SciStream: Architecture and Toolkit for Data Streaming between Federated Science Instruments

Joaquin Chung
chungmiranda@uchicago.edu
The University of Chicago
Chicago, IL, USA
Argonne National Laboratory
Lemont, IL, USA

Wojciech Zacherek
zacherw@rose-hulman.edu
Rose-Hulman Institute of Technology
Terre Haute, IN, USA

AJ Wisniewski
austinw6@illinois.edu
University of Illinois at Urbana-Champaign
Champaign, IL, USA

Zhengchun Liu
zhengchun.liu@anl.gov
Argonne National Laboratory
Lemont, IL, USA
The University of Chicago
Chicago, IL, USA

Tekin Bicer
tbicer@anl.gov
Argonne National Laboratory
Lemont, IL, USA
The University of Chicago
Chicago, IL, USA

Raj Kettimuthu
kettimut@mcs.anl.gov
Argonne National Laboratory
Lemont, IL, USA
The University of Chicago
Chicago, IL, USA

Ian Foster
foster@anl.gov
Argonne National Laboratory
Lemont, IL, USA
The University of Chicago
Chicago, IL, USA

ABSTRACT

Modern scientific instruments, such as detectors at synchrotron light sources, generate data at such high rates that online processing is needed for data reduction, feature detection, experiment steering, and other purposes. The same high data rates also demand memory-to-memory streaming from instrument to remote computer, because local computational capacity is limited and data transmissions that engage the file system introduce unacceptable latencies. But efficient and secure memory-to-memory data streaming is challenging to realize in practice, due to a lack of direct external network connectivity for scientific instruments; and authentication and security requirements. In response, we propose here `SciStream`, a middlebox-based architecture with appropriate control protocols to enable efficient and secure memory-to-memory data streaming between producers and consumers that lack direct network connectivity. We describe the protocols that `SciStream` uses to establish authenticated and transparent connections between producers and consumers, and the extensive experiments that we have conducted to evaluate alternative implementation approaches for key `SciStream`'s components. Experiments on the Chameleon Cloud show that `SciStream` improves the throughput of a streaming pipeline by an order of magnitude compared to state-of-the-art data transfer methods, and adds only ∼4 μsec latency compared to an ideal scenario in which producers and consumers have direct external connectivity.

HPDC '22, June 27–July 01, 2022, Minneapolis, MN

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

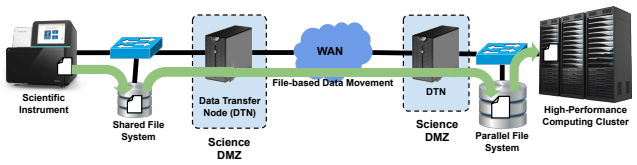
CCS CONCEPTS

• **Networks** → *Middle boxes / network appliances*; • **Security and privacy** → Access control.

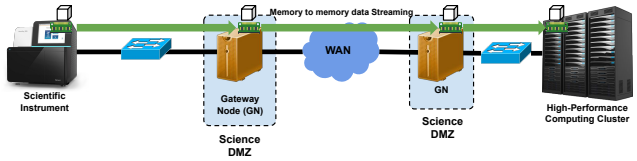
ACM Reference Format:
Joaquin Chung, Wojciech Zacherek, AJ Wisniewski, Zhengchun Liu, Tekin Bicer, Raj Kettimuthu, and Ian Foster. 2022. SciStream: Architecture and Toolkit for Data Streaming between Federated Science Instruments. In *Proceedings of The 31st International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC '22).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recent technological advances allow scientific instruments to generate data at rates that can exceed tens of gigabytes per second [25]. Rapid analysis of generated data, e.g., to permit real-time feedback and experiment steering, often requires computational capabilities greater than those available at the experimental facility—and/or the use of specialized computer systems [39]. Thus, the use of powerful remote computers (e.g., compute clusters, supercomputers, and in some circumstances, clouds: what we refer to here, without loss of generality, as high-performance computing or HPC) for analysis often becomes a necessity. The speed at which data can be moved between experiment and HPC then becomes important, particularly when rapid response is needed for experimental steering.



(a) State-of-the-art using file staging via intermediate file system



(b) Secure, memory-to-memory data streaming with intermediate hops.

Figure 1: Data movement approaches for remote data analysis in scientific environments

For remote data analysis, current practice, as shown in Figure 1(a), is often to stage data as files from experiment to HPC via an intermediate file system: typically, one accessible from dedicated data transfer nodes (DTNs) located in a Science DMZ [19] at the perimeter of the HPC facility's campus network. The incoming data from the wide area network (WAN) are received by services (e.g., Globus GridFTP [46]) running on the DTNs, which write the data to the file system; analysis codes running on HPC compute nodes then read the data from the file system. This use of the intermediate disk-based file system for data staging has the advantages of simplifying security and permitting separate optimization of wide area transfer and local data access [14], but can introduce significant performance degradation, even when using a high-performance parallel file system. In addition to the need to write and read the data, contention for the shared file system can result in significant overhead [27].

Real-time analysis of streaming data, a need highlighted in recent reports from federal agencies [1, 40, 51], requires (1) effective and reliable methods for acquisition of network and computing resources at a specific time for a specific period, (2) infrastructure for efficient and secure data streaming from scientific instruments to remote compute nodes, and (3) analysis of data streams at data generation rates so that timely decisions can be taken. Methods and tools exist for (1) [31] and (3) [11, 17], but have been lacking for (2). `SciStream` addresses this gap by providing the infrastructure (architecture, system software, etc.) necessary to enable memory-to-memory data streaming between scientific instruments and remote HPC.

We note that scientific applications differ from many other streaming applications in their high throughput requirements (a single application may require >10 Gbps) and the fact that the data producers (e.g., data acquisition applications on scientific instruments, simulations on supercomputers) and consumers (e.g., data analysis applications on HPC systems) are typically in different security domains (and thus require bridging of those domains). The `SciStream` system that we describe in this paper establishes the necessary bridging and end-to-end authentication between source and destination, while providing efficient memory-to-memory data streaming. `SciStream` focuses on addressing these challenges so that any data streaming tool that handles high volume and velocity can be readily used in scientific environments. In other words, `SciStream` is not a data streaming tool per se—rather, it is a tool to address the infrastructural challenges that otherwise hinder memory-to-memory data streaming in secure scientific environments.

Though NATs and other techniques provide mechanisms to punch holes through firewalls and middleboxes have been used to accelerate data streaming in other contexts [16], `SciStream` addresses unique needs that arise in scientific settings. These unique needs include a) the ability to handle transfers across multiple security domains with multiple user identities; b) the ability to support delegated authentication and authorization mechanism as data streaming is typically a part of larger scientific workflow; and c) the ability to decouple these sophisticated identity and access management functionality from the application so as to minimize the changes to the application as well as to reuse as much of the security architecture currently in place for file based communication. To meet these needs, `SciStream` will leverage federated identity systems such as InCommon [9] and identity and access management platforms such as Globus Auth [15], which is widely adopted for file transfers.

`SciStream` intends to utilize the Science DMZ [19] that connects internal and external networks, to host (on-demand) proxies on specialized gateway nodes (existing DTNs or software-defined switches can be used in place of gateway nodes where appropriate) for creating bridges between the internal instrument/HPC network and the external wide area network (WAN). The `SciStream` toolkit includes negotiation and control protocols that allow users to request resources from gateway nodes, as well as an implementation of the on-demand proxy based on a Layer-4 (TCP) proxy with a reconfigurable circular buffer.

This paper makes three major contributions:

- A middlebox-based architecture to enable (third-party initiated) data streaming between nodes (with no direct external network connection) in multiple security domains in scientific environments.
- A suite of protocols to establish authenticated and transparent connection between producer and consumer (in different security domains) via intermediate gateway nodes (aka middleboxes or proxies).
- A prototype and evaluation results that show `SciStream` improves the throughput of an streaming pipeline by an order of magnitude compared to file-system-based, state-of-the-art data transfer methods.

The rest of the paper is organized as follows. We provide application drivers on §2. We describe `SciStream`'s design considerations in §3, design in §4, and implementation in §5. Section 6 presents our evaluation results, we discuss challenges in §7, present related work in §8, and conclude in §9.

2 APPLICATION DRIVERS

Analysis of scientific data as they are being acquired is emerging as an important requirement in many science fields [23, 24]. Here we describe such examples to motivate the need in scientific environments for moving data with low latency and high bandwidth from memory of a scientific instrument to the memory of remote HPC.

2.1 Light source applications

Light sources are crucial tools for addressing grand challenge problems in the life sciences, energy, climate change, and information technology [10, 25]. For instance, the x-rays produced at the Advanced Photon Source (APS) enable scientists to study internal morphology of materials and samples with very high spatial (atomic and molecular scale) and temporal resolutions (<100 ps). These experiments can generate massive amounts of burst data. For example, tomographic imaging stations can collect 1,500 projections (images each with 2048×2448 pixels) in 9 seconds with an Oryx detector, generating data at >8 Gbps. These experiments are performed to observe time-dependent phenomena that might spread over long time periods, resulting in very large datasets. Real-time streaming and analysis of these experimental data enable scientists (or the control software) to 1) make timely decisions that, in turn, can significantly accelerate the scientific progress of lengthy experiments, 2) do smart experimentation, such as changing the parameters interactively or finalizing experimentation with only sufficient data, and therefore can enhance the overall efficiency of end-to-end scientific workflow, as demonstrated in [11, 18]. It has been demonstrated that such a capability can double data acquisition and analysis speed [11, 45].

2.2 Cosmology workflows

To understand the Universe, cosmologists use large telescopes to conduct observational surveys [50]. These surveys are becoming increasingly complex as telescopes reach deeper into space, mapping out the distributions of galaxies at farther distances. Cosmological simulations that track the detailed evolution of structure in the Universe over time are essential for interpreting these surveys. In order to achieve high-quality simulations, high temporal and spatial resolution are critical. Current (and next-generation) supercomputers (will) allow them to attain high spatial resolution in large cosmological volumes by simulating trillions of tracer particles. For example, a trillion-particle simulation with the Hardware/Hybrid Accelerated Cosmology Code (HACC) code [32] can generate huge amount of data (several PBs of data in each run) that they store only one in every five or ten snapshots. To overcome the storage issue, a pipelined data analysis has been proposed in which the files are moved to a HPC for analysis at the end of each snapshot during the simulation [39]. To analyze these snapshots as they are produced, the data needs to be streamed directly from the memory of the compute nodes that perform the simulation to the memory of the compute nodes that perform the analysis.

3 DESIGN CONSIDERATIONS

The considerable success of the Science DMZ architecture is due in large part to its clean separation of local and wide area concerns. It allows for the creation of high-throughput, friction-free wide area paths to data transfer nodes (DTNs) that are placed at the perimeter of the institutional network. The local communication between DTNs and the scientific instruments or computing systems typically happens through a file system that is mounted on both the DTNs and instruments. To support scientific streaming applications, we want to design an architecture that utilizes the science DMZ but bypasses the file system and stream data directly to/from the memory of instruments. Such an architecture needs to consider

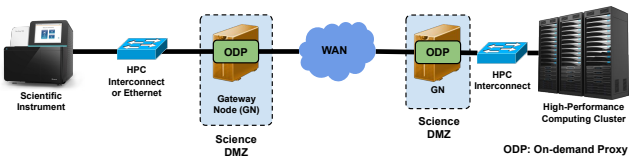


Figure 2: `SciStream` architecture for supporting efficient and secure data streaming from data producer's memory to remote data consumer's memory using gateway nodes and on-demand proxies (ODP).

the unique requirements that arise in scientific environments (as described in the previous section). We consider a middlebox-based architecture that creates *on-demand proxies* between an instrument's LAN and the WAN as shown in Figure 2. We use on-demand proxies because infrastructure requirements vary from one experiment to the other: (1) instruments and HPC resources are reserved for each individual experiment and (2) IP addressing schemes are not known until resources are provisioned. Thus we must provide a solution that can be instantiated for each new experiment request, and removed upon termination. We next describe high-level design considerations for the `SciStream` architecture.

3.1 Third-party streaming

One common data transfer pattern is streaming data between two remote computers (or instruments), initiated by a "third-party" user or application. This type of streaming allows users to initiate, manage, and monitor producer-to-consumer data streaming from anywhere. Further, it provides flexibility for integrating data streaming with commonly used scientific workflow engines such as Galaxy [29], Swift/T [57], Kepler [5], Panorama [20], and Pegasus [21], and their transparent execution on remote resources. Third-party streaming involves distinct control and data channels, with the control channel used for sending protocol messages between system components (e.g., to authenticate and authorize users, and to request initiation of streaming) and the data channel providing the link between producer and consumer processes for streaming data. The separation of the control and data channels improves the extensibility of the data channel in a transparent way and reduces the impact of a data channel crash. We want `SciStream` to support third-party streaming.

3.2 Secure streaming

Securing the channels is important especially because `SciStream` supports streaming data between geographically distributed facilities with each facility in a different administrative (and thus different security) domain. There are standard process for authenticating and authorizing users and (control and data) connections for third-party file transfers by using a campus identity through InCommon federation [9] as employed, for example, for Globus file transfers [15]. Such standard process is not sufficient for memory-to-memory data streaming with intermediate hops.

End-to-end data streaming in `SciStream` involves four control connections and three sets of data connections: more than used in standard services supported on campuses (e.g., Globus uses two control and one set of data connections for each transfer request). But `SciStream` employs four more internal connections - a control

connection between the application and SciStream control process and a data connection between the application and SciStream data process at both the producer and consumer side, more details in §4.4. The authentication and authorization of the connections internal to a facility require a different approach. SciStream requires a control connection between the application and SciStream control process and a data connection between the application and SciStream data process at both the producer and consumer side. SciStream should authenticate all control and data channels by default. It should provide the ability to encrypt communication on control and data channel on a need basis. The authentication and authorization approaches should be designed to avoid changes to streaming libraries and minimizes code changes to data producer and consumer applications. As described in §4.2, SciStream uses a shared key-based authentication for control connections and source-based authentication for data connections.

3.3 General and Transparent Streaming

Each end-to-end data stream must traverse three network segments—data producer to local proxy, local proxy to remote proxy, and remote proxy to consumer. We consider the following three approaches to bridging these network segments.

Network layer (L3) network address translation (NAT) or L3 tunnels: A NAT may be the easiest solution to deploy but is difficult to scale, as it requires load balancers or other supporting technologies to distribute traffic among several gateways. Furthermore, it is a known issue that when NAT is deployed on software, it may have a negative impact on throughtput performance. L3 tunnels avoid the need to maintain stateful connections, but may introduce larger header overhead to each IP packet.

Application layer (L7) proxies: L7 proxies would arguably be the least complicated in terms of connection management, but the gateway then needs to run multiple proxies, to support both standard streaming libraries (e.g., ZeroMQ, RabbitMQ, nanosec, DASH) and custom, application-specific approaches.

Transport layer (L4) proxies: An L4 proxy is agnostic to streaming libraries, but complicates connection management as we need to ensure that the number of transport layer connections for a given streaming request is the same in each of the three network segments listed above.

We want to make SciStream agnostic of streaming application and avoid HPC resource reconfiguration, while at the same time creating a flexible design that can support application-specific approaches. In addition, we want to make the chain of connections transparent to the applications as much as possible. In §5.2 we evaluate these approaches and provide our implementation of an on-demand proxy for SciStream.

3.4 Provisioned vs. Best-effort Resources

Ideally, all network and compute resources needed to stream and process data at generation rate would always already be acquired and ready for use when streaming is initiated on SciStream, making SciStream design less complicated. In fact, advance reservation of HPC resources is supported by many facilities, and some scientific computing facilities even have a real-time queue serving on-demand requests. Some research and education networks (e.g., ESnet, Internet2) support bandwidth reservation [31, 37]; other efforts extend bandwidth reservations to the last mile inside the campus [12, 36, 59]. Nevertheless, most applications still wait on batch queues and use best-effort network transport. We want to make sure that SciStream supports both scenarios. To support the scenario where HPC and network resources are provisioned on-demand, SciStream allows for provisioning the necessary gateway (proxy) resources. To support the best-effort scenario, the SciStream control design (see §4.4) allows (for a given streaming request) the third-party client, producer, and consumer to connect independently, whenever they become ready.

4 SCISTREAM DESIGN

We architect SciStream to meet the design considerations of §3. The overarching goal is that users be able to request resources for their streaming analyses; as each scientific facility is an independent administrative domain, we design SciStream as a federated system in which participating facilities make their resources available through programmatic interfaces. We describe in the following deployment options for gateway nodes (GNs) (§4.1), the software components (§4.2), negotiation (§4.3), and control (§4.4) protocols that compose SciStream.

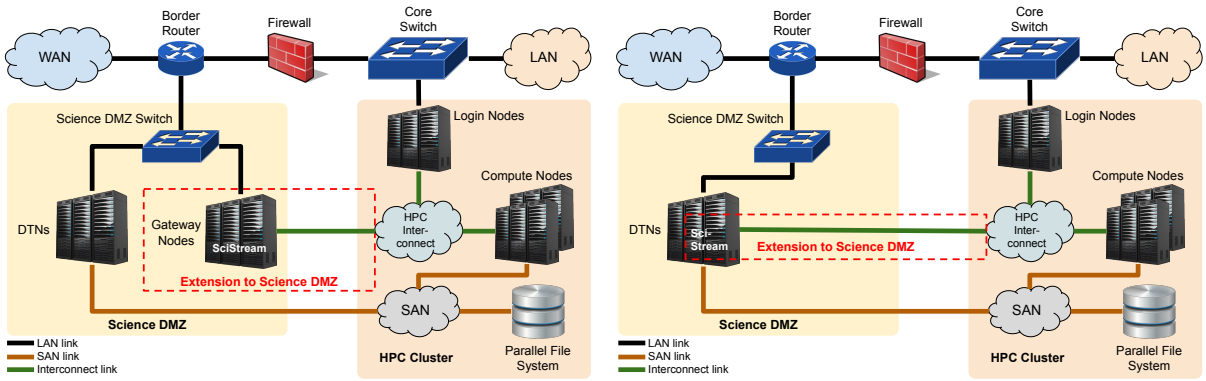
4.1 Gateway Node deployment options

To switch between the WAN and HPC interconnect, gateway nodes (GNs) may need to use IP over HPC interconnect and in some cases HPC interconnect stack over Ethernet (e.g., for RoCE). GNs should also be compatible with newer Ethernet-compatible interconnects for Exascale computers such as Slingshot and Omni-Path. Furthermore, a wide range of scientific institutions may deploy SciStream, and what works for a large scientific facility may not be best for a university campus. A dedicated set of GNs will provide better performance than repurposing existing DTNs to host SciStream at the Science DMZ, since file-based data transfers can introduce additional contention on the resources [43, 44, 47]. Dedicated GNs will also be easier to maintain and manage. However, repurposing existing DTNs may be more cost-effective for some institutions. Figure 3 illustrates two options for deploying gateway nodes on a Science DMZ. SciStream is designed to work in both scenarios.

4.2 Software components

We realize SciStream using three software components:

SciStream User Client (S2UC) is software with which the end user and/or workflow engines/tools acting on behalf of the user interact to orchestrate end-to-end data streaming. A request to S2UC must provide (at least) the location of producer and consumer, the number of streaming channels, and the required bandwidth in bps (either per streaming channel or aggregate). S2UC allows the user to fetch short-term proxy credentials through integration with federated identity management and certificate generation systems. It authenticates (on behalf of the user) with SciStream Control Server (see next) on the GNs and orchestrates the creation of end-to-end data channel. It also generates shared keys that the user must pass to producer and consumer applications in order for the applications to communicate



(a) Deploying gateway nodes (GNs)

(b) Repurposing DTNs

Figure 3: Design options for deploying `SciStream` on a Science DMZ

with control elements of `SciStream`. Once data channel connections are established, the `S2UC` provides monitoring information (e.g., streaming workflow status and throughput) to the user.

SciStream Control Server (`S2CS`) runs on a gateway node. It interacts with `S2UC`, data producer / consumer and `S2DS` (see next). It authenticates the user and producer / consumer applications (using pre-shared keys) and creates a mapping between them. It manages gateway node resources and `SciStream` Data Servers (`S2DS`), including initiating, monitoring, and terminating `S2DS` processes.

SciStream Data Server (`S2DS`) runs on gateway nodes. It acts as a proxy between the internal network (LAN or HPC interconnect) and the external WAN and authenticates external connections between remote facilities using proxy certificates that the user passes through `S2UC` and the internal connections (with the producer/consumer application) using source-based authentication methods.

4.3 Negotiation Protocol

Before a user can issue a request to `SciStream`, producer and consumer systems must agree on the number of streaming channels and bandwidth required to sustain the streaming data analysis. After receiving a user request, `S2UC` will generate a unique ID and pass the request to both producer and consumer `S2CS`. Each `S2CS` will check available resources and respond by either accepting the request, declining it, or offering a new allocation of resources. This new allocation should honor the requested bandwidth but can change the number of channels. It is possible that `S2DS` at either end need to use more than the requested number of channels to satisfy the bandwidth requirement. In such a scenario, `S2CS` will propose a greater number of channels than the requested by the user. `S2CS` will split the bandwidth equally among the number of channels. After receiving all offers, `S2UC` will compute the maximum of the number of requested channels by the user and the offers from both producer and consumer `S2CS`. The user can accept or decline the offer from `S2UC`. Upon acceptance, `S2UC` will initiate the control protocol as described in §4.4.

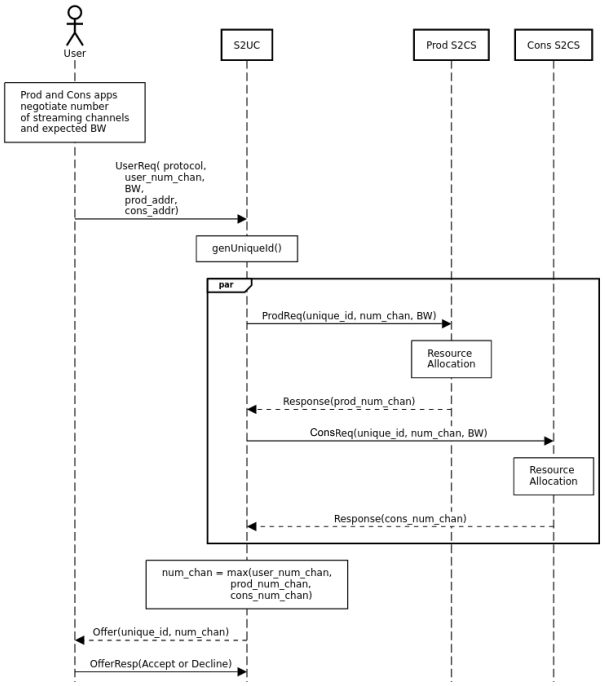


Figure 4: `SciStream` negotiation protocol

4.4 Control Protocol

Figure 5 outlines the control flow used by `SciStream` to establish streaming data channels between two facilities.

Here we assume that the data producer is `ProdAPP` running at Facility 1 (`F1`) on the left side and the data consumer is `ConsAPP` running at Facility 2 (`F2`) on the right. Before initiating an end-to-end data streaming setup request with `S2UC`, the user should have requested appropriate experimental / computing resources from both facilities. The steps to form authenticated and transparent end-to-end data streaming connections between `ProdAPP` and `ConsAPP` (labeled with numbers in Figure 5) are shown below:

① **Send request.** The user establishes a connection with `S2UC` and submits a new request, which includes information such

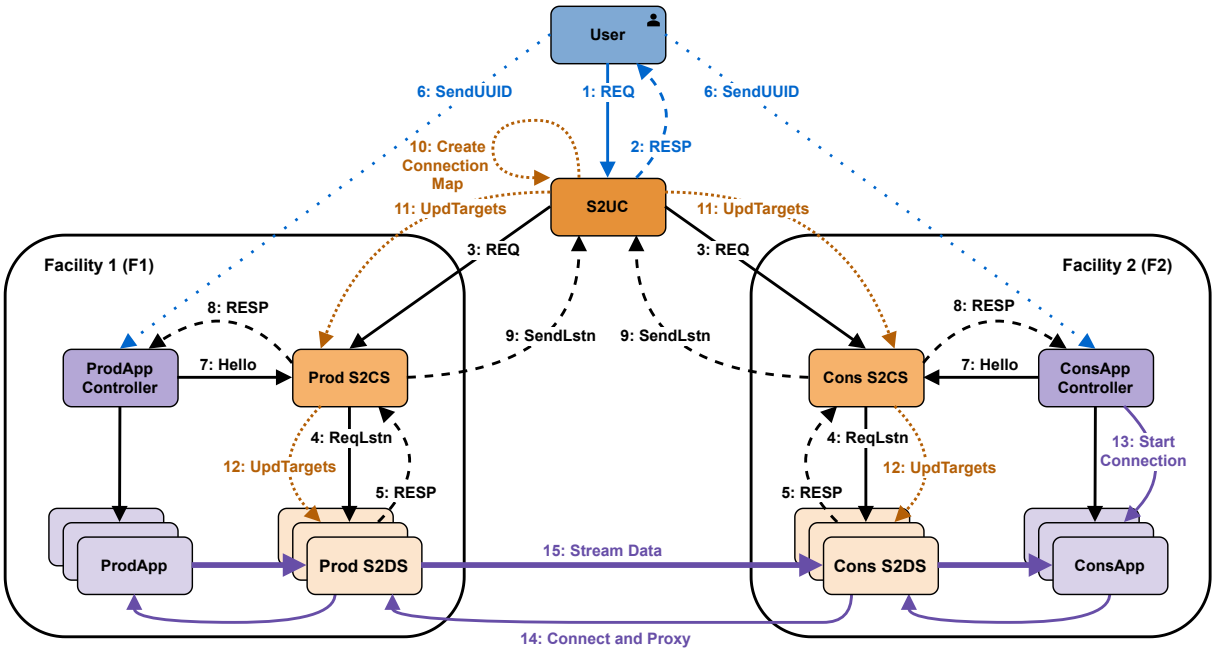


Figure 5: `SciStream` control protocol.

as connection details regarding the producer and consumer facilities and the required bandwidth. For fully provisioned scenarios, the request should also include a start and end time. The user authenticates with the facilities via `S2UC` so that `S2UC` obtains X.509 certificates [56], which `S2UC` can use to act on the user's behalf to authenticate and interact with `S2CS`s at the producer and consumer. The negotiation protocol of §4.3 happens within this step, but we omit it for simplicity.

② **Acknowledge request.** The `S2UC` issues a response to the user, including connection information about the two facilities as well as the unique id assigned to their request. The unique id will be used later by other components, since multiple incoming requests may be handled at once; it can also be used to manually release the request.

③ **Inform facilities.** The `S2UC` relays the request information to the `S2CS` located at each facility. Each `S2CS` then checks that enough resources are available to handle the request; if not, the `S2CS` raises an error, which is sent to the `S2UC` so that the user can be notified.

④ **Reserve resources.** If enough resources are available, the `S2CS` instantiates new `S2DS` instances and passes relevant request information to each. Note that these instances need not be hosted on the same machine.

⑤ **Return listening ports.** The `S2DS` allocates a port for data to be sent through and will return this information back to the `S2CS`. The port is chosen automatically by the operating system and can be reused upon termination of the `S2DS`.

⑥ **Send unique-id.** The user needs to give the unique id of the request to the `ProdAPP` and `ConsAPP` controllers so that they know which request is being set up. Note that this communication is out-of-band of the `SciStream` control protocol but still necessary (i.e., user does not communicate with `S2UC` in this step).

⑦ **Relay controller information.** Both application controllers establish communication with the `S2CS`. In particular, the `ProdAPP` controller includes the ports `ProdAPP` listens on, as well as other connection and resource information. Note that steps ⑥ and ⑦ are both performed irrespective of when the user submits the initial request.

⑧ **Acknowledge controllers.** Each `S2CS` issues a response to the respective application controllers. In particular, the `ConsAPP` controller includes the ports that the consumer `S2DS` instances are listening on.

⑨ **Centralize information.** Each `S2CS` then relays connection information to the `S2UC`. This step is needed in order to bridge the connection between producer and consumer `S2DS` over the WAN, but other connection information is also included to offer transparency to the user.

⑩ **Create connection map.** The `S2UC` creates a connection map defining each data movement path from `ProdAPP` to `ConsAPP`. These paths are based upon available `S2DS` bandwidths, in order to evenly distribute requested resources.

⑪ **Distribute connection map.** The `S2UC` sends the connection map to each `S2CS` so that it can update its internal structures and propagate the new information to other components.

⑫ **Update connections.** Each `S2CS` relays the update connection information to the respective `S2DS` instances. In particular, the producer `S2DS` are informed of which consumer `S2DS` they are to connect to, and send data to, over the WAN.

⑬ **Start connection.** The `ConsAPP` controller signals to `ConsAPP` that the setup is complete and data is ready to be streamed.

(14) **Connect and proxy.** Each component in the data movement path connects to the component from which it is to receive data, so as to ensure that data are sent to the correct place and will be received.

(15) **Stream data.** Once the `ProdAPP` launches the request associated with this request's unique-id and begins collecting data, it transmits the data to the established producer `S2DS` instances, which then forward the data to the consumer `S2DS`, and finally to the `ConsAPP`.

To release a request, the user again submits a request to the `S2UC` and includes the desired unique-id of the request to be released. The `S2UC` will then relay the information to each `S2CS` so that the appropriate resources can be released. This includes terminating `S2DS` instances and releasing any ports associated with the request. Finally, each `S2CS` will send a response to the `S2UC` so that it can clean up any of its own resources allocated to the request.

5 IMPLEMENTATION

We describe the `SciStream` control protocol implementation (§5.1) and evaluate `S2DS` implementation approaches (§5.2).

5.1 Control Protocol Implementation

The prototypes for the `S2UC`, `S2CS`, and the application controllers are implemented in Python. The `S2UC` and `S2CS` are designed as finite state machines to help with organization and for smoother error handling. The `S2UC` also uses multithreading to handle multiple incoming requests at once by assigning a dedicated worker thread to each request. For all the components, any allocated resources for a request are automatically cleaned up upon user request or error. Additionally, all resources allocated by a particular component are released upon program termination.

For preliminary benchmarks, the prototype architecture was tested locally using a virtual machine running Ubuntu 20.10 with 4GB of RAM. Initiating a data streaming request as seen in Figure 5 takes ~0.12 s per request, under the assumption that each step is executed consecutively. Similar performance was achieved regardless of there being tens of other requests already active. Releasing a request took on average only 0.003 s. The memory footprint for each component was also negligible, with values ranging from 10 MB for the `S2UC` to 9 MB per `S2CS` process.

5.2 `S2DS` Implementation Approaches

As mentioned in §3.3, `SciStream`'s `S2DS` can be implemented as NAT, an L3 tunnel, an L4 proxy, or an L7 proxy. To maintain `SciStream` application-agnostic, we would like to avoid the option of using L7 proxies as they require the GN to support user's libraries. As previously mentioned, NAT could be difficult to scale as we may need load balancers to steer the traffic to the right GN and if implemented in software it could hurt performance. Furthermore, L3 tunnels would not work on all scenarios. Consider for example the case in which only one end of the streaming pipeline lacks external connectivity. This means that one end of the tunnel should be deployed/configured inside the user's resources or application, breaking the transparency requirement. Moreover, any L3-based solution may require changes to the campus or Science DMZ network in terms of routing. Thus, L4 proxies are the most general solution

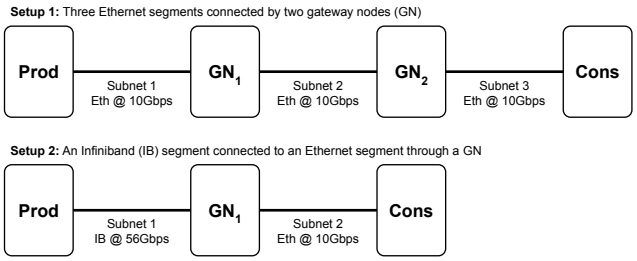


Figure 6: Experimental setup on Chameleon Cloud

to support memory-to-memory streaming analysis in an application-agnostic and transparent way. Nevertheless, in the rest of this section we present our evaluation of `S2DS`'s implementation approaches: NAT for L3, TCP proxy for L4, and ZMQ Pub/Sub proxy for L7.

We evaluate the performance of `S2DS` implementation approaches in terms of goodput (or the throughput measured at the consumer application level) for two scenarios: bridging between two Ethernet-based networks and between an Infiniband (IB) interconnect and an Ethernet network. We also evaluate how much latency `S2DS` adds to an streaming pipeline and how the presence of `S2DS` affects the inter-message delay variation (or jitter) of a scientific streaming pipeline. For our experiments, we assume that all resources have been provisioned by the negotiation (§4.3) and control protocols (§4.4). We conduct our evaluation on a LAN to avoid fluctuations and transients that could happen on a WAN environment. This ensures our results are reproducible and allows us to create controlled WAN conditions using Linux's `tc` and `netem` tools.

5.2.1 Experimental Setup. We conduct our experiments using Chameleon, a reconfigurable, open experimental platform for computer science research [38]. Chameleon has two sites connected by a 100 Gbps WAN: one at the University of Chicago (UC) in Chicago, IL, and the other at the Texas Advanced Computing Center (TACC) in Austin, TX. We perform our experiments at TACC as it has nodes with IB support. Each bare metal node has 48 cores, 128 GB of RAM, and two 10 Gigabit Ethernet network interface cards (NICs); the nodes with IB support have one 10 Gigabit Ethernet NIC and one 56 Gbps 4X FDR IB adapter card. Each node runs Linux Ubuntu 20.04 as the OS, Python 3.8 for emulating the data generation and consumption processes, and ZeroMQ [33] to implement a Pub/Sub streaming application pipeline in Python. For the IB nodes, we installed CentOS 7.9 as the OS.

Figure 6 shows our two experimental setups. **Setup 1** is composed of four bare metal servers and three Ethernet segments. The two edge nodes act as producer and consumer, while the inner nodes act as gateway nodes (GN). This setup can be reconfigured to use only three nodes in which only the middle node functions a GN. **Setup 2** is composed of three bare metal servers, an IB segment, and a Ethernet segment. Again, the edge nodes work as producer and consumer, while the inner node works as GN.

5.2.2 Methodology. To evaluate the effect of inserting `S2DS` on a scientific streaming analysis pipeline, we compare the application goodput of `SciStream` with the ideal scenario in which producer and consumer have direct connectivity over the network. We also

evaluate latency added by S2DS and how the presence S2DS affects the inter-message delay variation of a scientific streaming analysis pipeline. Our prototype of the S2DS is a transport layer (L4) proxy with a reconfigurable circular buffer. We hypothesize that the presence of this buffer will help amortize the jitter produced by WAN scenarios. For these experiments we focus on a TCP implementation, although S2DS could be implemented as a UDP proxy as well. We further compare this implementation with a L7 proxy that uses the Python implementation of ZMQ Pub/Sub proxy and L3 NAT using iptables. We repeat each experiment 10 times and present average values.

We conduct our evaluation on Chameleon's TACC LAN with no cross-traffic in the network. For the goodput evaluation, the producer generates samples as fast as possible (i.e., no sampling delay). In each experiment, we transfer 10 GB for a fixed sample size, which we vary across experiments from 512 bytes to 1 MB in power-of-two increments. For each experiment, we compute goodput as the size of the received dataset ($num\_samples \times sample\_size$) divided by the elapsed time between the arrival of the first sample and the arrival of the STOP message ($\Delta\ t = t_{stop} - t_{first\_sample}$). We configure the circular buffer of the L4 S2DS to be 10 MB for all experiments, corresponding to 10 messages for the largest sample size. We use default settings for L7 proxy.

For latency and inter-message delay variation experiments the producer generates 100,000 samples with a sampling delay of 1 ms. The round-trip time (RTT) of each Ethernet segment is 136 μs (measured with ICMP ping), so the accumulated RTT from producer to consumer, passing through two GNs is 408 μs. We compute inter-message delay by subtracting the arrival time of a message to the consumer minus the arrival time of the previous message ($t_{i+1} - t_i$) and store the result in an array. At the end of the experiment we compute the average and standard deviation of our measurements to obtain the latency and inter-message delay variation, respectively.

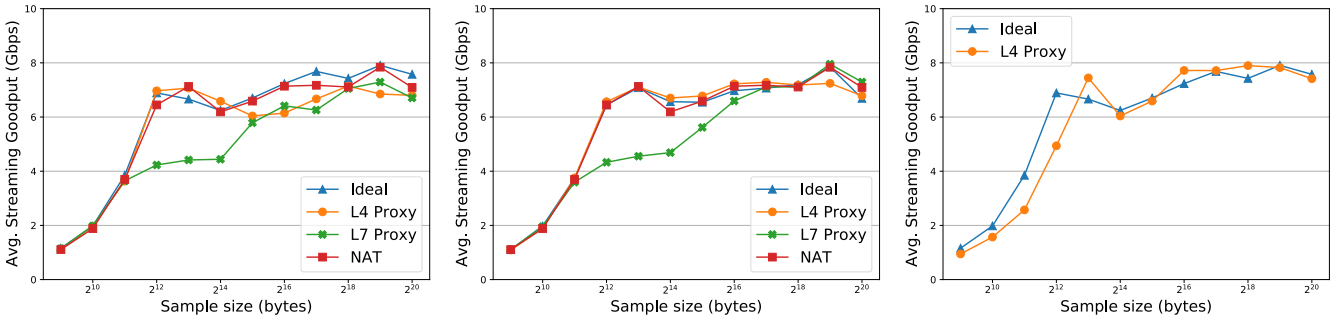
5.2.3 S2DS Goodput Evaluation. Our experimental results show that in the absence of cross-traffic and in the presence of a single instance of S2DS, both NAT and the L4 proxy closely follow the goodput of the ideal scenario in LAN environments (see Figure 7(a)). The L7 proxy shows a bad performance for small to medium sample sizes, but after 32 KB it starts catching up with the baseline and L4 proxy. We repeat this experiment with two GNs between producer and consumer (see Figure 6, **Setup 1**). For the proxies we just run an instance of the L4/L7 proxy on each GN, while for NAT we configure the appropriate iptables on each GN. Figure 7(b) shows that the goodput behavior is consistent with the one S2DS experiment. For completeness, we evaluate the IB scenario (see Figure 7(c)) using IP-over-IB (IPoIB).

Note that we optimized neither the OS nor our Pub/Sub application for IPoIB, and only collect measurements for the L4 proxy as this is best candidate for S2DS. For this experiment, we use the all-Ethernet ideal scenario as comparison, because in real scenarios the WAN connecting producer and consumer will always be Ethernet based. We observe that the results are consistent with the previous experiments and L4 S2DS closely follows the goodput of the ideal scenario. We conclude that SciStream does not add significant overhead to a scientific streaming analysis pipeline, although we recommend to avoid L7 proxies for small-sized messages. We note that, L4 proxy enables us to has a buffer to absorb the fluctuation of WAN congestion (will be validated in §5.2.5). As discussed in §3.2, L4 proxy also enables transparent secure streaming over WAN without adding complexity to producer and consumer applications.

5.2.4 Latency and Inter-message Delay Variation. Streaming analysis pipelines are highly sensitive to latency. Increased application latency may result in a workflow missing a phenomenon and thus not steering an experiment at the right time. We measure SciStream's added latency in the presence of two instances of S2DS over a LAN environment. For these experiments, we generate samples at 1 kHz on the producer side and measure the inter-message delay at the consumer side for sample sizes from 512 bytes to 1 MB. Figure 8(a) shows the average added latency for the ideal scenario (no SciStream) and our three S2DS implementations: NAT, L4 proxy, and L7 proxy. We compute the added latency as the average inter-message delay for each run minus 1 ms (sample generation period); this encompasses the network propagation delay and the added delay of the two S2DS instances. We observe that the added latency is negligible. On average, the NAT adds 3.65 μs, the L4 proxy adds 4.23 μs, and the L7 proxy adds 4.30 μs.

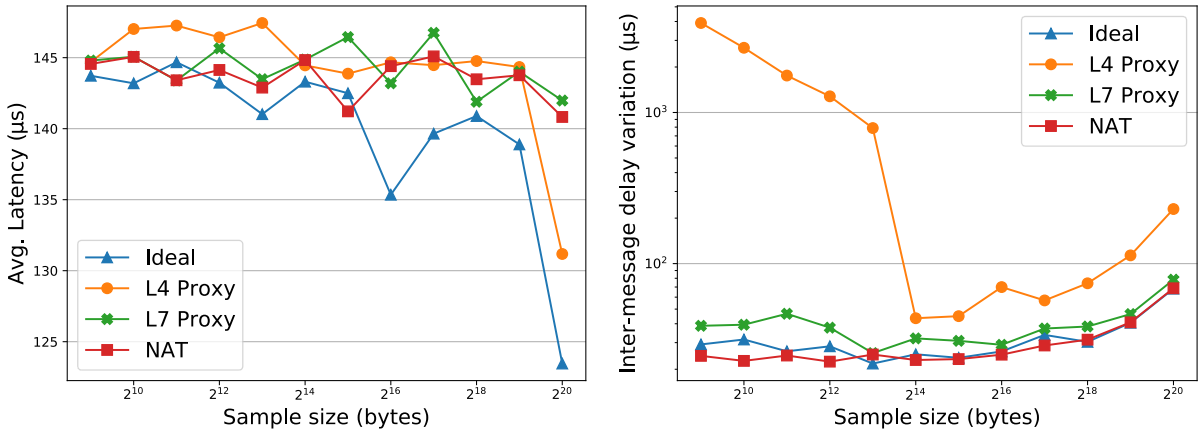
Jitter is an important metric for defining the QoS of real time streaming applications over IP-based networks. It is defined as the variantion of the inter-packet delay, and can severely affect VoIP or video streaming applications if it crosses certain threshold (e.g., 150 ms for VoIP). Analogous to network jitter, the inter-message delay variation can affect scientific streaming analysis pipelines. For instance, if samples generated at a remote instrument do not arrive on time to the supercomputer, precious computation time may be wasted. Figure 8(b) shows the inter-message delay variation experiment results. Again, we vary the message size from 512 bytes to 1 MB in power-of-two increments. We observe that the inter-message delay variation of the L4 proxy is almost two orders of magnitude larger than the ideal scenario for small message sizes and ~30 μs larger than the ideal scenario for medium-to-large message sizes. NAT and L7 proxy remain pretty close (±10 μs) to the ideal scenario. Nevertheless, when we look at absolute numbers, the worst case scenario for L4 proxy is ~4 ms, which is negligible when we consider that the typical WAN RTT for remote analysis is an order of magnitude larger.

5.2.5 Emulated WAN Evaluation. We study the effect of S2DS on WAN scenarios by emulating long delays between producer and consumer GNs using Linux's tc and netem tools. For these experiments we use a fixed message size of 256 KB as this provides one of the best goodput results on previous experiments. We vary the RTT from 10 ms to 100 ms in 10 ms increments. We measure both goodput (see Figure 9(a)) and inter-message delay variation (see Figure 9(b)). The value of zero in the X-axis of the plots represent the results observed on the WAN environment when no synthetic delay is added. The goodput results show that for RTTs between 10 ms and 50 ms, both L4 and L7 proxies perform better that the ideal scenario and NAT. We attribute this performance gain to the presence of buffering capabilities on the proxies, something that both the ideal scenario and NAT lack. For the inter-message delay variation results, the L4 proxy has a variation 400 μs smaller that the ideal scenario, demonstrating that the presence of the circular buffer



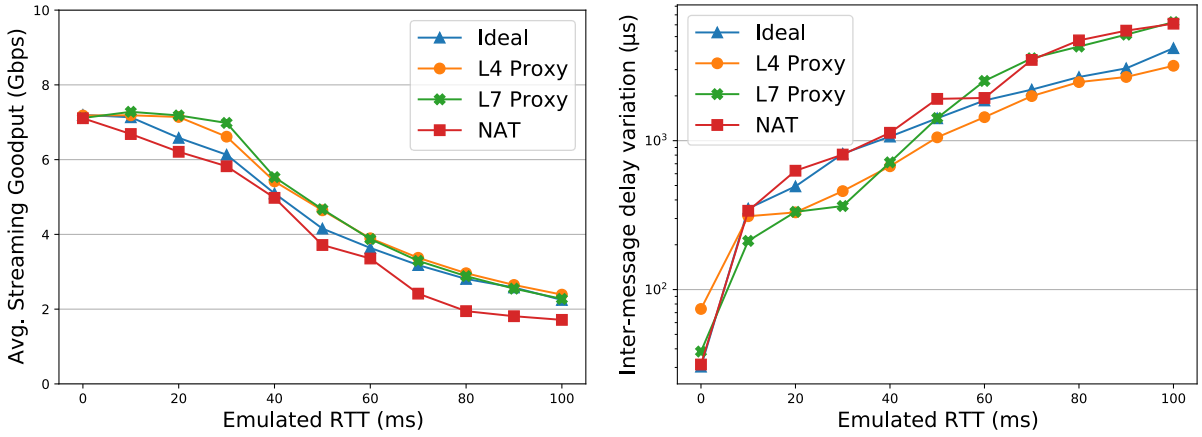
(a) One S2DS between two Eth segments (b) Two S2DS across three Eth segments (c) One S2DS between IB and Eth segments

Figure 7: Streaming goodput performance evaluation in the presence of SciStream over LAN environments.



(a) Added latency (b) Inter-message delay variation

Figure 8: Added latency and inter-message delay variance of a scientific streaming pipeline in the presence of two instances of S2DS. SciStream adds at most 4 ms of latency with max. ~30 μs inter-message delay variance.



(a) Goodput as a function of RTT (b) Inter-message delay variance as a function of RTT

Figure 9: S2DS evaluation on an emulated WAN with a streaming application using a message size of 256 KB.

helps amortize the jitter on the WAN, similar to the effectiveness of jitter buffer in VoIP scenarios [52, 53].

6 SCISTREAM EVALUATION

To demonstrate the benefits of SciStream, we compare a streaming application running on a state-of-the-art scientific environment configuration (i.e., DTNs and file-based data movement methods) with the same application running over a SciStream environment (i.e., GNs and memory-to-memory data streaming). As mentioned in §1, the state-of-the-practice for data production, consumption, and analysis in scientific environments involves generating all the samples at the producer before starting a file-based data transfer to the consumer. To enable remote streaming over the state-of-the-art environment, we had to modify the workflow so that a producer starts transferring samples as soon as they are completely written to disk. We conduct these experiments on the Chameleon Cloud testbed (see §5.2.1) using our **Setup 1** configuration shown in Figure 6. As Chameleon nodes do not have high-performance disks, we mount the disk to RAM using the tmpfs tool. For SciStream we use our implementation of L4 proxy with circular buffer.

In Figure 10, we emulate a set of synchrotron light source workflows to show the performance differences between variety of data transfer methods. Specifically, we emulate data acquisition of X-ray images with fixed sizes at $1/T$ Hz. We conduct experiments with three sample sizes (small: 1 MB, medium: 4 MB, and large: 10 MB) and three time gaps between each sample production ($T = \{1, 10, 100\}$ ms; i.e., 1000 Hz, 100 Hz, and 10 Hz.) For SciStream the buffer size is always 10 times the sample size. We produce 1000 samples in each experiment and run each experiment five times. For each run we measure the average completion time, i.e., the time from the generation of the first sample at the producer until the consumer has received the last sample. We divide the dataset size ($1000 \times sample\_size$) by the completion time to obtain the average goodput of each approach.

We present the average goodput for both approaches in Fig. 10, showing the maximum achievable throughput with 100 ms, 10 ms, and 1 ms gaps between samples, as well as the theoretical maximum achievable value for each gap value. We observe that the state-of-the-art method is always below the theoretical maximum for a 100 ms gap with any sample size. This demonstrate that read/write operations to the file systems introduce significant delay even when the file system is mounted in memory. On the contrary, SciStream outperforms the state-of-the-art method by an order of magnitude for 1 MB samples with 1 ms gap between sample production. SciStream maintains similar performance for the other configurations of the experiment, and it is always able to achieve a goodput significantly close to the theoretical maximum.

7 DISCUSSION

We discuss various aspects of the SciStream design.

7.1 Application Code Changes

We design SciStream to require only minimal application code changes. In particular, S2CS and S2DS processes and the negotiation and control protocols are designed so that application code changes are required only at the higher level of the producer and consumer applications and not in the streaming libraries that sit between the application and SciStream. For example, the consumer application (with minimal code changes) interacts securely with the S2CS and exchanges information that the streaming library (with no code changes) needs to connect to the local S2DS and start receiving the data stream from the producer.

7.2 Fault Detection and Recovery

We intend that SciStream provide mechanisms for detecting and reporting failures, such as packet losses or process failures, via SciStream control process (S2CS) and user client (S2UC) throughout the application life cycle. These mechanisms provide necessary information for applications to recover from failures. We think it is easier to implement efficient fault recovery methods at the application layer than in the infrastructure (SciStream), since applications have more information about their analysis pipelines. For example, because a simulation code knows the data structures that must be checkpointed during its execution, and a suitable checkpoint frequency, it is well positioned to store that internal state to persistent storage for failure recovery. In contrast, SciStream sees only a bit stream, without any application-specific information, and thus checkpointing within SciStream would be resource intensive and inefficient.

7.3 SciStream for Real-time Processing

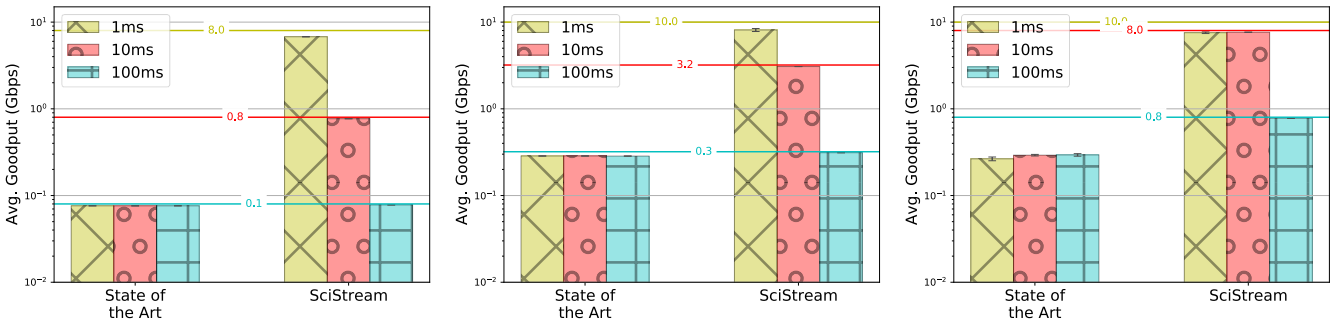
SciStream mechanisms can also be used to support line-rate data processing for applications such as Internet of Things (IoT), video processing, and measurement/monitoring systems where real-time stream-processing pipelines can be provisioned dynamically across cloud-like infrastructures. They can benefit such applications in two ways. First, individual compute nodes (even if they have external network connectivity) need not be built to handle WAN transfers optimally; second, individual compute nodes need not be exposed to the external world.

For instance, SciStream can be applied to a well-known design pattern in wireless sensor networks where alternate versions of TCP [58] and other protocols [54] are deployed in the last mile. These protocols work well for wireless sensor applications, but the end-to-end data path may also involve high-speed wired connections, and need to switch to standard TCP versions. With SciStream, GNs sit close to sensors, receive data from sensors over the wireless network using the alternate transport protocols, and send them to other resources using a transport protocol (e.g., HTCP [42], UDT [30]) that is optimized for wired WAN transport.

7.4 Resource Management

SciStream's control protocol is designed to allow S2UC, producer, and consumer to connect independently as they become available, to accommodate best-effort resources. Thus, some resources at the GN will be held until producer and consumer send their Hello message (see §4.4). However, if we hold GN resources for the time $\Delta = t_{Hello} - t_{REQ}$, these resources will sit idle. As Δ becomes larger, S2CS will start rejecting requests because of "unavailable resources," when in fact it could have been serving other requests.

For fully provisioned requests (i.e., those for which required compute resources and network bandwidth to process the data streams are provisioned for a specific duration), the user request must include a start and end time in addition to the required bandwidth (§4.4). The user request must also indicate whether the request is provisioned or



(a) Small samples (1 MB) (b) Medium samples (4 MB) (c) Large samples (10 MB)

Figure 10: Goodput performance evaluation of a streaming application running on a state-of-the-art scientific environment configuration (i.e., DTNs and file-based data movement methods) vs. running over the `SciStream` environment (i.e., GNs and memory-to-memory data streaming) over a LAN. We evaluate three sample sizes (small: 1 MB, medium: 4 MB, and large: 10 MB) and three time gaps between each sample production (1000 Hz, 100 Hz, and 10 Hz, i.e., T={1, 10, 100} ms.). Colored horizontal lines represent the theoretical goodput for each configuration, computed as min(8 × sample_size/T, 10 Gbps).

best-effort. If start and end time parameters are not provided (best-effort request), we assume start time as current time and assign a default maximum limit so that resources are not retained indefinitely. For future work, we will consider overbooking GNs for best effort cases so as to minimize resource wastage; however, that overbooking should not impact the performance of fully provisioned requests. We will investigate mechanism to identify misuse of provisioned option and flag dishonest users.

8 RELATED WORK

Data streaming and analysis methods have been studied and evaluated for a wide range of use cases in industry [4, 7]. In the scientific context, which is the focus of `SciStream`, the data rates of individual flows are much higher than the data rates of the individual live streaming flows handled by content delivery networks (CDNs), e.g., ~20 Gbps from a single (X-ray detector) source [11, 13, 22, 45]. Also, CDNs often deliver given content to thousands of users that are geographically distributed, whereas, in the scientific streaming context, data is usually streamed to a single (or a small number of) facility or cluster for analysis. The resources in the scientific context are also federated across administrative and security domains, and proxies often have to switch traffic between WAN and HPC interconnects. `SciStream` provides a suite of protocols to establish an authenticated and transparent connection between producer and consumer via intermediate GNs that are optimized for memory-to-memory data streaming.

Improving the streaming data throughput has been an active area of research. Ghasemi et al. [28] focus on performance issues and their characterizations in data streams, including server-side delays due to asynchronous disk-reads and cache misses, jitter, buffering delays, and dropped frames. Melette et al. propose a new network design, *Opera*, that can deliver high-throughput connection via a dynamic network and time-varying expander graphs [48]. *AViC* [2] is a caching algorithm for video streams that can perform request predictions for content delivery systems. Amaro et al. present mechanisms for *far memory*, which allow compute nodes to access other nodes memory within the cluster, to improve the turnaround time of the jobs [6]. *Bertha* is an API for offloading some of the application functionalities, such as serialization or fast-path transfers between containers, to network (or smart NICs) [49]. Ao et al. implemented a parallel data processing framework, Sprocket, that can utilize serverless cloud resources for video processing framework [8]. `SciStream` is a tool to address the infrastructural challenges that otherwise hinder memory-to-memory data streaming in secure scientific environments, and therefore `SciStream` will enable the aforementioned tools and optimizations to be used for streaming large-scale scientific data to remote HPC.

The ability to analyze streaming data in order to provide fast (quasi-real-time) feedback is crucial for many use cases. Several high profile streaming data analysis frameworks and systems have been developed [3, 26, 41, 55]. While these tools provide easy implementation of streaming data analysis pipelines, e.g., defining DAGs and mechanically establishing connections and transfers, they can also provide advanced optimizations such as via domain specific languages and compilers [34, 35]. Although generally developed for industrial problems, e.g., cybercrime detection [4] or recommendation systems [7], these systems can also be used for scientific computing. Their performance depends not only on the computational resources on which they operate but also on their use of efficient data transfer methods. As `SciStream` is application (and framework) agnostic, any of these frameworks can be efficiently used in the `SciStream` content with little user effort.

9 CONCLUSION

We have presented `SciStream`, a middlebox-based framework to enable secure data streaming from a producer's memory at one scientific facility to a consumer's memory at another remote facility. We described `SciStream`'s architecture and design, and defined negotiation and control protocols to establish authenticated and transparent connection between producer and consumer via intermediate gateway nodes. We demonstrated a prototype of `SciStream` and evaluated it on the Chameleon cloud on both LAN and emulated WAN environments. Our results show `SciStream` improves the throughput of an streaming pipeline by an order of magnitude

compared to state-of-the-art data transfer methods in scientific environments. Moreover, `SciStream` only adds $\sim 4\mu$ latency to a streaming pipeline compared to an ideal scenario in which producers and consumers have direct external connectivity.

ACKNOWLEDGMENTS

This material was based upon work supported by the U.S. National Science Foundation (NSF), under award 2019073. It was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

[1] Advanced Scientific Computing Research and Basic Energy Sciences, U.S. Department of Energy. 2015. Exascale Requirements Review. https://www.osti.gov/servlets/purl/1341721. Accessed: 2020-01-06.

[2] Zahaib Akhtar, Yaguang Li, Ramesh Govindan, Emir Halepovic, Shuai Hao, Yan Liu, and Subhabrata Sen. 2019. Avic: a cache for adaptive bitrate video. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 305–317.

[3] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. Millwheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1033–1044.

[4] Mohammed Ali Al-Garadi, Kasturi Dewi Varathan, and Sri Devi Ravana. 2016. Cybercrime detection in online communications: The experimental case of cyberbullying detection in the Twitter network. *Computers in Human Behavior* 63 (2016), 433–443.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004.* 423–424. https://doi.org/10.1109/SSDM.2004.1311241

[6] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[7] Xavier Amatriain. 2013. Big & personal: data and models behind netflix recommendations. In *Proceedings of the 2nd international workshop on big data, streams and heterogeneous source Mining: Algorithms, systems, programming models and applications*. 1–6.

[8] Lixiang Ao, Liz Izhikevich, Geoffrey M Voelker, and George Porter. 2018. Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*. 263–274.

[9] William Barnett, Von Welch, Alan Walsh, and Craig A Stewart. 2011. *A roadmap for using NSF cyberinfrastructure with InCommon*. Technical Report.

[10] Basic Energy Sciences 2007. Directing Matter and Energy: Five Challenges for Science and the Imagination. https://science.osti.gov/-/media/bes/pdf/reports/files/Directing_Matter_and_Energy_rpt.pdf. Accessed: 2021-09-06.

[11] T. Bicer, D. Gursoy, R. Kettimuthu, I. T. Foster, B. Ren, V. De Andrede, and F. De Carlo. 2017. Real-Time Data Analysis and Autonomous Steering of Synchrotron Light Source Experiments. In *IEEE 13th International Conference on e-Science*. 59–68. https://doi.org/10.1109/eScience.2017.53

[12] A. Bobyshev, M. Crawford, P. DeMar, V. Grigaliunas, M. Grigoriev, A. Moibenko, D. Petravick, R. Rechenmacher, H. Newman, J. Bunn, F. Van Lingen, D. Nae, S. Ravot, C. Steenberg, X. Su, M. Thomas, and Y. Xia. 2006. Lambda Station: On-Demand Flow Based Routing for Data Intensive Grid Applications Over Multitopology Networks. In *2006 3rd International Conference on Broadband Communications, Networks and Systems*. 1–9. https://doi.org/10.1109/BROADNETS.2006.4374315

[13] Jan-Willem Buurlage, Federica Marone, Daniël M Pelt, Willem Jan Palenstijn, Marco Stampanoni, K Joost Batenburg, and Christian M Schlepütz. 2019. Real-time reconstruction and visualisation towards dynamic feedback control during time-resolved tomography experiments at TOMCAT. *Scientific reports* 9, 1 (2019), 1–11.

[14] Kyle Chard, Eli Dart, Ian Foster, David Shifflett, Steven Tuecke, and Jason Williams. 2018. The Modern Research Data Portal: A design pattern for networked, data-intensive science. *PeerJ Computer Science* 4 (2018), e144.

[15] Kyle Chard, Steven Tuecke, and Ian Foster. 2016. Globus: Recent Enhancements and Future Plans. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale* (Miami, USA) *(XSEDE16)*. Association for Computing Machinery, New York, NY, USA, Article 27, 8 pages. https://doi.org/10.1145/2949550.2949554

[16] Junguk Cho, Hyunseok Chang, Sarit Mukherjee, T. V. Lakshman, and Jacobus Van der Merwe. 2017. Typhoon: An SDN Enhanced Real-Time Big Data Streaming Framework. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies* (Incheon, Republic of Korea) *(CoNEXT '17)*. Association for Computing Machinery, New York, NY, USA, 310–322. https://doi.org/10.1145/3143361.3143398

[17] J. Y. Choi, T. Kurc, J. Logan, M. Wolf, E. Suchyta, J. Kress, D. Pugmire, N. Podhorszki, E. Byun, M. Ainsworth, M. Parashar, and S. Klasky. 2016. Stream processing for near real-time scientific data analysis. In *2016 New York Scientific Data Summit (NYSDS)*. 1–8. https://doi.org/10.1109/NYSDS.2016.7747804

[18] J. Y. Choi, T. Kurc, J. Logan, M. Wolf, E. Suchyta, J. Kress, D. Pugmire, N. Podhorszki, E. Byun, M. Ainsworth, M. Parashar, and S. Klasky. 2016. Stream processing for near real-time scientific data analysis. In *2016 New York Scientific Data Summit (NYSDS)*. 1–8. https://doi.org/10.1109/NYSDS.2016.7747804

[19] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. 2014. The Science DMZ: A network design pattern for data-intensive science. *Scientific Programming* 22, 2 (2014), 173–185.

[20] Ewa Deelman, Christopher Carothers, Anirban Mandal, Brian Tierney, Jeffrey S Vetter, Ilya Baldin, Claris Castillo, Gideon Juve, Dariusz Król, Vickie Lynch, et al. 2017. PANORAMA: An approach to performance modeling and diagnosis of extreme-scale workflows. *The International Journal of High Performance Computing Applications* 31, 1 (2017), 4–18.

[21] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.

[22] Junjing Deng, Curt Preissner, Jeffrey A Klug, Sheikh Mashrafi, Christian Roehrig, Yi Jiang, Yudong Yao, Michael Wojcik, Max D Wyman, David Vine, et al. 2019. The velociprobe: An ultrafast hard x-ray nanoprobe for high-resolution ptychographic imaging. *Review of Scientific Instruments* 90, 8 (2019), 083701.

[23] DOE, NSF, and AFOSR. 2015. Streaming and Steering Applications: Requirements and Infrastructure STREAM2015 Workshop Final Report. http://streamingsystems.org/stream2015finalreport.html. Accessed: 2020-03-01.

[24] DOE, NSF, and AFOSR. 2016. STREAM2016: Streaming Requirements, Experience, Applications and Middleware Workshop Workshop Final Report. https://www.osti.gov/servlets/purl/1344785. Accessed: 2020-03-01.

[25] DOE-SC. 2013. The Report of the BES Advisory Subcommittee on Future X-ray Light Sources. https://science.osti.gov/-/media/bes/besac/pdf/Reports/Future_Light_Sources_report_BESAC_approved_72513.pdf. Accessed: 2021-09-06.

[26] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. 2017. Dhalion: self-regulating stream processing in heron. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1825–1836.

[27] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir. 2015. Scheduling the I/O of HPC Applications Under Congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 1013–1022. https://doi.org/10.1109/IPDPS.2015.116

[28] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. 2016. Performance characterization of a commercial video streaming service. In *Proceedings of the 2016 Internet Measurement Conference*. 499–511.

[29] Jeremy Goecks, Anton Nekrutenko, James Taylor, Galaxy Team, et al. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology* 11, 8 (2010), R86.

[30] Yunhong Gu and Robert L Grossman. 2007. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks* 51, 7 (2007), 1777–1799.

[31] Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney, and William Johnston. 2006. Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System. In *3rd International Conference on Broadband Communications, Networks, and Systems*.

[32] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, Venkatram Vishwanath, Zarija Lukić, Saba Sehrish, and Wei-keng Liao. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (2016), 49–65.

[33] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.".

[34] Martin Hirzel, Henrique Andrade, Bugra Gedik, Gabriela Jacques-Silva, Rohit Khandekar, Vibhore Kumar, Mark Mendell, Howard Nasgaard, Scott Schneider, Robert Soulé, et al. 2013. IBM streams processing language: Analyzing big data in motion. *IBM Journal of Research and Development* 57, 3/4 (2013), 7–1.

[35] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. 2014. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–34.

[36] J. Ibarra, J. Bezerra, H. Morgan, L. Fernandez Lopez, M. Stanton, I. Machado, E. Grizendi, and D.A. Cox. 2015. Benefits brought by the use of OpenFlow/SDN on the AmLight intercontinental research and education network. In *Integrated*

Network Management (IM), 2015 IFIP/IEEE International Symposium on. 942–947. https://doi.org/10.1109/INM.2015.7140415

[37] Internet2. 2014. Internet's Advanced Layer 2 Services. https://www.internet2.edu/products-services/advanced-networking/layer-2-services/.

[38] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, J Manbretti, Paul Rad, and R Paul. 2017. Chameleon: a scalable production testbed for computer science research. *Contemporary High Performance Computing* 3 (2017).

[39] Raj Kettimuthu, Zhengchun Liu, David Wheeler, Ian Foster, Katrin Heitmann, and Franck Cappello. 2017. Transferring a Petabyte in a Day. *4th International Workshop on Innovating the Network for Data Intensive Science (INDIS) 2017* (Nov. 2017), 1–11.

[40] Anne L. Kinney and Dawn M. Tilbury. 2018. Dear Colleague Letter: Data-Driven Discovery Science in Chemistry (D3SC). https://www.nsf.gov/pubs/2018/nsf18075/nsf18075.pdf. Accessed: 2021-03-01.

[41] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 239–250.

[42] Douglas Leith and Robert Shorten. 2004. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet*.

[43] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. 2019. Data transfer between scientific facilities–bottleneck analysis, insights and optimizations. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 122–131.

[44] Zhengchun Liu, Prasanna Balaprakash, Rajkumar Kettimuthu, and Ian Foster. 2017. Explaining Wide Area Data Transfer Performance. In *26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington, DC, USA) *(HPDC '17)*. ACM, New York, NY, USA, 167–178. https://doi.org/10.1145/3078597.3078605

[45] Zhengchun Liu, Tekin Bicer, Rajkumar Kettimuthu, and Ian Foster. 2019. Deep learning accelerated light source experiments. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. IEEE, 20–28.

[46] Zhengchun Liu, Rajkumar Kettimuthu, Joaquin Chung, Rachana Ananthakrishnan, Michael Link, and Ian Foster. 2021. Design and Evaluation of a Simple Data Interface for Efficient Data Transfer across Diverse Storage. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 6, 1 (2021), 1–25.

[47] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Nageswara SV Rao. 2018. Cross-geography scientific data transferring trends and behavior. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 267–278.

[48] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 1–18.

[49] Akshay Narayan, Aurojit Panda, Mohammad Alizadeh, Hari Balakrishnan, Arvind Krishnamurthy, and Scott Shenker. 2020. Bertha: Tunneling through the network api. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 53–59.

[50] NSF. 2012. Advancing Astronomy in the Coming Decade: Opportunities and Challenges. https://www.nsf.gov/mps/ast/portfolioreview/reports/ast_portfolio_review_report.pdf. Accessed: 2021-03-01.

[51] NSF's Big Ideas. 2017. Harnessing Data for 21st Century Science and Engineering. https://www.nsf.gov/news/special_reports/big_ideas/harnessing.jsp. Accessed: 2020-01-06.

[52] Boris Oklander and Moshe Sidi. 2008. Jitter buffer analysis. In *2008 Proceedings of 17th International Conference on Computer Communications and Networks*. IEEE, 1–6.

[53] Stefan Paulsen, Tadeus Uhl, and Krzysztof Nowicki. 2011. Influence of the jitter buffer on the quality of service VoIP. In *2011 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 1–5.

[54] Paulo Rogério Pereira et al. 2007. End-to-end reliability in wireless sensor networks: Survey and research challenges. In *EuroFGI Workshop on IP QoS and Traffic Control*, Vol. 54. Citeseer, 67–74.

[55] Bogdan Simion, Daniel N Ilha, Suprio Ray, Leslie Barron, Angela Demke Brown, and Ryan Johnson. 2015. Slingshot: A modular framework for designing data processing systems. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 421–430.

[56] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist. 2004. X.509 proxy certificates for dynamic delegation. In *3rd annual PKI R&D workshop*, Vol. 14.

[57] Justin M Wozniak, Timothy G Armstrong, Michael Wilde, Daniel S Katz, Ewing Lusk, and Ian T Foster. 2013. Swift/T – large-scale applicatino composiion via distributed-memory dataflow processing. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 95–102.

[58] Ye Tian, Kai Xu, and N. Ansari. 2005. TCP in wireless environments: Problems and solutions. *IEEE Communications Magazine* 43, 3 (March 2005), S27–S32. https://doi.org/10.1109/MCOM.2005.1404595

[59] Jason Zurawski, Robert Ball, Artur Barczyk, Mathew Binkley, Jeff Boote, Eric Boyd, Aaron Brown, Robert Brown, Tom Lehman, Shawn McKee, Benjeman Meekhof, Azher Mughal, Harvey Newman, Sandor Rozsa, Paul Sheldon, Alan Tackett, Ramiro Voicu, Stephen Wolff, and Xi Yang. 2012. The DYNES Instrument: A Description and Overview. *Journal of Physics: Conference Series* 396, 4 (2012), 042065. http://stacks.iop.org/1742-6596/396/i=4/a=042065