AWStream: Adaptive Wide-Area Streaming Analytics

Ben Zhang
University of California, Berkeley
benzh@cs.berkeley.edu

Xin Jin*
Johns Hopkins University
xinjin@cs.jhu.edu

Sylvia Ratnasamy
University of California, Berkeley
sylvia@eecs.berkeley.edu

John Wawrzynek
University of California, Berkeley
johnw@eecs.berkeley.edu

Edward A. Lee
University of California, Berkeley
eal@eecs.berkeley.edu

Abstract

The emerging class of wide-area streaming analytics faces the challenge of scarce and variable WAN bandwidth. Applications that solely rely on transport protocols like TCP and UDP suffer from increased latency or degraded accuracy. Existing approaches that adapt to network changes are often application-specific or require extensive developer effort.

We present AWStream, a stream processing system that simultaneously achieves low latency and high accuracy in the wide area, requiring minimal developer effort. 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 and precise profile that models accuracy and bandwidth tradeoff; and (*iii*) at runtime, it adjusts the application data rate to match the available bandwidth. We evaluate AWStream using three real-world applications: pedestrian detection, augmented reality, and monitoring log analysis. Our experiments show that AWStream achieves sub-second latency with nominal accuracy drop.

CCS Concepts • **Computer systems organization** → *Distributed architectures*; *Real-time system architecture*; • **Information systems** → *Multimedia streaming*; *Computing platforms*;

ACM Reference format:
Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. 2017. AWStream: Adaptive Wide-Area Streaming Analytics. In *Proceedings of , Shanghai, China, October 29–31, 2017 (Submitted to SOSP'17),* 17 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Also with UC Berkeley.

Submitted to SOSP'17, October 29–31, 2017, Shanghai, China

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 [40, 66]. Buildings are increasingly equipped with a wide variety of sensors to improve energy efficiency, occupant comfort, reliability, and maintenance [38]. Geo-distributed infrastructure, such as content delivery networks (CDNs), analyzes user requests from machine logs over the globe [45]. 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 [67], Spark Streaming [75], and VideoStorm [76], are capable of handling 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 [32, 72]. Moreover, WAN bandwidth growth has been decelerating for many years [65] while traffic demands are growing at a staggering rate [35].

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, 54, 60]. However, communication is not entirely avoidable: (*i*) some analytical jobs require joining or aggregating data from multiple geo-distributed sites [53, 70]; (*ii*) the edge benefits substantially from central computing resources such as GPUs and TPUs [2] in the cloud; and (*iii*) end-devices such as cameras and mobile devices suffer from limited bandwidth in last-hop wireless links when running processing on nearby edge infrastructure [3, 77].

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

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

Manual policies, such as sampling, allow developers to trade data fidelity for freshness [54]. However, it's difficult to write accurate policies without extensive expertise or

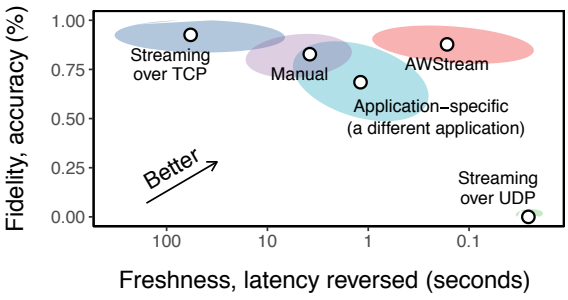


Figure 1. The trade-off space between data freshness and fidelity when facing insufficient bandwidth (details in §5.3).

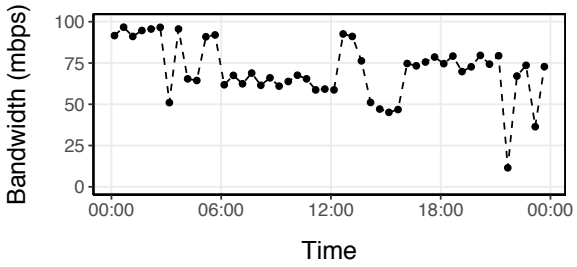


Figure 2. Bandwidth variations throughout the day between Amazon EC2 sites (from Ireland to California).

without considerable effort. In practice, developers write manual policies based on some set of 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 algorithm for one application will work poorly for another application, if performance metrics or data distributions change. For example, video streaming focuses on quality of experience (QoE) [74]. Because humans favor smoothness over image quality, video streaming maintains a high frame rate, e.g. 25 FPS, and reduces image resolutions under bandwidth limit. This is a poor match for machine-based analytics that rely on image details.

To achieve low latency and high accuracy simultaneously with minimal developer effort, we design and implement AWStream, a stream processing system for the wide area. The key idea is to build an accurate and precise performance model instead of relying on manual or application-specific policies. AWStream's solution is three-fold: easy-to-use APIs, 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 APIs are simple, modular 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 potential model drifts. We exploit parallelism and sampling-based profiling 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 network congestion, our rate adaptation algorithm increases the degradation level to reduce bandwidth demand, 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 for developers to control application behaviors, e.g., by limiting the maximum allowed WAN bandwidth. For multiple applications, the profiles allow bandwidth allocation among competing tasks for utility fairness.

To evaluate AWStream, we have built three streaming applications: pedestrian detection (PD), augmented reality (AR), and distributed Top-K analysis (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 a set of `maybe` operators to incorporate adaptation with existing stream processing models. Our programming abstraction is simple, modular 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 9×.
- Using runtime experiments on geo-distributed EC2 nodes, AWStream achieves low latency and high accuracy simultaneously for all applications—sub-second latency and 2% accuracy drop for video analytics, 4-second latency and 1% accuracy drop for TK.

2 Motivation

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

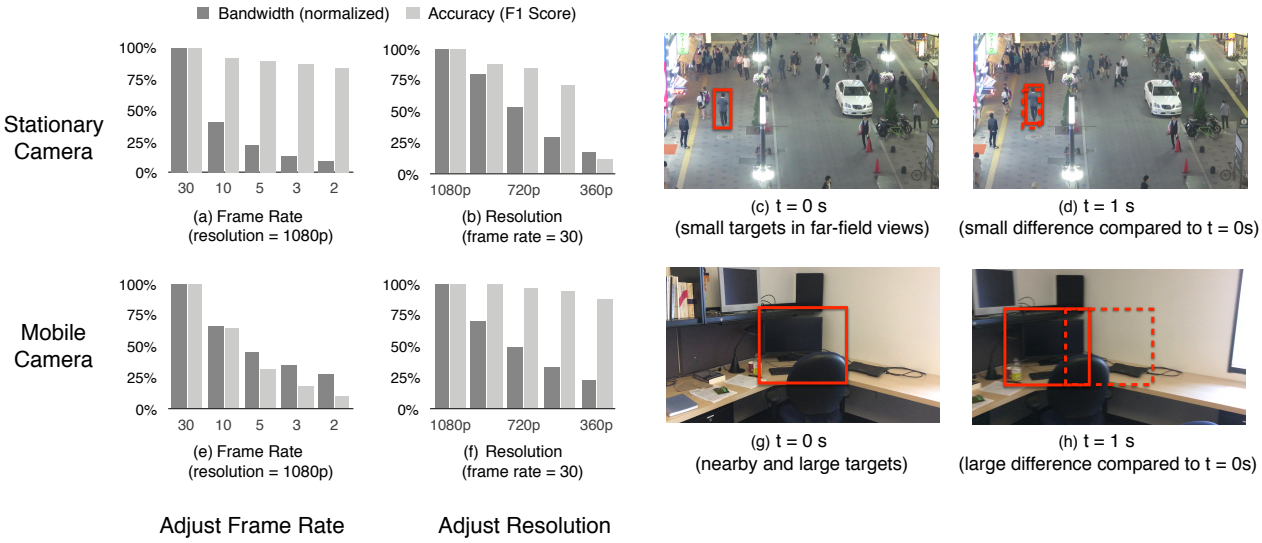


Figure 3. The measured bandwidth and application accuracy for two video analytics applications. (1) Different degradation strategies have different impacts on accuracy, e.g., degrading frame rate in (a) vs. degrading resolution in (b). (2) The same degradation strategy has different impacts on different applications, e.g., degrading frame rates works well for stationary camera (a), but not well for mobile camera (e). (c-g) shows example measurement frames.

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 [49]. Traditionally, video analysis has relied on humans, but recent advances in computer vision and deep learning have dramatically increased accuracy for automatic visual scene analysis, such as pedestrian detection [24], vehicle tracking [21], an facial recognition to locate people of interest [42, 52]. While some surveillance networks use dedicated links, an increasing number of surveillance systems, such as Dropcam [28] and Vigil [77], 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 [15]. This geo-distributed infrastructure continuously produces large volumes of data such as data access logs, server monitoring logs, and performance counters [8, 53, 72]. While most log analysis today runs in batch mode on a daily basis, there is a trend towards analyzing logs in real time for rapid optimization [54]. For example, CDNs can improve the overall efficiency by optimizing data placement if the access logs can be processed in real time.

2.2 Wide-area Bandwidth Characteristics

WAN bandwidth is insufficient and costly, as demonstrated by recent WAN-aware systems [32, 53, 71, 72]. 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 [32]. In terms of pricing, as of April 2017, the average WAN bandwidth cost is up to 38x of the cost of renting two machines [9].

In addition to the scarcity and cost, the large variability of WAN bandwidth also affects streaming workloads. We conducted a day-long measurement using iPerf [25] to measure the pair-wise bandwidth between four Amazon EC2 sites. The results show large variance in all pairs—Fig. 2 is one such pair. There are occasions when 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 [13], broadband access networks [30, 63] and cellular networks [47].

2.3 The Case for a System Approach

To address the bandwidth limits, existing solutions include manual policies and application-specific solutions. We discuss their drawbacks to make the case for a system approach.

Manual polices are sub-optimal. While existing systems such as JetStream [54] and DASH [61] allow adaptation, they require developers to write manual policies. We illustrate the problems with manual policies using an example [54]: *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.*

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 [57] is used, a reduction in frame rate increases the inter-frame difference, creating larger P-frames. Fig. 3e shows that by reducing the frame rate to one-third (from 30 FPS to 10 FPS), the bandwidth demand is still more than 50%.

Second, it is not scalable to specify rules in this way. When the policy involves multiple dimensions or developers desire a fine-grain control, the policy will end up with too many rules. Writing such rules 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 solutions don't generalize. Because each application has a different performance metric, relies on different features, and targets different data distributions, a fine-tuned policy for one application will often work poorly for others.

Analytical applications have their own goals, entailing different optimization metrics and different algorithms. In video analytics, for instance, object detection algorithms depend on the edge information [16, 41, 69] while object tracking [7] works best when the inter-frame difference is small. The former is sensitive to resolution changes while the latter frame rate changes.

Similar applications face different data distributions. Compare the stationary camera detecting pedestrians with the mobile camera recognizing objects (Fig. 3). In the former case, when we take the pedestrians' walking speed into consideration, a high frame rate is not necessary. But high-resolution images are crucial as surveillance cameras are far from the targets. In the latter case, mobile cameras move. Reducing the frame rate introduces significant errors.

3 AWStream Design

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

3.1 APIs for Structured Adaptation

The majority of stream processing applications today are constructed as a directed graph of operators [67, 75], 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.

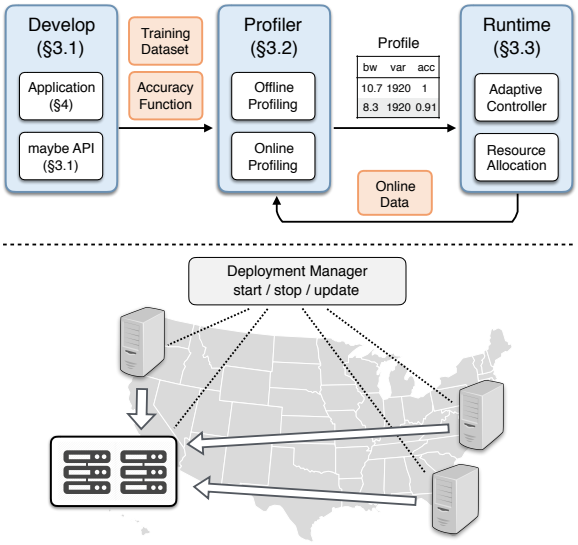


Figure 4. High-level overview of AWStream.

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 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 the Rust programming language:

```
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 subsequently encoded—for example, run-length encoding as in JPEG [73]—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 takes the top values of

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>
	timed_window (time: Duration, 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, Interger)>)	Stream<Image> ⇒ Stream<Image>
	...	...

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

a vector; and `maybe_downsample` can adjust the image resolution to a configured target. AWStream provides a number of such operations as a library for developers (Table 1).

With our APIs, 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 one 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 application 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.

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 all c such that there is no alternative configuration c' that requires less bandwidth while giving a higher accuracy. These configurations form the Pareto-optimal set $\mathbb{P}$, defined as follows:

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

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)
Q_E, Q_C	messages when `Queue` is empty or congested
$S_{\text{ProbeDone}}$	message when `Socket` finishes probing

Table 2. Notations used in this paper.

Because AWStream allows arbitrary functions as the degradation functions, it does not assume a closed-form relation 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 discuss examples of concrete knobs, configurations, accuracy functions when we present applications in §4.

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 evaluate all possible configurations. While all knobs form a combinatorial space, the offline profiling is only a one-time process. We exploits parallelism to reduce the profiling time. Without any *a priori* knowledge, all configurations are assigned randomly to available machines. Nevertheless, given certain workloads, we can leverage statistical methods to build accurate performance models while only exploring a small number of configurations [50, 68].

Online Profiling: AWStream runs an online profiling process continuously to refine the profile. The refinement handles *model drifts*, a problem when the learned profile fails to predict the performance accurately. There are two challenges with online profiling. (*i*) There is no ground-truth labels or reference data to compute accuracy. Because labelling data is prohibitively labor intensive and time consuming [59], AWStream currently uses raw data as the reference. If the application is run without any data degradation, the application data is used for online profiling. Otherwise, we allocate additional bandwidth to back-haul raw data, but only do so when it does not interfere the running application. (*ii*) Exhaustive profiling is expensive. If it takes too much time, the newly-learned profile may already be stale. AWStream uses a combination of parallelism and sampling techniques to speed up this process, as described below:

- Degradation-aware parallelization: Evaluating each configuration takes a different amount of time. Typically, an increase in the level of degradation leads to a decrease in computation, e.g. the smaller the FPS, the fewer images to process. Therefore, we collect processing times for each configuration from the offline process and use them for scheduling—for example longest-job first schedule (LFS) [36]—in parallelization.
- Sampling-based profiling: Online profiling can speed up if we only profile a subset of all data or a subset of all configurations. The former case reduces the amount of data to process, but at a cost of generating a less accurate profile. In the latter case, we can evaluate a subset of the configurations and compare their performances with the 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 performs application adaptation according to the learned profile. We choose to design our own runtime system and adaptation algorithm because prior solutions are not satisfactory: (*i*) network protocols adapt to available resources without application accuracy guarantee; (*ii*) JetStream [54] uses manual policies without the bandwidth demand of each rule, therefore it can only react slowly and adapt gradually, causing high latency; and (*iii*) video streaming adaptation, such as Huang et al. [33], relies on the playback buffer and incurs high latency.

Our runtime differs from prior systems in the ability to react with precision: it stays in a low-latency state and uses a Pareto-optimal configuration for high accuracy.

Fig. 5 shows our runtime system architecture. AWStream applications' source contains a `Maybe` module derived from all `maybe` operations. 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`. When the data generation rate exceeds `Socket`'s

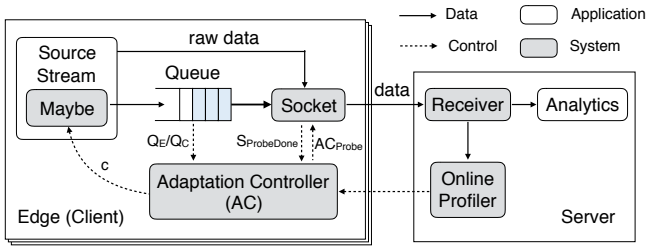
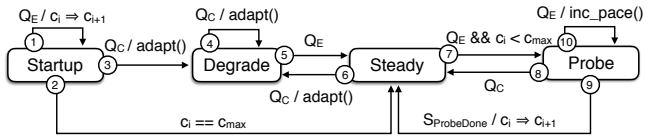


Figure 5. Runtime adaptation system architecture.

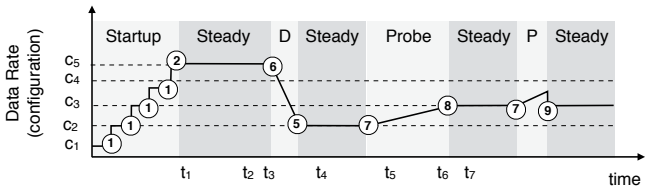
departure rate, the queue grows. In this case, the adaptation controller (AC) queries the estimated bandwidth from `Socket` and regulates the source stream by updating the configuration. After the data is sent through the network, the receiver extracts raw data to the online profiler and delivers data to the application analytics. Raw data is only transmitted when the queue is empty. When a new profile is learned, it is fed back to AC for subsequent adaptation.

We then describe the adaptation algorithm; Fig. 6 shows the state machine model and an example trace for illustration. 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 the `Queue`: message Q_E when the queue is empty and Q_C when queued items exceed a threshold. AC can query `Socket` for delivery rate R or request it to probe (AC_{Probe}) for a target bandwidth, often $B(c_{i+1})$. When $R > B(c_{i+1})$, `Socket` sends back $S_{ProbeDone}$. The algorithm follows a state machine as described below:

- **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 (to `Degrade`).
- **Degrade: reacting to congestion.** When `Queue` grows and exceeds a threshold, AC receives Q_C and runs the `adapt()` procedure. This involves two steps: (1) AC queries R from `Socket`; (2) AC updates `Maybe` with the maximum-allowed c that satisfies $B(c) < \alpha R, \alpha \in (0, 1)$. A smaller α allows a quicker draining of the queue. After the queue is drained, 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



(a) Rate adaptation as a state machine.



(b) An example illustrating the adaptation algorithm.

Figure 6. Runtime adaptation algorithm.

to BBR [18]). AWStream stops probing under two conditions: (1) upon $S_{ProbeDone}$, it advances c_i; (2) upon Q_C, it returns to `Steady`.

Resource Allocation and 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 pinpoint 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, useful to reduce WAN bandwidth usage and cost.

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

$$\begin{aligned} \underset{c^t}{\text{maximize}} \quad & \min(A^t(c^t)) \\ \text{subject to} \quad & \sum_t B^t(c^t) < R \end{aligned} \tag{2}$$

Solving this optimization is computationally hard. AWStream uses a heuristics approach. We start with c_1^t and improve the application t that has the worst accuracy. This process repeats until we have allocated all available bandwidth.

4 Implementation and Applications

While our proposed APIs are general and not language specific, we choose a safe language, Rust, to implement the core framework. AWStream is open-source on Github.[1] Applications built with AWStream run as a single process. The execution mode—profiling, runtime as client, or runtime as server—is configurable with command line arguments or environment variables. Our deployment manager is currently a shell script.

Application	Knobs	Accuracy	Dataset
Pedestrian Detection	resolution frame rate quantization	F1 score	MOT16 [44] training: MOT16-04 testing: MOT16-03
Augmented Reality	resolution frame rate quantization	F1 score	iPhone video clips training: office (24 s) testing: home (246 s)
Log Analysis (Top-K)	head (N) threshold (T)	Kendall's τ	SEC.gov access logs [48] training: 4 days testing: 16 days

Table 3. AWStream Applications

We've built three applications: pedestrian detection surveillance, an augmented reality, and a distributed log analysis to extract the Top-K most access files. Table 3 summarizes the application-specific part: knobs, accuracy function, and the dataset we used for training and testing.

Pedestrian Detection. This application analyzes streams of videos from installed CCTV cameras and detects pedestrians inside. We implement image-related operations with OpenCV 3.1 [14] and detect pedestrians using histogram of oriented gradients (HOG) [22] with the default linear SVM classifier. To ensure real-time processing of frames, we use the GPU-accelerated implementation. Video encoding employs H.264 because of its prevalence in existing systems. Our implementation uses GStreamer [64] with `x264enc` plugin. To integrate with AWStream, we first create a pipeline that exposes `appsrc` (to feed raw image data) and `appsink` (to get encoded bytes). The GStreamer main loop executes in a separate thread and AWStream communicates with it via Rust's channel. The `x264enc` uses the `zerolatency` preset and four threads. We use constant quality encoding and expose the quantization factor as another knob (in addition to image resolution and frame rate).

Pedestrian detection returns a list of bounding boxes, and each box is a rectangle with normalized coordinates on the image. We compare the detection against the reference result from raw data, and declare it success if the intersection over union (IOU) is greater than 50% [26]. We use F1 score (%) [58] as the accuracy function and MOT16 dataset [44] for training and testing.

Augmented Reality. We target mobile augmented reality applications that recognizes objects by offloading the heavy computation to resources elsewhere. We use a similar setup (OpenCV and GStreamer) as the pedestrian detection except for the analytical function. To recognize objects, we use YOLO [55, 56], a GPU-enabled pre-trained neural network for object detection. In this case, a successful detection requires

[1] Url elided for anonymity.

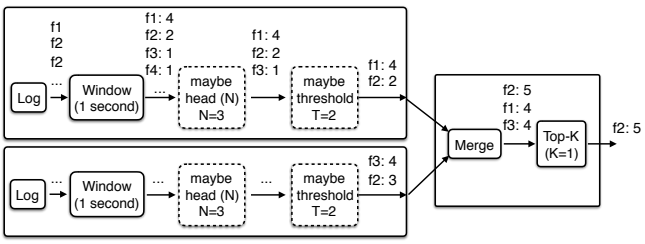


Figure 7. A distributed Top-K application with two degradation operations: `head` and `threshold`.

matching the object type in addition to IOU criteria. In terms of dataset, we've 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.

Distributed Top-K. Many monitoring applications need to answer the *Top-K* question [11], such as the Top-K most popular URLs, or the Top-K most access files. A distributed Top-K application aggregates information from geo-distributed servers to computer a final Top-K (Fig. 7).

Naively aggregating all the raw logs is not feasible because popular servers have millions of requests per second. Edge nodes can first perform a `Window` operation to generate data summary, such as key-value pairs of `<item, count>`. Even after this operation, however, the data size can still be too large because most real-world access patterns follow a long tail distribution. There is a large-but-irrelevant tail that contributes little to the final results.

The edge nodes perform two degradation operations: (1) a head (`N`) operation that only takes the top `N` entries; (2) a threshold `T` that filters small entries. These two operations are not orthogonal. Their impact on data size reduction and quality degradation depends on the data distribution. For the accuracy function, we use Kendall's τ [4], 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 the range of $[0, 1]$ with a linear transformation.

Our Top-K application aims to find the fifty most accessed files from web server logs. We use Apache log files that record and store user access statistics for the SEC.gov website. We split these logs 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 (Fig. 8).
- §5.2 Our parallel and sampling profiling speeds up offline and online profiling (Fig. 9, Fig. 10).
- §5.3 At runtime, AWStream applications achieve sub-second latency and nominal accuracy drop (Fig. 11).
- §5.4 AWStream profiles allow different resource allocations, such as utility fairness (Fig. 12).

5.1 Application Profiles

We run our offline profiling using the training dataset described in Table 3. Fig. 8 shows the learned profiles. In each figure, the cross dots represent the bandwidth demand and application accuracy of 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, their bandwidth requirements have two to three orders of magnitude of difference (note the x-axis is in log scale). For PD and AR, the most expensive configuration transmits videos at 1920x1080, 30 FPS and 0 quantization; it consumes 230 mbps. In constrast 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%. This allows AWStream applications to operate at a high accuracy configuration even under severe network deterioration.

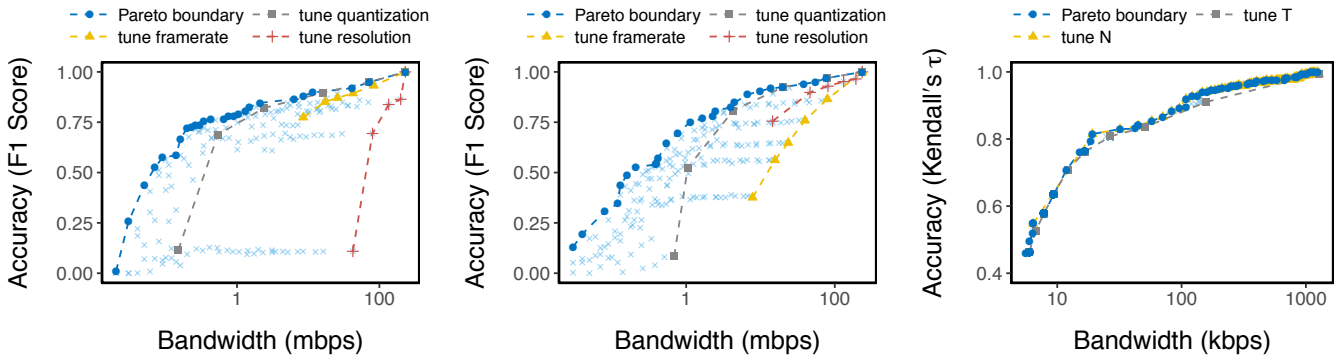
Multiple dimensions achieve the optimal. Comparing dashed lines in each profile, we see that the Pareto-optimal configurations are only achievable when multiple knobs are in effect. Tuning only one dimension often leads to sub-optimal performance. Although Top-K's Pareto-optimal boundary aligns with the performance of tuning `N`, i.e. tuning `T` doesn't contribute much for this dataset, for another dataset with a different distribution, the behavior will probably change.

Each dimension affects differently. 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 in comparison to tuning the frame rate (the yellow line). Comparing PD and AR, 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- and context-specific optimizations don't generalize.

Quantification with precision. The generated profiles are actionable configurations that control the knobs with precision. For example, if PD transmits video at 1920x1080 resolution, 10 FPS and a quantization of 20, it will consume 11.7 mbps of bandwidth, achieving roughly 90% accuracy. This saves developers from laboriously analyzing their application to compute manual policies.

5.2 Profiling Efficiency and Online Profiling

This section focuses on the AR application as a case study; our profiling techniques—parallelism and sampling—do not



(a) Pedestrian Detection (PD) **(b)** Augmented Reality (AR) **(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.

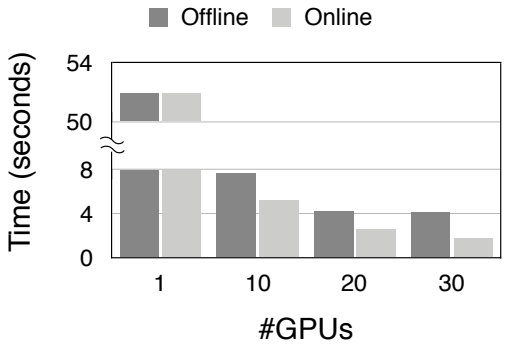


Figure 9. Parallelism speeds up both offline and online profiling. The y-axis shows the profiling time for 1-second video.

make assumptions about the application; therefore, the evaluation results can be generalized to other applications.

AR allows 216 different configurations: 6 resolutions, 6 frame rates and 6 quantization levels. Processing a frame with YOLO [56] on GeForce® GTX 970 takes roughly 30 ms.[2] But different configurations require different amounts of 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. §3.2 discusses techniques to improve the profiling efficiency; we present their evaluations as follows.

Parallelism reduces the profiling time (Fig. 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 currently uses LPT assignment and can reduce the overhead to 1.75X with 30 GPUs.

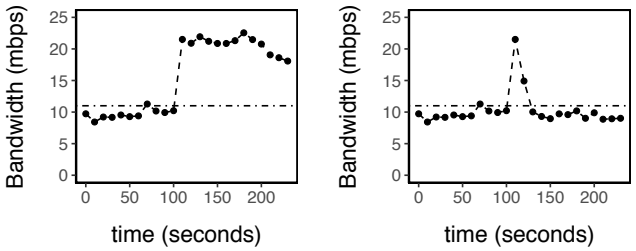
Sampling techniques speed up online profiling (Fig. 10). Before we evaluate the speed up, we validate *model drifts* with real-world data. We use the profile trained in an office environment. According to the profile, the application should operate at a configuration of 1280x720 resolution, 30 FPS and a quantization of 20 to meet 11 mbps available bandwidth. 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. 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. Fig. 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 drifts. The downside of continuous profiling, as discussed earlier, is the cost: 52X overhead with 1 GPU. In addition to parallelism, AWStream uses two other techniques for online profiling (improvements in Table 4):

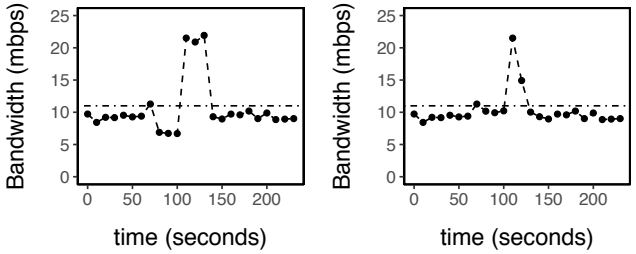
(i) Partial data: Instead of using all the past data, we run profiling with only a fraction of the raw data. Fig. 10c shows the bandwidth prediction if we only use 10 seconds of data out of the past 30 seconds. In this way, although the profile may be less accurate (the mis-prediction at t=80–100s), 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. 10d shows the bandwidth prediction if we evaluate 5 configurations continuously and trigger a full profiling when

[2] YOLO resizes an input image to a fixed neural network size. Evaluating all images (with different resolutions) takes a similar amount of 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). We omit accuracy predictions since in all schemes they are similarly high (~90%). (1) Online profiling is necessary to handle model drifts ((a) vs. (b-d)). (2) Sampling techniques—partial data (c) and partial configurations (d)—can correct model drifts with less profiling overhead (see Table 4), compared to continuous (b).

online scheme	overhead	improvements
continuous	52X	baseline
partial data	17X	3×
partial configurations	6X	8.7×

Table 4. Sampling techniques speed up the profiling. We use X to denote the overhead of processing one-second worth of data; and × for the improvements.

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. 10b and Fig. 10d). The overhead reduces to 6X because we run full profiling less often (only two times in this experiment).

When we combine the parallelization and sampling-based profiling, we can further reduce the profiling overhead. For example, scheduling five GPUs running 5 configurations continuously to check for model drifts will reduce the overhead to 1X. In practice, the amount of resources to use will depend on the budget and the importance of the job.

5.3 Runtime Adaptation

In this section, we evaluate the runtime performance for three applications. We first show that AWStream achieves low latency and high accuracy across all three applications (Fig. 11). We then focus on AR by evaluating how manual policies and application-specific solutions work poorly, i.e., describe the experiments for Fig. 1.

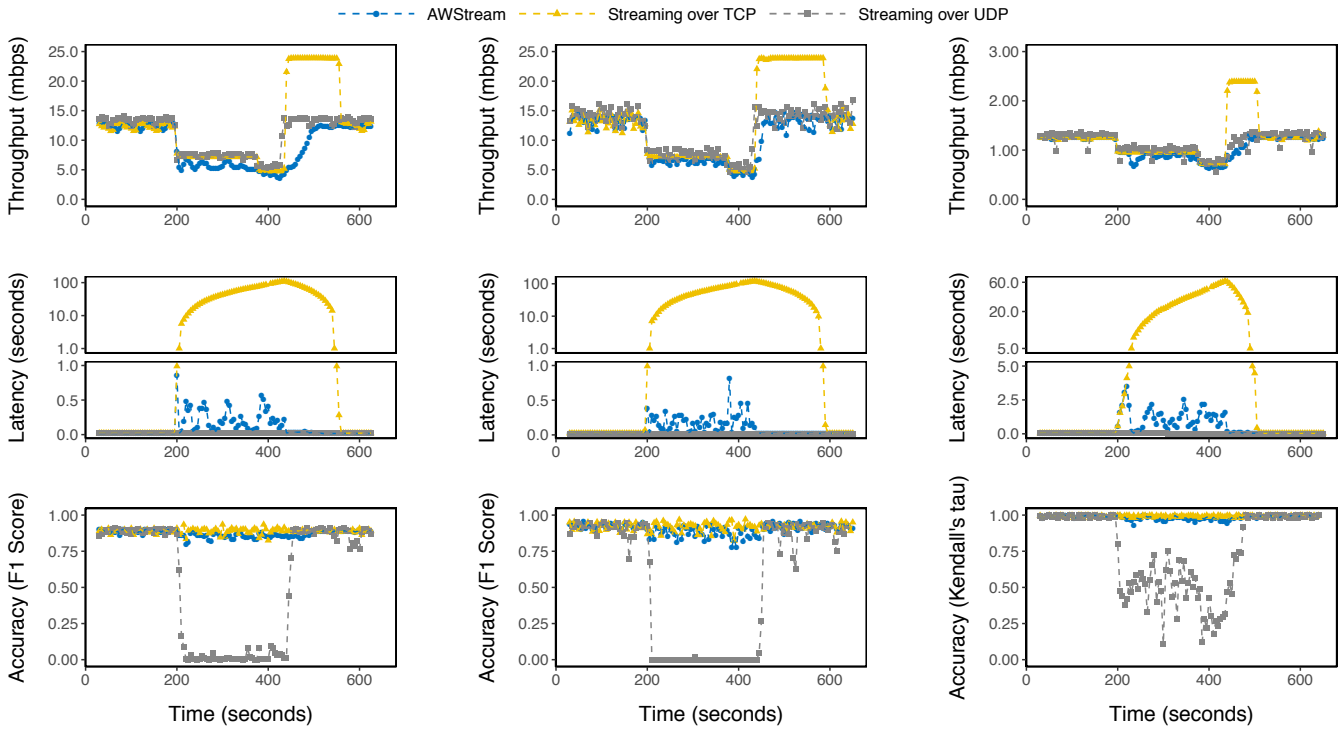
We conduct our experiments on five geo-distributed machines from Amazon EC2, spanning different AWS regions. Four act as worker nodes (*t2.micro* instances) and one acts as the aggregation server (*m4.xlarge* instance to guarantee enough bandwidth). During each experiment, the worker node transmits test data (Table 3) for about 10 mins. If the duration of the test data is less than 10 mins, it loops.

We compare AWStream applications with baseline applications built with vanilla TCP and UDP because they are universal for all applications. For streaming over TCP, we use AWStream but disable the adaptation. For streaming over UDP, our experiment setup is different: for video streaming (PD and AR), we use RTSP over UDP with ffmpeg [12]; for TK, we implemented a simple application protocol—packetization and a custom header—such that the receiver can still aggregate data in the presence of UDP packet loss.

Because the baselines don't adapt the configuration and $B(c_{\max})$ is prohibitively large—for example, raw videos consumes 230 mbps—we use a reasonable configuration to limit the maximum rate. For PD, $c_{\max}$ is 1920x1080 resolution, 10 FPS and a quantization of 20; it consumes about 12 mbps. For AR, $c_{\max}$ is 1600x900 resolution, 30 FPS and a quantization of 20; it consumes about 14 mbps. For TK, $c_{\max}$ is $N = 9750$ for `head` and $T = 0$ for `threshold`.

Our controlled-experiment design follows JetStream [54]. During the experiment, we use the Linux `tc` utility with HTB [23, 34] to control the clients' outgoing bandwidth. Our experiments involve four phases: (i) before t=200s, there is no shaping; (ii) at t=200s, we start traffic shaping for 3 minutes (7.5 mbps for video streaming and 1 mbps for TK); (iii) at t=380s, we further decrease the available bandwidth (5 mbps for video streaming and 0.75 mbps) for TK); (iv) at t=440s, we remove all traffic shaping. For UDP, HTB doesn't emulate the packet loss or out-of-order delivery; we use `netem` and configure the loss probability according to the delivery rate. Because different pair-wise connections have different capacity, we impose a *background* bandwidth limit—25 mbps for PD and AR, 2.5 mbps for TK—to simplify comparisons.

The throughput figures of Fig. 11 show the effect of our traffic shaping. During the shaping, TCP and UDP make full use of the available bandwidth; in comparison, AWStream is conservative. When we stop the shaping, TCP has a "catch-up" phase where it's sending all the queued items as fast as possible. AWStream increases the throughput gradually instead of drastically as TCP/UDP does. This is when AWStream is slowly probing for more bandwidth.



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

Figure 11. Runtime behavior of AWStream applications in comparison with streaming over TCP/UDP. AWStream simultaneously provides sub-second latency (similar to UDP) and high accuracy (similar to TCP).

The latency figures show that AWStream maintains a bounded latency. Note the figures split in the middle because of the large variation in the measured latency. Specifically, during the traffic shaping, TCP queues items at the sender side for up to tens or hundreds of seconds. We plot such high latency measurements in log scale. UDP always transmits as fast as possible, leading to a consistent low latency. AWStream's latency is slightly higher than UDP but considerable small in comparison to TCP. During the traffic shaping, AWStream controls the application's degradation to avoid data queueing up. For most of the time, we've maintained less than 500 ms latency for PD/AR, and 2.5 seconds latency for TK. Because TK's source generates data every second after the `window` operation, two objects in the queue would lead to two seconds latency. The worst case happens at the start of bandwidth drop: AWStream achieves sub-second latency for PD/AR and 5 seconds latency for TK at the worst case.

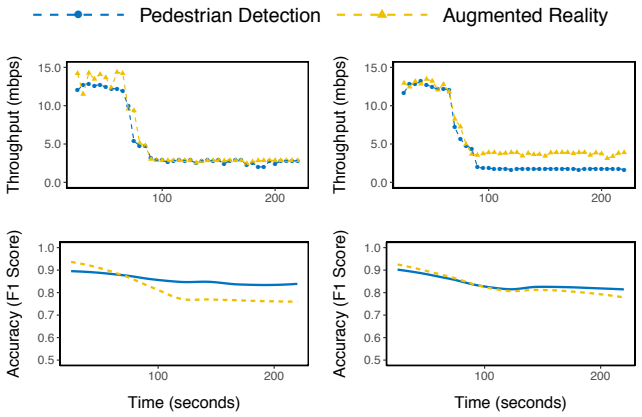
The accuracy figures show that AWStream maintains a high accuracy throughout the experiments. TCP without adaptation always sends data at high fidelity (~90% for PD/AR and ~100% for TK). It achieves the high fidelity at the expense of increase latency. UDP without adaptation suffers from packet loss, leading to a severe performance deterioration. In both PD and AR, the accuracy are ~0% as the recovered images contain large blocks of pixels without content details. For TK, the accuracy has dropped to ~50% during modest traffic shaping and dropped further to ~25% during the severe shaping. While AWStream also sees an accuracy drop during the traffic shaping, the amount is significant smaller: 2% drop on average in comparison to streaming over TCP. We attribute this to the knowledge of the profile and the use of Pareto-optimal configurations during traffic shaping.

In summary, Fig. 11 shows that AWStream's adaptation achieves low latency and high accuracy simultaneously.

Fidelity and freshness. Here we explain our experiments for Fig. 1. For streaming over TCP/UDP and AWStream, we use the latency and accuracy measurements for AR. For manual policies, we implement the policy from the example in JetStream [54]. For application-specific solutions, we apply PD's adaptation profile to control AR's behavior.

For comparison, we use data collected during traffic shaping (between t=200s and t=440s). When facing insufficient bandwidth, their behaviors reflect the design choices within the design space of data fidelity and freshness:

- Streaming over TCP favors fidelity over freshness.
- Streaming over UDP favors freshness over fidelity.
- Manual policies have limited rules with coarse-granularity. A sudden change from one rule to another incurs large latency (~5 seconds, similar to JetStream's result [54]).



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

Figure 12. AWStream supports a variety of resource allocation schemes (e.g., resource fairness and utility fairness).

- Application-specific solutions suffer from accuracy drop. PD's profile has configurations that reduce frame rate aggressively; these configurations work poorly for AR.
- AWStream achieves both data fidelity and freshness.

5.4 Resource Allocation and Fairness

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

We choose PD and AR as the example applications. We colocate the clients and servers of both applications 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 at 6 mbps. When we allocate resource equally between two applications (Fig. 12a), each application gets 3 mbps. Under this condition, PD runs with a higher accuracy (~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. 12b).

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 some limitations and our future work.

Fault-tolerance and failure recovery. AWStream tolerates bandwidth variation but not network partition or host failure. While the servers within the DCs can handle faults as existing systems—such as Spark Streaming [75]—do, it is difficult to make edge clients failure-oblivious. We leave failure detection and recovery of clients as a future work.

Profile modelling. AWStream currently do not attempt to model $B(c)$ and $A(c)$. Instead it performs an exhaustive search during the profiling. While parallelism and sampling techniques offer speed up, AWStream can benefit from using statistical techniques during the profiling. For example, Bayesian Optimization, as demonstrated by CherryPick [50], models black-box functions and reduces the search time. We plan to explore this direction to improve our profiling.

Context detection. AWStream is currently limited to one profile: the offline profiling uses one representative training dataset and the online profiling updates the single profile continuously. Real-world applications may produce data with a multi-modal distribution, where the mode changes upon context changes, such as indoor videos to outdoor videos. Therefore, one possible optimization to AWStream would be to allow multiple profiles for one application, detect context changes at runtime, and use the profile that best matches the current data distribution. Switching contexts could reduce, or even eliminate, the overhead during online 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 future bandwidth, we expect further improvements such as no latency spikes or a simplified probing state. Techniques such as model predictive control (MPC) have improved QoE in video streaming [74] with throughput prediction; we plan to investigate using MPC in AWStream as a future work.

7 Related Work

The work most closely related to AWStream is JetStream [54]. Both systems target at wide-area streaming analytics. JetStream proposes to use aggregation and *manual* degradation policies to achieve responsiveness in the presence of bandwidth fluctuation. We've discussed its limitations in §2 and made comparisons throughout the paper.

We divide other related works into stream processing systems, approximate analytics, WAN-aware systems and adaptive video streaming.

Stream processing systems. Streaming databases, such as Borealis [1], TelegraphCQ [19], 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 [67], Heron [39], Spark Streaming [75], Apache Flink [17], 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 the wide area and explicitly explores the trade-off between data fidelity and data 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 has also been explored in other contexts. For SQL queries, online aggregation [31],

BlinkDB [5] and GRASS [10] speed up queries with partial data based on a statistical model of SQL operators. For real-time processing, MEANTIME [27] uses approximation to guarantee satisfying real-time deadlines. For video processing, VideoStorm [76] optimizes video stream *processing* within lager clusters by exploiting approximation and delay-tolerance for resource management and allocation. The tradeoff between available resource and application accuracy is a common theme among all this work. 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 [20], 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 [46] 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 [71], Geode [72], Iridium [53], Pixida [37], Clarinet [70], etc. They minimize data movement by incorporating heterogeneous wide-area bandwidth information into query optimization and data/task placement. While effective in improving analytics efficiency, 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 focuses on streaming analytics that process streams of data continuously in real time.

Adaptive video streaming. Designing a good adaptation strategy to improve QoE for video streaming has been a hot topic, including research projects [62, 74] and industrial efforts (HLS [51], DASH [43, 61]). Because they optimize QoE for humans, their results are not directly transferable to wide-area streaming analytics. In comparison, AWStream generalizes adaptation by providing `maybe` APIs to allows more custom control over what parameters to tune. In this way, AWStream can effectively integrate existing and future techniques in video streaming, such as video encoding (H.264 [57] or VP9 [29]). The main contribution of AWStream here is to provide a system for a wide range of streaming analytics to benefit from adaptation.

8 Conclusion

To support the emerging class of wide-area streaming analytics in the face of scarce and variable bandwidth, we propose AWStream, a stream processing system specialized for wide area networks. AWStream supports a wide variety of applications by enabling developers to customize degradation operations with `maybe` APIs. Our automatic profiling tool generates an accurate profile that characterizes the tradeoff between bandwidth consumption and application accuracy. The profile allows the runtime to react with precision. Our evaluation with three applications shows that AWStream achieves low latency (sub-second) with nominal accuracy drop. In summary, AWStream allows resilient execution of wide-area streaming analytics continuously over time with minimal developer efforts.

Acknowledgments

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

The authors would like to thank Kaifei Chen and Pan Hu for providing feedback to an early version of this manuscript.

References

[1] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, and others. 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, and others. 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. DOI: http://dx.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] John G. Allen, Richard Y. D. Xu, and Jesse S. Jin. 2004. Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces. In *Proceedings of the Pan-Sydney Area Workshop on Visual Information Processing (VIP'05)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 3–7. http://dl.acm.org/citation.cfm?id=1082121.1082122

[8] 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

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

[10] 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

[11] 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. DOI: http://dx.doi.org/10.1145/872757.872764

[12] Fabrice Bellard, M Niedermayer, and others. 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. DOI: http://dx.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. DOI: http://dx.doi.org/10.1145/2504730.2504754

[16] John Canny. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), 679–698. DOI: http://dx.doi.org/10.1109/TPAMI.1986.4767851

[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, and others. 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. DOI: http://dx.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] 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.

[22] 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. DOI: http://dx.doi.org/10.1109/CVPR.2005.177

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

[24] 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.

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

[26] 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. DOI: http://dx.doi.org/10.1007/s11263-009-0275-4

[27] 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

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

[29] Adrian Grange, Peter de Rivaz, and Jonathan Hunt. 2016. VP9 Bitstream & Decoding Process Specification. *Google, March* (2016).

[30] 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. DOI: http://dx.doi.org/10.1145/2504730.2504736

[31] 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. DOI: http://dx.doi.org/10.1145/253260.253291

[32] 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

[33] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM'14)*. ACM, New York, NY, USA, 187–198. DOI: http://dx.doi.org/10.1145/2619239.2626296

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

[35] Cisco Visual Networking Index. 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).

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

[37] 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

[38] 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. DOI: http://dx.doi.org/10.1145/2422531.2422546

[39] 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. DOI: http://dx.doi.org/10.1145/2723372.2742788

[40] 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.

[41] David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110. DOI: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[42] 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

[43] 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.

[44] 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/

[45] 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

[46] 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. DOI: http://dx.doi.org/10.1145/502034.502052

[47] 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. DOI: http://dx.doi.org/10.1007/978-3-319-04918-2_2

[48] 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.

[49] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, and others. 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. DOI: http://dx.doi.org/10.1109/CVPR.2011.5995586

[50] 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

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

[52] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep Face Recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*, Mark W. Jones Xianghua Xie and Gary K. L. Tam (Eds.). BMVA Press, Article 41, 12 pages. DOI: http://dx.doi.org/10.5244/C.29.41

[53] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-Distributed Data Analytics. (2015), 421–434. DOI: http://dx.doi.org/10.1145/2785956.2787505

[54] 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

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

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

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

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

[59] 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. DOI: http://dx.doi.org/10.1007/s11263-007-0090-8

[60] 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. DOI: http://dx.doi.org/10.1109/MPRV.2009.82

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

[62] 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. DOI: http://dx.doi.org/10.1145/2934872.2934898

[63] 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

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

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

[66] 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.

[67] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, and others. 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

[68] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 363–378. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman

[69] 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. DOI: http://dx.doi.org/10.1109/CVPR.2001.990517

[70] 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

[71] 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. DOI: http://dx.doi.org/10.1145/2723372.2735365

[72] 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

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

[74] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. (2015), 325–338. DOI: http://dx.doi.org/10.1145/2785956.2787486

[75] 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. DOI: http://dx.doi.org/10.1145/2517349.2522737

[76] 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

[77] 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