# Transport Control Networking: Optimizing Efficiency and Control of Data Transport for Data-Intensive Networks (Invited Paper)

Jacob Dunefsky*
Mahdi Soleimani*
Yale University

Ryan Yang*
Choate Rosemary Hall

Jordi Ros-Giralt
Qualcomm Europe, Inc.

Mario Lassnig
CERN

Inder Monga
Energy Sciences Network

Frank K Wuerthwein
San Diego Supercomputer Center

Jingxuan Zhang
Tongji University

Kai Gao
Sichuan University

Y. Richard Yang
Yale University

## ABSTRACT

Data-intensive sciences are becoming increasingly important for modern sciences. The transport control plane (TC-Plane) of the networks supporting data-intensive sciences can be important to achieve efficient and controlled transport of data for data-intensive sciences. In this paper, we analyze FTS, which is the de facto TC-Plane of the largest data-intensive network, revealing both efficiency and resource control issues of the current design. We then present the design and initial evaluation of Transport Control Networking (TCN), a design that is based on FTS but introduces (1) network-application co-design/coordination, which uses ALTO to realize network-wide resource control , and (2) a general, efficient, flexible optimization framework for TC-Plane, which allows both zero-order and first-order (*e.g.*, bottleneck structure) gradient-based algorithms. We also discuss future work to engage the broad networking and data-intensive sciences communities.

## CCS CONCEPTS

• **Networks** → **Network design principles**;

## KEYWORDS

Network information exposure, ALTO, FTS, Rucio, CERN

**ACM Reference Format:**
Jacob Dunefsky* Mahdi Soleimani*, Ryan Yang*, Jordi Ros-Giralt, Mario Lassnig, Inder Monga, Frank K Wuerthwein, Jingxuan Zhang, Kai Gao, and Y. Richard Yang. 2022. Transport Control Networking: Optimizing Efficiency and Control of Data Transport for Data-Intensive Networks (Invited Paper). In *ACM SIGCOMM 2022 Workshop on Network-Application Integration (NAI '22), August 22, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538401.3548550

## 1 INTRODUCTION

Data-intensive sciences are becoming increasingly important as they drive the investigation of fundamental scientific questions through collection and processing a large amount of scientific data. In a large-scale data-intensive sciences setting, scientists from several projects collaborate to design an instrument infrastructure to generate the data, a data center infrastructure to store and process the data, and a network infrastructure to transfer the data. Since the data can be distributed widely, spanning multiple networks, the communities working in the sciences collaborate to design an overall network architecture and develop community networking tools and applications, resulting in a fundamental setting we call data-intensive networks. As a concrete example, CERN has constructed a worldwide distributed system of 170 data centers across 40 countries for the global transfer, storage, and processing of the large data sets generated by the LHC experiments: ALICE, ATLAS, CMS, and LHCb.

A core component of a data-intensive network is its transport control plane, which we denote as the TC-Plane. The TC-Plane receives higher layer transfer requests and decides, for each request, when it is scheduled and at what rate it should be transported. A well-design TC-Plane can lead to both *efficiency* (*e.g.*, application performance optimization) and *resource control* (*e.g.*, controlled infrastructure resource multiplexing).

One might think that the design of the TC-Plane should be relatively straightforward by using existing techniques developed by the networking community: including widely-deployed Internet transport control (*e.g.*, TCP congestion control [6, 9]), well-studied data distribution architectures such as CDN (*e.g.*, Akamai's network [14]), and recent advances in data-center flow scheduling (*e.g.*, [1, 7]) or wide-area bandwidth allocations by large Internet service providers (*e.g.*, [10, 12, 13]).

Unfortunately, the TC-Plane has its own unique challenges that the aforementioned existing techniques do not address. For example, one might assume that TCP can already conduct congestion control (CC) and hence achieve transport rate allocation. However, TCP CC is designed for independent, individual transfers, but TC-Plane should manage bandwidth allocation for sets of interdependent transfers (*e.g.*, those transfers for the same scientific experiments).

The recent advances in flow scheduling for data centers and wide area networking indeed consider sets of transfers. However, they are designed for a single administrative domain with a highly centralized, managed operating environment. The TC-Plane, on the other hand, should operate when a data transfer may span links in multiple autonomous network domains, and the end devices are heterogeneous, without single management control (*e.g.*, install the same operating system configuration). The data-intensive sciences communities introduced the two systems, FTS[4] and Rucio[5], which target the organization, management, access, and transfer of data across multiple networks and administrative domains. As we show in this work, the network transfer scheduling component of FTS has both efficiency and resource control issues.

This paper aims to present the initial design of Transport Control Networking (TCN), a highly effective TC-Plane for data-intensive networks. TCN adopts techniques developed by both the networking and the data-intensive sciences communities. TCN achieves both resource control and efficiency.

Specifically, TCN achieves resource control by allowing systematic specification of global resource control and using the IETF ALTO framework [3] to map transfers to resource usage, realizing global resource control through network-application co-design.

Following resource control, TCN introduces a highly effective transport scheduling framework and associated algorithms. The framework is based on the existing, widely deployed FTS framework but addresses FTS issues by introducing a gradient composition framework supporting both zero-order and first-order gradients (*e.g.*, gradients modeled by bottleneck structures [15]).

We conduct initial evaluations to demonstrate the benefits of TCN transport scheduling. We show that, in a simulated setting, TCN outperforms the FTS optimizer with respect to flow completion time, bandwidth utilization, and maximum utility on both a single link and on the ESnet topology. Additionally, we find that the ability to incorporate first-order gradients drastically increases convergence speed. These results suggest that TCN can be a promising system for handling transfers in data-intensive networks.

The rest of the paper is organized as follows. In §2, we provide background. In §3, we specify the main design requirements of TCN. In §4, we analyze FTS, the current *de facto* TC-plane used by CERN. §5 provides the overall architecture and core APIs. §6 provides the core scheduling algorithms. §7 conducts initial evaluations of the algorithms. We conclude and give the next steps in §8.

## 2 BACKGROUND

The design of TCN is based on the existing data transfer systems operated by CERN and collaborating organizations. To help readers understand TCN, we briefly introduce the components involved. These components have been deployed in production for years and are managing the data transmissions for major scientific experiments, including ATLAS, CMS, and LHCb.

**Infrastructure.** The infrastructure provides the hardware resources. Storage, computation, and internal networking connectivity are provided by participating organizations, including CERN, research institutes, and universities around the globe (referred to as *sites*), and the inter-site connectivity is provided by multiple National Research and Education Networks (NRENs), including ESnet, Geant, LHCONE, Internet2, SURFnet, and NORDUnet, among others.

**Transport Data Plane.** The transport data plane consists of software and protocols which provide the basic functionalities of data storage and transfer between two endpoints. A storage element is deployed at a site and globally addressable by the hostname, port, protocol, etc. At the same time, data transfers transmit files between storage elements (storage-to-storage transfers) or from a storage element to a machine for further processing (storage-to-transfer transfers) over network infrastructure. Currently deployed storage systems by the CERN experiments include EOS, dCache, and XRootD, and the majority of transfer protocols in use are HTTP/WebDAV, XRootD, and S3.

**Transport Control Plane.** This layer, which is the focus of this paper, receives *data transfer requests* (i.e., the files to be transferred, the source storage element, and the destination storage element) from the upper layer and determines *when and at what speed* to execute a request. The *de facto* transport scheduling control plane is the *File Transfer Service* (FTS) [4].

**Transport Management Plane.** This component specifies the resource sharing and performance goals of the transport system.

## 3 REQUIREMENTS

With the preceding background, we now specify the main requirements of the TC-Plane, focusing on a high-level description.

### 3.1 Software Architecture Requirements

S1 Controllability: The "control knobs" by which optimization is carried out must be widely available to support heterogeneous deployment settings.

S2 Ease of deployment: As opposed to heavy-weight control at the packet level, which is challenging to deploy, light-weight control at the transfer level is preferred.

S3 Modular, decoupling architecture: It allows a single rate-controlling component to interface with multiple higher-level selection mechanisms.

### 3.2 Performance Function Requirements

P1 Optimality: In scheduling sets of transfers, the system should utilize resources as fully as possible.

P2 Resource allocation: There must be a way to determine which resources, and how much of those resources each user has access to.

P3 Dependency coordination: When scheduling sets of transfers: the system should be able to take into account dependencies between individual transfers.

## 4 ANALYSIS OF CURRENT DESIGN

With the preceding requirements, we now analyze FTS, which is the *de facto* transport control plane for data-intensive transfers at CERN. Our analysis focuses on FTS against the requirements enumerated in §3. FTS shines in satisfying the software architecture requirements (§3.1), but the current design cannot achieve the functional requirements specified in §3.2.

### 4.1 FTS Software Architecture Analysis

S1 FTS uses the number of TCP connections between a source and a destination (we refer to a $(src, dst)$ pair as a *pipe*) as the control knob, which is universally available, and hence represents an excellent design option to achieve S1.

S2 FTS delegates the transfers to the transport data plane. Since there are multiple transfer tools available and at high bandwidth settings, rate limits at hosts or switches can become bottlenecks, FTS does not directly use packet-level control to limit transport rates. Thus, FTS has a lightweight control design.

S3 Although in CERN's system, transfers are generated by multiple different sources, FTS is the major coordinator responsible for coordinating global transfers. FTS thus embodies a modular software architecture.

## 4.2 FTS Performance Function Analysis

We next analyze FTS on the 3 performance function requirements. Since FTS considers only individual transfers, it does not handle P3. Hence, we focus on the optimality (P1) and resource control (P2) behaviors of FTS. Protocol 1 gives the FTS algorithm which we analyze. Although there are other parts of the optimizer (which take into account file size and success rate changes), this algorithm forms the core of the optimizer.

**Protocol 1** FTS Model Analyzed (Called for High Success Rate)

```
1:  Define $RL(x)$ = round($\log_B(x)$)
2:  procedure OPTIMIZEGOODSUCCESSRATE(state)
3:      if cur.ema < prev.ema then
4:          if RL(cur.ema) < RL(prev.ema) then
5:              decision = prevValue - decreaseStepSize
6:          else
7:              decision = prevValue
8:          end if
9:      else if cur.ema > prev.ema then
10:         decision = prevValue + increaseStepSize
11:     else                                        ▷ emas are equal
12:         decision = prevValue + increaseStepSize
13:     end if
14: end procedure
```

**Protocol 1 to Model.** Assume an interval model, and let $t+1$ denote the new interval (*cur*) in Protocol 1, and $t$ the preceding interval (*pre*). The system state at interval $t$ is the vector $n(t) = \{n_i(t)\}$, where $n_i(t)$ is the number of connections of pipe $i$ at interval $t$. FTS adapts each $n_i(t)$ individually and the Protocol 1 above is the algorithm. FTS includes $M_i$ and $m_i$, which are configurations limiting the maximum and minimum of $n_i(t)$ respectively. Let $T_i(t)$ denote the throughput of pipe $i$ during interval $t$. Then the exponential moving average (*ema* in Protocol 1) of pipe $i$, denoted as $E_i$ below, is computed as $E_i(t+1) = \alpha T_i(t+1) + (1-\alpha)E_i(t)$, where $\alpha$ is a constant.

With the notations. a formalization of Protocol 1 is the following:

$$n_i(t+1) = \begin{cases} n_i(t) - 1 & RL_B(E_i(t+1)) < RL_B(E_i(t));\ \text{Line 4} \\ n_i(t) + 1 & E_i(t+1) \geq E_i(t);\ \text{Lines 9,11} \\ n_i(t) & \text{else} \end{cases}$$

where $B$ is the base of the logarithm in Protocol 1; in actual FTS, we have $B = 10$. We refer to this model as the **Dynamical System Model**; when context makes it clear, we call it "the model" for short.

**Analysis** We consider an idealized model where $T_i(t)$ is a simple function of $n(t)$. The model is more accurate when (1) there is a greater persistent backlog at each queue (pipe), and (2) the transfer boundary effects are not large (*e.g.*, transfer sizes are large).

One challenge in the analysis is that the dynamical system model can exhibit complex behaviors, including both convergence and oscillations. We present the results in 2 settings: (1) when the model converges to a fixed point; and (2) a specific setting where the model oscillates between two states.

**Setting I (Fixed-Point Convergence Behavior)**
Consider the case where the model converges to a fixed point at a finite time $t_0$. We have the following result:

THEOREM 4.1 (STABLE STATE CHARACTERIZATION). *Let $n(t)$ be the vector $(n_1(t), \ldots, n_k(t))$ at time $t$. If, for all $t > t_0$, $n(t) = n(t_0)$, then $n_i(t) = M_i$ for all $i$.*

The theorem makes clear that if the model converges to a fixed point, then the only possible fixed point is the state at which all pipes reach their configured maximum numbers of connections. The intuition is that due to finite precision of floating point computation, regardless of the initial values, cur.ema ($E_i(t+1)$) eventually equals prev.ema ($E_i(t)$), at which point Case 2 of the dynamical system model increases $n_i$. Thus the only stable fixed point is the maximum.

However, our evaluations show that the model does not always converge to a fixed point. In the general case, an insightful result is the following conserved quantity proposition:

THEOREM 4.2 (CONSERVATION THEOREM). *Let $K = \max \frac{M_i}{m_i}$. Then as long as $B > (1 - \alpha + \alpha K^2)$, the quantity*

$$V_t(t) = n_i(t) - \text{round}(\log_B(E_i(t))$$

*only ever stays constant or increases.*

This provides valuable insight into the general behavior of Protocol 1: other than slight changes in round($\log_B(E_i(t))$), the number of connections generally cannot be decreased, and will either go to the maximum or increase until it begins oscillating.

The implication of these theorems on resource control is the following. Because the only possible fixed-point state is the state where $n_i = M_i$, the only mechanism for resource control is a judicious setting of the $M_i$. But it is challenging to set $M_i$ to achieve a desired throughput distribution, given complicating factors such as TCP RTT bias and complex network bottleneck structures.

**Setting II (Non-Fixed-Point Behavior)** It is also important to analyze cases in which we have oscillatory behavior. In particular, consider the following simple, concrete setting:

**Definition (Throughput-Deterioration Model)**: A single pipe traversing a single link with capacity $C$, where throughput $T(t)$ is given according to the following model:

$$T(t) = \max\left(0, \min\left(C, C - d \cdot (n(t) - n_0)\right)\right),$$

where $n$ is the number of connections. In this model, when the number of TCP connections is $\leq n_0$, the throughput is the capacity $C$. But when the number of connections is $> n_0$, various components are overwhelmed and the effective capacity decreases by $d$ for each new connection until it reaches a value of 0. Note that choosing any number of connections $\leq n_0$ always gives the optimal throughput.

However, under FTS, we have the following theorem regarding Protocol 1 behavior under the Throughput-Deterioration Model:

Theorem 4.3 (Oscillation Characterization). *Consider the Throughput-Deterioration Model setting. Define* $P(x) = B^{\left(round(\log_B x) - \frac{1}{2}\right)}$. *Then, there exists some* $t_0$ *such that for* $t > t_0$, *Protocol 1 is either always equal to the maximum, or it oscillates between* $N$ *and* $N+1$ *where* $N \geq n_0 + \left\lfloor \frac{C - P(C)}{d} \right\rfloor$.

Recall that the Throughput-Deterioration Model considers a single pipe traversing a single link. Oscillation requires a decrease in $RL_B(T)$, so a value of $T$ satisfying $RL_B(T(\cdot)) \leq \text{round}(\log_B C) - 1$ is needed. This happens for the first time at $N = n_0 + \left\lfloor \frac{C - P(C)}{d} \right\rfloor + 1$ since this $N$ value gives a throughput of $C - d \cdot (N - n_0) < P(C) = B^{\left(\text{round}(\log_B C) - \frac{1}{2}\right)}$.

As a corollary, we have the following important result:

Corollary 4.4. *In the Throughput-Deterioration setting, if* $C = B^p$ *for some integer* $p$, *then either Protocol 1 oscillates between points which have an achieved throughput* $1/\sqrt{B}$ *of the optimal, or it stabilizes to* $n = M$ *(which can have arbitrarily poor throughput depending on* $d$ *and* $n_0$*).*

In fact, by selecting $C$ to be the greatest floating point number less than $B^{p+1/2}$, we can cause achieved throughput to be almost a factor of $1/B$ of the optimal.

Intuitively, this is due to two causes: (1) Protocol 1 only decreases $n$ when throughput drops by an order of magnitude; (2) If decreasing $n$ increases throughput, then instead of decreasing again, Protocol 1 *increases* $n$. In particular, this means that Protocol 1 sometimes goes against the gradient $dT/dn$, causing an oscillation around the point where the order of magnitude changes.

**Summary**: Protocol 1 has four issues:

(1) As demonstrated in Corollary 4.4, Protocol 1 may produce throughputs that are suboptimal by almost a factor of $1/B$. Thus, it fails to meet the optimality requirement. This is because Protocol 1 does not always adjust $n$ in the direction of the gradient $dT/dn$, but can be considered a semi-gradient algorithm.
(2) Protocol 1 is a zero-order method, in the sense that it does not need an explicit model. However, zero-order can be less efficient than a first-order method that computes gradients when an *explicit* model is available.
(3) As seen in Setting I, the only fixed point of the system is when $n_i = M_i$ for all $i$. Hence, it does not include an effective resource control mechanism.
(4) Protocol 1 attempts to optimize the throughput of each pipe independently, resulting in a multi-objective framework. A multi-objective framework, however, can result in non-Pareto optimal solutions.

## 5 TCN ARCHITECTURE

We now present the overall design of TCN, which is based on FTS but includes extensions. In particular, TCN extends FTS in three core ways. (1) TCN introduces a component in the management plane to provide resource control specifications. (2) TCN utilizes a unifying data structure called the *run-time transport-control state* to address P3 and map logical resource usage to physical resource usage using ALTO. (3) TCN specifies a transport control scheduling framework to enhance native FTS substantially, addressing its efficiency and resource control issues.

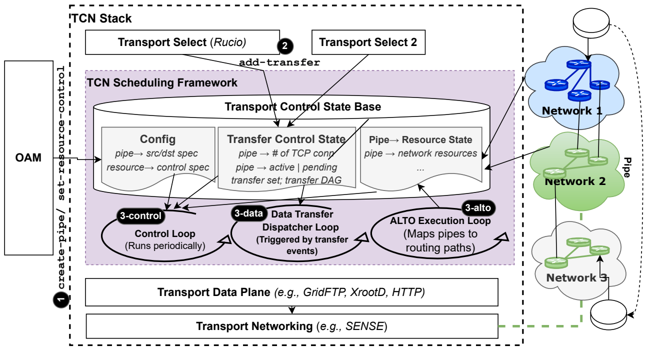


**Figure 1: Architecture and workflow of TCN.**

Below, we introduce the abstractions used by TCN, go over the typical workflow of TCN (Figure 1), and explain the components.

**TCN Abstractions**. There are four primary abstractions used in TCN: (1) *user*, which represents an organization or experiment (*e.g.*, CMS); (2) *pipe*, which represents a logical connection from a source to a destination; (3) *resource*, which can be anything from network bandwidth to storage; (4) *transfer*, which traverses a pipe.

Additionally, pipes are mapped to *resource-set*s, defining which resources the pipe uses (*e.g.*, which physical links or storage units are used). Resource control is achieved by setting bounds on how much of the resources in a given resource-set each user can use. The resource-sets associated with a given pipe are automatically updated in control loop 3-alto, as explained below.

**TCN Workflow**. First, operators specify the pipes and their resource control requirements (❶), with the following APIs:

```
create-pipe: src, dst, user -> pipe
set-rc: user, resource-set -> bound | rel-weight
etc.
```

Then, during operations, a transfer-select component adds sets of transfers (❷), with the following APIs:

```
add-transfer: data, pipe, order -> trans-id
etc.
```

Note that the add-transfer call is a push API, and TCN also allows a pull-based design. Additionally, unlike basic FTS, TCN allows each transfer to include ordering constraints (*i.e.*, which transfers must finish before a given transfer can start); hence, this design can integrate a powerful mechanism (*i.e.*, Sincronia [1]) to achieve co-flow scheduling optimization. (Note, however, that Sincronia assumes that there is no bottleneck inside of the network. In order to meet this assumption, the network must support a hose model [2] that allows clients to behave as if there is no bottleneck.)

The main functions of TCN are implemented in 3 control loops, which run concurrently:

- The ALTO execution loop (3-alto) continues to update (pull or push) the mapping from pipes to the network resources used by each pipe, using ALTO to obtain information about network resources;
- The control-state loop (3-control) runs periodically by invoking the scheduling framework (§6) to update the control state: the number of concurrent TCP connections[1];

[1]this control state can have a limited controllability space, but for this paper, we still focus only on this control state.

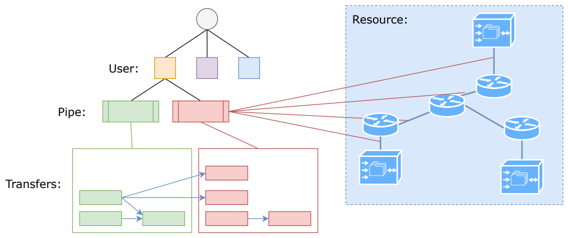


**Figure 2: Run-time transport control state.**

- The data-transfer dispatch loop (3-data) is triggered when a new transfer is added or when the data plane finishes a transfer. It is guided by the control state maintained by the control-state loop.

**TCN Core Data Structures**. The core data structure supporting the three control loops is shown in Figure 2; we refer to it as the *run-time transport control state*. Recall that pipes are mapped to resource-sets; this enables TCN to achieve performance requirement P2. Additionally, the run-time transport control state allows TCN to meet performance requirement P3 by storing transfers in a DAG and representing dependencies as directed edges.

**Summary:** Two main novelties of the TCN architecture are as follows: (1) Via ALTO integration, which makes network resources visible to the application, resource constraints can be explicitly accounted for by the scheduler; (2) By storing transfers in a DAG, TCN can encode complex dependencies between sets of transfers.

## 6 TCN SCHEDULING FRAMEWORK

Within the architecture outlined in the previous section, we now give the details of TCN's scheduling framework. The framework is based on FTS but includes four major differences: (1) instead of using semi-gradient, it always uses full gradient; (2) instead of only using zero-order, it uses first-order when explicit models are available; (3) instead of depending on specifying indirect connections bounds to achieve resource control, it allows generic resource constraints in optimization; (4) instead of using multi-objective gradients, it uses a single objective to avoid non Pareto-optimal inefficiency. Below, we first give the details of the framework (§6.1), and then we give the details of the full zero-order gradient (§6.2)

### 6.1 TCN Scheduling Framework

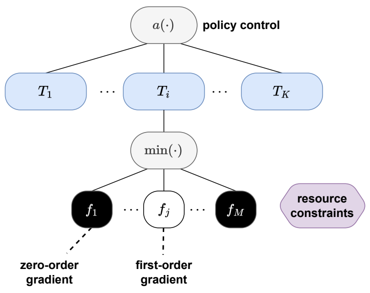


**Figure 3: TCN composition framework.**

Figure 3 specifies the complete composition framework, which consists of two levels of compositions. At the top level, it uses an aggregation function $a(\cdot)$ to map multiple objectives (throughputs) to a single objective. (For instance, $a(\cdot)$ might be summation.) At the second level, it uses the $\min(\cdot)$ function to handle multiple bottlenecks.

With the composition framework, the scheduling framework can be specified as optimizing the objective function: $a(T_1, T_2, \ldots, T_k)$ where $T_i(n) = \min(f_{i,1}(n), \ldots, f_{i,j}(n))$. Each of the $f_{i,j}$, in turn, represents a different bottleneck on the throughput of pipe $i$ (*e.g.*, a network bottleneck, a storage bottleneck, etc).

We use the notation from §4.2, and use $n = (n_1, \ldots, n_k)$ for the overall decision vector. Optimizing the objective function using gradients requires computing the gradient of $a(\cdot)$ with respect to the control vector $n$. Using the structure of the composition framework,

$$\frac{\mathrm{d}a}{\mathrm{d}n_i} = \sum_{j=1}^{K} \frac{\mathrm{d}a}{\mathrm{d}T_j} \cdot \frac{\mathrm{d}T_j}{\mathrm{d}n_i} \tag{1}$$

measures the change in $a$ with respect to assignment changes at pipe $i$ when there are a total of $K$ pipes. Since $a(\cdot)$ is explicitly expressed in terms of $T$, $\mathrm{d}a/\mathrm{d}T_j$ can be easily computed. The main difficulty therefore comes from computing the $\mathrm{d}T_j/\mathrm{d}n_i$ terms. Since $T_j = \min\{f_{j,1}(n), f_{j,2}(n), \cdots, f_{j,k}(n)\}$, if $k = \mathrm{argmin}_k f_{j,k}$, then $\mathrm{d}T_j/n_i = \mathrm{d}f_{j,k}/\mathrm{d}n_i$. For ease of use, $f_{j,k}$ will be used to refer to the bottleneck of pipe $j$. Finally, the derivative is computed with:

$$\frac{\mathrm{d}f_{j,k}}{\mathrm{d}n_i} = \begin{cases} \text{zero order estimate} & \text{if } f_{j,k} \text{ is blackbox} \\ \text{first order gradient} & \text{otherwise.} \end{cases} \tag{2}$$

First-order gradients for a given explicit function are available in general and can be found even in specialized cases (*e.g.*, taking into account the bottleneck structure of networks [15]). Below, we focus on computing zero-order estimates for blackbox $f_i$.

### 6.2 TCN Zero-Order Algorithm

Designing a correct, efficient zero-order gradient algorithm, however, is not easy. There are multiple approaches to computing zero-order gradients. TCN generates noise by first sampling $z' \sim N(0, I)$. Then, we set $z = int(z)$, where $int(\cdot)$ is defined below. Now, let $n' = n + z$. Then

$$G(n, z) = \frac{f(n') - f(n)}{\|z\|^2} \cdot z \tag{3}$$

is an estimator of the gradient.

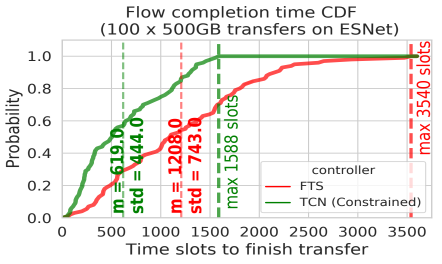
Utilizing the preceding estimator and accelerating using momentum, TCN scheduler implements a zero-order gradient algorithm, shown in Protocol 2. Specifically, in Protocol 2, line 15 updates the gradient using momentum; line 16 rounds the update with the following function:

$$int(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\ \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor. \end{cases} \tag{4}$$

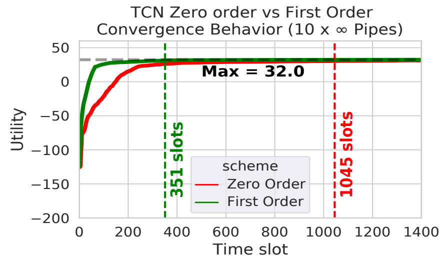
This is called "stochastic rounding" [11] and is necessary because the space of possible assignments is discrete. In expectation, $\mathbb{E}[int(x)] = x$, and this allows us to keep the true value of the momentum step.

## 7 EVALUATION

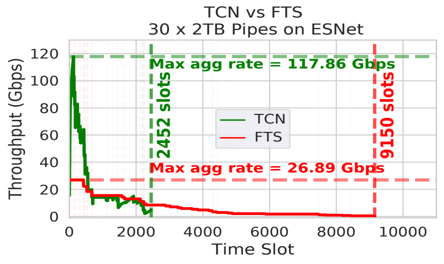
The TCN framework has four major components: (A) the aggregation of multiple optimality criteria into a single objective function, (B) zero-order gradient estimates for implicit throughput functions, (C) support for explicit first-order gradients, and (D) dynamic resource control constraints. We performed initial but systematic



(a) TCN vs FTS FCT (100×500GB Pipes).



(b) TCN 0-order vs 1st-order (10×∞ Pipes).



(c) TCN vs FTS (30×2TB Pipes).

**Figure 4: TCN Evaluations**

**Protocol 2** Momentum-Based Discrete Zero-Order Optimization

```
1:  $a(\mathbb{T})$ = some utility function of the throughputs $\mathbf{T}$
2:  cur.$\mathbf{n}$ = uniform(m, M, k)    ▷ Choose $k$ values for $n$ uniformly between $m$ and $M$
3:  $\alpha, \eta$ assigned hyperparameters
4:  while network is running do
5:      Evaluate $f_{i,k}$ functions
6:      for each $(n_i, T_j)$ pair do
7:          Find bottle neck $f_{j,k}$ for $T_j$
8:          if $f_{j,k}$ is explicit then
9:              Compute $\frac{df_{j,k}}{dn_i}$ = first-order gradient.
10:         else if $f_{j,k}$ is not explicit then
11:             Compute $\frac{df_{j,k}}{dn_i}$ according to Equation 3
12:         end if
13:     end for
14:     Compute $g = (\frac{da(n)}{dn_1}, \frac{da(n)}{dn_2}, \ldots, \frac{da(n)}{dn_K})$ according to Eq (1).
15:     $\mathbf{m} = (1-\alpha)\mathbf{m} + \alpha \cdot (\eta g)$    ▷ Update Momentum
16:     $n = cur.\mathbf{n} + int(\mathbf{m})$    ▷ Update Assignment
17: end while
```

evaluations to demonstrate how these components can improve the top-down performance metrics of scientific transfers. Our evaluations measured the following: (1) aggregated throughput and bandwidth utilization while adhering to the resource control constraints (§7.1), (2) individual flow completion times (FCT) (§7.2), (3) convergence speed, and converged states (§7.3), and (4) behavior under dynamic transfer arrival patterns (§7.4).

## 7.1 FTS vs TCN: Optimality and Constraints Satisfaction

First, we evaluate 2×20TB transfers on a single link with two pipes. We find that TCN with resource constraints achieves 3.33× bandwidth utilization compared to FTS. Additionally, we evaluate 30×2TB transfers on ESnet's topology with 30 pipes using FTS and TCN (zero-order, bandwidth constraints). We find that TCN achieves 4.38× maximum utility when compared to FTS, thanks to components Ⓐ and Ⓑ.

## 7.2 FTS vs TCN: Flow Completion Time

We evaluated 30×2TB transfers on ESnet's topology using FTS and TCN (zero-order, bandwidth constraints) and found that maximum FCT was 3.73× longer in FTS than TCN (fig. 4c).

Additionally, we evaluated 100 pipes, each executing 500GB transfers on the ESnet topology (fig. 4a). TCN with no constraints achieved a mean FCT 1.59× less than FTS, indicating that TCN improves fairness even without active bandwidth constraints. But when explicitly taking into account resource constraints (component Ⓓ), TCN achieved a mean FCT of 1.95× less than FTS.

## 7.3 First vs Zero Order Gradient Ascent

To compare zero-order and first-order versions of TCN, we scheduled 10 pipes with infinite backlogs on the ESnet topology. For concave objective functions, first-order methods (component Ⓒ) converge faster than zero-order methods (component Ⓑ) [8]. However, because the updates used by component Ⓑ involve steps randomly drawn from a unit Gaussian (§6.2), zero-order methods can escape local maxima when BW allocation functions are not concave. Thus, As shown in fig. 4b, with concave bandwidth allocation functions and convex constraints, both versions converge to the same global optimal state, while first-order converges 3.0× faster. However, when evaluated on non-convex BW constraints, zero-order TCN yields a 1.49× gain in final throughput compared to first-order TCN.

## 7.4 Dynamic Arrival Pattern

Due to continuous changes in BW control constraints, handling dynamic transfer arrivals is challenging for a gradient system. To evaluate the stability of TCN, we scheduled 50 dynamic transfers with arrival time drawn from Exp(1/200) with an average size of 4TB on ESnet topology. Component Ⓓ of TCN enables it to achieve a mean FCT of 7.0× lower than FTS.

## 8 CONCLUSION

We presented TCN: a system for handling transport in data-intensive settings. TCN utilizes ideas from current systems while enabling further gains in flexibility, modularity, generality, and performance. TCN takes cues from FTS' method of optimizing transfers, but addresses performance issues of FTS with a gradient-based optimization algorithm that accounts for heterogeneous bottlenecks and resource constraints. Initial evaluations indicate that TCN's optimization algorithm significantly outperforms FTS'. This algorithm is supported by an architecture that interfaces with pre-existing components to keep track of network state and control transfers. We believe that TCN can become an integral part of the systems underpinning data-intensive networks.

## REFERENCES

[1] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat. Sincronia: Near-optimal network design for coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 16–29, 2018.

[2] S. S. Ahuja, V. Gupta, V. Dangui, S. Bali, A. Gopalan, H. Zhong, P. Lapukhov, Y. Xia, and Y. Zhang. Capacity-efficient and uncertainty-resilient backbone network planning with hose. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 547–559, 2021.

[3] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Woundy. Application-layer traffic optimization (alto) protocol. RFC 7285, RFC Editor, 09 2014.

[4] A. Ayllon, M. Salichos, M. Simon, and O. Keeble. FTS3: new data movement service for WLCG. In *Journal of Physics: Conference Series*, volume 513, page 032081. IOP Publishing, 2014.

[5] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing, et al. Rucio: Scientific data management. *Computing and Software for Big Science*, 3(1):1–19, 2019.

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[7] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36, 2012.

[8] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

[9] S. Ha, I. Rhee, and L. Xu. Cubic: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[10] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 15–26, 2013.

[11] M. Hopkins, M. Mikaitis, D. R. Lester, and S. Furber. Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190052, jan 2020.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[13] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, et al. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 1–14, 2015.

[14] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: a platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[15] J. Ros-Giralt, N. Amsel, S. Yellamraju, J. Ezick, R. Lethin, Y. Jiang, A. Feng, L. Tassiulas, Z. Wu, M. Y. Teh, and K. Bergman. Designing data center networks using bottleneck structures. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM ’21, page 319–348, New York, NY, USA, 2021. Association for Computing Machinery.