AggNet: Cost-Aware Aggregation Networks for Geo-distributed Streaming Analytics

Dhruv Kumar
University of Minnesota, Twin Cities
dhruv@umn.edu

Sohaib Ahmad
University of Massachusetts, Amherst
sohaib@cs.umass.edu

Abhishek Chandra
University of Minnesota, Twin Cities
chandra@umn.edu

Ramesh K. Sitaraman
University of Massachusetts, Amherst
ramesh@cs.umass.edu

ABSTRACT

Large-scale real-time analytics services continuously collect and analyze data from end-user applications and devices distributed around the globe. Such analytics requires data to be transferred over the wide-area network (WAN) to data centers (DCs) capable of processing the data. Since WAN bandwidth is *expensive* and *scarce*, it is beneficial to reduce WAN traffic by partially aggregating the data closer to end-users. We propose *aggregation networks* for performing aggregation on a geo-distributed edge-cloud infrastructure consisting of edge servers, transit and destination DCs. We identify a rich set of research questions aimed at reducing the traffic costs in an aggregation network. We present an optimization formulation for solving these questions in a principled manner, and use insights from the optimization solutions to propose an efficient, near-optimal practical heuristic. We implement the heuristic in *AggNet*, built on top of Apache Flink. We evaluate our approach using a geo-distributed deployment on Amazon EC2 as well as a WAN-emulated local testbed. Our evaluation using real-world traces from Twitter and Akamai shows that our approach is able to achieve 47% to 83% reduction in traffic cost over existing baselines without any compromise in timeliness.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Information systems** → **Data analytics**; **Data streaming**.

KEYWORDS

Geo-distributed systems; Edge; Cloud; Stream processing.

ACM Reference Format:
Dhruv Kumar, Sohaib Ahmad, Abhishek Chandra, and Ramesh K. Sitaraman. 2021. AggNet: Cost-Aware Aggregation Networks for Geo-distributed Streaming Analytics. In *SEC '21: ACM/IEEE Symposium on Edge Computing, November-December, 2021, San Jose, CA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Real-time analytics is becoming increasingly important in areas such as web and social media analytics, IoT analytics, video analytics, and energy analytics. Consequently, a number of distributed stream processing systems [3, 12, 36, 37, 48, 72] have been developed in recent times to enable analysis of large quantities of data streams in real-time. The data streams for many of the above mentioned application domains originate in a **geographically distributed** manner. For example, large scale internet services such as Twitter [63], Akamai [49] and Netflix [46] continuously collect data from their users around the globe to analyze trending tweets, monitor the QoS experience of their users, and so on. These analytics tasks are often near-real-time in nature and hence, are **delay sensitive**. Sending the raw data directly from the user devices to enterprise cloud data centers (DCs) in order to minimize delay is a tempting solution but highly impractical given the large data volumes. For instance, Netflix gathers trillions of events and petabytes worth of data per day spread across the globe [47], while Twitter sees around 500 million tweets on a daily basis. In the IoT domain, smart cities and smart power grids deploy thousands of sensors. Readings from these sensors are generated at rates varying from tens to thousands of samples per sensor per second [51]. These readings are then required to be sent to the cloud DCs for analytical purposes. Since WAN bandwidth is highly *expensive* and *scarce* [25, 28, 32, 34, 38, 54, 55, 67], it is important to reduce WAN traffic between user (or IoT) devices and the cloud DCs.

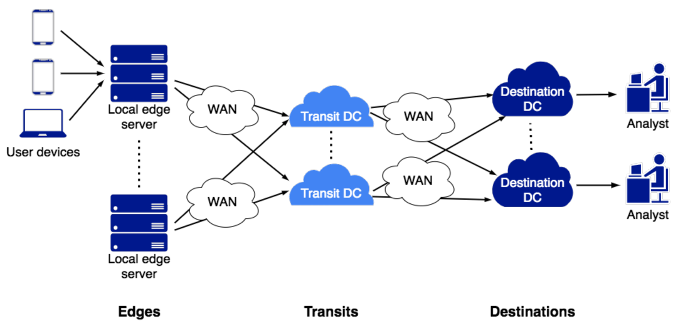


Figure 1: Aggregation Networks. ***Data arrives at the edges and is aggregated at the edges, transit DCs and destination DCs for consumption by the analysts at the destination DCs.***

Enterprises are, thus, increasingly moving towards a **geo-distributed edge-cloud infrastructure** [25, 26, 38, 69] for performing such analytics as close to the user devices as possible. Each edge server partially aggregates data streams from multiple user devices and sends the partially aggregated data streams to the destination DC for final aggregation leading to a reduction in the WAN traffic from the edge to the destination DC. It is also possible to utilize intermediate transit DCs [69] between the edge and destination DCs to assist in aggregation, further reducing the traffic over WAN links. As a concrete use-case, Akamai has a quarter-million CDN edge servers deployed across the globe. Enterprises such as video content providers use Akamai for streaming their video content to their end-users. Akamai's analytics services [2] allow content providers to monitor Quality of Service (QoS) metrics such as video quality and rebuffer rates in real-time to ensure the best quality of experience for their end-users. This real-time monitoring can be done by aggregating the QoS metrics arriving at the edge servers and forwarding the aggregated information to the destination DCs. The content provider often has a latency constraint (or ***delay budget***) within which any degradation in QoS should be fixed. This latency constraint is taken into account while deciding the amount of aggregation to be performed at the edge servers. Twitter offers a similar service [64].

In this work, we focus on performing **continuous aggregation** over data streams. Aggregation forms a prominent part of any data analytics system [10, 30, 71]. Some popular examples are the Reduce operation in MapReduce [16], GroupBy clause in SQL and LINQ [40] etc. These operations are building blocks in many typical analytical queries, used to gain insights from mounds of data as well as to reduce the data volumes involved in analytics [19, 20, 25, 38, 52, 53].

We propose **aggregation networks** to efficiently perform continuous aggregation on the distributed edge-cloud infrastructure. An aggregation network can perform aggregation at edges, transits, and destination DCs (see Figure 1). The goal would be to reduce the WAN traffic between the edges and destination DCs, while meeting the delay requirements of streaming analytics applications.

Aggregation networks have heterogeneity across multiple dimensions which leads us to a rich set of novel research questions. There is heterogeneity in bandwidth availability and compute resources across these networks. A less-studied aspect is the heterogeneity in the traffic cost from one region to another. For example, on AWS, it is 4.5 times costlier to transfer data out of Singapore than Virginia [5]. This cost heterogeneity leads to a tradeoff between traffic cost and traffic wherein the traffic cost could be reduced even at the expense of higher traffic. Further, different queries have different QoS requirements in terms of acceptable delay. This variation in QoS requirements leads to a tradeoff between delay and traffic wherein aggregation for a longer period of time at the edge (higher delay) can lead to lower WAN traffic and traffic cost. Additionally, these networks are highly dynamic, where node failures, bandwidth congestion, and WAN outages are common.

These dimensions lead to a number of interesting questions. *How do we construct efficient and cost-effective aggregation networks? How to orchestrate data movement and routing on these networks while ensuring that the resource constraints and QoS requirements are satisfied? How to handle network and workload dynamism and failures? Most importantly, while doing all this, how do we keep the traffic cost low in these aggregation networks?* In essence, aggregation networks enable a rich tradeoff between delay, WAN traffic, and cost, that we explore.

Our approach and key insights. We formulate an optimization problem to explore these tradeoffs in a principled manner, and derive several insights that can result in significant cost reduction in aggregation networks. For instance, it is common to focus on minimizing traffic ignoring the associated cost. But we find that cost-agnostic traffic reduction can significantly inflate the traffic cost. Additionally, although it is common in practice to route data streams from the edges to their nearest DCs, we find that routing the data streams to a common transit DC which may be further away from the edges, can result in greater reduction of traffic and its associated cost. At the same time, we show that simply selecting the cheapest common transit for all the edges also does not necessarily lead to minimum traffic cost. The optimal solution to the *path provisioning problem* (i.e. which transit to select for each edge) is non-trivial and depends on a variety of factors such as input arrival rates at the edges, cost difference across various network links, bandwidth availability, etc. The optimal solution may select a combination of cheapest common transit for some edges and nearest transits for the remaining edges. The other dimension of an aggregation network is *delay budgeting:* how much aggregation to perform at the edges vs. the transits. We show that performing the entire aggregation at the first stage of aggregation, i.e. at the edges, can be sub-optimal in comparison to splitting the amount of aggregation between the edges and transit DCs. We use these insights to design a practical, efficient and scalable heuristic that can be deployed in a real-world network.

Systems	**Workload Type**	**Cost Aware**
WANalytics [67], Iridium [54], Pixida [34], Clarinet [66], Tetrium [29], Yugong [28]	Batch	No
Kimchi [50]	Batch	Yes
JetStream [55], AWStream [73], ApproxIoT [69], Sana [32]	Streaming	No
AggNet (our contribution)	Streaming	Yes

Table 1: Geo-distributed Data Analytics Systems

Relation to Prior Work. Table 1 compares our proposed work with the existing geo-distributed data analytics systems. To the best of our knowledge, no existing work has looked at minimizing the traffic cost in aggregation networks for real-time geo-distributed streaming workloads. Much of the existing work in geo-distributed analytics [28, 29, 34, 54, 66, 67] has focused on batch processing workloads, with the main goal of reducing WAN traffic[1]. Existing work on geo-distributed streaming analytics [25, 26, 32, 55, 69, 73] has also primarily looked at reducing traffic from edge to the cloud but we argue in this work that *minimizing traffic does not always lead to minimization of traffic cost.* Kimchi [50] is a recent cost-aware geo-distributed analytics system that only considers batch workloads, and assumes that later stages do not contribute significantly to the overall cost. This assumption may not work well for streaming workloads, as we will show.

[1]Works such as Iridium [54] and Tetrium [29] discuss reducing the traffic cost but assume a uniform cost across all network links, thus, essentially reducing the traffic.

Research contributions. We study the problem of minimizing the traffic cost across an aggregation network while satisfying the resource constraints in the network as well as the delay sensitivity of the aggregation queries. Our main contributions are:

- We identify the tradeoffs and opportunities for cost reduction using an aggregation network comprising edge, transit and destination DCs.
- We formulate a mixed-integer non-linear optimization problem (MINLP) to minimize the traffic cost subject to a delay budget and resource constraints.
- We propose a fast, near-optimal and scalable heuristic, based on the insights from the optimization. This heuristic can be employed in a real system and can be used to minimize the traffic cost at a fine-grained level in a dynamic environment where both the resource availability and workloads are dynamic.
- We implement the proposed heuristic in *AggNet*, a prototype we built on top of Apache Flink, a popular stream analytics system.
- We extensively evaluate our approach using a real geo-distributed deployment on Amazon EC2 data centers as well as a WAN-emulated local testbed. Our evaluation uses both synthetic and real world traces from Akamai and Twitter. The proposed technique shows 47% to 83% reduction in traffic cost over existing baselines without any compromise in timeliness.

2 BACKGROUND AND PRELIMINARIES

System Model. We consider a multi-tiered edge-cloud topology [69] (as in Figure 1) consisting of:

- ***Edges.*** Edge servers are located nearest to the user devices. Each user device sends its data to the nearest edge server. Each edge server aggregates the data coming from multiple user devices and forwards the aggregated data stream to one or more transits.
- ***Transits.*** Transit DCs aggregate data streams coming from the edge servers and forward it to the destination DCs. Transit DCs could either be full-fledged regional cloud DCs or micro-clouds.
- ***Destinations.*** These are full-fledged cloud DCs which are the final destinations where all the data originating from the user devices is aggregated, analyzed and consumed by the analytics end-users (e.g., for identifying trends or for taking business critical decisions).

In this work, we consider a general setting where the results of a query may be needed at multiple destination DCs and where each destination DC may request data from all or a subset of edges based on the requirements of the analytics consumers. This redundancy may be needed to provide better reliability as well as lower latency of access to consumers distributed across the globe.

Stream Processing Model. Stream processing systems can be broadly classified into two categories based on their computational model: the dataflow model [4, 12, 45] and the bulk-synchronous parallel (BSP) model [13, 31, 72]. In this paper, we focus on the dataflow model where data streams flow continuously from one or more data sources into the system. Each record in the data stream is processed and transformed by a set of stream operators. However, our proposed techniques are not limited to the dataflow model, and can be applied to the BSP model also without any change.

Data Aggregation. Aggregation is a widely-employed operation within any analytics system [10, 30, 71]. Some prominent examples include the Reduce operation in MapReduce [16], GroupBy in SQL and LINQ [40] etc. Each record in the data stream is of the type (k, v): key k and value v. For performing aggregation over a key-value stream, all the records $(k, v_i), 1 \leq i \leq m$ belonging to the same key k are grouped into an aggregate record $(k, v_1 \oplus v_2 \oplus \cdots \oplus v_m)$, where $v_1, v_2, \cdots, v_m$ are the values received for key k up to time T and $\oplus$ is an application-defined associative binary operator. These operators can range from simple operators such as *sum* or *max* to more sophisticated operators like transforms and sketches (such as HyperLogLog), and user-defined functions. In this work, we focus on continuous aggregation at the destination DCs where the newly arrived data record (k, v) is immediately aggregated into its key k's aggregated value to provide the most updated aggregated result for the key k[2].

As an example, in the context of Akamai Download Analytics service, the analyst may be interested in the number of bytes successfully downloaded for each content provider in every location (e.g. USA, Europe and Asia). In this case, the key would be a combination of content provider id and geographic location and the value would be the number of bytes successfully downloaded for the content provider from that location. The SQL statement for such an aggregation query would look like:

```
SELECT LOCATION, CP_ID, SUM(BYTES_SUCCESS)
FROM Host.US, Host.EU, Host.Asia
GROUP BY LOCATION, CP_ID
```

TTL-based Aggregation. In this work, we employ Time-to-Live (TTL)-based aggregation [38]: a recently proposed aggregation model that provides a theoretical basis for modeling data aggregation in streaming analytics. This model employs a TTL aggregator (inspired by TTL caches) that assigns a TTL value to each key in the data stream. It holds and aggregates records belonging to each key for a time period equal to the key's TTL, as follows. Whenever a record (k, v) arrives at the TTL aggregator, if an aggregate for the key k currently does not exist in the aggregator (cache miss), the key k is inserted into the aggregator along with its value v and a timer equal to the key's TTL is set. Until the timer counts down to zero, all the records arriving at the TTL aggregator for key k are aggregated into the existing record (cache hit). Once the timer expires, the aggregated record is flushed out and the key k is removed from the aggregator.

Aggregation Network. An aggregation network comprises the multi-tiered edge-cloud topology as mentioned above (See an example network in Figure 1). Each edge, transit and destination in the network has compute capacity which is utilized to deploy a TTL-aggregator for aggregating the incoming data streams. This network spans across a WAN environment and hence, there is massive heterogeneity in resource availability in WAN environments such as WAN bandwidth. For instance, we measured the available bandwidth between different AWS EC2 sites and found it to vary between 20Mbps and 400Mbps. This variation is in line with the findings from prior work [29, 32, 39]. More interestingly, the dollar cost for WAN traffic also varies from one region to another. A portion of this cost variation taken from AWS [5] and Azure [7] is

[2]This encompasses windowed grouped aggregation which aggregates records within pre-defined time windows (window length could vary from minutes to hours to days). In such a case, the current window will be continuously updated to reflect the most updated aggregated result for any key.

shown in Table 2. As seen from this table, it can be 8x costlier to move the same amount of data out of SA Brazil site than US West site on AWS.

AWS				
Orig \ Dest	US West	Australia	AP India	SA Brazil
US West	0	0.02	0.02	0.02
Australia	0.14	0	0.14	0.14
AP India	0.086	0.086	0	0.086
SA Brazil	0.16	0.16	0.16	0

Azure Cloud				
Orig \ Dest	US West	Australia	AP India	SA Brazil
US West	0	0.087	0.087	0.087
Australia	0.12	0	0.12	0.12
AP India	0.12	0.12	0	0.12
SA Brazil	0.181	0.181	0.181	0

Table 2: Cost of data transfer in AWS and Azure Cloud (in $ per GB). Orig: Origin, Dest: Destination.

Metrics. While there can be several metrics of interest: cost, network traffic, energy consumption, etc., here, we focus on the following key metrics that are critical for executing an analytics query on an aggregation network.

- ***Traffic and Traffic cost.*** Traffic is the number of records per second transmitted over the entire aggregation network, while traffic cost is the cost per second incurred in transmitting traffic over the entire aggregation network.
- ***Delay.*** The delay of a record is the time lag between receiving that record at the edge and the first time this record is incorporated into (i.e., contributes to) the aggregate result at the destination[3]. Since we are considering streaming analytics, it is critical to get the results of an analytics query with low delay[4].
- ***Delay Budget.*** We define delay budget for a query to be the maximum tolerable delay for any record used by that query. That is, in order to satisfy the delay budget of a query, each record should get aggregated into the final aggregates for the query at the destination within < *delay_budget* > time duration. Delay budget is generally specified by the analyst based on the specific requirements of the application.

Aggregated Stream Rate. We derive the aggregated stream rate (outgoing traffic) out of a TTL-aggregator deployed at any edge or transit in the following manner: the TTL-based aggregation model [38] defines the outgoing traffic per record (μ) as a function of input arrival rate (λ) and TTL (T).

$$\mu(\lambda, T) = \frac{1}{1 + \lambda T} \tag{1}$$

Equation 1 says that for every cache miss, there will be λT cache hits that follow, before the aggregated record is flushed from cache and sent out over the outgoing link. This makes intuitive sense since a record that is inserted into cache as a "miss" will receive an average of λT other records that are "hits" within the timer expiration window of length T that follows. In this work, TTL is the maximum aggregation delay (τ) incurred by any record. We multiply the outgoing traffic per record (μ) by the input arrival rate λ to get the aggregated stream rate (λ_{agg}).

$$\lambda_{agg}(\lambda, \tau) = \mu(\lambda, \tau) \cdot \lambda = \frac{\lambda}{1 + \lambda\tau} \tag{2}$$

This provides a theoretical basis [5] to capture the tradeoff between delay and traffic, which we use in our problem formulation[6].

3 OPPORTUNITIES AND TRADEOFFS

In this section, we use simple examples to illustrate the tradeoffs and identify the opportunities for traffic cost reduction in aggregation networks[7]. In all examples, assume that all results are sent to all the destinations.

Tradeoff between traffic cost and traffic. While minimizing traffic is considered to be a critical goal by many providers, the cost heterogeneity among different links might lead to a traffic-cost tradeoff. To illustrate, consider the aggregation network shown in Figure 2: it consists of three edges, two transits and two destinations, with the cost for each link as indicated in the figure. Assume delay budget to be 100 ms and the entire aggregation happens at the transits.

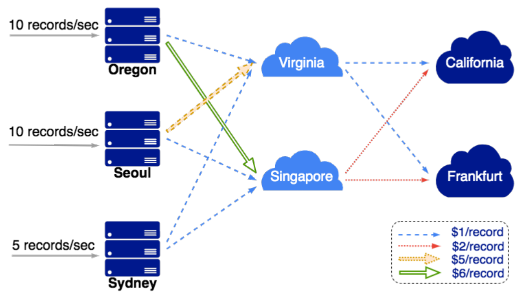


Figure 2: Aggregation network with cost heterogeneity.

Now consider two possible transit-to-edge assignments based on this network, shown in Table 3. Assignment 1 minimizes traffic but incurs higher traffic cost while Assignment 2 minimizes traffic cost but incurs higher traffic as it selects cheaper transits[8]. Thus, we note that *selecting cheaper transits may be helpful in reducing the traffic cost even at the expense of higher traffic.*

Transit selection for each edge. Traditionally traffic from an edge is routed to the nearest transit. While this may be beneficial for locality, it may result in sub-optimal cost for the aggregation

[3]Note that even though each record is not sent *individually* to the destination, it will always be incorporated into an aggregate that reaches the destination.

[4]We focus only on aggregation delay as it is a direct consequence of aggregation, and do not consider network delay since it is just a constant and the delay budget could be easily adjusted to include it.

[5]The insights presented in this work are also applicable to other aggregation models such as the windowed grouped aggregation [70]. But these models do not have a theoretical basis for trading off delay with traffic. Therefore, for such models, we would need to perform empirical measurements for approximating the tradeoff between delay and traffic.

[6] Although the theoretical model assumes the record arrivals to be Poisson in nature, the aggregation model works well in practice even for non-Poisson arrivals [38].

[7]Note that the examples are specifically constructed to illustrate the tradeoffs, but the insights apply to real-world scenarios (as shown later).

[8]Traffic and Traffic cost are computed using Equation 2 and unit cost multipliers in this section. As an example, the traffic cost for Assignment 2 in Table 3 is computed as: $10*1+10*1+5*1+\frac{10+5}{1+0.1*(10+5)}*2*1+\frac{10}{1+0.1*10}*2*2 = \mathbf{57}$

	Assignment 1	**Assignment 2**
Goal	Minimize traffic	Minimize traffic cost
Transits selected	Oregon ⇒ Virginia Seoul ⇒ Virginia Sydney ⇒ Virginia	Oregon ⇒ Virginia Seoul ⇒ Singapore Sydney ⇒ Virginia
Traffic	**39**	47
Traffic cost	79	**57**

Table 3: Cheaper transits can reduce the traffic cost.

network (given the delay constraint). Consider the aggregation network in Figure 3 consisting of 3 edges, 3 transits, and 3 destinations. Here, two edges (Seoul and Singapore) have co-located transits, and thus, zero traffic cost to these local transits. Assume delay budget

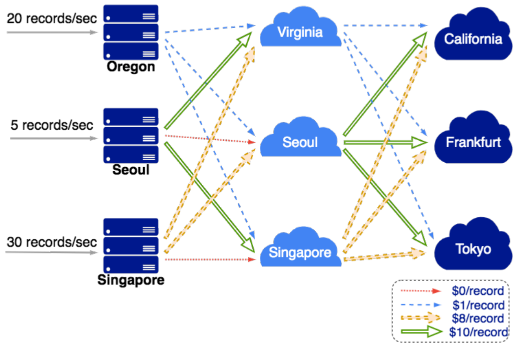


Figure 3: Aggregation network with some co-located edges and transits.

to be 200 ms and the entire aggregation happens at the transits.

Consider three transit-to-edge assignments based on this network, shown in Table 4. Here, Assignment 1 selects the nearest transit for each edge (where available) and Assignment 2 selects the cheapest common transit for all edges. Assignment 3 selects a combination of nearest (for the Singapore edge) and common (non-local for the Oregon and Seoul edges) transits that results in *lowest* traffic cost among the three assignments. The reason for this cost reduction is as follows. There are two parts to the traffic cost: *edge-transit* cost and *transit-destination* cost. The edge-transit cost is minimized by allocating the co-located/nearest transit to each edge while the transit-destination cost is minimized by allocating a common transit to multiple edges, even if it is non-local. Hence, depending on the exact values of these two costs, *the optimal solution may select a common transit for some edges, while choosing a co-located transit for other edges.*

	Assignment 1	**Assignment 2**	**Assignment 3**
Transit selection strategy	Nearest Transit	Cheapest Common Transit	Hybrid: Common and Nearest Transit
Transits selected	Oregon ⇒ Virginia Seoul ⇒ Seoul Singapore (SG)⇒ SG	Oregon ⇒ Virginia Seoul ⇒ Virginia SG ⇒ Virginia	Oregon ⇒ Virginia Seoul ⇒ Virginia SG ⇒ SG
Edge-Transit cost	**20**	310	70
Transit-Destination cost	190	**14**	115
Total cost	210	324	**185**

Table 4: Selecting a combination of common and nearest transits may reduce the traffic cost.

Amount of aggregation at edge vs. transit. Traditionally, pushing as much aggregation towards the edge is considered beneficial for reducing network traffic. However, given an end-to-end **delay budget** (the delay tolerance for the application), there is a tradeoff in terms of how much aggregation should be performed at each tier, as doing *more aggregation results in higher delay* (Equation 2). Consider another aggregation network similar to Figure 2 but having six edges instead of two, and a uniform cost for all the links ($1 per record). Assume a delay budget of 500 ms. Suppose we perform partial aggregation at each edge: we define β to be the fraction of aggregation performed at each edge (thus, $(1-\beta)$ is the fraction at each transit), so that $\beta = 0$ (resp., 1) corresponds to all the aggregation being performed at the transits (resp., edges). Figure

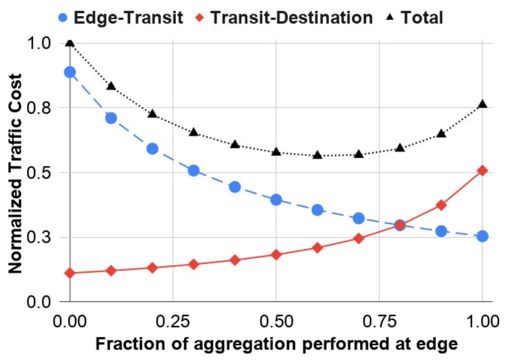


Figure 4: Variation of Traffic cost with β. ***Distributing the amount of aggregation between the edge and transit may help reduce the traffic cost.***

4 plots the total traffic cost along with the edge-transit cost and transit-destination cost as the value of β is varied. It can be seen from this figure that as β increases, the edge-transit cost decreases while the transit-destination cost increases. This shows that there is a tradeoff between the edge-transit cost and transit-destination cost with respect to β and the optimal value of β that minimizes total traffic cost depends on this tradeoff. Hence, we conclude that *distributing the amount of aggregation between the edge and transit may help reduce the traffic cost.*

4 ALGORITHMS FOR CONSTRUCTING AGGREGATION NETWORKS

To explore the tradeoffs illustrated above, we consider the problem of constructing aggregation networks that minimize the traffic cost while obeying the delay budget and resource constraints. We first formulate the problem as a Mixed Integer Linear Program (MINLP) and solve the problem using a constraint solver (§ 4.1). Since this approach is not practical, we use the insights gained from the optimization to derive efficient near-optimal heuristics (§ 4.2).

4.1 Optimization Formulation and Solution

Constructing an aggregation network involves solving two subproblems: (1) **path provisioning**: determining what edge-transit and transit-destination links to use, and (2) **delay budgeting**: how much aggregation to perform at each edge and transit. Our objective is to construct a network that *minimizes the total traffic cost for sending the data from the edges to all the destinations via transits with a delay budget of K seconds*[9].

Item	Notation
Set of edges	E
Set of transits	T
Set of destination	D
Set of keys	N
Indicator function	I
Cost for sending a record from edge i to transit j	c^s_{ij}
Cost for sending a record from transit j to destination k	c^t_{jk}
Delay budget	K
Fraction of delay budget allocated to each edge for key n	β_n
Arrival rate (in records/sec) of key n arriving at edge i	λ^n_i
Available bandwidth (in records/sec) from edge i to transit j	b^s_{ij}
Available bandwidth (in records/sec) from transit j to destination k	b^t_{jk}
Binary variable for selecting transit j for sending key n from edge i to destination k	p^n_{ijk}
Outgoing data rate for key n from edge i to transit j	μ^n_{ij}
Outgoing data rate for key n from transit j to destination k	y^n_{jk}

Table 5: Notation used in the optimization formulation.

$$\min \quad \sum_n COST(\beta_n) \tag{3}$$

$$\text{s.t.,} \quad COST(\beta_n) = \sum_j \sum_i c^s_{ij}\mu^n_{ij} + \sum_k \sum_j c^t_{jk} y^n_{jk} \quad \forall n \in N \tag{4}$$

$$\sum_j p^n_{ijk} = 1 \quad \forall i \in E, k \in D, n \in N \tag{5}$$

$$\mu^n_{ij} = I_{(\sum_k p^n_{ijk} > 0)} \frac{\lambda^n_i}{1 + \lambda^n_i \beta_n K} \quad \forall i \in E, j \in T, n \in N \tag{6}$$

$$y^n_{jk} = \frac{\sum_i p^n_{ijk}\mu^n_{ij}}{1 + \sum_i p^n_{ijk}\mu^n_{ij}(1-\beta_n)K} \quad \forall j \in T, k \in D, n \in N \tag{7}$$

$$\sum_n \mu^n_{ij} \le b^s_{ij} \quad \forall i \in E, j \in T \tag{8}$$

$$\sum_n y^n_{jk} \le b^t_{jk} \quad \forall j \in T, k \in D \tag{9}$$

$$p^n_{ijk} \in \{0,1\} \quad \forall i \in E, j \in T, k \in D, n \in N \tag{10}$$

$$0 \le \beta_n \le 1 \quad \forall n \in N \tag{11}$$

We formulate our optimization in the following manner. The notation used in the formulation is described in Table 5. c^s_{ij}, c^t_{jk}, K, and λ^n_i are inputs to the problem. p^n_{ijk} is a binary variable that indicates the path taken by key n from the edges to destinations via the transits, solving the *path provisioning* sub-problem, while β_n decides how to split the delay budget for key n among the edges and transits, solving the *delay budgeting* sub-problem.

[9]For simplicity, we assume end-results to be replicated across all the destinations, though in practice, replication would be done across a subset, and the ideas explored here will still apply.

The optimization formulation above is a mixed integer non-linear program (MINLP) with a linear objective function and non-linear constraints. The objective function (Equation 3) sums over the traffic cost ($COST(\beta_n)$) of each key n. As shown in Equation 4, $COST(\beta_n)$ has two components: the first term denotes the edge-transit traffic cost for key n over all the edge-transit links (i, j) while the second term denotes the transit-destination traffic cost of key n over all the transit-destination links (j, k). Constraint 5 ensures that there is only one transit selected for each edge-destination pair for each key. Constraint 6 computes the outgoing data rate for each key on each edge-transit link. It uses an indicator function I to decide whether a key should be sent from an edge to a transit. Constraint 7 computes the outgoing data rate for each key on each transit-destination link. Constraints 6 and 7 are derived using Equation 2. Both these constraints are non-linear because of the non-linear relationship between the amount of data reduction, the incoming data rate and the aggregation delay (TTL). Constraints 8 and 9 ensure satisfying the bandwidth constraint on every edge-transit and transit-destination link respectively.

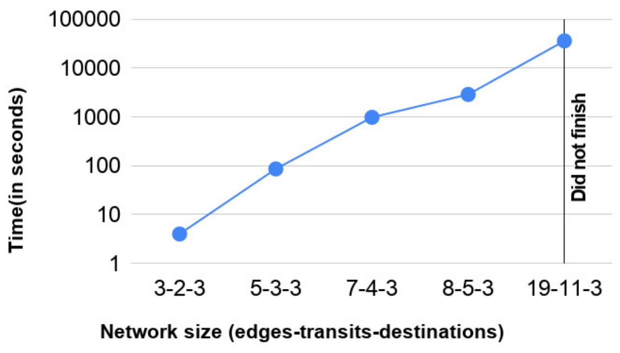


Figure 5: Growth of solution time for the optimization model with network size. *The optimization model is impractical and unscalable for a real system deployment.*

4.1.1 Computational Efficiency of Optimization. The optimization model can be understood to jointly solve for two sets of variables: the path provisioning variables p^n_{ijk} and the delay budgeting variable β_n. Note that the path provisioning and delay budgeting variables are affected by each other and cannot be solved independently. The resulting joint optimization involves solving an MINLP and is hence inefficient. Figure 5 shows how the time required to solve the optimization grows with the network size. For a realistic aggregation network of 8 edges, 5 transits, and 3 destinations, it takes more than 45 minutes to find the optimal solution for one key. For another aggregation network of 19 edges, 11 transits and 3 destinations, the optimization did not find the optimal solution even after running for 10 hours. A real streaming workload may have hundreds and thousands of keys with different arrival rates and different origins (i.e. different keys will have different arrival rates at different edges). Furthermore, the arrival rates may vary with time [21, 59] as well as the resource availability such as network bandwidth may vary significantly across time (every few mins) in a WAN environment [55, 68, 73]. Hence, it is desirable to solve the path provisioning and delay budgeting problem frequently (every few mins) on a per-key level rather than on the aggregate level, in order to find accurate optimal solution. Therefore, to be able to do this in practice, we turn to more efficient heuristics based on the insights provided by the optimization results.

4.2 iCAPP: A Practical Heuristic Approach

We generated the optimization results for real traces using a real testbed (the traces and testbed are described in more detail in §6) and the results confirm the tradeoffs discussed in §3. We use the results of the optimization to extract the following insights that we use in formulating our heuristic.

- Cheaper transits are preferred over costlier transits for cost reduction.
- A common transit for multiple edges helps in reducing the transit-destination cost while a co-located transit for an edge helps in reducing the edge-transit cost. The optimal solution may select a combination of common transit for some edges and co-located transits for the remaining edges to minimize the total traffic cost.
- Allocating the entire delay budget to (performing full aggregation at) either the edge or transit may be sub-optimal, and the minimum cost is achieved at some value $0 \leq \beta_n \leq 1$ of the delay budget distribution between edge and transit for a given key n.

In summary, the exact set of transits selected by the optimization model along with the exact delay budget split between edge and transit is dependent on a variety of factors such as input arrival rates, presence of co-located transits, cost difference across various edge-transit and transit-destination links, and available resources such as bandwidth availability.

In addition to the insights provided by the optimization results, we also observe that the traffic costs used in practice by major cloud providers such as AWS [5], Azure[7] and Google Cloud [23] are origin-based. That is, the cost of sending the data from an origin to any destination is the same. This can also be seen in Table 2. We use this observation along with the insights given by the optimization results to propose a heuristic that we call **Iterative Cost-Aware Path Provisioning (iCAPP)**.

Instead of jointly finding the optimal solution to delay budgeting (β_n) and path provisioning (p^n_{ijk}), our proposed heuristic takes an iterative approach to reduce the solution space (See Heuristic 1 for the pseudocode). For a given value of β_n, the heuristic solves the path provisioning problem in the following manner:

(1) For each transit, compute the sum of the unit traffic costs from this transit to all the destinations.
(2) Sort the transits in the increasing order of the sum computed in Step 1. If there is a tie between two or more transits, then sort the tied transits in the decreasing order of the total incoming arrival rate at each transit's co-located edges. Repeat the following until no edge is left unassigned to any transit.
    (a) Select the next transit (i.e. the cheapest) from the sorted list of transits. Assign this transit as the common transit for as many unassigned edges as feasible given the transit's bandwidth constraints.
(3) For each edge with a co-located(local) transit, check if assigning the co-located transit will reduce the total cost compared to using the common transit for that edge. If yes, then assign the edge to its co-located transit.

Using the above path provisioning process, the heuristic determines the traffic cost $COST(\beta_n)$, for a given fixed value of β_n. Now it iteratively finds the best value of β_n using a hill-climbing algorithm [57]. Specifically, it starts by initializing $\beta_n = \beta_0$, where β_0 can be any value between 0 and 1. In each iteration, it increments (or decrements) the value of β_n by step size γ, if doing so decreases the cost. At any iteration, if the cost cannot be decreased any further by moving to $\beta_n + \gamma$ or $\beta_n - \gamma$, a minimum is reached and the heuristic stops.

Heuristic 1: Iterative Cost-Aware Path Provisioning (iCAPP)

```
Result: Path variables $p^n_{ijk}$ and delay budget split $\beta_n$
        $\forall i \in E, j \in T, k \in D, n \in N$
foreach key $n \in N$ do
    $\beta_n \leftarrow \beta_0$
    $\Delta COST = \infty$
    while $\Delta COST > 0$ do
        $T_{sorted} \leftarrow sort\_by\_cost(T)$
        $E_{unallocated} \leftarrow E$
        while $E_{unallocated} \neq \phi$ do
            $e \leftarrow E_{unallocated}[0]$
            $t \leftarrow T_{sorted}[0]$
            if $t$ has available bandwidth then
                Route $e$ through $t$
                $E_{unallocated} \leftarrow E_{unallocated} - e$
            else
                $T_{sorted} \leftarrow T_{sorted} - t$
        for $e \in E$ do
            if $e$ has local transit $t_{local}$ & $t_{local}$ reduces
              $COST(\beta_n)$ then
                Route $e$ through $t_{local}$
        $\beta_{prev} = \beta_n$
        if $COST(\beta_n + \gamma) < COST(\beta_n - \gamma)$ then
            $\beta_n = \beta_n + \gamma$
        else
            $\beta_n = \beta_n - \gamma$
        $\Delta COST = COST(\beta_{prev}) - COST(\beta_n)$
```

Note that we compute this heuristic on a per-key level i.e. for each key, we select the β_n and set of transits. This heuristic is very efficient and can be practically employed in dynamic environments where the transit selection and β_n may need to be re-computed based on changes in workloads and resource availability (§6.7).

Convergence of iCAPP. The hill-climbing algorithm in iCAPP can theoretically get stuck in a local minimum which may not be a global minimum. We use hill-climbing with random restarts (restarting R times randomly selecting an initial β_n) which is considered to give a reasonably good solution after a small number of iterations [57](§6.7).

5 IMPLEMENTATION

We implement the proposed heuristic in a system we call *AggNet*, on top of Apache Flink [12], a popular stream processing engine. Apache Flink uses dataflow model [4] for its computations (§2). AggNet (See Figure 6) has a *global manager* which runs in one site and interacts with the site managers running at each aggregation site for fetching the most up-to-date arrival rates and resource availability. A Flink cluster runs at each edge, transit and destination. Each edge

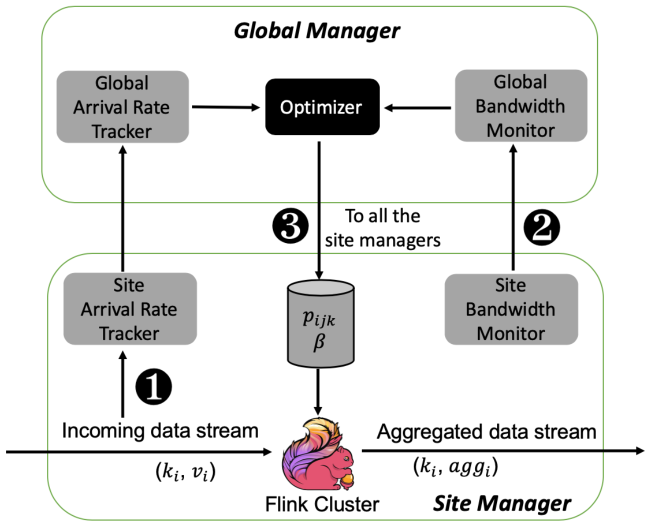


Figure 6: AggNet Architecture

and transit perform aggregation using the `TTLAggregation` operator [38]. The global manager also comprises an optimizer module for computing the path provisioning matrix and delay budget split (β) for each key.

Arrival Rate Tracking. Each edge has an arrival rate tracker which continuously tracks the arrival rates of incoming keys. This information is shared with the global arrival rate tracker running at the global manager (❶).

Bandwidth Monitoring. Each edge and transit has a bandwidth monitor for measuring the available bandwidth at its site, periodically (every few minutes) as in [54] and shares this information with the global bandwidth monitor running at the global manager (❷).

Optimizer. The global manager has an optimizer module which periodically receives the arrival rate and bandwidth availability information from the global arrival rate tracker and global bandwidth monitor. It then computes the path provisioning matrix and delay budget split β for each key and shares them with each site manager for performing aggregation in the desired manner (❸). We use the mlrose [24] library to implement the hill-climbing algorithm in iCAPP.

Adaptation to dynamic workloads. Since the arrival rates for each key will vary depending on various factors such as time of the day, occurrence of any special event etc., AggNet recomputes the path provisioning matrix and delay budget split for a key if its arrival rate changes beyond a threshold ($\triangle_\lambda$). Sometimes, the changes in arrival rates can be transient and hence, AggNet adapts at two different time scales: first, it only recomputes the delay budget split for a key since changes in delay budget split does not require any changes to the path of the aggregated data streams and can be easily done at each site. Second, if the changes in arrival rates are

Edges	Virginia, Oregon, Ireland, Tokyo, Seoul, Mumbai, Sydney, Sao Paulo
Transits	Virginia, Ireland, Tokyo, Sydney, Sao Paulo
Destinations	California, Frankfurt, Singapore

Table 6: Aggregation network used for evaluation.

Origin	**Virginia, Oregon, Ireland**	**Tokyo**	**Seoul**	**Mumbai**	**Sydney**	**Sao Paulo**
Cost (in $ per GB)	0.02	0.09	0.08	0.086	0.14	0.16

Table 7: Cost of data transfer in AWS (origin-based).

persistent (for a period $\triangle_p$), it recomputes the path provisioning matrix for each key. If the estimated traffic cost using new path provisioning matrix is lower than that of the current path provisioning matrix by a threshold ($\triangle_c$), the new path provisioning matrix are communicated to each site manager.

Adaption to variability in bandwidth availability. In scenarios where the available bandwidth to and from a transit changes beyond a certain threshold ($\triangle_b$), the optimizer recomputes the path provisioning matrix and delay budget splits for all the keys and communicates them to each site manager.

Fault tolerance. Each aggregation site uses the default fault tolerance mechanism provided by Flink wherein it periodically checkpoints its processing state. This allows it to restore its execution from the last checkpointed state upon recovering from failures. Flink provides exactly-once consistency guarantees for stateful computations [11].

6 EVALUATION

6.1 Experimental Setup

We evaluate AggNet using a network (Table 6) consisting of 8 edges, 5 transits, and 3 destinations, based on AWS site locations. As mentioned in §5, a Flink cluster runs on each of the edge, transit and destination sites. We run a data streamer locally on each edge site to generate raw input data streams which are sent to the respective edge clusters.

Each edge site continuously ingests the incoming data stream from its local data streamer, performs aggregation using the `TTLAggregation` operator and then forwards the (partially) aggregated data to the transits as decided by the optimizer. Each transit receives the individual streams from the edges, performs aggregation using the `TTLAggregation` operator and then forwards the aggregated data to the destinations as decided by the optimizer. Each destination receives individual streams from the transits, performs a final aggregation and then saves the final results into a database.

We demonstrate the effectiveness and practicality of the proposed approach through a real deployment of AggNet on two testbeds.

AWS EC2 Testbed. We use 11 AWS EC2 geographically distributed sites [5] for our experiments. At each site, we use one t2.xlarge instance type. The edge, transit and destination sites are chosen based on the aggregation network mentioned above. The available

WAN bandwidth data between any two EC2 sites follows similar trend as in [39].

Emulated Testbed. We also run our experiments on a local 11-node testbed which emulates the WAN bandwidth characteristics measured across the AWS EC2 sites. Each site is allocated one node for running the Flink job. Each node is a 6-core Intel(R) Xeon CPU E5-2620 v3 @ 2.40GHz. All nodes are connected by a 1Gbps ethernet. We emulate WAN characteristics using Linux `tc` utility [41]. Experiments in §6.3 use AWS testbed while other experiments use the emulated testbed.

Cost Structure. We consider the dollar costs for traffic from AWS [5] as shown in Table 7.

Evaluation Metrics. As explained in §2, we measure and compare the *traffic* and *traffic cost* for different approaches. We ensure that *delay budget* is satisfied in all the experiments.

6.2 Datasets and Queries

We use real as well as synthetic datasets to evaluate our proposed approach. For real datasets, we use a trace consisting of anonymized beacon logs from Akamai's download analytics service, and the real tweet data from Twitter. For synthetic datasets, we generate records for a group of keys where the arrivals for each key follow Poisson distribution.

Twitter Trace. We use the Twitter Streaming APIs [65] to collect real tweet data from Twitter for three days during the month of December 2015. It consists of approximately 12 million tweets. Each tweet in the trace contains information such as name, gender, age, location of the user along with actual tweet contents. We show results for the popular word count query which computes the frequency of each word appearing in tweets. Similar conclusions hold for other queries such as the trending topics query which computes the frequency of tweets grouped by topic but omitted due to space constraints.

Akamai Trace. We use a month-long trace from Akamai's download analytics service [1]. Content providers consume this service for collecting information such as the popularity of a particular content, the download performance experienced by the end users, the type of devices used for downloading, the number of successful complete downloads, and so on. The trace contains anonymized information about the content provider, user's geography and IP address, download start time, url, content size, number of bytes downloaded, server's geography. *Traffic sizes, time durations etc, are normalized for ensuring confidentiality in our experiments.* We consider the query which computes the total number of bytes downloaded grouped by content provider id and last mile bandwidth.

Synthetic Traces. We also generate synthetic traces assuming Poisson arrivals for each key. We vary the arrival rates for keys from 0.05 to 10 records per second. Relevant details are provided with the experiments where these traces are used.

Wherever applicable, we distribute the data records across the edge sites based on their geographic location associated with the record (or tweet).

6.3 Baseline System Comparison

We consider one set of baseline heuristics for path provisioning and another set for delay budgeting. We then combine each path provisioning baseline with each delay budgeting baseline for comparison with our proposed heuristic `iCAPP`. For path provisioning, we consider the following baselines:

- **Nearest transit selection (NTS)** selects the nearest transit for each edge based on edge-transit latency. If the nearest transit is unable to serve its edge due to resource constraints, the next nearest transit is selected for that edge.
- **Common Transit Selection (CTS)** selects a common transit (at random) for all the edges. If no transit is able to serve all the edges due to resource constraints, then we allocate as many edges to it as feasible based on its resource constraints, and iteratively pick another common transit for the remaining edges.
- **Cheapest Common Transit Selection (CCTS)** selects the cheapest common transit for all the edges. Resource constraints are met as in CTS, except transits are picked in increasing order of their cost.

For delay budgeting, we consider:

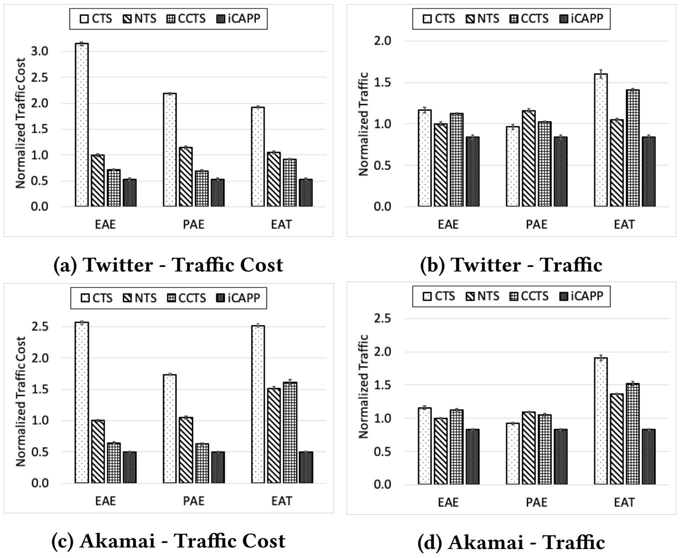
- **Entire Aggregation at Edge (EAE)** allocates the entire delay budget to each edge, resulting in no aggregation at the transits ($\beta = 1$).
- **Entire Aggregation at Transit (EAT)** allocates the entire delay budget to each transit, resulting in no aggregation at the edges ($\beta = 0$).
- **Partial Aggregation at Edge (PAE)** allocates β portion of delay budget to each edge and the remaining $(1 - \beta)$ to each transit. We set $\beta = 0.5$ for this baseline.

Relation to existing state-of-the-art. In the above set of heuristics, NTS-EAE and CTS-EAE represent state-of-the-art on geo-distributed streaming analytics which includes JetStream [55], AW-Stream [73] and ApproxIoT [69]. These works are cost-agnostic and send the data stream from the edge to its nearby transit or to a common DC from where the results are sent to other DCs wherever they are needed. Moreover, they try to do maximum aggregation at the first level i.e. the edges. *Note that the CCTS-PAE heuristic is not derived from prior work but is based on our insights presented in §3 that cheap, common transits and distribution of the delay budget between edge and transit may assist in reducing the traffic cost in an aggregation network.*

Parameter settings for iCAPP. `iCAPP` always utilizes the optimal β computed using Heuristic 1 discussed in §4.2. We set number of random restarts $R = 0$ and step size $\gamma = 0.05$ for the hill-climbing algorithm in iCAPP (See §6.7 for details).

Wherever applicable, we show results as an average of 5 runs, plotted with 95% confidence intervals.

Twitter Trace. We use delay budget equal to 30 seconds for this trace. Figures 7 (a) and 7 (b) shows the overall comparison of various aggregation approaches for the Twitter trace. We can see that `iCAPP` results in 72% to 83% reduction in traffic cost as compared to CTS while 47% - 53% reduction in traffic cost as compared to NTS. This is because both CTS and NTS are cost-agnostic and hence, end up selecting costlier transits such as Sao Paulo, resulting in higher traffic cost. On the other hand, we note that `iCAPP` results in 23% to 42% reduction in traffic cost as compared to CCTS. This is due to the fact that while CCTS and `iCAPP` both select the cheapest common transit (Virginia), `iCAPP` further optimizes the traffic cost by selecting a more accurate delay budget split β. In terms of traffic also, iCAPP results in 13% to 48% reduction as compared to other approaches. *Across all approaches,* iCAPP *simultaneously results in the minimum traffic cost as well as the minimum traffic since it attempts to jointly perform transit provisioning as well as finding the best β for distributing the delay budget between edge and transit.*



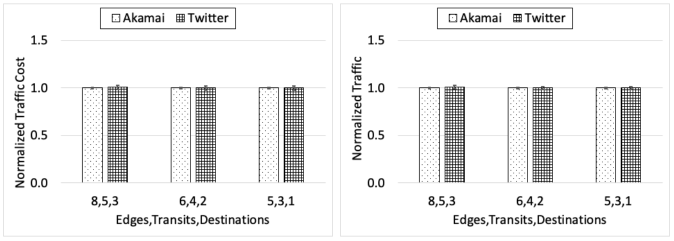
(a) Twitter - Traffic Cost (b) Twitter - Traffic (c) Akamai - Traffic Cost (d) Akamai - Traffic

Figure 7: Baseline comparison for Twitter and Akamai Traces. Lower is better. Normalized by NTS-EAE for respective traces. ***Across all aggregation approaches,*** **iCAPP** ***simultaneously leads to least traffic cost and least traffic.***

Akamai Trace. We use delay budget equal to 10 seconds for this trace. Figures 7 (c) and 7 (d) shows the overall comparison of various aggregation approaches for the Akamai trace. Similar to the results for Twitter trace, we can see that iCAPP results in 50% to 81% reduction in traffic cost as compared to CTS and NTS. In comparison to CCTS, iCAPP results in 20% to 69% reduction in traffic cost. iCAPP also results in 10% to 56% reduction in traffic as compared to other approaches. *Across all approaches,* iCAPP *simultaneously leads to least traffic cost and least traffic.* The reasons for reduction in traffic cost and traffic are same as in Twitter trace.

Comparison with Optimization Model. Since the optimization model proposed in §4 is impractical and not scalable for use in a real system, we use the optimization model to solve path provisioning and delay budgeting problems on an aggregate level, i.e., considering the entire data stream as one key (using the average arrival rate of the stream).

For appropriate comparison, we do the same for the iCAPP heuristic. Figure 8 show the traffic cost and traffic incurred by the iCAPP heuristic (normalized with respect to the corresponding metric for optimization model) for both Akamai and Twitter trace for three aggregation networks. It can be seen that the iCAPP heuristic incurs traffic cost and traffic within 1% of that incurred by the optimization model across all networks for both traces. *This shows that the* iCAPP *heuristic can achieve near-optimal performance.*

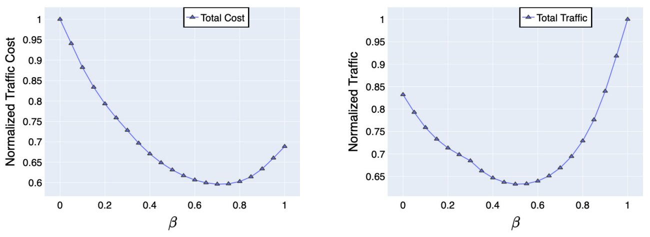


(a) Traffic Cost (b) Traffic

Figure 8: Traffic cost and traffic incurred by iCAPP (normalized by the corresponding metric incurred by optimization model) for Akamai and Twitter traces for different aggregation networks. **iCAPP** ***is able to achieve near-optimal performance in all scenarios.***

6.4 Impact of β and Transit Selection

Taking the same aggregation network as in §6.3 and using Twitter trace, we analyze the following:



(a) Traffic Cost (b) Traffic

Figure 9: Variation of Traffic cost and Traffic with β.

Impact of delay budget distribution β on traffic cost. In this experiment, we analyze the variation in traffic cost for one of the keys for different values of β between 0 and 1 as shown in Figure 9. For each β, we compute the best solution for path provisioning using iCAPP. The minima for the traffic cost and traffic are achieved at $\beta = 0.75$ and $\beta = 0.5$ respectively. This again reinforces the fact that *minimizing traffic does not necessarily lead to minimization of traffic cost.* Moreover, the traffic cost at $\beta = 0.75$ is 14% and 40% lower than that at $\beta = 1$ and $\beta = 0$ respectively. Similarly, the traffic at $\beta = 0.5$ is 38% and 25% lower than that at $\beta = 1$ and $\beta = 0$ respectively. This shows that the minima for both traffic and traffic cost are achieved by distributing the delay budget between edge and transit rather than always allocating it entirely to edge or transit. Similar conclusions hold true for other keys in the Twitter trace but omitted due to space constraints.

Impact of transit selection on traffic cost. In this experiment, we analyze the variation in traffic cost for one of the keys for different transit selection scenarios as shown in Figure 10. For each transit selection scenario, we use the optimal β computed using iCAPP. It can be seen that the scenario where the cheapest common transit (Virginia (VA) or Ireland (IR)) is selected gives the lowest traffic cost as compared to other scenarios which include selecting any of the costlier transits as the common transit, selecting

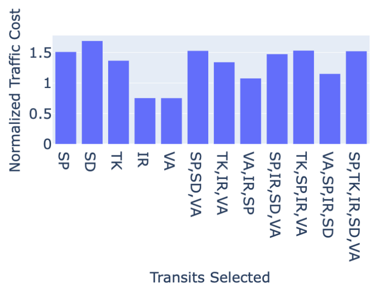


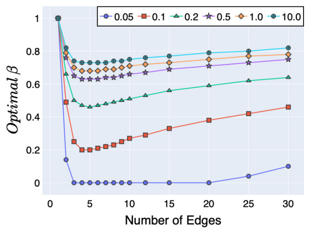
Figure 10: Traffic cost variation with transit selection. ***Lowest traffic cost is achieved by selecting the cheapest common transit (VA or IR).***

a common transit for a few edges and co-located transits for the other edges, or selecting the nearest/co-located transit for every edge. Similar conclusions hold true for other keys in the Twitter trace but omitted due to space constraints.

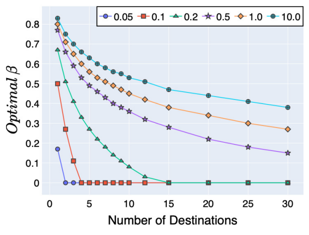
6.5 Variation of optimal β

Using the synthetic trace (§6.2) and iCAPP for computing the optimal β, we analyze the following:

Variation of optimal β with the number of edges. In this experiment, we consider an aggregation network consisting of one transit and 3 destinations. The number of edges are varied from 1 to 30. Figure 11a analyzes the variation of optimal β with the number of edges for different arrival rates. When there is just one edge, optimal $\beta = 1$ (full aggregation at the edge) since there is no gain achieved by aggregating the data stream at the transit. As we increase the number of edges, initially, the optimal β decreases because the reduction in transit-destination cost due to partial aggregation at the transit outweighs the increase in edge-transit cost. After a certain number of edges, the optimal β goes on increasing because now even though it is beneficial to partially aggregate the data streams at the transit, the increase in edge-transit cost becomes dominant. Also, note that the optimal β is never equal to 1 except when the number of edges is just 1.



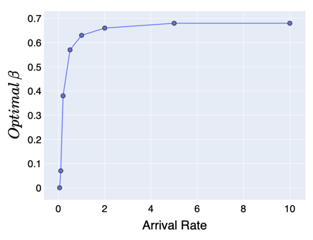
(a) Varying #edges



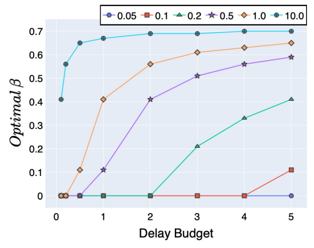
(b) Varying #destinations

Figure 11: Variation of Optimal β with the number of edges and destinations. ***Optimal β first decreases and then increases with the increase in the number of edges while it continuously decreases with the increase in the number of destinations.***

Variation of optimal β with the number of destinations. In this experiment, we consider an aggregation network consisting of one transit and 10 edges. The number of destinations are varied from 1 to 30. Figure 11b shows the variation of optimal β with the number of destinations for different arrival rates. As the number of destinations increases, the optimal value of β decreases. This is because the transit-destination cost becomes dominant with the increasing number of destinations. Thus, pushing more aggregation at the transit helps reduce the total cost. Note that the optimal β is never equal to 1.



(a) Varying input arrival rate.



(b) Varying delay budget.

Figure 12: Optimal β versus arrival rate and delay budget. ***Optimal β increases with increase in arrival rate and delay budget.***

Variation of optimal β with the input arrival rate. Figure 12a shows the variation of optimal β with increase in arrival rate for an aggregation network comprising 8 edges, 1 transit and 3 destinations. The optimal β is low for low arrival rates because the edge-transit cost is going to be low for such data streams and hence, it is beneficial to perform more aggregation at the transit. As the arrival rate increases, the optimal β goes on increasing so as to reduce the edge-transit cost, thereby, reducing the total traffic cost.

Variation of optimal β with the delay budget. Figure 12b shows the variation of optimal β with increase in delay budget for an aggregation network comprising 8 edges, 1 transit and 3 destinations. For a fixed arrival rate at the edge, as the delay budget increases, optimal β also increases. This is because a higher delay budget gives more opportunity for aggregation at the edge (incurring higher delay), but this opportunity has diminishing returns beyond a point.

6.6 Adaptation to dynamism

Since dynamics in both workload and resource availability are common in WAN environments, we now analyze how well AggNet performs in the face of such dynamics by emulating variable stream arrival rates and a link failure. We consider an AggNet deployed over 8 edges, 2 transits and 3 destinations. All the edge-transit and transit-destination links have equal unit cost except the links from Transit-2 to all destinations which are 4 times costlier than other links, so that all edges are assigned to Transit-1 by the iCAPP heuristic. We set the adaptation parameters mentioned in §5 as: $\triangle_\lambda$= 10%, $\triangle_p$= 10 mins, $\triangle_c$= 1%, $\triangle_b$= 10%.

Figure 13 shows the performance of AggNet in a dynamic environment for a synthetic trace which has varying arrival rates. As shown in the top subfigure, the arrival rate is initially low but as time progresses, the arrival rate increases until a point after which

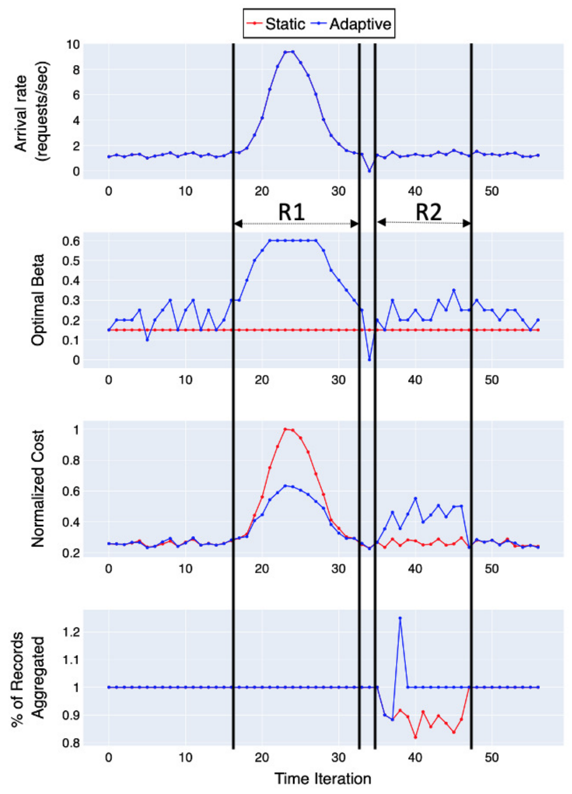


Figure 13: AggNet in a dynamic environment. ***The adaptive iCAPP heuristic is fast enough to adapt with the changes in arrival rates and resource availability.***

it again starts decreasing. The duration of continuously changing arrival rates are marked by region R1 in the figure. In region R2, we emulate a link failure (the link between Edge-7 and Transit-1). We consider two strategies using the iCAPP heuristic: (1) *adaptive*, which periodically recomputes the transit selection and optimal β by considering the recent arrival rate information gathered from all the edges and the bandwidth availability on all the links, and (2) *static* which computes the transit selection and optimal β only once at the beginning assuming the arrival rate and bandwidth availability remains constant. The second, third, and fourth subfigures in Figure 13 show the optimal β values, the normalized cost, and the fraction of records aggregated by each strategy in the presence of these variations. It can be seen that in region R1, the static approach continuously sets a lower β as compared to the adaptive approach and thus, incurs up to 40% higher traffic cost as compared to the adaptive approach. We note that the adaptive heuristic is fast enough to adapt quickly with the changing arrival rates. Further, in region R2, when Edge-7 is not able to send traffic to Transit-1, the adaptive strategy identifies such a bottleneck and shifts the Edge-7 to the second option, Transit-2. This results in a higher traffic cost due to the loss of the cheapest transit for Edge-7, but there is only a short period of data loss/backup. The static approach, on the other hand, is not able to shift Edge-7 to Transit-2 since it is not aware

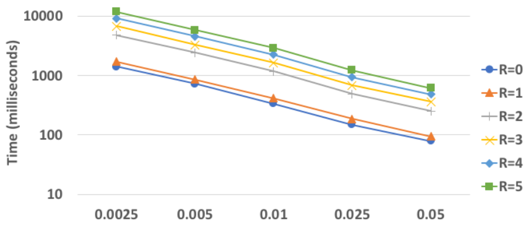


Figure 14: Average solution time (per key) for iCAPP decreases with increasing γ and increases with increasing R. ***For $\gamma = 0.05$ and $R = 0$, iCAPP's computational overhead is modest (below 100 ms) while its solution is within 1% of the optimal solution.***

of the changes in the bandwidth availability. As a result, the static approach leads to loss of data from Edge-7 during the period R2.

These results show that *adaptive* iCAPP *can efficiently adapt to changes in workload and resource availability.*

6.7 Convergence and Computational Overhead of iCAPP

The hill-climbing algorithm in iCAPP has two important parameters: step size γ and number of random restarts R. In order to select the γ and R for our experiments, we take the following approach. We vary R from 0 to 5 and for each R, we vary γ from 0.0025 to 0.05. Then, for each pair of (R, γ), we use iCAPP to solve the path provisioning and delay budgeting problem for both Twitter and Akamai traces on an aggregate level (as we did in §6.3 for comparison with the optimization model). We find that the traffic costs computed by iCAPP for all the (R, γ) pairs are within 1% of the optimal traffic cost computed by the optimization model[10] (similar to our finding in §6.3). We conclude that *although the hill climbing algorithm used by iCAPP can theoretically get stuck in a local minimum and does not guarantee a global minimum, it is able to converge to a near-optimal solution in our experiments over real datasets.* We demonstrate the tradeoff between the solution time and the choice of parameters in Figure 14, using the Twitter dataset as an example. We find that as we increase γ from 0.0025 to 0.05 while keeping R fixed, the solution time continuously decreases. We do the same for each R from 0 to 5, and we find that the solution time increases as R is increased, while keeping γ constant. For $R = 0$ and $\gamma = 0.05$, iCAPP computes the final solution for every key in less than 100 milliseconds on average and hence, iCAPP's computational overhead is modest. *This makes iCAPP practical for real system deployments.*

7 RELATED WORK

Stream Processing Systems. Due to increasing demand for stream processing systems, a number of distributed stream processing systems [3, 12, 36, 37, 48, 72] have been developed in the last few years, providing low latency and high throughput. They are primarily meant for intra-data center environment where the bandwidth is not constrained. Our work focuses on geo-distributed environment where the WAN bandwidth is both scarce and expensive. We utilize

[10]For very large step sizes ($\gamma > 0.05$), iCAPP's deviation from the optimal solution starts increasing and hence, we restrict the γ to 0.05.

Apache Flink, one of these stream processing systems, for efficiently processing the data streams at each edge, transit and destination.

Geo-distributed Data Analytics (GDA). A major portion of the prior work on GDA has proposed techniques for efficiently processing batch workloads in geo-distributed environments. Yugong [28], Pixida [34], Iridium [54], Tetrium [29], Clarinet [66], WANalytics [67] focus on reducing WAN bandwidth consumption while also minimizing query execution time in the context of batch analytics. Kimchi [50] is a recent cost-aware GDA system which explores the traffic cost-performance trade off but is only applicable for batch workloads. Kimchi proposes a stage-by-stage optimization and assumes that later stages do not contribute significantly to the total cost. But as we show in this work, this assumption does not hold true in case of streaming workloads.

For wide-area streaming workloads, JetStream [55], AWStream [73], and ApproxIoT [69] trade off accuracy with bandwidth consumption. These works are cost-agnostic and hence, can significantly inflate the bandwidth usage cost as we show in this work. Furthermore, these works assume that it is generally beneficial to route the data streams from the edges to its nearest DC or a common DC but we show in this work that this may not be the best choice. Finally, these works assume that it is always beneficial to perform maximum aggregation at the first stage i.e. at the edges. Our work shows that performing a portion of the aggregation at the later stage (i.e. transit) can be more beneficial. Sana [32] focuses on multi-query optimization for streaming analytics in a wide-area environment so as to minimize WAN consumption and is also cost-agnostic. Moreover, techniques such as quality degradation, sampling and multi-query optimization proposed by some of the above mentioned works compliment our proposed techniques.

Aggregation. Aggregation forms a prominent component of data analytics. Existing work has looked at aggregation from diverse perspectives. CoopStore [19] propose techniques for minimizing error in approximate aggregation queries subject to memory constraints, while Gan et al. [20] uses statistical moments to speed up approximate aggregation queries. GRETA [52] and COGRA [53] propose algorithmic improvements to optimize real-time event trend aggregation. FiBA [61] proposes usage of finger trees for sliding window aggregation over out-of-order data streams. LIGHTSABER [62] presents a streaming engine for exploiting both parallelism and incremental processing for efficient window aggregation on multi-core processors. All of the above mentioned works optimize different metrics such as error, memory usage, query execution time etc. within a single centralized data center environment where WAN bandwidth is not a constraint. On the contrary, our work focuses on minimizing traffic cost in a geo-distributed environment with constrained and expensive WAN bandwidth.

Kumar et al. [38] propose a TTL-based aggregation mechanism for geo-distributed streaming analytics. It provides a theoretical basis for relating delay with WAN traffic which we use a foundational building block in our problem formulation. Heintz et al. [25, 26] study tradeoffs between delay, traffic and accuracy in the context of windowed grouped aggregation for geo-distributed environment. These works consider a simple hub-and-spoke model for aggregation and are cost-agnostic whereas we propose aggregation networks which perform aggregation over a general multi-tiered edge-cloud topology in a cost-aware manner. Aggregation has also been studied in sensor networks [56] which focuses on energy efficient data aggregation to enhance network lifetime while we focus on a different delay-traffic cost tradeoff in a WAN environment.

Query Optimization. Query optimization is a widely studied topic in relational databases [8, 9, 14, 43, 44, 58]. The relevant literature covers techniques for optimizing general relational queries with the main goal of minimizing the query execution time for databases in intra-data center environments where network bandwidth is homogeneous and sufficient. On the contrary, our work specifically focuses on optimizing the aggregation-based operators with the goal of minimizing the traffic cost in a geo-distributed streaming environment where WAN bandwidth is scarce and expensive.

Relationship to Network Flow. While on the surface our problem may appear similar to network flow [18, 22, 33], transshipment [27], and other related problems [15, 74], our problem is very different from these problems. The main reason is that flow conservation does not hold in our problem, while flow conservation holds and is key to solving network flow and other related problems. In our problem, the amount of outgoing traffic flow is smaller than the incoming traffic flow due to data aggregation. Further, the reduction in flow is non-linear and is a function of several variables like the arrival rate of keys, duration of aggregation, etc. This makes our problem very different from the prior work.

Relationship to Content Delivery Networks (CDNs). Our work on aggregation networks is different from the existing work on CDNs [17, 42, 49]. CDNs route data (i.e., web and video content) from the cloud where the data originates to user devices via edges and transits. Aggregation networks have exactly the opposite flow of data: from user devices to the cloud via edges and transits. Further, unlike CDNs, aggregation networks provide a non-linear reduction in the traffic flow from edge to the cloud. However, the real-time live video streaming architecture of a CDN is often also a multi-tiered network [6, 35, 42, 60] of "entry points" where stream enters the network and "reflectors" that play the role of transit nodes to relay the stream to the edges that in turn serve user devices.

8 CONCLUSION

We studied aggregation networks for streaming workloads in the context of traffic cost minimization. We identify the various opportunities for reducing the traffic cost such as selecting cheaper and common transits for as many edges as possible and also distributing the delay budget between edge and transit in an intelligent manner. We provide optimization-based and heuristic algorithms to minimize the traffic cost of aggregation, while obeying resource constraints and the delay budget. Our extensive evaluation shows a 47% to 83% reduction in traffic cost compared to existing approaches.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for many constructive comments and suggestions that greatly improved the quality of this paper. This work was sponsored in part by NSF under Grants CNS-1717834 and CNS-1717179, as well as by DARPA contract HR001117C0049.

REFERENCES

[1] Akamai Download Analytics solution. Accessed: 2020-08-30. https://www.akamai.com/us/en/multimedia/documents/product-brief/akamai-

download-delivery-product-brief.pdf.
[2] Akamai Media Analytics. Accessed: 2020-08-30. https://www.akamai.com/us/en/multimedia/documents/product-brief/media-analytics-product-brief.pdf.
[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. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 1033–1044. https://doi.org/10.14778/2536222.2536229
[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, out-of-Order Data Processing. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1792–1803. https://doi.org/10.14778/2824032.2824076
[5] Amazon EC2 Pricing. Accessed: 2020-03-22. https://aws.amazon.com/ec2/pricing/on-demand/.
[6] Konstantin Andreev, Bruce M Maggs, Adam Meyerson, and Ramesh K Sitaraman. 2003. Designing overlay multicast networks for streaming. In *ACM SPAA*. 149–158.
[7] Azure Pricing. Accessed: 2020-03-22. https://azure.microsoft.com/en-us/pricing/details/bandwidth/.
[8] Brian Babcock and Surajit Chaudhuri. 2005. Towards a Robust Query Optimizer: A Principled and Practical Approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, Maryland) *(SIGMOD '05)*. Association for Computing Machinery, New York, NY, USA, 119–130. https://doi.org/10.1145/1066157.1066172
[9] Philip A. Bernstein and Dah-Ming W. Chiu. 1981. Using Semi-Joins to Solve Relational Queries. *J. ACM* 28, 1 (Jan. 1981), 25–40. https://doi.org/10.1145/322234.322238
[10] Oscar Boykin, Sam Ritchie, Ian O'Connell, and Jimmy Lin. 2014. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1441–1451. https://doi.org/10.14778/2733004.2733016
[11] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1718–1729. https://doi.org/10.14778/3137765.3137777
[12] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38, 4 (2015), 28–38. http://sites.computer.org/debull/A15dec/p28.pdf
[13] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. 2010. FlumeJava: Easy, Efficient Data-Parallel Pipelines. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation* (Toronto, Ontario, Canada) *(PLDI '10)*. Association for Computing Machinery, New York, NY, USA, 363–375. https://doi.org/10.1145/1806596.1806638
[14] Surajit Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Seattle, Washington, USA) *(PODS '98)*. Association for Computing Machinery, New York, NY, USA, 34–43. https://doi.org/10.1145/275487.275492
[15] Brian Cho and Indranil Gupta. 2011. Budget-Constrained Bulk Data Transfer via Internet and Shipping Networks. In *Proceedings of the 8th ACM International Conference on Autonomic Computing* (Karlsruhe, Germany) *(ICAC '11)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/1998582.1998595
[16] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492
[17] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. 2002. Globally distributed content delivery. *IEEE Internet Computing* 6, 5 (2002), 50–58.
[18] Lisa Fleischer and Martin Skutella. 2007. Quickest Flows Over Time. *SIAM J. Comput.* 36, 6 (2007), 1600–1630. https://doi.org/10.1137/S0097539703427215 arXiv:https://doi.org/10.1137/S0097539703427215
[19] Edward Gan, Peter Bailis, and Moses Charikar. 2020. CoopStore: Optimizing Precomputed Summaries for Aggregation. *Proc. VLDB Endow.* 13, 11 (2020), 2174–2187. http://www.vldb.org/pvldb/vol13/p2174-gan.pdf
[20] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-Based Quantile Sketches for Efficient High Cardinality Aggregation Queries. *Proc. VLDB Endow.* 11, 11 (July 2018), 1647–1660. https://doi.org/10.14778/3236187.3236212
[21] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. 2007. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *2007 IEEE 10th International Symposium on Workload Characterization*. 171–180.
[22] Andrew V Goldberg, Éva Tardos, and Robert E Tarjan. 1989. *Network flow algorithms*. Technical Report. PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE.
[23] Google Cloud Pricing. Accessed: 2020-03-22. https://cloud.google.com/network-tiers/pricing.
[24] G Hayes. Accessed: 2020-03-22. mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose.
[25] Benjamin Heintz, Abhishek Chandra, and Ramesh K. Sitaraman. 2016. Trading Timeliness and Accuracy in Geo-Distributed Streaming Analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 361–373. https://doi.org/10.1145/2987550.2987580
[26] Benjamin Heintz, Abhishek Chandra, and Ramesh K. Sitaraman. 2020. Optimizing Timeliness and Cost in Geo-Distributed Streaming Analytics. *IEEE Trans. Cloud Comput.* 8, 1 (2020), 232–245. https://doi.org/10.1109/TCC.2017.2750678
[27] Bruce Hoppe and Éva Tardos. 2000. The Quickest Transshipment Problem. *Mathematics of Operations Research* 25, 1 (2000), 36–62. http://www.jstor.org/stable/3690422
[28] Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, and Jingren Zhou. 2019. Yugong: Geo-Distributed Data and Job Placement at Scale. *Proc. VLDB Endow.* 12, 12 (Aug. 2019), 2155–2169. https://doi.org/10.14778/3352063.3352132
[29] Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. 2018. Wide-Area Analytics with Multiple Resources. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 12, 16 pages. https://doi.org/10.1145/3190508.3190528
[30] Jean-François Im, Kishore Gopalakrishna, Subbu Subramaniam, Mayank Shrivastava, Adwait Tumbde, Xiaotian Jiang, Jennifer Dai, Seunghyun Lee, Neha Pawar, Jialiang Li, and Ravi Aringunram. 2018. Pinot: Realtime OLAP for 530 Million Users. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 583–594. https://doi.org/10.1145/3183713.3190661
[31] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (Lisbon, Portugal) *(EuroSys '07)*. Association for Computing Machinery, New York, NY, USA, 59–72. https://doi.org/10.1145/1272996.1273005
[32] Albert Jonathan, Abhishek Chandra, and Jon Weissman. 2018. Multi-Query Optimization in Wide-Area Streaming Analytics. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 412–425. https://doi.org/10.1145/3267809.3267842
[33] Bettina Klinz and Gerhard J. Woeginger. 2004. Minimum-cost dynamic flows: The series-parallel case. *Networks* 43, 3 (2004), 153–162. https://doi.org/10.1002/net.10112 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.10112
[34] Konstantinos Kloudas, Margarida Mamede, Nuno Preguiça, and Rodrigo Rodrigues. 2015. Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics. *Proc. VLDB Endow.* 9, 2 (Oct. 2015), 72–83. https://doi.org/10.14778/2850578.2850582
[35] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. 2004. A transport layer for live streaming in a content delivery network. *Proc. IEEE* 92, 9 (2004), 1408–1419.
[36] KSQL: Streaming SQL for Kafka. Accessed: 2018-10-29. https://www.confluent.io/product/ksql/.
[37] 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* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 239–250. https://doi.org/10.1145/2723372.2742788
[38] Dhruv Kumar, Jian Li, Abhishek Chandra, and Ramesh Sitaraman. 2019. A TTL-Based Approach for Data Aggregation in Geo-Distributed Streaming Analytics. In *ACM SIGMETRICS*. New York, NY, USA, Article 29, 27 pages. https://doi.org/10.1145/3341617.3326144
[39] Fan Lai, Mosharaf Chowdhury, and Harsha Madhyastha. 2018. To Relay or Not to Relay for Inter-Cloud Transfers?. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotcloud18/presentation/lai
[40] LINQ (Language Integrated Query). Accessed: 2020-03-22. https://docs.microsoft.com/en-us/dotnet/standard/using-linq.
[41] Linux traffic control. Accessed: 2020-08-30. https://man7.org/linux/man-pages/man8/tc.8.html.
[42] Bruce M Maggs and Ramesh K Sitaraman. 2015. Algorithmic nuggets in content delivery. *ACM SIGCOMM CCR* 45 (2015), 52–66.
[43] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (July 2019), 1705–1718. https://doi.org/10.14778/3342263.3342644
[44] J. K. Mullin. 1990. Optimal Semijoins for Distributed Database Systems. *IEEE Trans. Softw. Eng.* 16, 5 (May 1990), 558–560. https://doi.org/10.1109/32.52778

[45] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A Timely Dataflow System. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 439–455. https://doi.org/10.1145/2517349.2522738

[46] Netflix. Accessed: 2020-03-19. https://www.netflix.com.

[47] Netflix Keystone Real-time Stream Processing Platform. Published: 2018-09-10. https://netflixtechblog.com/keystone-real-time-stream-processing-platform-a3ee651812a.

[48] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. 2017. Samza: Stateful Scalable Stream Processing at LinkedIn. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1634–1645. https://doi.org/10.14778/3137765.3137770

[49] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. 2010. The Akamai Network: A Platform for High-performance Internet Applications. *SIGOPS Oper. Syst. Rev.* 44, 3 (2010), 2–19.

[50] Kwangsung Oh, Abhishek Chandra, and Jon B. Weissman. 2020. A Network Cost-aware Geo-distributed Data Analytics System. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020, Melbourne, Australia, May 11-14, 2020*. IEEE, 649–658. https://doi.org/10.1109/CCGrid49817.2020.00-28

[51] Meikel Poess, Raghunath Nambiar, Karthik Kulkarni, Chinmayi Narasimhadevara, Tilmann Rabl, and Hans-Arno Jacobsen. 2018. Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 1519–1530. https://doi.org/10.1109/ICDE.2018.00170

[52] Olga Poppe, Chuan Lei, Elke A. Rundensteiner, and David Maier. 2017. GRETA: Graph-Based Real-Time Event Trend Aggregation. *Proc. VLDB Endow.* 11, 1 (Sept. 2017), 80–92. https://doi.org/10.14778/3151113.3151120

[53] Olga Poppe, Chuan Lei, Elke A. Rundensteiner, and David Maier. 2019. Event Trend Aggregation Under Rich Event Matching Semantics. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 555–572. https://doi.org/10.1145/3299869.3319862

[54] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-Distributed Data Analytics *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 421–434. https://doi.org/10.1145/2785956.2787505

[55] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. 2014. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 275–288. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/rabkin

[56] Ramesh Rajagopalan and Pramod K. Varshney. 2006. Data-aggregation techniques in sensor networks: A survey. *IEEE Commun. Surv. Tutorials* 8, 1-4 (2006), 48–63. https://doi.org/10.1109/COMST.2006.283821

[57] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education. http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html

[58] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data* (Boston, Massachusetts) *(SIGMOD '79)*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/582095.582099

[59] Zhiming Shen, Qin Jia, Gur-Eyal Sela, Ben Rainero, Weijia Song, Robbert van Renesse, and Hakim Weatherspoon. 2016. Follow the Sun through the Clouds: Application Migration for Geographically Shifting Workloads. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 141–154. https://doi.org/10.1145/2987550.2987561

[60] Ramesh K Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. 2014. Overlay networks: An akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services* (2014), 305–328.

[61] Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. 2019. Optimal and General Out-of-Order Sliding-Window Aggregation. *Proc. VLDB Endow.* 12, 10 (June 2019), 1167–1180. https://doi.org/10.14778/3339490.3339499

[62] Georgios Theodorakis, Alexandros Koliousis, Peter Pietzuch, and Holger Pirk. 2020. LightSaber: Efficient Window Aggregation on Multi-Core Processors. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2505–2521. https://doi.org/10.1145/3318464.3389753

[63] Twitter. Accessed: 2020-03-19. https://www.twitter.com.

[64] Twitter Analytics. Accessed: 2020-08-30. https://business.twitter.com/en/analytics.html.

[65] Twitter Developer APIs. Accessed: 2020-08-30. https://developer.twitter.com/en/docs.

[66] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. CLARINET: WAN-Aware Optimization for Analytics Queries *(OSDI'16)*. USENIX Association, USA, 435–450.

[67] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. 2015. WANalytics: Geo-Distributed Analytics for a Data Intensive World. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1087–1092. https://doi.org/10.1145/2723372.2735365

[68] Hao Wang, Di Niu, and Baochun Li. 2018. Dynamic and Decentralized Global Analytics via Machine Learning. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 14–25. https://doi.org/10.1145/3267809.3267812

[69] Zhenyu Wen, Do Le Quoc, Pramod Bhatotia, Ruichuan Chen, and Myungjin Lee. 2018. ApproxIoT: Approximate Analytics for Edge Computing. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*. IEEE Computer Society, 411–421. https://doi.org/10.1109/ICDCS.2018.00048

[70] Windows API in Apache Flink. Accessed: 2020-10-23. https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/windows.html.

[71] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. 2014. Druid: A Real-Time Analytical Data Store. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/2588555.2595631

[72] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 423–438. https://doi.org/10.1145/2517349.2522737

[73] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. 2018. AWStream: Adaptive Wide-Area Streaming Analytics *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 236–252. https://doi.org/10.1145/3230543.3230554

[74] Linquan Zhang, Chuan Wu, Zongpeng Li, Chuanxiong Guo, Minghua Chen, and Francis C. M. Lau. 2013. Moving Big Data to The Cloud: An Online Cost-Minimizing Approach. *IEEE J. Sel. Areas Commun.* 31, 12 (2013), 2710–2721. https://doi.org/10.1109/JSAC.2013.131211