage elements, so FDT is not actually deployed by CMS at this time.
- In order to get the best performance out of the systems, files that are to be transfered need to be spread among different disks on both source and destination storage servers. This adds more complexity to the sites' configuration and operation.

Despite these limitations - or rather because of them - this testbed is a valuable resource for understanding the behaviour of PhEDEx when we attempt to push its transfer performance to the limit.

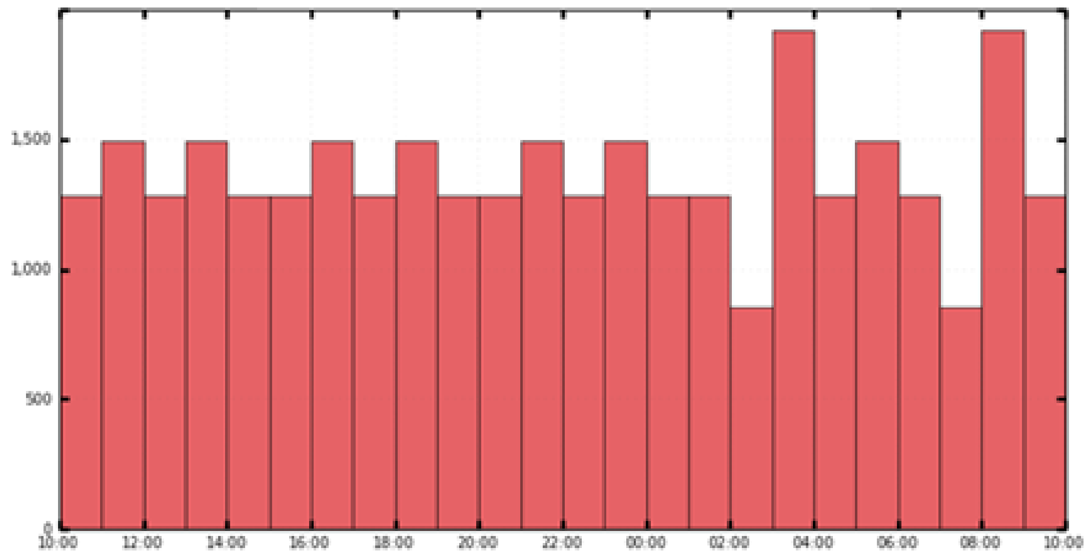

Figure 5: PhEDEx transfers over 24hrs using FDT as reported by PhEDEx. Transfer rates are given in MB/sec, with one hour bins. This, coupled with the fact that rates are only reported at the end of job transfers, make this plot look bumpy.

6. Virtual circuits at the FileDownload Agent Level

6.1 Changes to the FileDownload agent

For this prototype we decided to integrate all of the control and circuit awareness logic in the FileDownload agent, thus eliminating the need for an additional site agent or any changes to the DB schema.
To ease development and to facilitate testing, we fake calls to circuit management APIs which in turn, return IPs of already established static circuits. From PhEDEx' point of view it's as if a new path has been created. If some conditions are met, PhEDEx switches to using this path, as it would with a dynamic circuit.

6.1.1 Making use of a circuit

PhEDEx transfers files in bulk, copying several files with each transfer command. The number of files varies from site to site depending on local constraints, but it is typically in the range of a few tens of files per transfer job. Each transfer job, therefore, contains information about the source and destination Physical File Names (PFNs) of several files.
A PFN contains the protocol that needs to be used, the hostname or IP of the file and the local path of that file. This means that if we want PhEDEx to use circuits, we need a mechanism to replace the original hostname or IP in each source/destination PFN with the source IP and destination IP of the circuit.
The FileDownload agent actually receives only Logical File Names (LFNs) from the database, plus the name of the chosen source site. It constructs the PFNs from the LFNs and a lookup-table per-site which each site maintains and uploads to the database. After calculating the source and

destination PFNs, files are bundled into transfer jobs and queued for submission with whichever transfer tool has been configured between the two sites.
When the transfer jobs are taken from the local queue, but before they are actually submitted, the agent will check to see if a circuit has been established between the two endpoints. If so, it manipulates the source and destination PFNs, replacing the hostnames or IP numbers of the endpoints with the IP numbers corresponding to the circuit. This manipulation is completely transparent to the actual transfer tool in use.
It could also happen that a circuit becomes available, while the FileDownload agent marks tasks ready for transfer. This means that part of the tasks in a job will contain source/destination PFNs having the original hostname/IPs, while others will use the circuit IPs. When the FDT backend is used, it will automatically launch two different jobs: one for the files transferred over the normal path and the other for files transferred over the circuit. We still need to test how the other backends react. In any case, even if a transfer job would fail because of this, PhEDEx will just retry, as it would for any failure, and the transfer would succeed the second time.

6.1.2 Circuit awareness and lifecycle management

Since the standard FileDownload agent doesn't have knowledge about more than one transfer paths over a given link we needed to add this circuit awareness and a way to manage the lifecycle of a circuit on a given link.
PhEDEx agents are event-driven, using the Perl Object Environment (POE[12]) framework. To implement circuit-control we added several POE controlled events. Here are the main ones:

- check_workload
- request_circuit
- handle_request
- teardown_circuit
- check_circuit_setup

check_workload This is a recurrent event at 60 second intervals. It is used to estimate the amount of work that remains to be done based on the current size of the download queue. In order to do this, it needs to know the average rate that the current link is capable of and the total number of pending bytes. The latter is calculated based on the PENDING tasks that it currently holds. The former can be retrieved in two ways:

- if the agent has recently transfered tasks it will retain information about previously DONE tasks, which will then be used for the average rate calculation
- if nothing has been transferred, it will try to get this information based on the link average rate calculated by PhEDEx itself.

If an average rate source-destination pair has been calculated, it will estimate the amount of work needed to be done.
Going through all of the links, we check if all of the conditions below had been met:

- the amount of work pending is above a given amount[2]
- a circuit has not been established

[2]This is arbitrarily set to five hours

- a circuit request is not pending
- a circuit request hasn't failed within a given time-window[3]
- transfers haven't previously failed, within a given time-window, while a circuit was active

If this is the case, we trigger the request_circuit event for that particular link.

request_circuit As the name suggests this event is used to request a circuit.
It does the following:

- creates a state-file for this request. This file contains the names of the PhEDEx nodes involved in the request, the time of the request and the lifetime of the circuit. This file is used to maintain knowledge of circuits across restarts of the FileDownload agent
- sets a timer to cancel the request if there is no timely response. If we don't get a circuit within 10 minutes it's very likely that we won't get it at all.
- in the case of this prototype the "handle_request" event is triggered directly, however in the production version, this event will actually be a callback from the API call where a circuit is being requested

handle_request This event is a callback from the API call to request a circuit.
It will:

- trigger the event "remove_circuit_request"
- if the circuit creation failed, flag it and return
- modify and save the state file corresponding to this request by adding relevant information concerning the circuit that has just been established: time the circuit was established, time the circuit expires, endpoint IPs of the established circuit.
- starts a timer for the "teardown_circuit" event if an expiration time has been specified for the circuit

check_circuit_setup This is another recurrent event at 60 second intervals and is used to provide sanitation in case of errors. There are two important scenarios that we are interested in:

1. Download agent has crashed and is restarted
   (a) In-memory state information is lost, but circuit-request state file(s) exist on disk:
   For each circuit-request state file that has no matching in-memory data, try to cancel the circuit request and remove the file from disk
   (b) In-memory state information is lost, but circuit-established state file(s) exist on disk:
   For each circuit-established state file that has no matching in-memory data, check the expiration time in the state file:
      - if the expiration time is not defined, or is defined but didn't expire yet, the circuit can be reused. Internal data is populated based on state file and if the expiration time is defined, the teardown_circuit timer is set.
      - if the expiration time is defined but it expired, remove the state file and try to tear down the circuit
2. Reconsider failed circuits with failed requests or transfers:
   Creating a circuit on a particular link may have been blacklisted due to requests for circuits

[3]This time-window is arbitrarily set to one hour

failing or due to files transfers failing on that circuit. After a given time these circuits are removed from the blacklist and the system is free to try to use them again.

handle_request_timeout Triggered after a timeout, cancels a request and cleans its internal state

remove_circuit_request Cleans up internal data and state file concerning this circuit request

teardown_circuit Cleans up internal data and state file concerning the circuit and calls the API for tearing down the circuit.

6.2 Setup and test

The prototype was tested on ANSE's testbed (Figure 1) using our two PhEDEx sites: T2_ANSE_Amsterdam and T2_ANSE_Geneva. As we mentioned earlier each LSI controller manages 8 SSDs split in two RAID 0 arrays. Since the tests are write intensive and were scheduled to run for around 24 hours, we decided to only use 1 LSI controller, in order to minimize the negative effects on the life of the SSDs. Two 10Gbps virtual circuits were created for the purposes of this test. Figure 6 depicts the setup used.

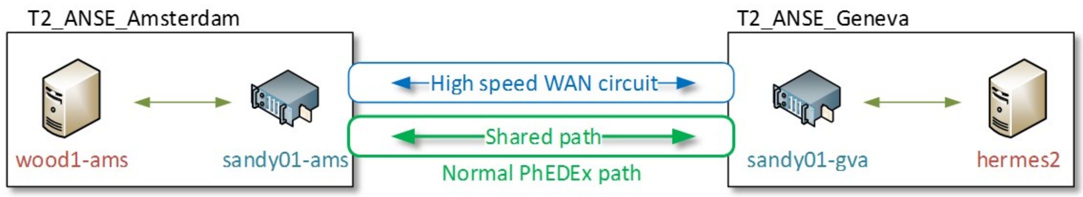

Figure 6: Diagram of the testbed used by the prototype

One of the circuits was used to model a shared link in which PhEDEx had to compete with other traffic. This background traffic was generated by Iperf and consisted of a continuous stream of UDP packets at 5Gbps . The other circuit served as the dedicated link.
The main purpose of this test wasn't to show that we can saturate a 10Gbps link with PhEDEX (although we came close with just 1 LSI controller), but that a PhEDEx FileDownload agent is able to switch to using a new path in a transparent manner and with no down time.
The first part of the test consisted of a 10 hour run with PhEDEx transfers on the shared link. After this time, PhEDEx switched to using the dedicated circuit and continued transfers for another 10 hours. We set up PhEDEx to run a single 450 GB transfer job at a time, each one comprised of 30 files of 15 GB each.

6.3 Results

The results of the first half of the test are shown in Figure 7. A quick glance at this plot, shows that the 10 Gbps link was saturated by the two competing transfers. Between 23:00 and 0:00 (beginning of the plot) and 10:00 and 11:00 (end of the plot), only UDP traffic was sent across the network, running at a steady 5 Gbps. This effectively leaves only 5 Gbps of PhEDEx traffic. PhEDEx transfers start around 0:00 and quickly saturates the 10 Gbps. The seesaw look of this plot is due to the fact that PhEDEx has a delay between finishing one job and starting the next. This is due to various factors: pre/post validation, preparation of copyjobs or even time spent by the backend itself before actually launching a transfer. Because of these delays, the average rates reported by

PhEDEx will always be lower than the average rates of each individual transfer job. Nevertheless we get average transfer rates of 9.5 Gbps for the whole link and consequently 4.5 Gbps for PhEDEx transfers (10% penalty from gap between jobs).

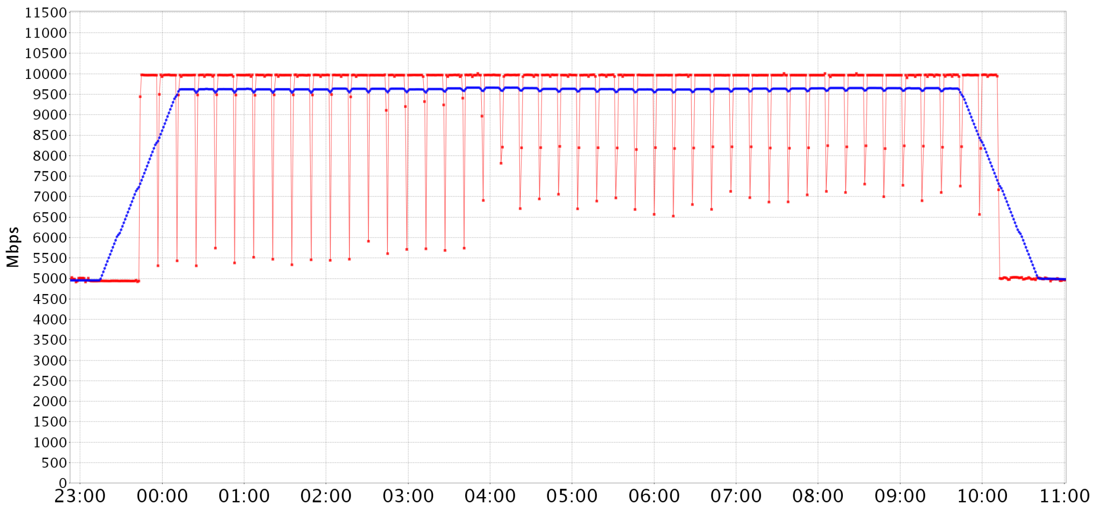

Figure 7: PhEDEx transfers on the shared path - competing with 5Gbps UDP traffic. As with the previous case, the red line represents network throughput in Mbps while the blue line represents a moving average of one hour of that network throughput.

Around 10:00 PhEDEx switches to using the dedicated link. The results of the second half of the test are shown in Figure 8.
Although most of the time we are able to saturate the 10Gbps link with PhEDEx traffic alone we sometimes see dips in transfer rates. These dips can be attributed to the storage not being able to sustain such high write rates.
This time, the average link rates drop from 9.5 Gbps to 8.5 Gbps, however actual PhEDEx transfers go up from 4.5 Gbps to 8.5 Gbps! The reason for this drop in average link rate is twofold:

- Transferring a job at higher speeds means that it will take less time for it to complete, hence more jobs will be completed in one hour. However, as we previously explained, there is a fixed overhead for starting each new job, which means that more delays will be introduced into the system.
- The storage system (using 1 LSI controller) sometimes cannot sustain write rates of 10 Gbps to disk.

Figure 9 and Figure 10 show the network activity for PhEDEx traffic alone on both links. In this plot we can observe that PhEDEx switches to using the new link without any interruption in service, doing it seamlessly and in a transparent way for other site-agents (switch occurs around 10:00).

7. Summary and future plans

PhEDEx has been very successful at managing data-flows on the WAN for the CMS collaboration. Nonetheless, it's architecture is based on design decisions that are no longer valid, and in order to continue to scale and perform for the future, it must evolve, taking advantage of new technologies.

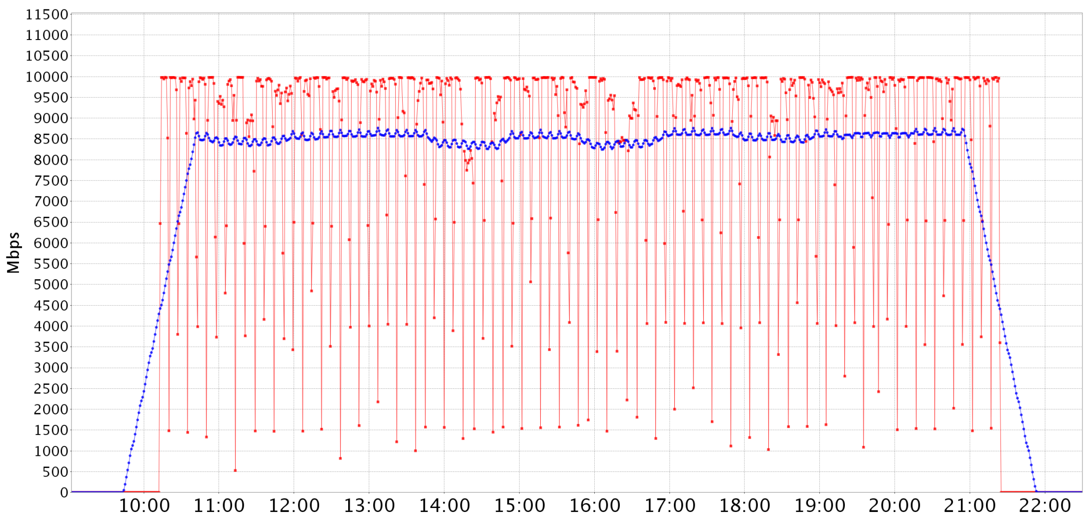

Figure 8: PhEDEx transfers on the dedicated path

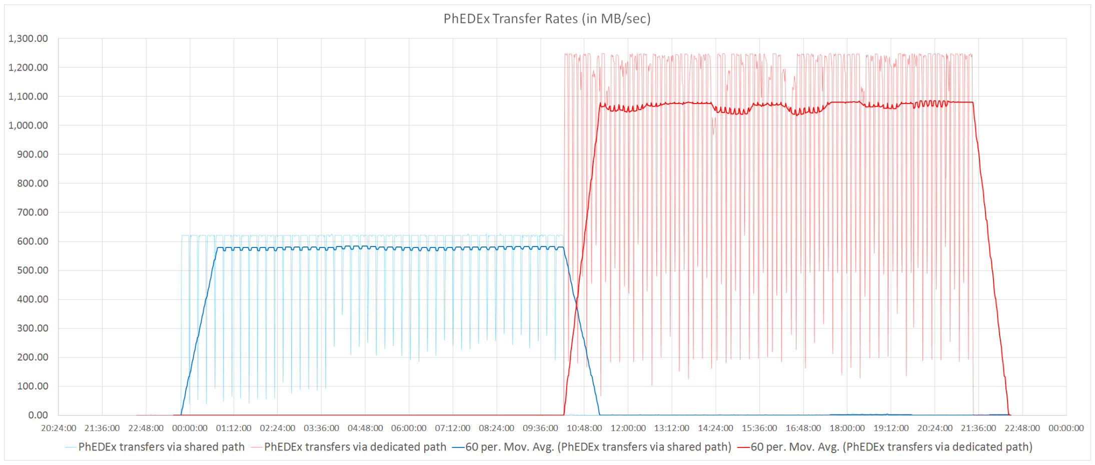

Figure 9: View of PhEDEx-only transfers on both the shared and dedicated path

Over the course of the past year, the ANSE project has made significant progress towards integrating network-awareness into PhEDEx. We have created and tested a prototype which integrates the use of virtual-circuits into PhEDEx at the level of the FileDownload agent, i.e. per destination-site. This is transparent for the other site-agents, with no modifications to the database schema and with all the logic contained in the FileDownload agent. Nor does it depend on network-circuits being infallible, the existing failure and retry mechanisms can cope with any underlying instability in the network.

Our immediate goal is to adopt a circuit management system and use its API to control and operate real circuits. This will be followed by extensive tests, on a potentially expanded ANSE testbed and ultimately in production data-transfers.

The next step is to integrate the use of virtual circuits at the FileRouter level. Here, we gain a global view of CMS-wide flows, which means we can make informed (optimal) decisions about which circuits need to be requested or not.

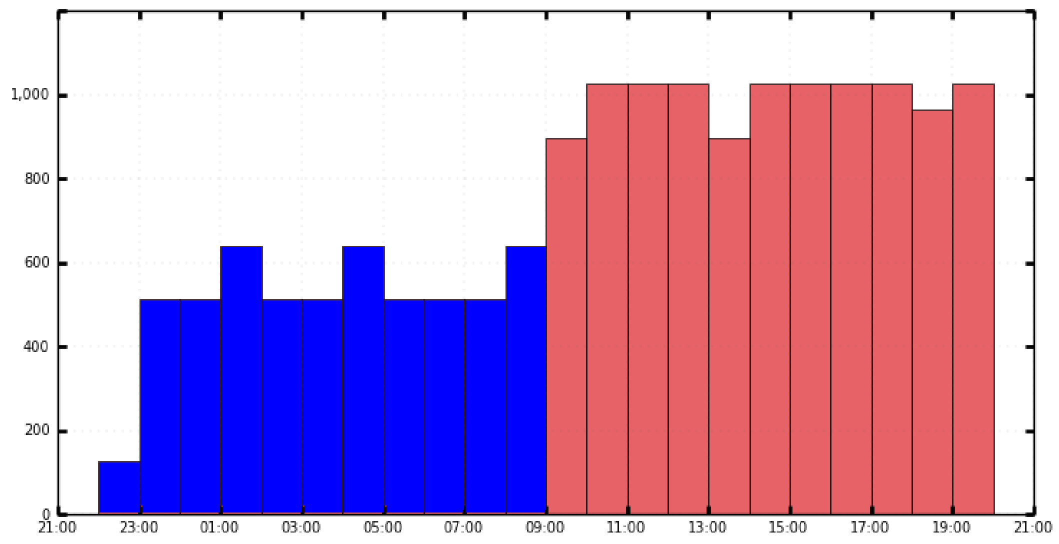

Figure 10: View of PhEDEx only transfers on both the shared and dedicated path

References

[1] Egeland R, Metson S and Wildish T 2008 Data transfer infrastructure for CMS data taking, *XII Advanced Computing and Analysis Techniques in Physics Research (Erice, Italy: Proceedings of Science)*

[2] The CMS Collaboration 2008 The CMS experiment at the CERN LHC *JINST* **3** *S08004*

[3] Alvarez Ayllon A, Kamil Simon M, Keeble O and Salichos M 2013 FTS3 - Robust, simplified and high-performance data movement service for WLCG *submitted to CHEP 2013*

[4] Legrand I, Newman H, Voicu R, Cirstoiu C, Grigoras C, Dobre C, Muraru A, Costan A, Dediu M and Stratan C 2009 MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems *Computer Physics Communications, Volume 180, Issue 12, December 2009, Pages 2472 - 2498*

[5] Fast Data Transfer (FDT) *http://fdt.cern.ch/*

[6] Eck C *et al.* 2005 LHC Computing Grid Technical Design Report *CERN-LHCC-2005-024*

[7] Bonacorsi D and Wildish T 2013 Challenging CMS Computing with Network-Aware Systems *submitted to CHEP 2013*

[8] LHCONE Point-to-Point Service Workshop, December 2012 *http://indico.cern.ch/event/215393/session/1/contribution/8/material/slides/1.pdf*

[9] MONitoring Agents using a Large Integrated Services Architecture (MonALISA) *http://monalisa.caltech.edu/*

[10] Campana S et.al. 2013 Deployment of a WLCG network monitoring infrastructure based on the perfSONAR-PS technology *submitted to CHEP 2013*

[11] Guok C, Robertson D, Thompson M, Lee J, Tierney B, Johnston W 2006: Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System *ICBCNS 2006*

[12] The Perl Object Environment (POE) *http://poe.perl.org/*