AWStream: Adaptive Wide-Area Streaming Analytics

Ben Zhang
UC Berkeley

Xin Jin
Johns Hopkins University

Sylvia Ratnasamy
UC Berkeley

John Wawrzynek
UC Berkeley

Edward A. Lee
UC Berkeley

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%).

CCS CONCEPTS

• **Networks** → **Application layer protocols**; **Cross-layer protocols**; **Network performance modeling**; **Wide area networks**; **Network dynamics**; **Public Internet**;

KEYWORDS

Wide Area Network; Adaptation; Learning; Profiling

ACM Reference Format:
Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. 2018. AWStream: Adaptive Wide-Area Streaming Analytics. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3230543.3230554

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5567-4/18/08…$15.00
https://doi.org/10.1145/3230543.3230554

1 INTRODUCTION

Wide-area streaming analytics are becoming pervasive, especially with emerging Internet of Things (IoT) applications. Large cities such as London and Beijing have deployed millions of cameras for surveillance and traffic control [46, 85]. Buildings are increasingly equipped with a wide variety of sensors to improve energy efficiency and occupant comfort [44]. Geo-distributed infrastructure, such as content delivery networks (CDNs), analyze requests from machine logs across the globe [54]. These applications all transport, distill, and process streams of data across the wide area, in real time.

A key challenge that the above applications face is dealing with the scarce and variable bandwidth in the wide area [38, 91]. As many have observed, WAN bandwidth growth has been decelerating for many years while traffic demands are growing at a staggering rate [21, 56, 84]. In addition, scarcity in last-mile bandwidth remains a problem across wireless [13], cellular [58], and even broadband [35, 82] networks. Finally, as we elaborate on in §2, not only is WAN bandwidth scarce, it is also relatively expensive, and highly variable.

For all of the above reasons, it is important that streaming applications be *adaptive*, incorporating the ability to optimally trade-off accuracy for bandwidth consumption and hence a key system challenge is to design the *programming abstractions and tools* that simplify the development of such adaptive applications.

In recent years, systems such as Storm [86], Spark Streaming [95], and VideoStorm [96], have emerged in support of stream processing. These systems enable efficient processing of large streams of data, but are designed to work within a single datacenter cluster (where network bandwidth is typically not the bottleneck) and hence they do not focus on support for adapting to the vagaries of WAN bandwidth.

Recent research on WAN-aware systems promote pushing computation to the network edge [65, 73]. However, even with edge computing, the need for adaptation remains because end-devices such as cameras and mobile phones still suffer from limited bandwidth in the last-hop infrastructure [3, 97]. In addition, edge computing is not a panacea as wide-area

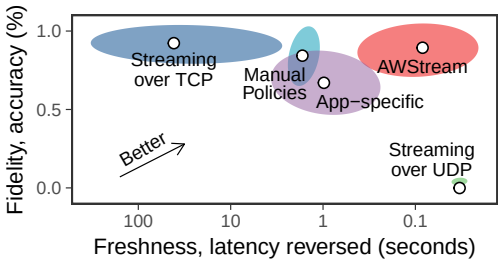


Figure 1. The trade-off space between data freshness and fidelity when facing insufficient bandwidth (details in §5.3).

communication is often not entirely avoidable: e.g., some analytical jobs require joining or aggregating data from multiple geo-distributed sites [64, 89], while in some cases processing benefits substantially from specialized computing resources such as GPUs and TPUs [2] in the cloud.

The core difficulty with adaptive streaming analytics is that, when bandwidth is scarce, developers are faced with the decision of how to reconcile data fidelity (i.e., not losing any data) with data freshness (i.e., sending data as quickly as possible). A deterioration in either fidelity or freshness can impact application accuracy but the exact impact varies depending on the application.[1] Figure 1 illustrates this trade-off with a few sample points in the design space.

Applications that simply use existing protocols without any attempt at adaptation can result in extreme design points. For example, streaming over TCP ensures reliable delivery (hence high fidelity) but backlogged data delays the delivery of data (hence freshness suffers). On the other hand, streaming over UDP minimizes latency by sending packets as fast as possible, but uncontrolled packet loss can devastate data fidelity.

Manual policies, such as sampling, allow developers to trade data fidelity for freshness [65]. However, it's difficult to write accurate policies without extensive domain expertise or considerable effort. In practice, developers write manual policies based on heuristics rather than quantitative measurements and, as we show in §5, such policies can lead to sub-optimal performance in terms of both freshness and fidelity.

Furthermore, application-specific optimizations often 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) [52, 62, 94]. 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. However, low resolution images can lead to poor accuracy for video analytics that rely on the image details (e.g., face detection [88]).

[1] For example, an application tracking the current value of a variable might prioritize freshness while one that is computing an average might prioritize fidelity.

In this paper, we present AWStream, a framework for building adaptive stream processing applications that simultaneously simplifies development *and* improves application accuracy in the face of limited or varying wide-area bandwidth. AWStream achieves this with three novel contributions:

1. AWStream introduces new programming abstractions by which a developer expresses *what* degradation functions can be used by the framework. Importantly, developers do not have to specify exactly when and how different degradation functions are to be used which is instead left to the AWStream framework.
2. Rather than rely on manual policies, AWStream automatically *learns* a Pareto-optimal policy or strategy for when and how to invoke different degradation functions. For this, we design a methodology that uses a combination of offline and online training to build an accurate model of the relationship between an application's accuracy and its bandwidth consumption under different combinations of degradation functions. Our solution exploits parallelism and sampling to efficiently explore the configuration space and learn an optimal strategy.
3. AWStream's final contribution is the design and implementation of a runtime system that continually measures and adapts to network conditions. AWStream matches the streaming data rate to the measured available bandwidth, and achieves high accuracy by using the learned Pareto-optimal configurations. Upon encountering network congestion, our adaptation algorithm increases the degradation level to reduce the data rate, such that no persistent queue builds up. To recover, it progressively decreases the degradation level after probing for more available bandwidth.

We implement AWStream and use it to prototype 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. We show that AWStream's data-driven approach generates accurate profiles and that our parallelism and sampling techniques can speed up profiling by up to 29× and 8.7× respectively.

With the combination of AWStream's ability to learn better policies and its well-designed runtime, our evaluation shows that AWStream significantly outperforms non-adaptive applications: achieving a 40–100× reduction in packet delivery times relative to applications built over TCP, or an over 45–88% improvement in data fidelity (application accuracy) relative to applications built over UDP. We also compare AWStream to JetStream [65], a state-of-the-art system for building adaptive streaming analytics that is based on manual policies. Our results show that besides the benefit of generating optimal policies *automatically*, AWStream achieves a 15-20× reduction in latency and 1-5% improvement in accuracy simultaneously relative to JetStream.

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

We focus on wide-area streaming analytics, especially the emerging IoT applications. We give two concrete examples.

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 [60]. Recent advances in computer vision have dramatically increased the accuracy for automatic visual scene analysis, such as pedestrian detection [26], vehicle tracking [22], and facial recognition to locate people of interest [50, 63]. While some surveillance cameras use dedicated links, an increasing number of surveillance systems, such as Dropcam [34] and Vigil [97], use the public Internet and wireless links to reduce the cost of deployment and management.

Infrastructure Monitoring. Large organizations today are managing tens of datacenters and edge clusters worldwide [15]. This geo-distributed infrastructure continuously produces large volumes of data such as data access logs, server monitoring logs, and performance counters [7, 64, 91]. 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 [65]. 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 [11, 33].

2.2 Wide-area Bandwidth Characteristics

WAN bandwidth is insufficient and costly, as shown by other systems [38, 64, 90, 91]. 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 [38]. In terms of pricing, the average WAN bandwidth cost is up to 38x of the cost of renting two machines [8, 38].

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 [28] 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—Figure 2 is one such pair. 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.

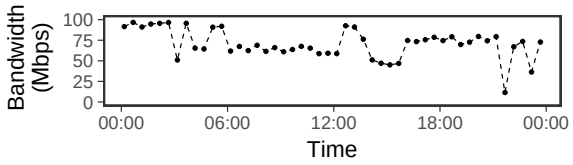


Figure 2. Bandwidth variations throughout the day between Amazon EC2 sites (from Ireland to California).

Similar scarcity and variations exist in wireless networks [13], broadband access networks [35, 82] and cellular networks [58].

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 [65] is the first to use degradation to address bandwidth limits in wide area. While effective in comparison to non-adaptive systems, JetStream requires developers to write manual policies, for example, *"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 do not 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 [69] is used, a reduction in frame rate increases the inter-frame difference and creates P-frames with larger sizes. Figure 3e shows that when reducing the frame rate to 33% (from 30 FPS to 10 FPS), the bandwidth use 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. Besides, applications can degrade in multiple dimensions and each dimension has different impacts (compare Figure 3a with Figure 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 each application has different performance metrics and relies on different features, a fine-tuned policy for one application will often work poorly for another. For example, DASH [79] optimizes QoE for video streaming; it keeps a high frame rate and reduces resolutions for adaptation. Its policy that lowers the resolution works poorly for video analytics

[2] Excerpt from JetStream §4.3 [65].

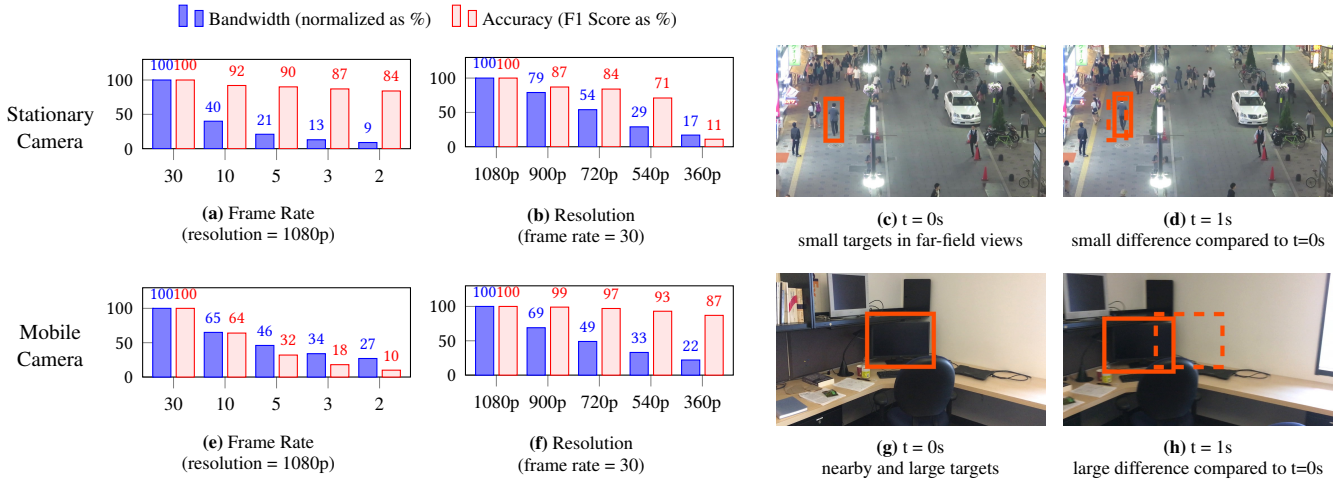


Figure 3. The measured bandwidth and application accuracy for two video analytics applications. (1) Manual policies lack precision without measurements and need to handle multiple dimensions, as in (a-b) and (c-d). (2) Application-specific optimizations do not generalize: degrading frame rates works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

that relies on image details [49, 88]. In Figure 3b, we show that pedestrian detection accuracy drops fast when reducing resolutions as pedestrian are small in the scenes.

Similar applications face different data distributions, as shown in Figure 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 camera moves, reducing the frame rate introduces significant errors.

3 AWSTREAM DESIGN

To address the issues with manual policies or application-specific optimizations, AWStream structures 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—we call it 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). Figure 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 [86, 95]. Each operator transforms input streams into new streams. AWStream borrows the same computation model and can support normal operators found in existing stream processing systems such as JetStream [65] (see example operators in Table 1).

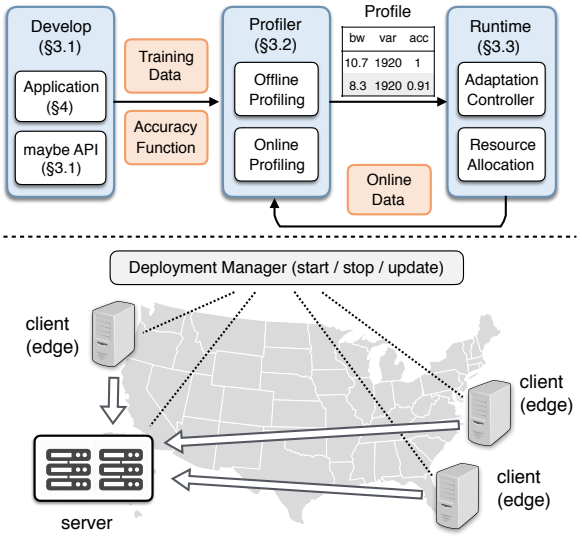


Figure 4. AWStream's phases: development, profiling, and runtime. AWStream also manages wide-area deployment.

To integrate adaptation as a first-class abstraction, AWStream introduces `maybe` operators that degrade data quality, yielding potential bandwidth savings. Our API design has three considerations. (*i*) To free developers from specifying exact rules, the API should allow specifications with options. (*ii*) To allow combining multiple dimensions, the API should be modular. (*iii*) To support flexible integration with arbitrary degradation functions, the API should take user-defined functions. 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:

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`.

```
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—for example, run-length encoding as in JPEG [92]—for transmission, the data size will depend on the level of degradation.

Based on the `maybe` primitive, one can implement additional degradation operators for common data types. For instance, `maybe_head` will optionally take the top values of a list; `maybe_downsample` can resize the image to a configured resolution. 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 1600x900 or 1280x720 resolution, and the other that skips every 2 or 5 frames, resulting in 30/(2+1)=10 FPS or 30/(5+1)= 6 FPS. This example then displays degraded images. 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 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.

Symbol	Description
n	number of degradation operations
k_i	the i-th degradation knob
$c = [k_1, k_2, \ldots k_n]$	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.

Profiling formalism. Suppose a stream processing application has n `maybe` operators. Each operator introduces a knob k_i. The combination of all knobs forms a *configuration* $c = [k_1, k_2, \ldots k_n]$. 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 these symbols.

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

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

We show examples of knobs, configurations, and accuracy functions when we present applications in §4 and visualize the profile of sample applications in Figure 8.

Offline Profiling. We first use an offline process to build a bootstrap profile (or default profile). Because AWStream allows arbitrary functions as the degradation functions, it does

not assume a closed-form relationship for $B(c)$ and $A(c)$. AWStream takes a data-driven approach: profiling applications with developer-supplied training data. $B(c)$ is measured as the data rate (bps) at the point of transmission. The accuracy $A(c)$ is measured either against the groundtruth, or the reference results when all degradation operations are off.

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 supports online profiling to continuously 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 [71], 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 offline profiling and schedule online profiling with longest first schedule (LFS) [41] during parallelization.
- Sampling-based profiling. Online profiling can speed up when we sample data or configurations. Sampling data reduces the amount of data to process, but at a cost of generating a less accurate profile. When sampling configuration, we can evaluate a subset of the Pareto-optimal configurations and compare their performances with an existing profile. 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. This section focuses on the details of our runtime design. We defer the evaluation and comparisons with existing systems (e.g., JetStream) to §5.3.

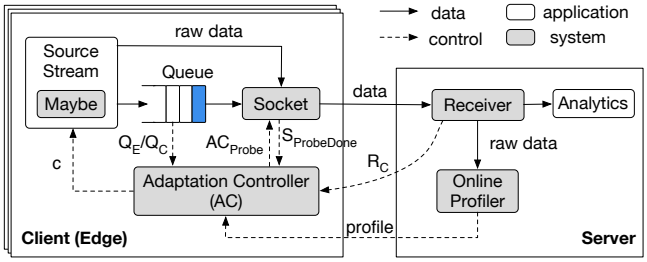
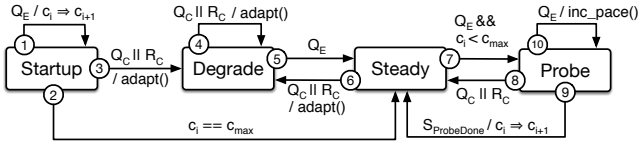
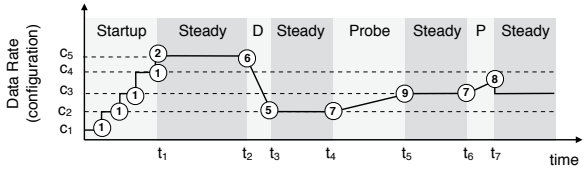


Figure 5. Runtime adaptation system architecture.



(a) Rate adaptation as a state machine.



(b) An example illustrating the adaptation algorithm.

Figure 6. Runtime adaptation algorithm.

Figure 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. `Socket` uses TCP as the underlying protocol for congestion control and estimates the available bandwidth using application-level delivery rate. 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` [18]) and reports sudden application-level latency spikes to clients as congestion signals (R_C). If a new profile is learned by the online profiler, it is fed back to AC for subsequent adaptation.

Figure 6a shows the adaptation algorithm with a state machine model and Figure 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 each state and transitions.

- **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 `Steady`. It changes to `Degrade` when congestion occurs. If $c < c_{max}$ and it receives Q_E, AC starts `Probe` to check for more available bandwidth.
- **Probe: more bandwidth for a higher accuracy.** Advancing c_i directly may cause congestion if $B(c_{i+1}) \gg B(c_i)$. To allow a smooth increase, AC requests `Socket` to probe by sending additional traffic controlled by `probe_gain` (in `inc_pace()`, similar to BBR [18]). Raw data is used for probing if available, otherwise we inject dummy traffic. 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`. The explicit `Probe` phase stabilizes feedback loop and prevents oscillation.

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 bound on the bandwidth consumption (e.g., do not exceed 1 Mbps) or a lower bound on application accuracy (e.g., do not fall below 75%).

For multiple applications, their profiles allow novel bandwidth allocation schemes such as utility fairness. Different from 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:

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

Solving this optimization is computationally hard. AWStream uses heuristics similar to VideoStorm [96]: it starts with c_1^t and improves the application t with the worst accuracy; this process iterates until all bandwidth is allocated. In this paper, we demonstrate resource allocation with two applications as a proof-of-concept in §5.4 and leave an extensive study regarding scalability for future.

Application	Knobs	Accuracy	Dataset
Augmented Reality	resolution frame rate quantization	F1 score [70]	iPhone video clips training: office (24 s) testing: home (246 s)
Pedestrian Detection	resolution frame rate quantization	F1 score	MOT16 [53] training: MOT16-04 testing: MOT16-03
Log Analysis (Top-K, K=50)	head (N) threshold (T)	Kendall's τ [4]	SEC.gov logs [59] training: 4 days testing: 16 days

Table 3. Application details.

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.

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 parts: knobs, accuracy functions, and datasets.

Augmented Reality. We target at augmented reality applications running on mobile phones that recognize objects by offloading the heavy computation elsewhere (e.g., the cloud).

Our implementation uses OpenCV [14] for image-related operations and YOLO [66, 67], a GPU-enabled pre-trained neural network, for object recognition. Videos are encoded with H.264 [69]. Our implementation uses GStreamer [83] with `x264enc` plugin (`zerolatency` and constant quality). The quantization factor affecting encoding quality becomes a knob in addition to image resolutions and frame rates.

Object recognition returns a list of bounding boxes with the object type. Each bounding box is a rectangle with normalized coordinates on the image. We compare the detection against

[3] https://github.com/awstream/awstream

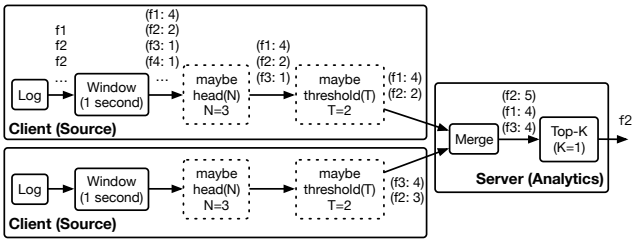


Figure 7. A distributed Top-K application with two degradation operations: `head` and `threshold`. In this example, `f2`, which is not in Top-1 for either client, becomes the global Top-1 after the merge. It would have been purged if the clients use threshold T=3, demonstrating degradation that reduces data sizes affects fidelity.

the reference result from raw data, and declare it success if the intersection over union (IOU) is greater than 50% [29] and the object type matches. We use F1 score [70] as the accuracy function. In terms of dataset, we collected our own video clips: the training data is a 24-second long video of an office environment; the test data is a 246-second long video of a home environment.

Pedestrian Detection. This application analyzes streams of videos from installed CCTV cameras and detects pedestrians inside. We use a similar setup (OpenCV and GStreamer) as our augmented reality application except for the analytical function. To detect pedestrians, we use GPU-accelerated histogram of oriented gradients (HOG) [24] with the default linear SVM classifier from OpenCV. Because we do not recognize individual pedestrians, a successful detection in this case only requires matching the bounding box. Our evaluation uses MOT16 dataset [53] for both profiling and runtime.

Distributed Top-K. This application aggregates machine logs from geo-distributed servers to find out the Top-K most accessed files, similar to many Top-K queries [10].

Figure 7 illustrates our processing pipeline with two degradation operations. First each source node summarizes the log using `Window` operator to reduce the data size, a preprocessing step. As many real-world access patterns follow a long tail distribution, there can be a large-but-irrelevant tail that contributes little to the final Top-K. Each source node then filters the tail: (1) head(`N`) takes the top `N` entries; (2) threshold(`T`) filters small entries whose count is smaller than `T`. These two operations affect the final result and the exact impact depends on data distribution. We implement these two operators by using AWStream's `maybe` abstraction.

To measure the accuracy, we need to compare the correlation between two ranked list. Kendall's τ [4] is a correlation measure of the concordance between two ranked list. The output ranges from -1 to 1, representing no agreement to complete agreement. To integrate with AWStream, we convert Kendall's τ to [0, 1] with a linear transformation. For our evaluation, we set K as 50 and use Apache log files that record and store user access statistics for the SEC.gov website. The logs are split into four groups, simulating four geo-distributed nodes monitoring web accesses. To match the load of popular web servers, we compress one hour's logs into one second.

5 EVALUATION

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

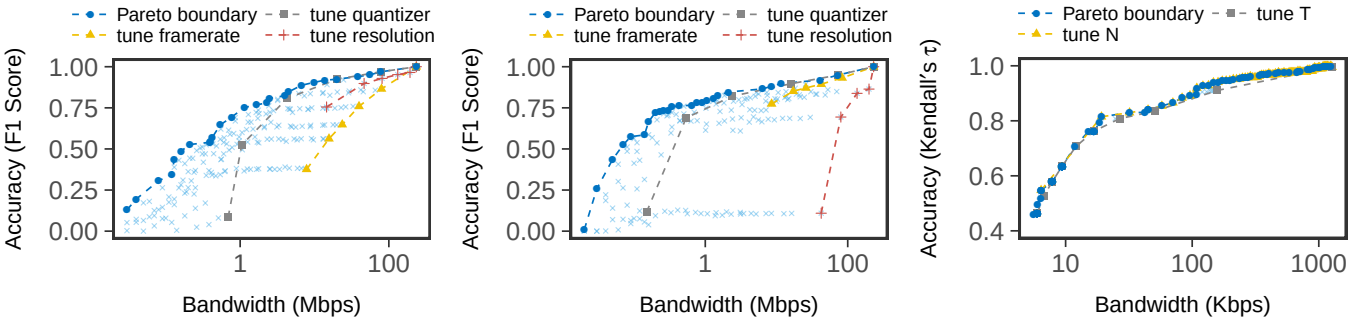
- §5.1 AWStream generates Pareto-optimal profiles across multiple dimensions with precision (Figure 8).
- §5.2 Our parallel and sampling techniques speeds up offline and online profiling (Figure 9, Figure 10).
- §5.3 At runtime, AWStream achieves sub-second latency and nominal accuracy drop for all applications (Figure 11, Figure 12) and across various network conditions (Figure 13).
- §5.4 AWStream profiles allow different resource allocations: resource fairness and utility fairness (Figure 14).

5.1 Application Profiles

We run offline profiling using the training dataset described in Table 3 and show the learned profiles in Figure 8. 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.



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

Figure 8. 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.

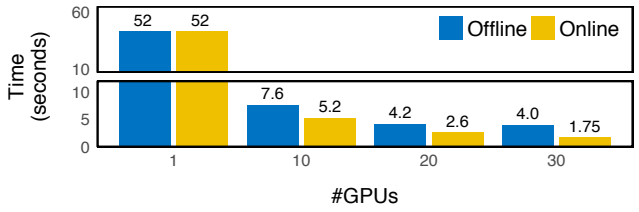


Figure 9. Parallelism speeds up both offline and online profiling. The y-axis shows the profiling time for 1-second video.

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.

In AR, there are 216 different configurations: 6 resolutions, 6 frame rates and 6 quantization levels. AR uses YOLO [67], 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 discusses parallel and sampling techniques to improve the profiling efficiency; we present their evaluations as follows.

Parallelism reduces the profiling time (Figure 9). 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 [41] to minimize the makespan and reduces the overhead to 1.75X with 30 GPUs (29× gain).

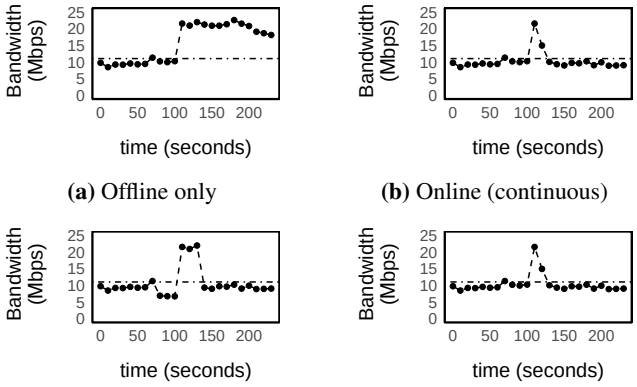
Sampling techniques speed up online profiling (Figure 10). Before we evaluate the speed up, we validate *model drift* with real-world data. When using the profile trained in an office environment, the application should use a configuration of 1280x720, 30 FPS and 20 quantization to meet an 11 Mbps goal. We test it against a home environment; but 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 bandwidth, as illustrated in Figure 10a.

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. Figure 10b 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. Figure 10c 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 $\mathbb{P}$, the savings can be substantial. A full profiling is only triggered if there is a significant difference. Figure 10d 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 Figure 10b and Figure 10d). The

[4] YOLO resizes images to fixed 416×416 resolutions as required by the neural network. Evaluating images with different resolutions takes similar time.



(a) Offline only **(b)** Online (continuous)

(c) Partial data **(d)** Partial configurations

Figure 10. 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×.

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 to further reduce the profiling overhead. For example, scheduling 5 GPUs running 5 configurations 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 bandwidth across geo-distributed sites and compare AWStream with baselines including streaming over TCP/UDP, JetStream, and video streaming. Due to limited space, we discuss AR in depth and only present the results of PD/TK.

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 20 quantization; it consumes about 14 Mbps.

Our bandwidth control scheme follows JetStream [65]. During the experiment, we use the Linux `tc` utility with HTB [25, 40] 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 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—such that all clients can use at most 25 Mbps of network bandwidth.

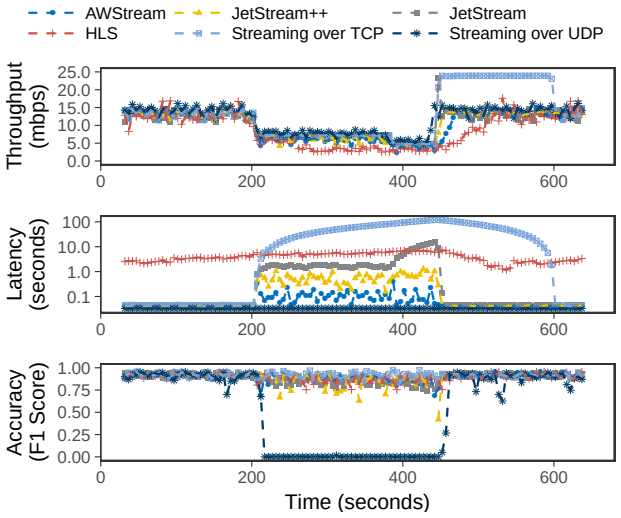
We compare AWStream with the following baselines:

- Streaming over TCP/UDP (non-adaptive). For TCP, we reuse AWStream runtime that runs over TCP but disable the adaptation. For UDP, we use FFmpeg [12] to stream video: RTP/UDP [74] for media and RTSP for signaling [75]; as in typical video conferencing and IP cameras [27, 42].
- Adaptive video streaming. We use HTTP Live Streaming (HLS) to represent popular adaptive video streaming techniques. Our setup resembles personalized live streaming systems [93] but uses a smaller chunk for low latency (1 second instead of typical 2-10 seconds).
- JetStream with the manual policy described in §2.3.
- JetStream++, a modified version of JetStream that uses the profile learned by AWStream.

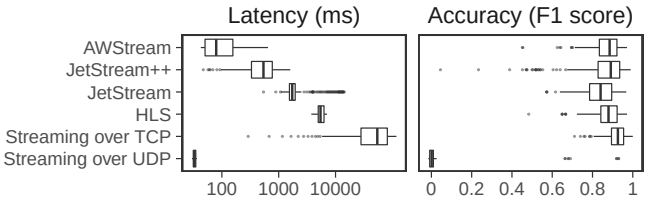
At runtime, AWStream differs from JetStream in both policy and adaptation. JetStream++ improves over JetStream by using our Pareto-optimal profile. AWStream improves the performance further with two major changes: (*i*) AWStream directly measures the delivery rate to select an appropriate configuration 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. Figure 11a shows the runtime behavior of AWStream and all baselines in time series. Figure 11b summarizes the latency and accuracy with box plots during bandwidth shaping (between t=200s and t=440s).

The throughput figure (Figure 11a) 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



(a) Time-series plot of the runtime behaviors: throughput (top), showing the effect of bandwidth shaping; latency (middle) in log scale; and accuracy (bottom). Overlapped lines may be hard to read; we present the same results in Figure 11b for clarity.



(b) Latency and accuracy during the traffic shaping (t=200s–440s).

Figure 11. For AR, AWStream simultaneously achieves low latency and high accuracy (accuracy has a smaller variation).

adaptation (see their throughput drops). When we stop shaping at t=440s, TCP catches up by 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's throughput increases 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 Figure 11a and Figure 11b) 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.[5] HLS's latency fluctuates around 4-5 seconds due to chunking, buffering, and network variations, on par with recent literature [93]. Both JetStream and JetStream++ are able to adapt during traffic shaping. With a more precise and fine-grain policy, JetStream++ 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 easily overcompensate and

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

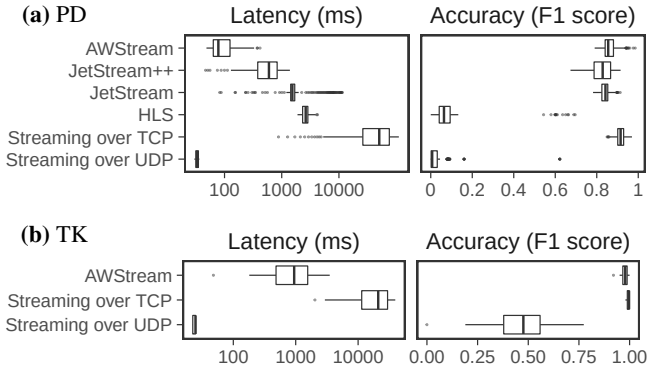


Figure 12. PD and TK performance summary. Box plot shows latency and accuracy during the traffic shaping (i.e., t=200s-440s).

exhibit oscillation 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 Figure 11a and Figure 11b) show that other than UDP, most schemes are able to maintain high accuracy. streaming over TCP 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 (6% accuracy as in Figure 12a).

In summary, Figure 11 shows that AWStream achieves low latency and high accuracy simultaneously. The latency improvement over JetStream allows interactive applications, such as AR, to feel responsive rather than interrupted [57]. We show the results *during shaping* in a different form in Figure 1 to discuss the trade-off between fidelity and freshness.[6]

Pedestrian Detection. The setup for PD is the same with AR: three clients and one server on EC2. $c_{\max}$ is 1920x1080 resolution, 10 FPS and 20 quantization; it consumes about 12 Mbps. For PD, AWStream learns that resolution is more important than frame rate. Hence it favors 1080p with 10FPS over 900p with 30FPS. We use the same bandwidth shaping schedule and baselines as AR. Figure 12a shows the result and most observations about latency/accuracy are the same as AR. HLS has a poor accuracy because it reduces resolution

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

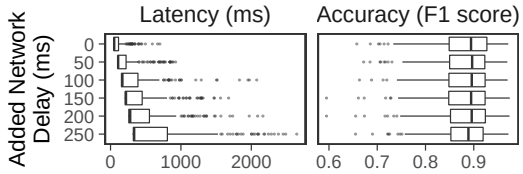


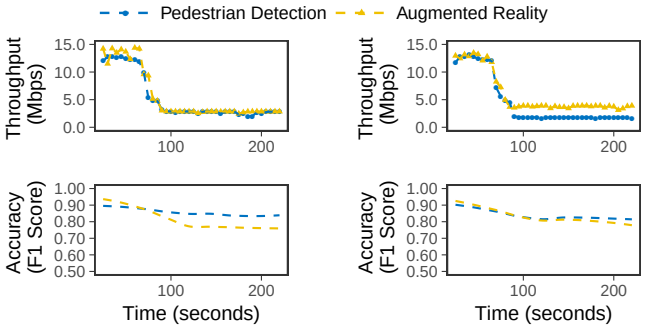
Figure 13. AWStream maintains low latency and high accuracy under different network delay conditions.

and encoding quality during adaptation. AWStream is able to achieve the lowest latency (78 ms) with small accuracy drop (86%, 6% drop in comparison to TCP). In comparison to JetStream, AWStream improves the latency by 20× (from 1535 ms to 78 ms) and accuracy by 1% (from 84% to 85%).

Top-K. For TK, we use four clients and one server because our logs are split into four groups. $c_{\max}$ is $N = 9750$ for `head` and $T = 0$ for `threshold`; it consumes about 1.2 Mbps. Because the overall bandwidth consumption is much smaller than video analytics, we modify the bandwidth parameter: during t=200-380s, we limit the bandwidth to 750 Kbps; during t=380-440s, the bandwidth is 500 Kbps; the background limit is 2.5 Mpbs. We only compared AWStream with streaming over TCP and UDP. JetStream's Top-K is based on TPUT [16] that targets at queries asked hourly or daily. We did not implement our Top-K pipeline (Figure 7) with JetStream because video analytics suffice the purpose of comparison. Figure 12b shows the evaluation results. Streaming over TCP has the highest accuracy (99.7%) but the worst latency (up to 40 seconds). Streaming over UDP has the lowest latency but the worst accuracy (52%). AWStream achieves low latency (1.1 seconds) and high accuracy (98%) simultaneously. Notice that because TK's source generates data every second after `Window`, one object in the queue leads to one second latency.

Performance with Varying Network Delays
AWStream targets at wide area whose key characteristic is the large variation in latency [48]. While we have conducted experiments using real-world setup on 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, up to 250 ms each way, so the added RTT can be as high as 500 ms. The delay follows a normal distribution where the variation is 10% (e.g., 250 ± 25ms).

Figure 13 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, because the congestion detection is slow 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.



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

Figure 14. AWStream allows different resource allocation schemes.

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 (Figure 14a), 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 (Figure 14b).

6 DISCUSSION AND FUTURE WORK

Reducing Developer Effort. While AWStream simplifies developing adaptive applications, there are still application-specific parts required for developers: wrapping appropriate `maybe` calls, providing training data, and implementing accuracy functions. Because AWStream's API is extensible, we plan to build libraries for common degradation operations and accuracy functions, similar to machine learning libraries.

Fault-tolerance and 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 [95], it is difficult to make edge clients failure-oblivious. We leave failure detection and recovery as a future work.

Profile Modeling. AWStream currently performs an exhaustive search when profiling. While parallelism and sampling are effective, profiling complexity grows exponentially with the number of knobs. Inspired by recent success of using Bayesian Optimization [61, 78, 80] to model black-box functions, we are currently exploring multi-objective Bayesian

Optimization [37] that can find *near-optimal* configurations without exhaustive search.

Bandwidth Estimation and Prediction. Accurately estimating and predicting available bandwidth in wide area remains a challenge [39, 98]. AWStream uses network throughput and behaves cautiously to avoid building up queues: congestion is detected at both sender/receiver; data rate only increases after probing. Recent research on adaptive video streaming explores model predictive control (MPC) [81, 94] and neural network [51]. We plan to explore these techniques next.

7 RELATED WORK

JetStream. JetStream is the first to use degradation to reduce latency for wide-area streaming analytics. Compared to JetStream, AWStream makes five major contributions: (1) a novel API design to specify degradation in a simple and composable way; (2) automatic offline profiling to search for Pareto-optimal configurations; (3) online profiling to address model drift; (4) an improved runtime system achieving sub-second latency (comparison in §5.3); (5) support for different resource allocation policies for multiple applications.

Stream Processing Systems. Early streaming databases [1, 19] have explored the use of dataflow models with specialized operators for stream processing. Recent research projects and open-source systems [6, 17, 45, 86, 95] primarily focus on fault-tolerance in the context of a single cluster. When facing back pressure, Storm [86], Spark Streaming [95] and Heron [45] throttle data ingestion; Apache Flink [17] uses edge-by-edge back-pressure techniques similar to TCP flow control; Faucet [47] leaves the flow control logic up to developers. While our work has a large debt to prior streaming work, AWStream targets at the wide area and explicitly explores the trade-off between data fidelity and freshness.

Approximate Analytics. The idea of degrading computation fidelity for responsiveness is also explored elsewhere, such as SQL queries [5, 9], real-time processing [30], and video processing within large clusters [96]. They employ techniques such as programming language support [72], sampling [32], sketches [23], and online aggregation [36]. The trade-off between available resource and application accuracy is a common theme among all these systems. AWStream targets at wide-area streaming analytics, an emerging application domain especially with the advent of IoT.

WAN-Aware Systems. Geo-distributed systems need to cope with high latency and limited bandwidth. While some systems assume the network can prioritize a small amount of critical data under congestion [20], most systems reduce data sizes in the first place to avoid congestion (e.g., LBFS [55]). Over the past two years, we have seen a plethora of geo-distributed analytical frameworks [43, 64, 89–91] that incorporate heterogeneous wide-area bandwidth into query optimization to minimize data movement. While effective in speeding up analytics, these systems focus on batch tasks such as MapReduce jobs or SQL queries. Such tasks are often executed infrequently and without real time constrain. In contrast, AWStream processes streams continuously in real time.

(Adaptive) Video Streaming. Multimedia streaming protocols (e.g., RTP [74]) aim to be fast instead of reliable. While they can achieve low latency, their accuracy can be poor under congestion. Recent work has moved towards HTTP-based protocols and focused on designing adaptation strategy to improve QoE, as in research [51, 81, 94] and industry (HLS [62], DASH [52, 79]). These adaptation strategies are often pull-based: client keeps checking the index file for changes. And clients have to wait for the next chunk (typically 2-10 seconds). In addition, as we have shown in §5.3, their adaptation is a poor match 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 low latency.

QoS. Most QoS work [31, 76, 77] in the 1990s focuses on network-layer solutions that are not widely deployable. AWStream adopts an end-host application-layer approach ready today for WAN. For other application-layer approaches [87], 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 [68]. Fundamentally, AWStream does not provide hard QoS guarantees; instead AWStream maximizes achievable accuracy (application performance) and minimizes latency (system performance) with respect to bandwidth constraints: 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.

ACKNOWLEDGMENTS

We thank the anonymous SIGCOMM reviewers and our shepherd Mosharaf Chowdhury for their thoughtful feedback on this paper. Kaifei Chen and Pan Hu provided valuable feedback to an early version of this manuscript.

This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. Xin Jin is supported by NSF under grant CRII-NeTS-1755646 and a Facebook Communications & Networking Research Award.

REFERENCES

[1] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. 2005. The Design of the Borealis Stream Processing Engine. In *CIDR*, Vol. 5. Asilomar, CA, 277–289.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[3] Omid Abari, Dinesh Bharadia, Austin Duffield, and Dina Katabi. 2017. Enabling High-Quality Untethered Virtual Reality. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 531–544. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/abari

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

[5] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. ACM, 29–42. https://doi.org/10.1145/2465351.2465355

[6] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: Fault-tolerant Stream Processing at Internet Scale. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1033–1044. https://dl.acm.org/citation.cfm?id=2536229

[7] Sara Alspaugh, Bei Di Chen, Jessica Lin, Archana Ganapathi, Marti A Hearst, and Randy H Katz. 2014. Analyzing Log Analysis: An Empirical Study of User Log Mining. In *Proceedings of the 28th USENIX Conference on Large Installation System Administration (LISA'14)*. USENIX Association, Berkeley, CA, USA, 53–68. http://dl.acm.org/citation.cfm?id=2717491.2717495

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

[9] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. 2014. GRASS: Trimming Stragglers in Approximation Analytics. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 289–302. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/ananthanarayanan

[10] Brian Babcock and Chris Olston. 2003. Distributed Top-K Monitoring. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*. ACM, New York, NY, USA, 28–39. https://doi.org/10.1145/872757.872764

[11] Naveen Balani and Rajeev Hathi. 2016. *Enterprise IoT: A Definitive Handbook*. CreateSpace Independent Publishing Platform.

[12] Fabrice Bellard, M Niedermayer, et al. 2012. FFmpeg. https://www.ffmpeg.org/. (2012).

[13] Sanjit Biswas, John Bicket, Edmund Wong, Raluca Musaloiu-e, Apurv Bhartia, and Dan Aguayo. 2015. Large-scale Measurements of Wireless Network Behavior. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*. ACM, New York, NY, USA, 153–165. https://doi.org/10.1145/2785956.2787489

[14] G. Bradski. 2000–2017. The OpenCV Library. *Doctor Dobbs Journal* (2000–2017). http://opencv.org

[15] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. 2013. Mapping the Expansion of Google's Serving Infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC'13)*. ACM, New York, NY, USA, 313–326. https://doi.org/10.1145/2504730.2504754

[16] Pei Cao and Zhe Wang. 2004. Efficient Top-K Query Calculation in Distributed Networks. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 206–215.

[17] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Data Engineering* 38, 4 (2015). https://flink.apache.org/

[18] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, et al. 2017. BBR: Congestion-based Congestion Control. *Commun. ACM* 60, 2 (2017), 58–66. http://dl.acm.org/citation.cfm?id=3042068.3009824

[19] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J Franklin, Joseph M Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R Madden, Fred Reiss, and Mehul A Shah. 2003. TelegraphCQ: Continuous Dataflow Processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*. ACM, New York, NY, USA, 668–668. https://doi.org/10.1145/872757.872857

[20] Brian Cho and Marcos K Aguilera. 2012. Surviving Congestion in Geo-Distributed Storage Systems. In *USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX, 439–451. https://www.usenix.org/conference/atc12/technical-sessions/presentation/cho

[21] Cisco. 2013. The Zettabyte Era: Trends and Analysis. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. *Cisco White Paper* (2013).

[22] Benjamin Coifman, David Beymer, Philip McLauchlan, and Jitendra Malik. 1998. A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance. *Transportation Research Part C: Emerging Technologies* 6, 4 (1998), 271–288.

[23] Graham Cormode. 2011. Sketch Techniques for Massive Data. *Synposes for Massive Data: Samples, Histograms, Wavelets and Sketches* (2011), 1–3.

[24] Navneet Dalal and Bill Triggs. 2005. Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01 (CVPR '05)*. IEEE Computer Society, Washington, DC, USA, 886–893. https://doi.org/10.1109/CVPR.2005.177

[25] Martin Devera. 2001–2003. HTB Home. http://luxik.cdi.cz/~devik/qos/htb/. (2001–2003). Accessed: 2017-04-08.

[26] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. 2012. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE transactions on pattern analysis and machine intelligence* 34, 4 (2012), 743–761.

[27] Arjan Durresi and Raj Jain. 2005. RTP, RTCP, and RTSP-Internet Protocols for Real-Time Multimedia Communication. (2005).

[28] ESnet. 2014-2017. iPerf: The TCP/UDP bandwidth measurement tool. http://software.es.net/iperf/. (2014-2017). Accessed: 2017-03-07.

[29] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision* 88, 2 (June 2010), 303–338. https://doi.org/10.1007/s11263-009-0275-4

[30] Anne Farrell and Henry Hoffmann. 2016. MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 421–435. https://www.usenix.org/conference/atc16/technical-sessions/presentation/farrell

[31] Domenico Ferrari and Dinesh C Verma. 1990. A Scheme for Real-time Channel Establishment in Wide-area Networks. *IEEE journal on Selected Areas in communications* 8, 3 (1990), 368–379.

[32] Minos N. Garofalakis and Phillip B. Gibbon. 2001. Approximate Query Processing: Taming the TeraBytes. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 725–. http://dl.acm.org/citation.cfm?id=645927.672356

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

[34] Google. 2009-2017. Nest Cam Indoor. https://www.dropcam.com. (2009-2017). Accessed: 2017-04-03.

[35] Sarthak Grover, Mi Seon Park, Srikanth Sundaresan, Sam Burnett, Hyojoon Kim, Bharath Ravi, and Nick Feamster. 2013. Peeking Behind the NAT: An Empirical Study of Home Networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC'13)*. ACM, New York, NY, USA, 377–390. https://doi.org/10.1145/2504730.2504736

[36] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. 1997. Online Aggregation. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*. ACM, New York, NY, USA, 171–182. https://doi.org/10.1145/253260.253291

[37] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. 2016. Predictive Entropy Search for Multi-Objective Bayesian Optimization. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 1492–1501. http://proceedings.mlr.press/v48/hernandez-lobatoa16.html

[38] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, USENIX Association. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh

[39] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*. ACM, New York, NY, USA, 225–238. https://doi.org/10.1145/2398776.2398800

[40] Bert Hubert. 2002. Linux Advanced Routing & Traffic Control. http://lartc.org/. (2002). Accessed: 2017-04-06.

[41] David Karger, Cliff Stein, and Joel Wein. 2010. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC. 20–20 pages. http://dl.acm.org/citation.cfm?id=1882723.1882743

[42] Joel W King. 2009. 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).

[43] Konstantinos Kloudas, Margarida Mamede, Nuno Preguiça, and Rodrigo Rodrigues. 2015. Pixida: optimizing data parallel jobs in wide-area data analytics. *Proceedings of the VLDB Endowment* 9, 2 (2015), 72–83. https://dl.acm.org/citation.cfm?id=2850582

[44] Andrew Krioukov, Gabe Fierro, Nikita Kitaev, and David Culler. 2012. Building Application Stack (BAS). In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys '12)*. ACM, New York, NY, USA, 72–79. https://doi.org/10.1145/2422531.2422546

[45] 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 (SIGMOD'15)*. ACM, New York, NY, USA, 239–250. https://doi.org/10.1145/2723372.2742788

[46] Frank Langfitt. 2013. 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.

[47] Andrea Lattuada, Frank McSherry, and Zaheer Chothia. 2016. Faucet: a User-Level, Modular Technique for Flow Control in Dataflow Engines. In *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. ACM, 2.

[48] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC'10)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/1879141.1879143

[49] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

[50] Chaochao Lu and Xiaoou Tang. 2015. Surpassing Human-level Face Verification Performance on LFW with Gaussian Face. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 3811–3819. https://dl.acm.org/citation.cfm?id=2888245

[51] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 197–210. https://doi.org/10.1145/3098822.3098843

[52] MG Michalos, SP Kessanidis, and SL Nalmpantis. 2012. Dynamic Adaptive Streaming over HTTP. *Journal of Engineering Science and Technology Review* 5, 2 (2012), 30–34.

[53] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. 2016. MOT16: A Benchmark for Multi-Object Tracking. *arXiv preprint arXiv:1603.00831* (2016). https://motchallenge.net/

[54] Matthew K Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. 2015. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 311–324. https://dl.acm.org/citation.cfm?id=2787475

[55] Athicha Muthitacharoen, Benjie Chen, and David Mazières. 2001. A Low-bandwidth Network File System. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP'01)*. ACM, New York, NY, USA, 174–187. https://doi.org/10.1145/502034.502052

[56] Cisco Visual Networking. 2016. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. *Cisco White Paper* (2016).

[57] Jakob Nielsen. 1994. *Usability Engineering*. Elsevier.

[58] Ashkan Nikravesh, David R Choffnes, Ethan Katz-Bassett, Z Morley Mao, and Matt Welsh. 2014. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362 (PAM 2014)*. Springer-Verlag New York, Inc., New York, NY, USA, 12–22. https://doi.org/10.1007/978-3-319-04918-2_2

[59] The Division of Economic and Risk Analysis (DERA). 2003–2016. EDGAR Log File Data Set. https://www.sec.gov/data/edgar-log-file-data-set. (2003–2016). Accessed: 2017-01-25.

[60] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. 2011. A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video. In *Proceedings of the*

2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11). IEEE Computer Society, Washington, DC, USA, 3153–3160. https://doi.org/10.1109/CVPR.2011.5995586

[61] Omid Alipourfard and Hongqiang Harry Liu and Jianshu Chen and Shivaram Venkataraman and Minlan Yu and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 469–482. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[62] Roger Pantos and William May. 2016. HTTP Live Streaming. (2016). https://tools.ietf.org/html/draft-pantos-http-live-streaming-19

[63] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep Face Recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, Article 41, 12 pages. https://doi.org/10.5244/C.29.41

[64] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-Distributed Data Analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*. ACM, New York, NY, USA, 421–434. https://doi.org/10.1145/2785956.2787505

[65] 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 *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 275–288. http://dl.acm.org/citation.cfm?id=2616448.2616474

[66] Joseph Redmon. 2013–2017. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/. (2013–2017).

[67] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242* (2016). http://arxiv.org/abs/1612.08242

[68] Reza Rejaie, Mark Handley, and Deborah Estrin. 2000. Layered Quality Adaptation for Internet Video Streaming. *IEEE Journal on Selected Areas in Communications* 18, 12 (2000), 2530–2543.

[69] Iain E. Richardson. 2010. *The H.264 Advanced Video Compression Standard* (2nd ed.). Wiley Publishing.

[70] C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.

[71] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. 2008. LabelMe: a Database and Web-based Tool for Image Annotation. *Int. J. Comput. Vision* 77, 1-3 (May 2008), 157–173. https://doi.org/10.1007/s11263-007-0090-8

[72] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 164–174.

[73] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. https://doi.org/10.1109/MPRV.2009.82

[74] H Schulzrinne, S Casner, R Frederick, and V Jaconson. 2006. RTP: A Transport Protocol for Real-Time. (2006).

[75] H Schulzrinne, A Rao, and R Lanphier. 1998. RTSP: Real time streaming protocol. *IETF RFC2326, april* (1998).

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

[77] Scott Shenker, R Braden, and D Clark. 1994. Integrated services in the Internet architecture: an overview. *IETF Request for Comments (RFC)* 1633 (1994).

[78] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in neural information processing systems*. 2951–2959.

[79] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming over the Internet. *IEEE MultiMedia* 18, 4 (Oct. 2011), 62–67. https://doi.org/10.1109/MMUL.2011.71

[80] Benjamin Solnik, Daniel Golovin, Greg Kochanski, John Elliot Karro, Subhodeep Moitra, and D Sculley. 2017. Bayesian Optimization for a Better Dessert. (2017). https://research.google.com/pubs/archive/46507.pdf

[81] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, ACM, 272–285. https://doi.org/10.1145/2934872.2934898

[82] Srikanth Sundaresan, Sam Burnett, Nick Feamster, and Walter De Donato. 2014. BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 383–394. http://dl.acm.org/citation.cfm?id=2643634.2643673

[83] GStreamer Team. 2001–2017. GStreamer: Open Source Multimedia Framework. (2001–2017). https://gstreamer.freedesktop.org/

[84] TeleGeography. 2016. Global Internet Geography. https://www.telegeography.com/research-services/global-internet-geography/. (2016). Accessed: 2017-04-10.

[85] James Temperton. 2015. One nation under CCTV: the future of automated surveillance. http://www.wired.co.uk/article/one-nation-under-cctv. (2015). Accessed: 2017-01-27.

[86] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 147–156. https://dl.acm.org/citation.cfm?id=2595641

[87] Bobby Vandalore, Wu-chi Feng, Raj Jain, and Sonia Fahmy. 2001. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Real-Time Imaging* 7, 3 (2001), 221–235.

[88] Paul Viola and Michael Jones. 2001. 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*, Vol. 1. I–511–I–518 vol.1. https://doi.org/10.1109/CVPR.2001.990517

[89] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. Clarinet: WAN-Aware Optimization for Analytics Queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, GA, 435–450. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/viswanathan

[90] 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 (SIGMOD'15)*. ACM, New York, NY, USA, 1087–1092. https://doi.org/10.1145/2723372.2735365

[91] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. 2015. Global Analytics in the Face of Bandwidth and Regulatory Constraints. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 323–336. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/vulimiri

[92] Gregory K Wallace. 1991. The JPEG Still Picture Compression Standard. *Commun. ACM* 34, 4 (April 1991), 30–44. https://doi.org/10.1145/103085.103089

[93] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y Zhao. 2016. Anatomy of a Personalized Livestreaming System. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 485–498.

[94] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. 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 (SIGCOMM'15)*. ACM, New York, NY, USA, 325–338. https://doi.org/10.1145/2785956.2787486

[95] 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 (SOSP'13)*. ACM, New York, NY, USA, 423–438. https://doi.org/10.1145/2517349.2522737

[96] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[97] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 426–438. https://dl.acm.org/citation.cfm?id=2790123

[98] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha. 2015. Can Accurate Predictions Improve Video Streaming in Cellular Networks?. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile '15)*. ACM, New York, NY, USA, 57–62. https://doi.org/10.1145/2699343.2699359