AWStream: Adaptive Wide-Area Streaming Analytics

Paper #8

Abstract

The emerging class of wide-area streaming analytics faces the challenge of scarce and variable WAN bandwidth. Non-adaptive applications built with TCP or UDP suffer from increased latency or degraded accuracy. State-of-the-art approaches that adapt to network changes require developer writing sub-optimal manual policies or are limited to application-specific optimizations.

We present AWStream, a stream processing system that simultaneously achieves low latency and high accuracy in the wide area, requiring minimal developer efforts. To realize this, AWStream uses three ideas: (*i*) it integrates application adaptation as a first-class programming abstraction in the stream processing model; (*ii*) with a combination of offline and online profiling, it automatically learns an accurate profile that models accuracy and bandwidth trade-off; and (*iii*) at runtime, it carefully adjusts the application data rate to match the available bandwidth while maximizing the achievable accuracy. We evaluate AWStream with three real-world applications: augmented reality, pedestrian detection, and monitoring log analysis. Our experiments show that AWStream achieves sub-second latency with only nominal accuracy drop (2-6%).

1 Introduction

Wide-area streaming analytics are becoming pervasive, especially with the emerging class of Internet of Things (IoT) applications. Large cities such as London and Beijing have deployed millions of cameras for surveillance and traffic control [37, 69]. Buildings are increasingly equipped with a wide variety of sensors to improve energy efficiency and occupant comfort [35]. Geo-distributed infrastructure, such as content delivery networks (CDNs), analyzes requests from machine logs over the globe [44]. These applications need to transport, distill, and process streams of data across the wide area in real time.

Although existing stream processing systems, such as Storm [70], Spark Streaming [79], and VideoStorm [80], can handle large streams of data, they are designed to work within a single cluster, where the network is not the bottleneck. In contrast, the wide area network (WAN) has limited bandwidth [29, 75]. Moreover, WAN bandwidth growth has been decelerating for many years [68] while traffic demands are growing at a staggering rate [31].

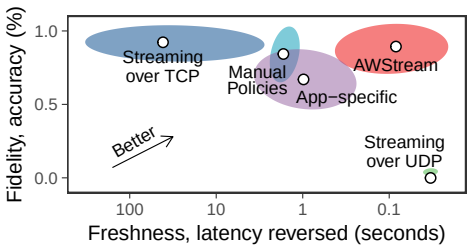


Figure 1: The trade-off space between data freshness and fidelity when facing insufficient bandwidth (details in §5.3).

Limited WAN bandwidth makes it neither practical nor efficient to back-haul all data to a central location. Recent research on WAN-aware systems promotes pushing computations towards the edge [53, 59]. However, communication is not entirely avoidable: (*i*) some analytical jobs require joining or aggregating data from multiple geo-distributed sites [52, 73]; (*ii*) the edge can benefit substantially from central computing resources such as GPUs and TPUs [2] in the cloud; and (*iii*) end devices, such as cameras and mobile phones, still suffer from limited bandwidth in last-hop wireless links even when the processing runs on nearby edge infrastructure [3, 81].

When facing insufficient bandwidth, application developers need to make a decision within the design space of data fidelity versus freshness, as illustrated in Fig. 1.

Applications using existing protocols without adaptation result in extreme design points. Streaming over TCP ensures a reliable delivery but backlogged data increases latency. On the other hand, streaming over UDP minimizes latency by sending packets as fast as possible, but uncontrolled packet loss devastates application accuracy.

Manual policies, such as sampling, allow developers to trade data fidelity for freshness [53]. However, it's difficult to write accurate policies without extensive expertise or without considerable effort. In practice, developers write manual policies based on heuristics rather than quantitative measurements. These inaccurate policies lead to sub-optimal performance for both freshness and fidelity.

Furthermore, application-specific optimizations do not generalize. A fine-tuned adaptation algorithm for one application works poorly for a different application, if performance metrics or data distributions change. For example, video streaming focuses on quality of experience (QoE) [42, 50, 78]. Because humans favor smoothness over image quality, these systems maintain a high frame

rate, e.g. 25 FPS, and reduce the resolution under bandwidth limitation. Low resolution images lead to poor accuracy for video analytics that rely on image details [39, 72].

In this paper, we design and implement AWStream, a stream processing system for the wide area that achieves low latency and high accuracy simultaneously with minimal developer effort. The key idea is to *automatically* build an accurate and precise *performance model* instead of relying on manual policies or application-specific optimizations. AWStream's solution is three-fold: an easy-to-use API, automatic profiling, and a low-latency runtime.

AWStream augments existing stream processing operators with a new `maybe` operator. Its basic form takes a list of values as a knob and a function that degrades the input stream. The knob specifies the degradation level that affects data size and data fidelity. We extend the basic form with a library of specialized operators for common data types, such as `maybe_downsample` for images. Our API is *simple*, *composable* and *extensible*. Developers do not need to be an expert in the application domain as the knobs tolerate approximate specifications. Multiple operators form a configuration that affects the adaptation jointly. Arbitrary functions and external libraries can be embedded with our operators.

AWStream then uses a data-driven approach to automatically build application performance profiles with minimal developer effort. The profiles accurately capture the relationship between application accuracy and bandwidth consumption under different *combinations* of data degradation operations. We use an *offline* process to bootstrap our system with developer-supplied training data, and continuously refine the profile *online* to handle *model drift*. We exploit parallelism and sampling-based techniques to efficiently explore the configuration space and learn a *Pareto-optimal* adaptation strategy.

At runtime, AWStream achieves low latency by matching data rate to available bandwidth, and high accuracy by using Pareto-optimal configurations from the profile. Upon encountering network congestion, our adaptation algorithm increases the degradation level to reduce data rate, such that no persistent queue builds up. To recover, it progressively decreases the degradation level after probing for more available bandwidth. The runtime also provides additional options to control application behaviors, such as limiting the maximum allowed WAN bandwidth. For multiple applications, the profiles can be used to allocate bandwidth among competing tasks for *utility fairness*.

To evaluate AWStream, we have built three streaming applications: augmented reality (AR), pedestrian detection (PD), and distributed Top-K (TK). We use real-world data to profile these applications and evaluate their runtime performance on a geo-distributed public cloud. Our contributions and evaluation results are as follows.

- We propose `maybe` operators to incorporate adaptation with existing stream processing models. Our programming abstraction is simple, composable and extensible.
- We show that AWStream's data-driven approach generates an accurate and precise profile for each application. Parallelism and sampling techniques can speed up the profiling substantially, up to 29× and 8.7×.
- Using runtime experiments on geo-distributed EC2 nodes, AWStream achieves sub-second latency and nominal accuracy drop for all applications. Compared with the state-of-the-art system JetStream, AWStream constructs adaptation policies automatically (both offline and online) and improves the runtime performance: latency by 15-20×, accuracy by 1-5%.

2 Motivation

In this section, we first examine the gap between high application demands and limited WAN bandwidth. We then show that neither manual policies nor application-specific optimizations solve the problem.

2.1 Wide-area Streaming Applications

Video Surveillance. We envisage a city-wide monitoring system that aggregates camera feeds, from stationary ground cameras and moving aerial vehicles, and analyzes video streams in real time for surveillance, anomaly detection, or business intelligence [48]. Recent advances in computer vision have dramatically increased the accuracy for automatic visual scene analysis, such as pedestrian detection [20], vehicle tracking [18], and facial recognition to locate people of interest [40, 51]. While some surveillance cameras use dedicated links, an increasing number of surveillance systems, such as Dropcam [26] and Vigil [81], use the public Internet and wireless links to reduce the cost of deployment and management.

Infrastructure Monitoring. Large organizations today are managing 10–100s of data centers (DCs) and edge clusters worldwide [13]. This geo-distributed infrastructure continuously produces large volumes of data such as data access logs, server monitoring logs, and performance counters [7, 52, 75]. While most log analysis today runs in a batch mode on a daily basis, there is a trend towards analyzing logs in real time for rapid optimization [53]. For example, CDNs can improve the overall efficiency by optimizing data placement if the access logs can be processed in real time. In Industrial IoT, large-scale real-time sensor monitoring is becoming pervasive to detect anomalies, direct controls, and predict maintenance [10, 25].

2.2 Wide-area Bandwidth Characteristics

WAN bandwidth is insufficient and costly, as demonstrated by recent WAN-aware systems [29, 52, 74, 75].

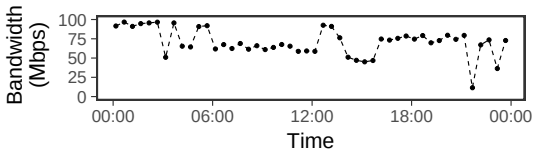


Figure 2: Bandwidth variations throughout the day between Amazon EC2 sites (from Ireland to California).

Using Amazon EC2 as a case study, the WAN bandwidth capacity is 15x smaller than their LAN bandwidth on average, and up to 60x smaller in the worst case [29]. In terms of pricing, the average WAN bandwidth cost is up to 38x of the cost of renting two machines [8, 29].

In addition to the scarcity and cost, the large variability of WAN bandwidth also affects streaming workloads. We conducted a day-long measurement with iPerf [22] to study the pair-wise bandwidth between four Amazon EC2 sites (N. California, N. Virginia, Tokyo, Ireland). The results show large variance in almost all pairs—Fig. 2 is one such pair.[1] There are occasions when the available bandwidth is below 25% of the maximum bandwidth.

The back-haul links between EC2 sites are better—if not at least representative—in comparison to general WAN links. Similar scarcity and variations have been reported in wireless networks [12], broadband access networks [27, 67] and cellular networks [46].

2.3 Motivation for AWStream

To address bandwidth limits, existing solutions use manual policies or application-specific solutions. We discuss their drawbacks to motivate AWStream (design in §3).

Manual polices are sub-optimal. JetStream [53] is the first wide-area analysis system that uses data degradation to address bandwidth limits. While effective in comparison to non-adaptive systems, JetStream requires developers to write manual policies, e.g. *"if bandwidth is insufficient, switch to sending images at 75% fidelity, then 50% if there still isn't enough bandwidth. Beyond that point, reduce the frame rate, but keep the image fidelity."*[2] We discuss the problems with manual policies below and present quantitative evaluations in §5.3.

First, this policy is not accurate. Developers write such rules based on heuristics and don't back them up with measurements. Images with 75% fidelity do not necessarily lead to 75% application accuracy. In terms of bandwidth, naively one would think that reducing the frame rate by half will also half the data rate. But if video encoding such as H.264 [56] is used, a reduction in frame rate increases the inter-frame difference and creates P-frames with larger sizes. Fig. 3e shows that when reducing the frame rate to 33% (from 30 FPS to 10 FPS), the bandwidth demand can still be more than 50%.

Second, it is not scalable to specify rules one by one. A fine-grain control requires many rules in the policy. Applications can degrade in multiple dimensions and each dimension has different impacts (compare Fig. 3a with Fig. 3b). Specifying rules in detail and across dimensions manually is a tedious and error-prone process.

Lastly, this abstraction is too low-level. It forces developers to study and measure the impact of individual operations, prohibiting its wide adoption in practice.

Application-specific optimizations do not generalize. Because applications have different performance metrics and rely on different features, a fine-tuned policy for one application will often work poorly for others. For example, DASH [65] optimizes QoE for video streaming; it keeps a high frame rate and reduces resolutions for adaptation. Its policy that lowers the resolution is a poor match for video analytics that relies on image details [39, 72]. For example, Fig. 3b shows that pedestrian detection accuracy drops fast when reducing resolutions as targets are small.

Similar applications face different data distributions, as shown in Fig. 3 between stationary cameras detecting pedestrians (up) and mobile cameras recognizing objects (bottom). For stationary cameras, when we consider the slow walking speed of pedestrians, a high frame rate is not necessary. But high-resolution images are crucial because these surveillance cameras are far away from the targets. In the mobile camera case, because the cameras move, reducing the frame rate introduces significant errors.

3 AWStream Design

AWStream avoids the problems with manual policies or application-specific optimizations by structuring adaptation as a set of approximate, modular and extensible specifications (§3.1). The well-defined structure allows us to build a generic profiling tool that learns an accurate relationship, the profile, between bandwidth consumption and application accuracy (§3.2). The profile then guides the runtime to react with precision: achieving low latency and high accuracy when facing insufficient bandwidth (§3.3). Fig. 4 shows the high-level overview of AWStream.

3.1 API for Structured Adaptation

Most stream processing systems construct applications as a directed graph of operators [70, 79], where each operator transforms input streams into new streams. AWStream borrows the same computation model. We list some example operators, such as `map` and `skip`, in Table 1.

To integrate adaptation as a first-class abstraction, AWStream introduces `maybe` operators that degrade data quality, yielding potential bandwidth savings. Our API de-

[1] The measurements of other pairs are in the appendix.
[2] Excerpt from JetStream §4.3 [53].

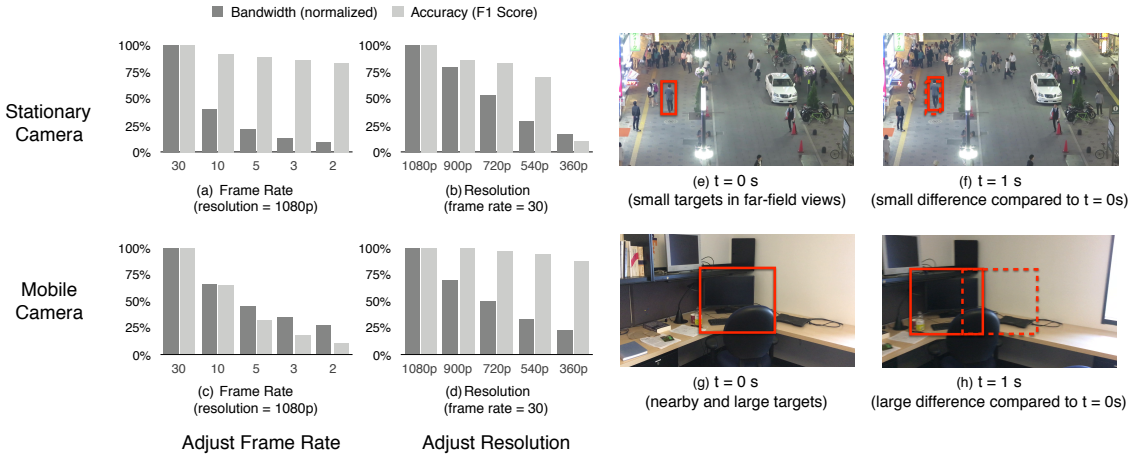


Figure 3: The measured bandwidth and application accuracy for two video analytics applications. (1) Manual policies lack precision without measurements and need to handle multiple dimensions (as in a-d). (2) Application-specific optimizes does not generalize: degrading frame rates works well for stationary camera (a), but not for mobile camera (c). (e-h) shows example frames.

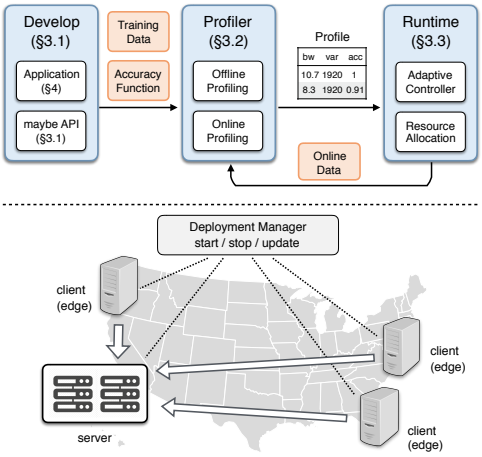


Figure 4: AWStream has three phases: development, profiling, and runtime. AWStream also manages wide-area deployment.

sign has three considerations. (*i*) To free developers from specifying exact rules, the API should tolerate approximate specifications. (*ii*) To allow combining multiple dimensions, the API should be modular: each operator is a unit and the developer can chain multiple operators. (*iii*) To support flexible integration with arbitrary degradation functions, the API should take user-defined functions (UDFs). Therefore, our API is,

```
maybe(knobs: Vec<T>, f: (T, I) => I)
```

We illustrate the use of the `maybe` operator with an example that quantizes a stream of integers in Rust:

```
let quantized_stream = vec![1, 2, 3, 4].into_stream()
  .maybe(vec![2, 4], |k, val| val.wrapping_div(k))
  .collect();
```

The snippet creates a stream of integers, chains a degradation operation, and collects the execution result. In this example, the knob is [2, 4], and the degradation function performs a wrapping (modular) division where the divisor is the chosen knob. The knob value modifies the quantization level, affecting the output: [1, 2, 3, 4] (no degradation), [0, 1, 1, 2] (k=2), or [0, 0, 0, 1] (k=4). If the stream is then encoded—e.g. run-length encoding as in JPEG [76]—for transmission, the bandwidth consumption will change according to the level of degradation.

Based on the `maybe` primitive, one can implement wrappers of degradation operations for common data types. For instance, `maybe_head` will optionally take the top values of a list; and `maybe_downsample` can adjust the image resolution to a configured target. AWStream provides a number of such operations as a library to simplify application development (Table 1).

With our API, the example mentioned in §2.3 can now be implemented as follows:

```
let app = Camera::new((1920, 1080), 30)
  .maybe_downsample(vec![(1600, 900), (1280, 720)])
  .maybe_skip(vec![2, 5])
  .map(|frame| frame.show())
  .compose();
```

This snippet first instantiates a `Camera` source, which produces `Stream<Image>` with 1920x1080 resolution and 30 FPS. Two degradation operations follow the source: one that downsamples the image to either 1600x900 or 1280x720 resolution, and the other that skips frames with a parameter of 2 or 5, resulting in 30/(2+1)=10 FPS or 30/(5+1)= 6 FPS. This example then shows degraded images on the display. In practice, operators for further processing, such as encoding and transmission, can be chained.

3.2 Automatic Profiling

After developers use `maybe` operators to specify potential degradation operations, AWStream automatically builds an accurate profile. The profile captures the relationship

Normal Operators	*map* (f: I ⇒ O)	Stream<I> ⇒ Stream<O>
	skip (i: Integer)	Stream<I> ⇒ Stream<I>
	sliding_window (count: Integer, f: Vec<I> ⇒ O)	Stream<I> ⇒ Stream<O>
	...	...
Degradation Operators	*maybe* (knobs: Vec<T>, f: (T, I) ⇒ I)	Stream<I> ⇒ Stream<I>
	maybe_skip (knobs: Vec<Integer>)	Stream<I> ⇒ Stream<I>
	maybe_head (knobs: Vec<Integer>)	Stream<Vec<I>> ⇒ Stream<Vec<I>>
	maybe_downsample (knobs: Vec<(Integer, Integer)>)	Stream<Image> ⇒ Stream<Image>
	...	...

Table 1: Stream processing operators in AWStream. `Vec<T>` represents a list of elements with type `T`.

Symbol	Description
n	number of degradation operations
k_i	the i-th degradation knob
$c = [k_1, k_2, ...k_n]$	one specific configuration
$\mathbb{C}$	the set of all configurations
$B(c)$	bandwidth requirement for c
$A(c)$	accuracy measure for c
$\mathbb{P}$	Pareto-optimal set
c_i, c_{i+1}, $c_{\max}$	current/next/maximal configuration at runtime
R	network delivery rate (estimated bandwidth)
$\mathrm{Q_E}$, $\mathrm{Q_C}$	messages when `Queue` is empty or congested
$\mathrm{R_C}$	message when `Receiver` detects congestion
$\mathrm{AC_{Probe}}$	message when `AC` requests probing
$\mathrm{S_{ProbeDone}}$	message when `Socket` finishes probing

Table 2: Notations used in this paper.

between *application accuracy* and *bandwidth consumption* under different combinations of data degradation operations. We describe the formalism, followed by techniques that efficiently perform offline and online profiling.

Profiling formalism. Suppose a stream processing application has n `maybe` operators. Each operator introduces a knob k_i and their combination forms a *configuration* $c = [k_1, k_2, ...k_n]$. The set of all possible configurations $\mathbb{C}$ is the space that the profiling explores. For each configuration c, there are two mappings that are of particular interest: a mapping from c to its bandwidth consumption $B(c)$ and its accuracy measure $A(c)$. Table 2 summarizes the notations used in this paper.

The profiling looks for Pareto-optimal configurations; that is, for any configuration c in the Pareto-optimal set, there is no alternative configuration c' that requires less bandwidth and offers a higher accuracy. The Pareto-optimal set $\mathbb{P}$ is defined mathematically as follows:

$$\mathbb{P} = \{c \in \mathbb{C} : \{c' \in \mathbb{C} : B(c') < B(c), A(c') > A(c)\} = \varnothing\} \quad (1)$$

Because AWStream allows arbitrary functions as the degradation functions, it does not assume a closed-form relationship for $B(c)$ and $A(c)$. Instead, AWStream takes a data-driven approach: profiling applications with developer-supplied training data. We measure $B(c)$ at the point of *transmission*. The accuracy $A(c)$ is measured either against the *ground-truth*, or the reference results when *all degradation operations are off*. We show examples of concrete knobs, configurations, accuracy functions when we present applications in Table 3.

Offline Profiling. We first use an offline process to build a bootstrap profile (or default profile). AWStream makes no assumptions on the performance models, and thus evaluates all possible configurations. While all knobs form a combinatorial space, the offline profiling is only a one-time process. We exploit parallelism to reduce the profiling time. Without any *a priori* knowledge, all configurations are assigned randomly to available machines.

Online Profiling: AWStream runs an online profiling process continuously to refine the profile. The refinement handles *model drift*, a problem when the learned profile fails to predict the performance accurately. There are two challenges with online profiling. (i) There are no ground-truth labels or reference data to compute accuracy. Because labeling data is prohibitively labor intensive and time consuming [58], AWStream currently uses raw data (data without degradation) as the reference. At runtime, if the application streams raw data, it is used for online profiling. Otherwise, we allocate additional bandwidth to transmit raw data, but only do so when there is spare capacity. (ii) Exhaustive profiling is expensive. If the profiling takes too much time, the newly-learned profile may already be stale. AWStream uses a combination of parallelization and sampling to speed up profiling, as below:

- Parallelization with degradation-aware scheduling. Evaluating each configuration takes a different amount of time. Typically, an increase in the level of degradation leads to a decrease in computation; for example, a smaller FPS means fewer images to process. Therefore, we collect processing times for each configuration from the offline process and schedule online profiling with longest first schedule (LFS) [32] during parallelization.
- Sampling-based profiling. Online profiling can speed up if we only profile a subset of all data or a subset of all configurations. Sampling data reduces the amount of data to process, but at a cost of generating less accurate profiles. When sampling configuration, we can evaluate a subset of the Pareto-optimal configurations and compare their performances with the existing pro-

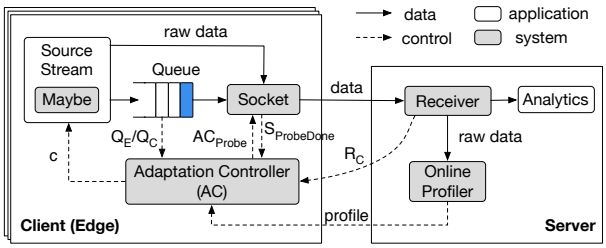


Figure 5: Runtime adaptation system architecture.

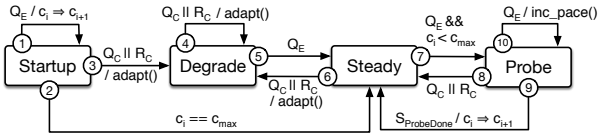
file. A substantial difference, such as more than 1 Mbps of bandwidth estimation, triggers a full profiling over all configurations to update the current profile.

3.3 Runtime Adaptation

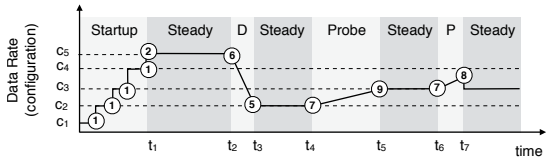
At runtime, AWStream matches data rate to available bandwidth to minimize latency and uses Pareto-optimal configurations to maximize accuracy.

Fig. 5 shows our runtime system architecture. AWStream applications' source contains a `Maybe` module derived from all `maybe` operators. This module allows the controller to update the level of degradation. Data generated by the source is then enqueued to `Queue` and subsequently dequeued by `Socket`, which sends data over the network using TCP. When the data generation rate exceeds `Socket`'s departure rate, the queue grows. In this case, the adaptation controller (AC) queries the estimated bandwidth from `Socket` and regulates the source stream by updating the configuration. After the data is sent through the network, `Receiver` delivers data to the application analytics. `Receiver` also performs congestion detection and extracts raw data, if it is present. It tracks the minimal latency (similar to how BBR tracks `RTprop` [15] within a filter window) and reports sudden latency spikes to clients as congestion signals. Raw data is only transmitted when the queue is empty and online profiling is enabled. If a new profile is learned by the online profiler, it is fed back to AC for subsequent adaptation.

Fig. 6a shows the adaptation algorithm with a state machine model and Fig. 6b shows the state transitions with an example. We first describe all symbols. AC loads the profile and sorts all configurations with an ascending order of bandwidth demand, resulting in a list $[c_1, \ldots, c_{\max}]$. These configurations follow a total order: $c_i < c_j$ if $B(c_i) < B(c_j)$. We denote the current configuration as c_i and the next c_{i+1}. AC receives messages from other modules: Q_E when `Queue` is empty; Q_C when queued items exceed a threshold; and R_C when `Receiver` detects congestion. AC can query `Socket` for delivery rate R (arrow not shown) or request it to probe (AC_{Probe}) for a target bandwidth, often $B(c_{i+1})$. If there is no congestion during the probing and $R > B(c_{i+1})$, `Socket` sends back $S_{ProbeDone}$. Below, we describe states and transitions.



(a) Rate adaptation as a state machine.



(b) An example illustrating the adaptation algorithm.

Figure 6: Runtime adaptation algorithm.

- **Startup: rapid growth.** AWStream starts with c_1 and grows the rate ($c_i \Rightarrow c_{i+1}$) upon each Q_E. The growth stops at $c_{\max}$ (to `Steady`) or if it receives Q_C/R_C (to `Degrade`).
- **Degrade: reacting to congestion.** Congestion is detected in two ways: (1) when `Queue` grows and exceeds a threshold, AC receives Q_C; (2) when `Receiver` detects latency spikes, AC receives R_C. During congestion, AC runs the `adapt()` procedure by updating `Maybe` with the maximum-allowed c that satisfies $B(c) < \alpha R$, where $\alpha \in (0,1)$ and R is `Socket`'s current delivery rate. A smaller α allows a quicker draining of the queue. After the congestion is resolved (Q_E received), AWStream changes to `Steady`.
- **Steady: low latency delivery.** AWStream achieves low latency by spending most of the time in the `Steady` state. It changes to `Degrade` when congestion occurs. If $c < c_{\max}$ and it receives Q_E, AC enters the `Probe` state to check for more available bandwidth.
- **Probe: more bandwidth for a higher accuracy.** Advancing the configuration directly causes a drastic latency increase when $B(c_{i+1}) \gg B(c_i)$. To allow a smooth increase, AC requests `Socket` to probe by sending additional traffic controlled by `probe_gain` (in `inc_pace()`, similar to BBR [15]). AWStream stops probing under two conditions: (1) upon $S_{ProbeDone}$, it advances c_i; (2) upon Q_C or R_C, it returns to `Steady`.

3.4 Resource Allocation & Fairness

In addition to rate adaptation, the profile is also useful for controlling a single application's bandwidth usage or allocating resources among competing tasks.

For individual applications, developers can pin-point a configuration for a given bandwidth or accuracy goal. They can also specify a criterion to limit effective configurations. For example, AWStream can enforce an upper

Application	Knobs	Accuracy	Dataset
Augmented Reality	resolution frame rate quantization	F1 score [57]	iPhone video clips training: office (24 s) testing: home (246 s)
Pedestrian Detection	resolution frame rate quantization	F1 score	MOT16 [43] training: MOT16-04 testing: MOT16-03
Log Analysis (Top-K, K=50)	head (N) threshold (T)	Kendall's τ [4]	SEC.gov logs [47] training: 4 days testing: 16 days

Table 3: AWStream Applications

bound on the bandwidth consumption. In this way, developers can control applications' bandwidth costs while using the profile to maximize accuracy.

For multiple applications, their profiles allow novel bandwidth allocation schemes such as utility fairness. Different from traditional resource fairness with which applications get an equal share of bandwidth, utility fairness aims to maximize the *minimal* application accuracy. With the profiles, bandwidth allocation is equivalent to finding proper configuration c^t for application t. We formulate utility fairness as follows:

$$\max_{c^t} \min(A^t(c^t)) \text{ s.t. } \sum_t B^t(c^t) < R \tag{2}$$

Solving this optimization is computationally hard. AWStream uses heuristics similar to VideoStorm [80]: it starts with c_1^t and improves the application t with the worst accuracy; this process iterates until all bandwidth is allocated.

4 Implementation

While our proposed API is general and not language specific, we have implemented AWStream prototype in Rust (~4000 lines of code). AWStream is open source on GitHub.[3] Applications use AWStream as a library and configure the execution mode—profiling, runtime as client, or runtime as server—with command line arguments. We have implemented the deployment manager that can start, stop, and update clients/servers as shell scripts.

Using AWStream, we have built three applications: augmented reality (AR) that recognizes nearby objects on mobile phones, pedestrian detection (PD) for surveillance cameras, and a distributed log analysis to extract the Top-K mostly accessed files (TK). Table 3 summarizes the application-specific part: knobs, accuracy function, and the dataset we used for training and testing. To conserve space, we leave application details in the appendix.

5 Evaluation

In this section, we show the evaluations of AWStream, summarizing the results as follows.

[3]URL elided for anonymity.

§5.1 AWStream generates Pareto-optimal profiles across multiple dimensions with precision (Fig. 7).

§5.2 Our parallel and sampling techniques speeds up offline and online profiling (Fig. 8, Fig. 9).

§5.3 At runtime, AWStream applications achieve sub-second latency and nominal accuracy drop across different network conditions (Fig. 10, Fig. 11, Fig. 12).

§5.4 AWStream profiles allow different resource allocations: resource fairness and utility fairness (Fig. 13).

5.1 Application Profiles

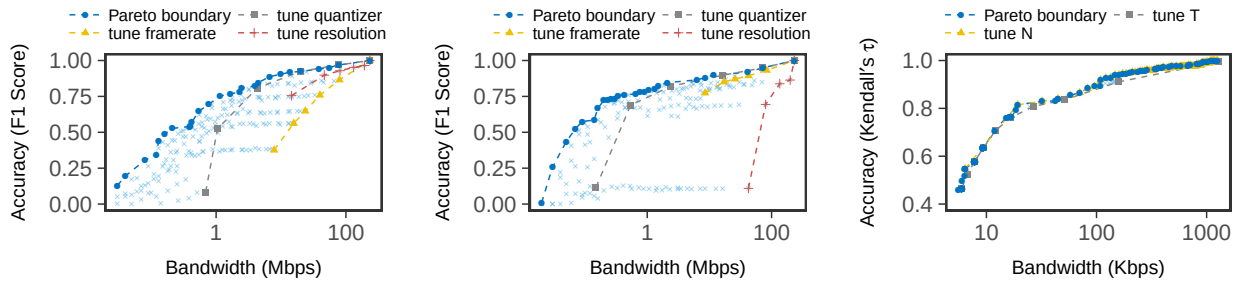
We run offline profiling using the training dataset described in Table 3 and show the learned profiles in Fig. 7. In each figure, the cross dots represent the bandwidth demand and application accuracy for one configuration. We highlight the Pareto-optimal boundary $\mathbb{P}$ with blue dashed lines. To understand each dimension's impact on the degradation, we highlight configurations from tuning only *one* dimension. From these profiles, we make the following observations:

Large bandwidth variation. For all three applications, The bandwidth requirements of all three applications have two to three orders of magnitude of difference (note the x-axis is in log scale). For AR and PD, the most expensive configuration transmits videos at 1920x1080, 30 FPS and 0 quantization; it consumes 230 Mbps. In contrast to the large bandwidth variation, there is a smaller variation in accuracy. In PD, for example, even after the bandwidth reduces to 1 Mbps (less than 1% of the maximum), the accuracy is still above 75%. The large variation allows AWStream to operate at a high accuracy configuration even under severe network deterioration.

Distinct effects by each dimension. Comparing dashed lines in each profile, we see that the Pareto-optimal configurations are only achievable when multiple knobs are in effect. Tuning only one dimension often leads to sub-optimal performance. Within a single profile, the difference between tuning individual dimensions is evident. For PD, tuning resolution (the red line) leads to a quicker accuracy drop than tuning frame rate (the yellow line). Comparing AR and PD, the same dimension has different impact. Tuning resolution is less harmful in AR than PD; while tuning frame rate hurts AR more than PD. This echoes our initial observation in §2.3 that application-specific optimizations do not generalize.

5.2 Profiling Efficiency & Online Profiling

This section focuses on the AR application as a case study; our profiling techniques—parallelism and sampling—do not make assumptions about the application; therefore, the evaluation results can be generalized to other applications.



(a) Augmented Reality (AR) **(b)** Pedestrian Detection (PD) **(c)** Top-K (TK)

Figure 7: Application profiles of three applications. Each cross point is one configuration c's performance $(B(c),A(c))$. All figures show the Pareto boundary as well as the performance if only tuning one dimension. Note the x-axis is in log scale.

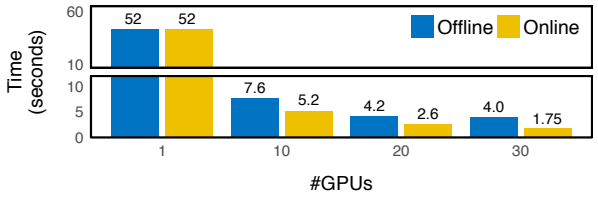


Figure 8: Parallelism speeds up both offline and online profiling. The y-axis shows the profiling time for 1-second video.

In AR, there are 216 different configurations: 6 resolutions, 6 frame rates and 6 quantization levels. AR uses YOLO [54], a neural network model for object detection. It takes roughly 30 ms to process one frame on GeForce® GTX 970.[4] But different configurations require different times for processing. For example, a 10 FPS video has 1/3 of the frames to process in comparison to a 30 FPS video. In our experiment, to evaluate all 216 configurations, it takes 52 seconds for 1 second worth of data. We denote such overhead as 52X. Section 3.2 have discussed parallel and sampling techniques to improve the profiling efficiency; we present their evaluations as follows.

Parallelism reduces the profiling time (Fig. 8). Because evaluating each individual configuration is independent of other configurations, we parallelize the profiling task by assigning configurations to GPUs. (i) Our offline profiling assigns configurations randomly. With the increased number of GPUs, the overhead reduces from 52X to 4X with 30 GPUs. (ii) Our online profiling assigns configurations based on the processing times collected during offline. AWStream uses LFS [32] to minimize the makespan and reduces the overhead to 1.75X with 30 GPUs (29× gain).

Sampling techniques speed up online profiling (Fig. 9). Before we evaluate the speed up, we validate *model drift* with real-world data. We use the profile trained in an office environment. According to the profile, the application should operate at a configuration of 1280x720, 30 FPS and 20 quantization to meet an 11 Mbps goal. We test it against a home environment, and at about t=100s, the camera points out of the window to detect objects on the street. Because of the scene change, the configuration fails to predict runtime bandwidth, as illustrated in Fig. 9a.

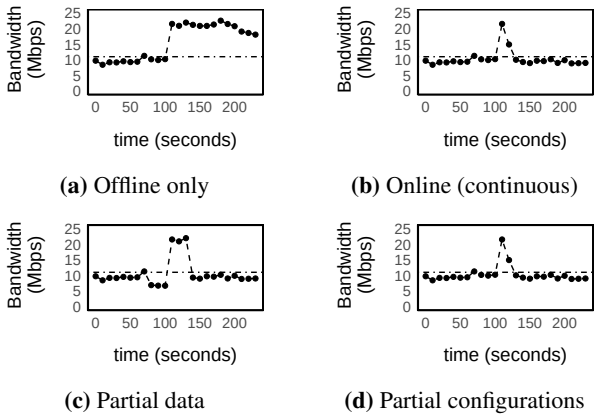
To correct the profile, if we continuously run the profiling online and update the profile, the application will choose the right configuration to meet the bandwidth limit. Fig. 9b shows the bandwidth prediction when we continuously profile with the past 30 seconds of video. At time t=120s, the new prediction corrects the drift. The downside of continuous profiling, as discussed earlier, is the cost: 52X overhead with 1 GPU. In addition to parallelism, AWStream uses sampling techniques for online profiling (improvements in Table 4):

(i) Partial data. Instead of using all the past data, we run profiling with only a fraction of the raw data. Fig. 9c shows the bandwidth consumption if the profiling uses only 10 seconds of data out of the past 30 seconds. In this way, although the profile may be less accurate (the mis-prediction at t=80-100s), and there is a delay in reacting to data change (the mis-prediction is corrected after t=125s), we save the online profiling by 3× (from 52X to 17X).

(ii) Partial configurations. If we use the past profile as a reference and only measure a subset of all Pareto-optimal configurations, the savings can be substantial. A full profiling is only triggered if there is a significant difference. Fig. 9d shows the bandwidth prediction if we evaluate 5 configurations continuously and trigger a full profiling when the bandwidth estimation is off by 1 Mbps or the accuracy is off by 10%. For our test data, this scheme is enough to correct model drifts by predicting an accurate bandwidth usage (compare Fig. 9b and Fig. 9d). The overhead reduces to 6X because we run full profiling less often (only two full profiling). It is an 8.7× gain.

Note that these techniques—parallelization, sampling data, and sampling configurations—can be combined together to further reduce the profiling overhead. For example, scheduling five GPUs running 5 configurations

[4] YOLO resizes an input image to a fixed resolution (416x416) as required by the neural network. It takes a similar amount of time to Evaluating each image (with different resolutions).



(a) Offline only

(b) Online (continuous)

(c) Partial data

(d) Partial configurations

Figure 9: The horizontal reference line is the target bandwidth (11 Mbps). (1) Online profiling is necessary to handle model drift ((a) vs. (b-d)). (2) Sampling techniques—partial data (c) and partial configurations (d)—can correct model drift with less profiling overhead (see Table 4), compared to continuous (b). We omit accuracy predictions since in all schemes AWStream finds configurations that achieve similarly high accuracy (~90%).

Online scheme	Overhead	Improvements
Continuous	52X	Baseline
Partial data	17X	3×
Partial configurations	6X	8.7×

Table 4: Compared to the continuous profiling baseline (52X overhead), our sampling techniques speed up by 3× or 8.7×.

continuously to check for model drift will reduce the overhead to 1X. In practice, the amount of resources to use depends on the budget and the importance of the job. AWStream currently requires developers to configure the application with proper online profiling techniques.

5.3 Runtime Adaptation

In this section, we evaluate the runtime performance by controlling available bandwidth across geo-distributed sites and compare AWStream with baselines including streaming over TCP/UDP, JetStream, and video streaming. Our evaluation shows that AWStream achieves low latency and high accuracy simultaneously across all applications. Due to space constraints, we discuss AR in full and only present the results of PD/TK (more in appendix).

Experiment setup. We conduct our experiments on four geo-distributed machines from Amazon EC2, spanning four different regions. Three (at N. Virginia, Ohio, Oregon) act as worker nodes and one (at N. California) acts as the analytics server. The average RTTs from the workers to the server are 65.2 ms, 22.2 ms, and 50.3 ms.

During the experiment, each worker transmits test data (Table 3) for about 10 mins. If the duration of the test data is less than 10 mins, it loops. Because $B(c_{\max})$ is prohibitively large (raw videos consumes 230 Mbps), we use a reasonable configuration to limit the maximum rate. In our AR experiment, $c_{\max}$ is 1600x900 resolution, 30 FPS and a quantization of 20; it consumes about 14 Mbps.

Our bandwidth control scheme follows JetStream [53]. During the experiment, we use the Linux `tc` utility with HTB [19, 30] to control the clients' outgoing bandwidth. Each experiment involves four phases: (i) before t=200s, there is no shaping; (ii) at t=200s, we limit the bandwidth to 7.5 Mbps for 3 minutes; (iii) at t=380s, we further decrease the available bandwidth to 5 Mbps; (iv) at t=440s, we remove all traffic shaping. For UDP, HTB doesn't emulate the packet loss or out-of-order delivery; so we use `netem` and configure the loss probability according to the delivery rate. Because each pair-wise connection has a different capacity, we impose a *background* bandwidth limit—25 Mbps—to simplify comparisons.

We compare AWStream with the following baselines:

- Streaming over TCP/UDP (non-adaptive). (i) For TCP, we re-use AWStream's runtime that runs over TCP but disable the adaptation. (ii) For UDP, we use FFmpeg [11] to stream video: RTP/UDP [60] for media and RTSP for signaling [61]. This is a common setup for video conferencing and IP cameras [21, 33].
- Adaptive video streaming. We use HTTP Live Streaming (HLS) to represent popular adaptive video streaming techniques. Our setup resembles personalized live streaming systems [77] but uses a smaller chunk for low latency (1 second instead of typical 2-10 seconds).[5]
- JetStream with the manual policy described in §2.3.
- JetStream++, a modified version of JetStream that uses the profile learned by AWStream.[6]

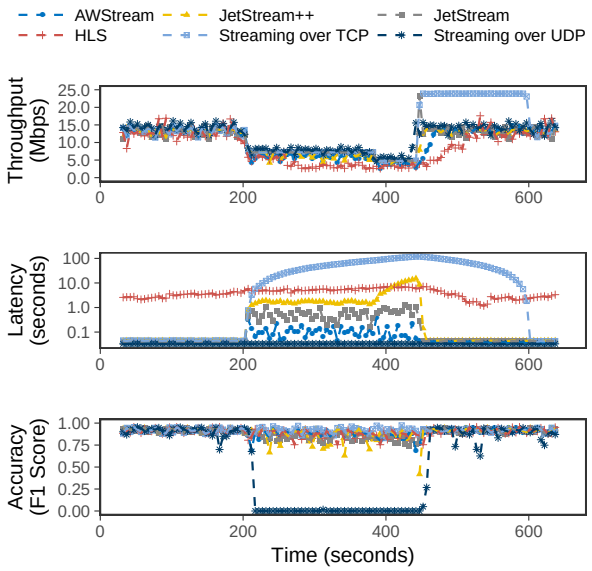
At runtime, AWStream differs from JetStream in both policy and adaptation. JetStream++ already improves over JetStream by using our Pareto-optimal profile. AWStream improves the performance further with two major changes: (i) AWStream directly measures the delivery rate to select an appropriate configuration to match available bandwidth while JetStream employs a latency-based measure of capacity ratio. (ii) AWStream has an explicit probe phase while JetStream changes its policy immediately after capacity ratio changes.

Results. Fig. 10a shows the runtime behavior of AWStream and all baselines in time series. Fig. 10b summarizes the latency and accuracy with box plots during bandwidth shaping (between t=200s and t=440s).

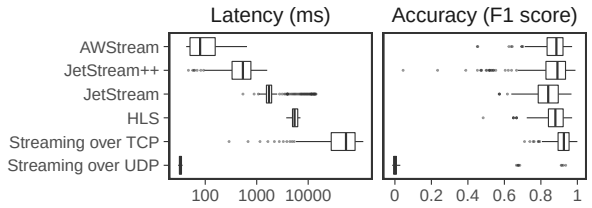
The throughput figure shows the effect of traffic shaping. During the shaping, TCP and UDP make full use of the available bandwidth; in comparison, AWStream, JetStream, JetStream++, and HLS are conservative because of adaptation (see their throughput drops). When we stop

[5]The appendix includes details about our HLS setup.

[6]The appendix includes details about how we modified JetStream.



(a) Time-series plot of the runtime behaviors: throughput (top), showing the effect of bandwidth shaping; latency (middle) in log scale; and accuracy (bottom).



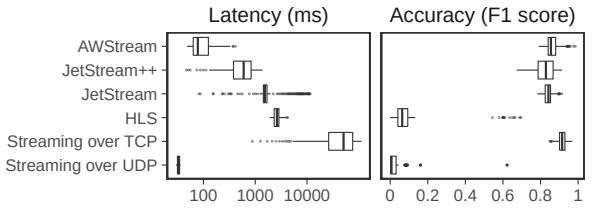
(b) Summary of latency and accuracy during the traffic shaping (between t=200s and t=440s).

Figure 10: For our AR application, AWStream simultaneously achieves low latency and high accuracy (with smaller variation).

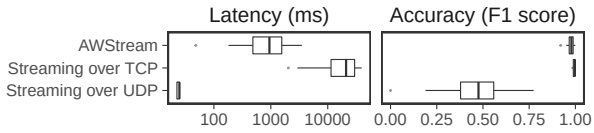
the shaping at t=440s, TCP has a "catch-up" phase when it is sending all queued items as fast as possible. JetStream also has queued items because the policy in use (with only three rules) cannot sustain 5 Mpbs bandwidth. AWStream increases the throughput gradually due to the explicit probing phase. HLS is the most conservative scheme; it does not recover from degradation until t=500s.

The latency figures (both Fig. 10a and Fig. 10b) show that AWStream is able to maintain sub-second latency. During the traffic shaping, TCP queues items at the sender side for up to hundreds of seconds. In contrast, UDP always transmits as fast as possible, leading to a consistent low latency.[7] HLS's latency fluctuates around 4-5 seconds due to chunking, buffering, and network variations, on par with recent literature [77]. Both JetStream and JetStream++ are able to adapt during traffic shaping. With a more precise and fine-grain policy, JetStream++

[7]FFmpeg discards packets that miss a deadline (33 ms for 30 FPS).



(a) PD's latency and accuracy during traffic shaping.



(b) TK's latency and accuracy during traffic shaping.[9]

Figure 11: For PD and TK, AWStream achieves low latency and high accuracy in comparison to baselines (details in appendix).

achieves a lower latency (median 539 ms) in comparison to JetStream (median 1732 ms). Because JetStream's runtime reacts instantaneously when the congestion condition changes, both baselines oscillate among polices during the experiment. AWStream effectively addresses the oscillation with probing and achieves a much lower latency (median 118 ms, 15× improvement over JetStream and 5× improvement over JetStream++).

The accuracy figures (both Fig. 10a and Fig. 10b) show that other than UDP, most schemes are able to maintain high accuracy. TCP without adaptation always sends data at high fidelity, achieving the highest accuracy (median 93%), but at a cost of high latency. JetStream uses a manual policy that are sub-optimal in comparison to our learned profile, so its accuracy is low (median 84%). Using Pareto-optimal configurations, JetStream++ is able to achieve a higher accuracy (median 89%); but because JetStream's runtime oscillates the policy, the accuracy has a large variation (standard deviation 14%). In contrast, AWStream chooses configurations carefully to stay in a steady state as much as possible. It achieves a high accuracy of 89% with a small variation (standard deviation 7.6%). HLS also achieves reasonable accuracy (median 87%) because its adaptation of tuning resolution and encoding quality is effective in AR. However, HLS's adaptation works poorly for PD (median 6% as in Fig. 11a).

In summary, Fig. 10 shows that AWStream achieves low latency and high accuracy simultaneously. We show the results during shaping in a different form in Fig. 1 to discuss the trade-off between fidelity and freshness.[8]

Runtime performance of PD and TK (Fig. 11). For

[8]We obtain Fig. 1's app-specific data by feeding PD's profile to AR. We refer to JetStream as manual policies in Fig. 1.

[9]TK's baselines are only streaming over TCP/UDP because HLS and JetStream are not directly comparable (see appendix).

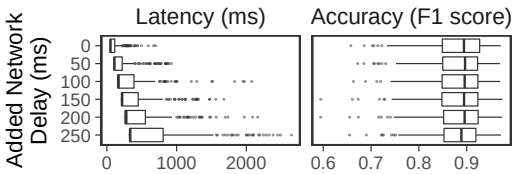


Figure 12: AWStream maintains low latency and high accuracy under different network delay conditions.

PD, AWStream achieves low latency (median 78 ms), which is 20× improvement over JetStream and 7.7× over JetStream++. AWStream achieves an accuracy of 86%, which is 6% drop compared to TCP and 1% improvement over JetStream. For TK, AWStream also manages sub-second latency (median 946 ms) and nominal accuracy drop (only 2% in comparison to TCP).

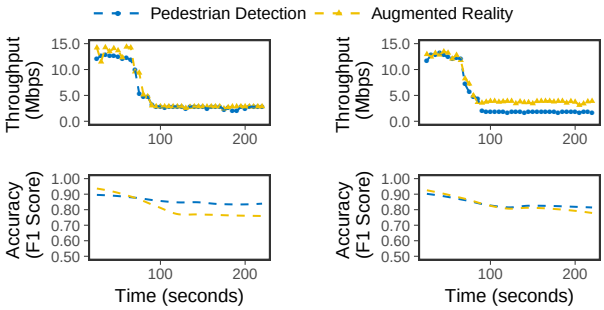
Performance with various network delays. AWStream targets at wide area whose key characteristic is the large variation in latency [38]. While we've conducted experiments using real-world setup on Amazon EC2, the latency between EC2 sites is relatively low. To evaluate how AWStream performs with increased network delays, we conducted another experiment with one pair of client and server under different network conditions. We use `netem` to add delays in addition to our bandwidth shaping. The additional one-way delays ranges from 0 to 250 ms, following a normal distribution where the variation is 10% of the added delay, e.g. 250 ± 25ms. We add delays in both direction, so the added RTT can be as high as 500 ms.

Fig. 12 shows the runtime behavior with various added network delays. While the latency increases with the added delay, AWStream mostly manages to achieve sub-second latency for all conditions. We see a higher variation in latency and more outliers as network delay increases. This is caused by a slow congestion detection when the RTT is high. In terms of accuracy, because AWStream mostly stays in `Steady` state and accuracy only depends on the level of degradation, AWStream achieves similar accuracy for different network delays.

5.4 Resource Allocation and Fairness

We evaluate resource allocations with two applications. In this way, the result also covers the case of a single application, and can generalize to more applications.

We choose AR and PD as the example applications. The clients and servers of both applications are co-located so that they share the same bottleneck link. The experiment starts with sufficient bandwidth. At t=60s, we start traffic shaping to limit the total bandwidth to 6 Mbps. When we allocate resource equally between two applications (Fig. 13a), each application gets 3 Mbps. Under this condition, PD runs with a higher accuracy of 85% while AR only achieves 77%. In addition to resource fairness, AWStream supports utility fairness: it chooses configurations that maximize the minimal accuracy. In this experiment, PD receives 2 Mbps and AR receives 4 Mbps; and both achieve 80% accuracy (Fig. 13b).



(a) Resource Fairness **(b)** Utility Fairness

Figure 13: AWStream supports a variety of resource allocation schemes: resource fairness (a) and utility fairness (b).

6 Discussion and Future work

We have presented AWStream, a stream processing system achieving low latency and high accuracy for the wide area. This section discusses limitations and future work.

Reducing developer effort. While AWStream focuses on reducing developer effort from tedious manual policy construction, there are still application-specific aspects for developers: writing appropriate `maybe` calls, providing training data, and specifying the accuracy function. Because AWStream's API is extensible, we plan to build more libraries for common degradation operators and accuracy functions, similar to machine learning libraries.

Fault-tolerance and failure recovery. AWStream tolerates bandwidth variation but not network partition or host failure. Although servers within data centers can handle faults in existing systems, such as Spark Streaming [79], it is difficult to make edge clients failure-oblivious. We leave failure detection and recovery as a future work.

Profile modeling. AWStream currently does not model $B(c)/A(c)$. It performs an exhaustive search when profiling. While parallelism and sampling techniques can speed this up, AWStream can benefit further from statistical techniques. Bayesian Optimization [64], as demonstrated by CherryPick [49], models black-box functions and speeds up profiling by searching for near-optimal configurations. We plan to explore this direction to improve our profiling.

Predicting bandwidth changes. Our runtime currently does not predict future bandwidth. While reacting to bandwidth changes is enough to achieve sub-second latency, if AWStream can accurately predict bandwidth changes, we expect further improvements such as no latency spikes or a simplified runtime design. Techniques such as model predictive control (MPC) have improved QoE in video

streaming with throughput prediction [78]; we plan to investigate using MPC in AWStream as a future work.

7 Related Work

JetStream. JetStream pioneered the use of degradation to reduce latency for wide-area streaming analytics. Compared to JetStream, AWStream makes the following major contributions: (1) a novel API design to specify degradation operations in a simple and composable manner; (2) automatic offline profiling to search for Pareto-optimal configurations; (3) online profiling to address model drift; (4) an improved runtime implementation that utilizes the learned profile and achieves sub-second latency (comparison made in §5.3); (5) support for different resource allocation policies for multiple applications.

Stream processing systems. Streaming databases, such as Borealis [1], TelegraphCQ [16], are the early academic explorations. They pioneered the use of dataflow models with specialized operators for stream processing. Recent research projects and open-source systems, such as MillWheel [6], Storm [70], Heron [36], Spark Streaming [79], Apache Flink [14], primarily focus on fault-tolerant streaming in the context of a single cluster. While our work has a large debt to the prior streaming works, AWStream targets at the wide area and explicitly explores the trade-off between data fidelity and freshness. In contrast, these stream processing systems often use TCP and choose to throttle the source when back pressure happens.

Approximate analytics. The idea of degrading computation fidelity for responsiveness is also explored elsewhere. For SQL queries, online aggregation [28], BlinkDB [5] and GRASS [9] speed up queries with partial data based on a statistical model of SQL operators. For real-time processing, MEANTIME [23] uses approximation to guarantee satisfying real-time deadlines. For video analytics, VideoStorm [80] optimizes video stream *processing* within lager clusters by exploiting approximation and delay-tolerance for resource management and allocation. The trade-off between available resource and application accuracy is a common theme among all these works. AWStream targets at wide-area streaming analytics, an emerging application domain especially with the advent of IoT.

WAN-aware systems. The main challenge in designing geo-distributed systems is to cope with high latency and limited bandwidth. While some research projects, such as Vivace [17], assume the network can prioritize a small amount of critical data to avoid delays under congestion, most systems reduce data transfer in the first place to avoid congestion. For example, LBFS [45] exploits similarities between files or versions of the same file. Over the past two years, we have seen a plethora of geo-distributed analytical frameworks: WANalytics [74], Geode [75], Iridium [52], Pixida [34], Clarinet [73], etc. They minimize data movement by incorporating heterogeneous wide-area bandwidth information into query optimization and data/task placement. While effective in speeding up analytics, these systems focus on batch tasks such as MapReduce jobs or SQL queries. Such tasks are often executed once, without real time constrain. In contrast, AWStream processes streams continuously in real time.

(Adaptive) video streaming. Multimedia streaming protocols, such as RTP [60], aim to be fast instead of reliable. While they can achieve low latency, their accuracy can be poor without adaptation. Recent work on multimedia streaming has moved towards using TCP (primarily HTTP-based) and focused on designing good adaptation strategy to improve QoE, as in research projects [41, 66, 78] and industrial efforts (HLS [50], DASH [42, 65]). These adaptation strategies work well for video on demand (VoD), but they are not for ultra low latency streaming: (*i*) they use pulling: the client keeps refreshing index file and fetching new chunks; (*ii*) chunking introduces latency: each chunk has to be ready before the client fetches. In addition, because they optimize QoE for humans, the adaptation can be a mismatch for analytics that rely on image details (e.g. 6% accuracy for PD). In contrast, AWStream learns an adaptation strategy for each application (also not limited to video analytics); and the runtime is optimized for ultra low latency.

QoS. Most QoS work [24, 63, 62] in the 1990s focuses on network-layer solutions that are not widely deployable. AWStream adopts an end-host and application-layer approach ready today for WAN. For existing application-layer approaches [71], AWStream's API can incorporate their techniques, such as encoding the number of layers as a knob to realize the layered approach in Rejaie et al [55]. Fundamentally, AWStream does not provide hard QoS guarantees; instead AWStream maximizes achievable accuracy (application performance) with respect to bandwidth constraints, such that latency is minimized (system performance): a multidimensional optimization.

8 Conclusion

This paper presents AWStream, a stream processing system for a wide variety of applications by enabling developers to customize degradation operations with `maybe` operators. Our automatic profiling tool generates an accurate profile that characterizes the trade-off between bandwidth consumption and application accuracy. The profile allows the runtime to react with precision. Evaluations with three applications show that AWStream achieves sub-second latency with nominal accuracy drop. AWStream enables resilient execution of wide-area streaming analytics with minimal developer effort.

References

[1] ABADI, D. J., AHMAD, Y., BALAZINSKA, M., CETINTEMEL, U., CHERNIACK, M., HWANG, J.-H., LINDNER, W., MASKEY, A., RASIN, A., RYVKINA, E., ET AL. The Design of the Borealis Stream Processing Engine. In *CIDR* (Asilomar, CA, 2005), vol. 5, pp. 277–289.

[2] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., ET AL. TensorFlow: A System for Large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 265–283.

[3] ABARI, O., BHARADIA, D., DUFFIELD, A., AND KATABI, D. Enabling High-Quality Untethered Virtual Reality. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 531–544.

[4] ABDI, H. The Kendall Rank Correlation Coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), 508–510.

[5] AGARWAL, S., MOZAFARI, B., PANDA, A., MILNER, H., MADDEN, S., AND STOICA, I. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013), EuroSys '13, ACM, pp. 29–42.

[6] AKIDAU, T., BALIKOV, A., BEKIROĞLU, K., CHERNYAK, S., HABERMAN, J., LAX, R., MCVEETY, S., MILLS, D., NORDSTROM, P., AND WHITTLE, S. MillWheel: Fault-tolerant Stream Processing at Internet Scale. *Proceedings of the VLDB Endowment 6*, 11 (2013), 1033–1044.

[7] ALSPAUGH, S., DI CHEN, B., LIN, J., GANAPATHI, A., HEARST, M. A., AND KATZ, R. H. Analyzing Log Analysis: An Empirical Study of User Log Mining. In *Proceedings of the 28th USENIX Conference on Large Installation System Administration* (Berkeley, CA, USA, 2014), LISA'14, USENIX Association, pp. 53–68.

[8] AMAZON. Amazone EC2 Pricing. https://aws.amazon.com/ec2/pricing/, 2017. Accessed: 2017-04-12.

[9] ANANTHANARAYANAN, G., HUNG, M. C.-C., REN, X., STOICA, I., WIERMAN, A., AND YU, M. GRASS: Trimming Stragglers in Approximation Analytics. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, 2014), USENIX Association, pp. 289–302.

[10] BALANI, N., AND HATHI, R. *Enterprise IoT: A Definitive Handbook*. CreateSpace Independent Publishing Platform, 2016.

[11] BELLARD, F., NIEDERMAYER, M., ET AL. Ffmpeg. https://www.ffmpeg.org/, 2012.

[12] BISWAS, S., BICKET, J., WONG, E., MUSALOIU-E, R., BHARTIA, A., AND AGUAYO, D. Large-scale Measurements of Wireless Network Behavior. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM'15, ACM, pp. 153–165.

[13] CALDER, M., FAN, X., HU, Z., KATZ-BASSETT, E., HEIDEMANN, J., AND GOVINDAN, R. Mapping the Expansion of Google's Serving Infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC'13, ACM, pp. 313–326.

[14] CARBONE, P., KATSIFODIMOS, A., EWEN, S., MARKL, V., HARIDI, S., AND TZOUMAS, K. Apache flink: Stream and batch processing in a single engine. *Data Engineering 38*, 4 (2015).

[15] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., ET AL. BBR: Congestion-based Congestion Control. *Communications of the ACM 60*, 2 (2017), 58–66.

[16] CHANDRASEKARAN, S., COOPER, O., DESHPANDE, A., FRANKLIN, M. J., HELLERSTEIN, J. M., HONG, W., KRISHNAMURTHY, S., MADDEN, S. R., REISS, F., AND SHAH, M. A. TelegraphCQ: Continuous Dataflow Processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2003), SIGMOD'03, ACM, pp. 668–668.

[17] CHO, B., AND AGUILERA, M. K. Surviving Congestion in Geo-Distributed Storage Systems. In *USENIX Annual Technical Conference (USENIX ATC 12)* (2012), USENIX, pp. 439–451.

[18] COIFMAN, B., BEYMER, D., MCLAUCHLAN, P., AND MALIK, J. A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance. *Transportation Research Part C: Emerging Technologies 6*, 4 (1998), 271–288.

[19] DEVERA, M. HTB Home. http://luxik.cdi.cz/~devik/qos/htb/, 2001–2003. Accessed: 2017-04-08.

[20] DOLLAR, P., WOJEK, C., SCHIELE, B., AND PERONA, P. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE transactions on pattern analysis and machine intelligence 34*, 4 (2012), 743–761.

[21] DURRESI, A., AND JAIN, R. RTP, RTCP, and RTSP-Internet Protocols for Real-Time Multimedia Communication., 2005.

[22] ESNET. iPerf: The TCP/UDP bandwidth measurement tool. http://software.es.net/iperf/, 2014-2017. Accessed: 2017-03-07.

[23] FARRELL, A., AND HOFFMANN, H. MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (Denver, CO, 2016), USENIX Association, pp. 421–435.

[24] FERRARI, D., AND VERMA, D. C. A Scheme for Real-time Channel Establishment in Wide-area Networks. *IEEE journal on Selected Areas in communications 8*, 3 (1990), 368–379.

[25] GE. Industrial Internet Insights. https://www.ge.com/digital/industrial-internet, 2017. Accessed: 2017-09-23.

[26] GOOGLE. Nest Cam Indoor. https://www.dropcam.com, 2009-2017. Accessed: 2017-04-03.

[27] GROVER, S., PARK, M. S., SUNDARESAN, S., BURNETT, S., KIM, H., RAVI, B., AND FEAMSTER, N. Peeking Behind the NAT: An Empirical Study of Home Networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC'13, ACM, pp. 377–390.

[28] HELLERSTEIN, J. M., HAAS, P. J., AND WANG, H. J. Online Aggregation. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1997), SIGMOD'97, ACM, pp. 171–182.

[29] HSIEH, K., HARLAP, A., VIJAYKUMAR, N., KONOMIS, D., GANGER, G. R., GIBBONS, P. B., AND MUTLU, O. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), USENIX Association, USENIX Association.

[30] HUBERT, B. Linux Advanced Routing & Traffic Control. http://lartc.org/, 2002. Accessed: 2017-04-06.

[31] INDEX, C. V. N. The Zettabyte Era: Trends and Analysis. *Cisco white paper* (2013).

[32] KARGER, D., STEIN, C., AND WEIN, J. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC, 2010.

[33] KING, J. W. Cisco ip video surveillance design guide. https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Video/IPVS/IPVS_DG/IPVS-DesignGuide.pdf, 2009.

[34] KLOUDAS, K., MAMEDE, M., PREGUIÇA, N., AND RODRIGUES, R. Pixida: optimizing data parallel jobs in wide-area data analytics. *Proceedings of the VLDB Endowment 9*, 2 (2015), 72–83.

[35] KRIOUKOV, A., FIERRO, G., KITAEV, N., AND CULLER, D. Building Application Stack (BAS). In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings* (New York, NY, USA, 2012), BuildSys '12, ACM, pp. 72–79.

[36] KULKARNI, S., BHAGAT, N., FU, M., KEDIGEHALLI, V., KELLOGG, C., MITTAL, S., PATEL, J. M., RAMASAMY, K., AND TANEJA, S. Twitter Heron: Stream Processing at Scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD'15, ACM, pp. 239–250.

[37] LANGFITT, F. In China, Beware: A Camera May Be Watching You. http://www.npr.org/2013/01/29/170469038/in-china-beware-a-camera-may-be-watching-you, 2013. Accessed: 2017-04-04.

[38] LI, A., YANG, X., KANDULA, S., AND ZHANG, M. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC'10, ACM, pp. 1–14.

[39] LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision 60*, 2 (Nov. 2004), 91–110.

[40] LU, C., AND TANG, X. Surpassing Human-level Face Verification Performance on LFW with Gaussian Face. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015), AAAI'15, AAAI Press, pp. 3811–3819.

[41] MAO, H., NETRAVALI, R., AND ALIZADEH, M. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 197–210.

[42] MICHALOS, M., KESSANIDIS, S., AND NALMPANTIS, S. Dynamic Adaptive Streaming over HTTP. *Journal of Engineering Science and Technology Review 5*, 2 (2012), 30–34.

[43] MILAN, A., LEAL-TAIXÉ, L., REID, I., ROTH, S., AND SCHINDLER, K. MOT16: A Benchmark for Multi-Object Tracking. *arXiv preprint arXiv:1603.00831* (2016).

[44] MUKERJEE, M. K., NAYLOR, D., JIANG, J., HAN, D., SESHAN, S., AND ZHANG, H. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 311–324.

[45] MUTHITACHAROEN, A., CHEN, B., AND MAZIÈRES, D. A Low-bandwidth Network File System. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2001), SOSP'01, ACM, pp. 174–187.

[46] NIKRAVESH, A., CHOFFNES, D. R., KATZ-BASSETT, E., MAO, Z. M., AND WELSH, M. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362* (New York, NY, USA, 2014), PAM 2014, Springer-Verlag New York, Inc., pp. 12–22.

[47] OF ECONOMIC, T. D., AND (DERA), R. A. EDGAR Log File Data Set. https://www.sec.gov/data/edgar-log-file-data-set, 2003–2016. Accessed: 2017-01-25.

[48] OH, S., HOOGS, A., PERERA, A., CUNTOOR, N., CHEN, C.-C., LEE, J. T., MUKHERJEE, S., AGGARWAL, J., LEE, H., DAVIS, L., ET AL. A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2011), CVPR '11, IEEE Computer Society, pp. 3153–3160.

[49] OMID ALIPOURFARD AND HONGQIANG HARRY LIU AND JIANSHU CHEN AND SHIVARAM VENKATARAMAN AND MINLAN YU AND MING ZHANG. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 469–482.

[50] PANTOS, R., AND MAY, W. HTTP Live Streaming, 2016.

[51] PARKHI, O. M., VEDALDI, A., AND ZISSERMAN, A. Deep Face Recognition. In *Proceedings of the British Machine Vision Conference (BMVC)* (September 2015), BMVA Press, pp. 41.1–41.12.

[52] PU, Q., ANANTHANARAYANAN, G., BODIK, P., KANDULA, S., AKELLA, A., BAHL, P., AND STOICA, I. Low Latency Geo-Distributed Data Analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 421–434.

[53] RABKIN, A., ARYE, M., SEN, S., PAI, V. S., AND FREEDMAN, M. J. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 275–288.

[54] REDMON, J., AND FARHADI, A. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242* (2016).

[55] REJAIE, R., HANDLEY, M., AND ESTRIN, D. Layered Quality Adaptation for Internet Video Streaming. *IEEE Journal on Selected Areas in Communications 18*, 12 (2000), 2530–2543.

[56] RICHARDSON, I. E. *The H.264 Advanced Video Compression Standard*, 2nd ed. Wiley Publishing, 2010.

[57] RIJSBERGEN, C. J. V. *Information Retrieval*, 2nd ed. Butterworth-Heinemann, Newton, MA, USA, 1979.

[58] RUSSELL, B. C., TORRALBA, A., MURPHY, K. P., AND FREEMAN, W. T. LabelMe: a Database and Web-based Tool for Image Annotation. *Int. J. Comput. Vision 77*, 1-3 (May 2008), 157–173.

[59] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing 8*, 4 (Oct. 2009), 14–23.

[60] SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACONSON, V. RTP: A Transport Protocol for Real-Time, 2006.

[61] SCHULZRINNE, H., RAO, A., AND LANPHIER, R. Rtsp: Real time streaming protocol. *IETF RFC2326, april* (1998).

[62] SHENKER, S. Fundamental Design Issues for the Future Internet. *IEEE Journal on selected areas in communications 13*, 7 (1995), 1176–1188.

[63] SHENKER, S., BRADEN, R., AND CLARK, D. Integrated services in the internet architecture: an overview. *IETF Request for Comments (RFC) 1633* (1994).

[64] SNOEK, J., LAROCHELLE, H., AND ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.

[65] SODAGAR, I. The MPEG-DASH Standard for Multimedia Streaming over the Internet. *IEEE MultiMedia 18*, 4 (Oct. 2011), 62–67.

[66] SUN, Y., YIN, X., JIANG, J., SEKAR, V., LIN, F., WANG, N., LIU, T., AND SINOPOLI, B. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference* (2016), ACM, ACM, pp. 272–285.

[67] SUNDARESAN, S., BURNETT, S., FEAMSTER, N., AND DE DONATO, W. BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 383–394.

[68] TELEGEOGRAPHY. Global Internet Geography. https://www.telegeography.com/research-services/global-internet-geography/, 2016. Accessed: 2017-04-10.

[69] TEMPERTON, J. One nation under CCTV: the future of automated surveillance. http://www.wired.co.uk/article/one-nation-under-cctv, 2015. Accessed: 2017-01-27.

[70] TOSHNIWAL, A., TANEJA, S., SHUKLA, A., RAMASAMY, K., PATEL, J. M., KULKARNI, S., JACKSON, J., GADE, K., FU, M., DONHAM, J., ET AL. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), ACM, pp. 147–156.

[71] VANDALORE, B., FENG, W.-C., JAIN, R., AND FAHMY, S. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Real-Time Imaging 7*, 3 (2001), 221–235.

[72] VIOLA, P., AND JONES, M. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (2001), vol. 1, pp. I–511–I–518 vol.1.

[73] VISWANATHAN, R., ANANTHANARAYANAN, G., AND AKELLA, A. Clarinet: WAN-Aware Optimization for Analytics Queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 435–450.

[74] VULIMIRI, A., CURINO, C., GODFREY, P. B., JUNGBLUT, T., KARANASOS, K., PADHYE, J., AND VARGHESE, G. WANalytics: Geo-Distributed Analytics for a Data Intensive World. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD'15, ACM, pp. 1087–1092.

[75] VULIMIRI, A., CURINO, C., GODFREY, P. B., JUNGBLUT, T., PADHYE, J., AND VARGHESE, G. Global Analytics in the Face of Bandwidth and Regulatory Constraints. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), pp. 323–336.

[76] WALLACE, G. K. The JPEG Still Picture Compression Standard. *Commun. ACM 34*, 4 (Apr. 1991), 30–44.

[77] WANG, B., ZHANG, X., WANG, G., ZHENG, H., AND ZHAO, B. Y. Anatomy of a Personalized Livestreaming System. In *Proceedings of the 2016 ACM on Internet Measurement Conference* (2016), ACM, pp. 485–498.

[78] YIN, X., JINDAL, A., SEKAR, V., AND SINOPOLI, B. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM'15, ACM, pp. 325–338.

[79] ZAHARIA, M., DAS, T., LI, H., HUNTER, T., SHENKER, S., AND STOICA, I. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP'13, ACM, pp. 423–438.

[80] ZHANG, H., ANANTHANARAYANAN, G., BODIK, P., PHILIPOSE, M., BAHL, P., AND FREEDMAN, M. J. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 377–392.

[81] ZHANG, T., CHOWDHERY, A., BAHL, P. V., JAMIESON, K., AND BANERJEE, S. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), ACM, pp. 426–438.