ePulsar: Control Plane for Publish-Subscribe Systems on Geo-Distributed Edge Infrastructure

Harshit Gupta
harshitg@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Tyler C. Landle
tlandle3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Umakishore Ramachandran
rama@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

Emerging applications such as autonomous drones and massively multiplayer gaming require real-time communication between multiple geo-distributed participating entities. A publish-subscribe system deployed on a geo-distributed edge infrastructure would provide a scalable messaging middleware for such applications. However state-of-the-art publish-subscribe systems like Apache Pulsar and Kafka perform inefficiently in a geo-distributed deployment due to heterogeneous client-broker latencies and constant client mobility. We present a novel *control-plane architecture* for geo-distributed publish-subscribe systems that is capable of adaptive topic partitioning to enable low-latency messaging for such applications. We leverage a peer-to-peer network coordinate protocol for scalable estimation of network latencies between publish-subscribe brokers and clients. Client-broker latency and workload metrics are continuously collected from brokers and used to detect latency violations or workload imbalance, which triggers reassignment of topics. We develop **ePulsar**, which incorporates the control-plane architecture ideas into the popular Apache Pulsar publish-subscribe system, retaining Pulsar's data-plane APIs. We evaluate the efficacy and overheads of the proposed control plane using workload scenarios representative of typical edge-centric applications on an emulated geo-distributed infrastructure.

SEC '21, Dec 14–17, 2021, San Jose, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

CCS Concepts

• **Computer systems organization** → **n-tier architectures**; **n-tier architectures**; • **Networks** → **Cloud computing**.

Keywords

edge computing, publish-subscribe systems, network coordinates

ACM Reference Format:

Harshit Gupta, Tyler C. Landle, and Umakishore Ramachandran. 2021. **ePulsar**: Control Plane for Publish-Subscribe Systems on Geo-Distributed Edge Infrastructure. In *SEC '21: ACM/IEEE Symposium on Edge Computing, Dec 14–17, 2021, San Jose, CA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

Applications such as drone control and Massively Multiplayer Online Games (MMOG) need to support large amounts of clients, retaining high throughput and low latency communication. The synchronization decoupling provided by publish-subscribe systems [14, 38] makes them an ideal messaging middleware for supporting these applications. Typically, pub-sub systems consist of broker *middleware nodes* that are responsible for message exchange within the system. Popular pub-sub systems such as Apache Kafka and Pulsar are commonly used for supporting low latency and high throughput messaging for datacenter applications. Pub-sub systems have been shown to be suitable for sharing game state updates in MMOGs [6], swarm synchronization for autonomous robots (drones) [4], and data distribution for large-scale stream processing [16]. However, contemporary applications such as MMOGs, large scale IoT, and Unmanned Aerial Vehicle (UAV) coordination pose latency constraints that make cloud-based publish-subscribe system deployments unsuitable due to the high WAN latency between clients and middleware nodes. Given the proximal nature of edge resources, they can be utilized for hosting

pub-sub middleware close to clients and thereby provide low end-to-end pub-sub latency.

However, adapting state-of-the-art cloud-based pub-sub systems like Kafka and Pulsar to a geo-distributed edge infrastructure poses peculiar challenges. Such systems typically are topic-based and they partition topics among brokers by computing consistent-hash of topic name. Consistent hashing ensures even distribution of load among brokers - which is key to manageable end-to-end latency in datacenters where the network topology is more or less homogeneous. However, in edge infrastructure, the physical location of a client has a significant impact on client-broker network latency, and latency-agnostic consistent hashing does not account for network proximity. In addition, client mobility requires constant adaptation of topic partitioning to continuously provide low end-to-end latency. The network infrastructure itself could experience changes (e.g., increased latency between servers) which might affect end-to-end latency. Finally, due to the capacity constraints and limited statistical multiplexing at edge sites, load-aware topic partitioning is important to avoid workload hotspots and minimize end-to-end latency [22].

To address the above challenges, we design a novel *edge-centric control plane architecture* for pub-sub systems. The elements of this architecture include the following features, which are the primary contributions of this paper: (1) A network-proximity monitoring technique that leverages network coordinates so that workload distribution can be latency-sensitive. The technique uses a decentralized network coordinates protocol [9, 23] to scalably obtain pairwise network latency estimates between system entities. (2) A latency and workload-aware adaptive topic partitioning policy that keeps the end-to-end latency under the threshold set by the application.

To put the above contributions into practice, we extend Apache Pulsar – a popular cloud-based pub-sub system – to build **ePulsar**, which offers the same functionality as Pulsar alongside agile and adaptive topic partitioning to provide low end-to-end latency. We evaluate **ePulsar** against realistic workload scenarios on an emulated geo-distributed infrastructure to show the performance improvements over off-the-shelf deployment of a cloud-based Pulsar. The comprehensive evaluation study forms the third contribution of this paper. While the control plane ideas have been incorporated into Pulsar as a proof of concept, the ideas are general and can be applied to any edge-centric pub-sub system.

The rest of the paper is structured as follows. We set the context for this work in Section 2 and enumerate the control-plane requirements of an edge-centric pub-sub system in Section 2.4. The architecture of **ePulsar** is presented in Section 3, followed by implementation details in Section 4. Section 5 presents the results of microbenchmarking **ePulsar**; and end-to-end evaluations of two exemplar applications with an emulated infrastructure. We conclude with a discussion of future work in Section 6.

2 Context

We first describe the characteristics of the geo-distributed edge infrastructure which would host both the edge-centric pub-sub system and client applications (Section 2.1). We then illustrate examples of applications that will benefit from an edge-centric pub-sub system (Section 2.2). The state-of-the art in pub-sub systems and their limitations in supporting such geo-distributed applications are discussed next (Section 2.3). The section concludes with the control-plane requirements for an edge-friendly pub-sub system (Section 2.4).

2.1 Assumptions about Edge Infrastructure

We consider an edge computing infrastructure that consists of multiple geo-distributed sites in each metropolitan area. These sites comprise multiple server racks and could be owned by service providers (such as AT&T and Comcast [28]), or by emerging edge solution providers (such as EdgeMicro [26] and VaporIO [21]). Since these sites are located just a few hops away from end-devices, they can be accessed with low network latency. Such sites in different cities along with cloud datacenters form a computational continuum to serve latency-critical applications to a geo-distributed set of clients.

2.2 Exemplar Applications

We describe two applications that would benefit from a geo-distributed pub-sub system.

2.2.1 UAV Swarm Coordination

Operating swarms of unmanned aerial vehicles (UAVs) have applications in smart cities, surveillance, etc. A common way to operate UAV swarms is to assign one of the UAVs as the *swarm leader* and the others as *followers* [34]. The leader receives commands from a control station and translates them into actionable tasks, which are then conveyed to the followers to guide their motion. In addition, the swarm leader also receives updates of relevant objects (e.g., obstacles) from the followers and instructs the swarm members to update their trajectory. Such communication has previously been modeled using a pub-sub abstraction [4]. Ensuring low-latency communication between leaders and followers, and between leader and ground control is essential for the efficient operation of the swarm.

Swarm mobility leads to changes in the routing path of packets through cellular networks, causing increased communication delay between the drones and the pub-sub brokers [18]. Therefore, a latency-sensitive pub-sub system is necessary to ensure proper swarm control.

2.2.2 Massively Multiplayer Online Games (MMOG)

Cloud-based control of MMOG reduces users' Quality of Experience (QoE) due to the high network latency between the clients and the game server [8]. It has been shown [30] that users would experience a much better QoE (25% lower state update latency) in MMOG if cooperating game servers are hosted on edge infrastructure. Geographically distributed end-users (*avatars*) connect to their geographically nearby game servers. The game servers use a peer-to-peer architecture [3] with dedicated pairwise connections amongst them to maintain the game state information, and communicate game state updates to the avatars in their respective *Area-of-Interest (AoI)*.

An edge-centric pub-sub system would offer a scalable alternative to maintaining the game state among the server instances, and the subscription of avatars to each server instance. As gameplay progresses, commensurate with their mobility pattern, avatars would unsubscribe and re-subscribe to server instances based on AoI.

2.3 State-Of-The-Art Pub-Sub Systems

Apache Kafka and Apache Pulsar are popular cloud-based pub-sub systems. Their simple and easy-to-use semantics coupled with scalable performance for data communication (high message throughput and low latency) make them attractive platforms for structuring cloud-based applications. However, the use cases identified earlier pose unique challenges due to the fact that the communicating entities are mobile and geo-distributed. Thus a cloud-based pub-sub system would not cater to their need for end-to-end low latency guarantees between the communicating entities. While the data plane of Apache Kafka and Apache Pulsar offer very good performance, their control plane decisions (e.g., for broker and bookie placement) assume all the communicating entities reside in the cloud with uniform communication latencies among the nodes. With mobility of clients in the aforementioned applications, it is imperative that the control plane decisions take into account network proximity of edge nodes to clients to ensure meeting end-to-end latency constraints. Further, continuous monitoring of violations of latency constraints is necessary to support such applications.

There has been prior work in building edge-centric pub-sub systems, which include EMMA [31], FogMQ [1], and MutiPub [17]. However, these systems do not meet the data communication and/or the scalability needs of the aforementioned applications. EMMA does not handle message reliability guarantees or at-least-once/exactly-once semantics that are typically offered by commercial pub-sub systems. FogMQ relies on creating a clone in the proximity of each device to handle communication on behalf of that device. With a large number of participating clients, this design decision will be a huge resource burden on the already scarce edge resources making the system non-scalable. MultiPub aims to provide latency guarantees for multi-region pub-sub systems by relying on having detailed information of inter-region latencies, as well as the network latency between every client-broker pair. Although this might be tractable for deployments with a handful of cloud regions, the much denser distribution of edge sites makes monitoring and maintaining such fine-grained latency information infeasible.

From an analysis of the state-of-the-art, we conclude that cloud-based pub-sub systems owing to their maturity have the best data plane and scalability attributes. However, a careful rethink of the control plane for pub-sub systems is needed to make such pub-sub systems operate adequately for supporting novel geo-distributed edge applications like the ones discussed in Section 2.2.

2.4 Control-Plane Requirements

We summarize the control-plane requirements for an edge-friendly pub-sub system:

- It should facilitate latency-aware topic partitioning for meeting application-specified end-to-end pub-sub latency thresholds.
- It should offer scalable inter-site latency estimation to support a large community of communicating entities.
- It should provide agile reconfiguration of topic partitioning to minimize violation of end-to-end application-specified latency thresholds.

The goal of this paper is to architect an agile control-plane for an edge-friendly pub-sub system which should meet the requirements mentioned above. To show the efficacy of this control plane, we implement these ideas in Apache Pulsar. However, the ideas are general and can be incorporated into any edge-centric pub-sub system.

3 ePulsar Architecture

Fig. 1 shows the main architectural components of **ePulsar** and how it extends Pulsar's control-plane to achieve the requirements posed by geo-distribution. Given that we use Apache Pulsar to evaluate the efficacy of our contributions, we begin with a description of Pulsar's control-plane design. In the subsequent subsections, we describe how the architectural elements of **ePulsar** are integrated into Pulsar.

3.1 Control-plane Design of Apache Pulsar

Pulsar supports two types of topics - *persistent* and *non-persistent*. Messages on persistent topics are logged on durable storage of *bookie* nodes (instances of Apache BookKeeper) for reliability [2], while non-persistent topics are not. Pulsar groups topics into *bundles* – the unit of monitoring and

topic partitioning. This design choice was made to amortize the amount of metadata needed to be tracked by the system. Topics are assigned to bundles by performing consistent hashing on the topic name. Each bundle is assigned to a unique broker through topic partitioning. Pulsar uses a ZooKeeper [20] instance to store cluster configuration data such as active brokers, bookies, bundles, and broker to bundle mapping. The ZooKeeper instance also serves as a repository for monitoring data which is periodically updated by each broker. Monitoring data comprises per-bundle traffic load and each broker's resource usage. The Load Manager module of Pulsar processes the monitoring data to determine candidate brokers for hosting a bundle, as well as to check if the current topic partitioning needs to be updated. Such a reconfiguration is needed when a broker is overloaded, in which case some bundles are migrated to another broker. Bundle migration in Pulsar is carried out by first having the current broker relinquish ownership of them followed by those bundles being *lazily* re-assigned to a less loaded broker determined by the same topic partitioning policy.

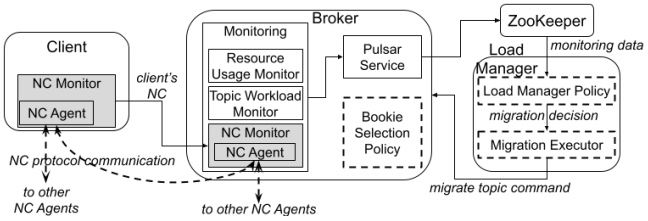


Figure 1: Architecture of **ePulsar**. The shaded components are the unique enhancements in **ePulsar**. The entities with dashed outlines represent baseline Pulsar's components that have been replaced with edge-centric implementations.

3.2 Per-Topic Load and Latency Monitoring

One fundamental design departure from Pulsar is not bundling topics into bundles, so that we can monitor and handle broker-assignment of each topic independently. This design choice is reflected in the monitoring module at each broker. As in Pulsar, each broker hosts modules for monitoring resource usage (CPU, memory, and network bandwidth) at the broker as well as per-topic traffic characteristics (including message rate and number of clients). Additionally, we also monitor the latency attributes of the broker and clients. These monitored metrics are periodically reported to the ZooKeeper instance. We now discuss in more detail the latency monitoring module.

Decentralized Network Coordinate protocol. Scalable and accurate measurement of network latencies between clients and brokers is essential for the selection of a suitable broker and ensuring low pub-sub latency. Network coordinate (NC) systems are distributed protocols to scalably determine the network proximity between a pair of nodes in a distributed system without performing direct measurements [12]. Such systems embed nodes in a geometric space such that the network latency between any two nodes can be estimated by calculating the Euclidean distance between their positions (coordinates) in this space. Doing so avoids the network overhead of pair-wise direct measurements. We employ a popular decentralized network coordinate protocol, Vivaldi [9] with some enhancements proposed by Ledlie, et al. [23] and Lee, et al. [24]. Prior art has shown that NC protocols provide efficient, accurate, and stable latency estimates in the wild [23].

Notation	Definition
t	topic
$P(t)$	producers of topic t
$C(t)$	consumers of topic t
$L_{th}(t)$	end-to-end latency constraint for topic t
$P[t]$	broker hosting topic t
$NC(i)$	network coordinate of entity i
$\overline{NC}(I)$	centroid network coordinate of entities in I
$d(nc_1, nc_2)$	distance between network coordinates
$W(t)$	Deviation of client NC from centroids of topic t
$E(t)$	Worst-case end-to-end latency for topic t

Table 1: Notations used.

Each broker's Latency Monitoring module contains an agent of the NC protocol, which interacts with its peer agents in other brokers. NC agents are also run on client nodes and they too form a part of the *peer-to-peer (P2P)* network of agents along with the brokers. Through periodic communication with a finite set of peers, each agent converges on a stable network coordinate. Each client periodically queries the coordinate of its NC agent and reports it to the broker that currently hosts its topic. Each broker periodically reports the NC of its agent, along with the coordinates reported by clients of its hosted topics to the ZooKeeper instance. This combination of NCs for broker and clients is used to compute the end-to-end pub-sub latency for a given topic.

Aggregation of Per-topic Latency Data. We reduce the amount of per-topic monitoring data sent to ZooKeeper by aggregating the network coordinates of multiple clients. For each topic t that is hosted on broker b, we report the following data items through monitoring. Table 1 provides a summary of notations used.

- **Producer and Consumer Centroid.** The producer and consumer centroids provide an approximate location of the network *location* of a topic's clients. We compute the centroid producer coordinate $\overline{NC_P}(t)$ and centroid consumer coordinate $\overline{NC_C}(t)$ as follows.

$$\overline{NC_P}(t) = \overline{NC}(\{i : i \in P(t)\})$$

$$\overline{NC_C}(t) = \overline{NC}(\{i : i \in C(t)\})$$

- **Maximum Deviation from Centroids.** To cope with the loss of information with centroids, we send the maximum deviation of the clients' coordinates from their corresponding centroid. We denote this as $W(t)$ and it is computed as shown below.

$$\max_{p \in P(t)} d\left(NC(p), \overline{NC_P}(t)\right) + \max_{c \in C(t)} d\left(NC(c), \overline{NC_C}(t)\right)$$

- **Worst-case End-to-end Latency.** For each topic t, we compute the worst-case pub-sub latency across all producer-consumer pairs. We denote this as $E(t)$ and it is computed as follows.

$$\max_{p \in P(t)} d\left(NC(p), \overline{NC}(P[t])\right) + \max_{c \in C(t)} d\left(NC(c), \overline{NC}(P[t])\right)$$

  This information is used to determine whether the current broker, denoted by $P[t]$ is meeting the topic's end-to-end pub-sub latency threshold.

One key point to take note of is that the volume of per-topic monitoring data generated is independent of the number of clients using that topic. Given the high heterogeneity of broker and client network locations in a geo-distributed setting, this aggregation technique significantly reduces monitoring data traffic. By contrast, a naive approach which records the network coordinates of all clients would incur network traffic proportional to the number of clients of each topic.

3.3 Latency and Load-Aware Topic Partitioning

ePulsar's topic partitioning policy uses the fine-grained per-topic latency and load monitoring data collected from brokers to meet end-to-end latency guarantees. We implement this policy by extending the Load Manager module in Pulsar, which periodically processes the latest monitoring data and determines whether the currently observed system state requires an update to the topic partitioning. An update to topic partitioning may be required for one of two reasons.

(1) The end-to-end pub-sub latency for a topic $E(t)$ exceeds the topic's prescribed threshold $L_{th}(t)$.
(2) Workload capacity on a broker exceeds a high watermark, resulting in inflated processing latency on the broker. A broker is said to be overloaded in case the consumption of any one of the hardware resources (CPU, memory or network bandwidth) exceeds a threshold, or if one of the aggregate traffic parameters (e.g., output message rate) exceeds a threshold. Through a comprehensive profiling of Pulsar's data-plane, we have identified the most relevant traffic parameters and their respective thresholds that indicate broker overload. In this scenario, the overloaded broker's per-topic traffic load is analyzed and a subset of topics are selected for migration, such that their removal from the broker will result in reduction of load below the high watermark.

The pseudocode of this algorithm is presented in Algorithm 1. First, we iterate through each topic and determine if its worst-case end-to-end latency violates its threshold and if so we add it to the set of topics to be migrated (lines 15-17). Next, we determine the set of brokers that are overloaded (line 18). For each such broker, we extract a set of topics whose migration would result in avoiding overload (lines 19-21). We compute repartitioning for these topics and final migration commands (reconfigurations) is then executed by the Load Manager. The logic for computing repartitioning of topics is present in the *PlaceTopics* procedure. For each topic we first determine a ranked set of latency-feasible brokers (Section 3.4.1). Next, in the order of increasing size of the candidate set, we try to place each topic on the candidate brokers, such that the topic placement does not result in broker overload (lines 5-10).

The policy for determining overload of brokers, selecting topics for migration, and finding candidate brokers are extensible; we discuss them in detail in Section 3.4. In the event that the Load Manager detects the need for topic repartitioning, a new broker is selected for that topic, and a reconfiguration of topic ownership is carried out (Section 3.6).

3.4 Topic Partitioning Policies

The latency and load-aware topic partitioning algorithm discussed above provides an extensible framework for implementing various policies. In this section we provide detail into the policies we use for each of the constituent decision-making steps in Algorithm 1.

3.4.1 Selecting Broker Based on End-to-End Latency

As a result of the fine-grained per-topic data collected by the latency monitoring discussed in Section 3.2, we propose the following policy to select a broker that keeps end-to-end latency under a specified threshold. For each broker b that is a potential candidate for hosting topic t, we compute $W(t, b)$, the expected worst-case end-to-end latency that broker b would be able to offer.

$$W(t,b) = d\left(\overline{NC_P}(t), NC(b)\right) + d\left(\overline{NC_C}(t), NC(b)\right) + W(t)$$

We then filter the set of brokers for which the approx. worst-case end-to-end latency $W(t, b)$ is under the topic's threshold $L_{th}(t)$. If the set of latency-feasible brokers $B(t)$ is empty, we return the entire broker list as candidates. We rank the candidate brokers by increasing $W(t, b)$.

3.4.2 Filtering Overloaded Brokers

We label a broker as overloaded if one of the following conditions holds true.

(1) The utilization of broker's hardware resources, namely CPU, memory and network bandwidth exceeds a threshold of 85% (same as Pulsar).
(2) Aggregate output message rate (messages/sec) at broker exceeds the threshold R_{out}^{th}. This condition ensures that

Algorithm 1 Topic Manager algorithm. Inputs are P_0 (initial topic partitioning) and M (monitoring data)

```
1:  procedure PlaceTopics(T_migr, P, M)
2:      F ← dict()
3:      for t ∈ T_migr do
4:          F[t] ← get_feasible_brokers_by_latency(t, M)
5:      sort T_migr by increasing |F[t]|
6:      for t ∈ T_migr do
7:          for b_cand ∈ F[t] do
8:              if can_host(b_cand, t, P, M) then ▷ no broker overload
9:                  P[t] ← b_cand
10:                 break
11:     return P
12: procedure PerformRepartitioning(P_0, M)
13:     R ← {}                          ▷ set of migration commands
14:     T_migr ← {}                     ▷ topics to migrate from curr broker
15:     for t ∈ P_0.topics do
16:         if latency_violated(t) then
17:             T_migr ← T_migr ∪ {t}
18:     B_overload ← get_overloaded_brokers(P_0, M)
19:     for b ∈ B_overload do           ▷ resource-based detection
20:         T_b ← get_topics_to_migrate(b, P_0, M)
21:         T_migr ← T_migr ∪ T_b
22:     P ← PlaceTopics(T_migr, P_0, M)
23:     R ← {}                          ▷ set of repartitioning commands
24:     for t ∈ P_0.topics do
25:         if P[t] ≠ P_0[t] then
26:             R ← R ∪ {(t, P_0[t], P[t])}
27:     execute_reconfigs(R)
```

processing latency on the broker does not impact the end-to-end latency. Through extensive profiling of **ePulsar**'s data-plane (as described in Section 5.4), we determine both the metric and the threshold for checking overload.

We note that more complex policies for determining broker overload such as the one proposed by Khare, et al. [22] could also be used. However, we defer the exploration of such policies to future work.

3.4.3 Topic Selection to Migrate from Brokers

For simplicity, we assume that two topics with an identical output message rate (msgs/sec) would incur identical load on a broker. $R_{out}(t)$ represents total incoming and outgoing message rates for a topic t. To minimize the number of migrations required, we prioritize migrating topics with higher $R_{out}(t)$. If the trigger for broker overload was high resource usage, then we determine the target aggregate message rate that topic migration should achieve to lower the resource utilization below the threshold. We keep marking topics as candidate for migration (in decreasing order of $R_{out}(t)$) until the target message rate is achieved. If the trigger for broker overload was high aggregate output message rate, then we remove topics (similar to the above description) until the total output message rate falls below the threshold R^{th}_{out}.

3.5 Reducing Message Processing Latency

Broker communicating with a remote system entity in the critical path of message processing can significantly impact end-to-end latency, even though the broker has been chosen keeping network proximity in mind. Such a workflow exists in Pulsar's data-plane, wherein serving persistent topics requires the broker logging individual messages on the durable storage of one or more BookKeeper nodes (bookies) before the producer can be acknowledged. Thus latency-aware broker selection needs to be augmented with a suitable bookie selection policy for end-to-end latency satisfaction.

ePulsar selects bookie nodes that are resident on the same edge site as the broker to avoid remote communication in the critical path. For better reliability, bookie nodes from distinct racks on the same edge site (Section 2.1) are selected. We allow developers to specify the number of bookie nodes to persist a topic's messages on, and the size of the write quorum (number of bookie acks needed before acknowledging the producer). This design enables developer to choose a tradeoff between fault-tolerance and latency overhead.

3.6 Control-Plane Agility for Reconfigurations

One of the requirements of **ePulsar** is to be responsive when detecting and adapting to workload dynamism via topic repartitioning. **ePulsar** achieves this requirement by (a) better coordination between Load Manager and brokers to reduce per-topic migration time, and (b) exploiting concurrency to increase topic migration throughput. We discuss both these design choices in this section.

Enhanced Coordination. We perform topic unloading from old broker and loading on new broker together as part of the same workflow. We add coordination steps in the communication between the Load Manager and the brokers involved in a topic migration. Fig. 2 shows the steps involved in **ePulsar**'s migration workflow. The Load Manager sends a command to the current broker to release the topic (1). The old broker releases ownership of the topic by updating ZooKeeper (2). Upon successful write to ZooKeeper, it terminates the connections to the clients of that topic, providing them the new broker's address (3), and informs the newly chosen broker to acquire ownership of this topic (4). Clients reconnect to the new broker (7) and resume operation when their connections are established. Step 3 is an expanded control action in **ePulsar** compared to baseline Pulsar. Specifically, notifying the clients of the new broker is not part of the workflow of baseline Pulsar. Further, step 4 does not exist at all in baseline Pulsar. Step 3 lets clients know the next broker for the

topic under migration and eliminates the need for clients to perform topic lookup from the Load Manager. Step 4 allows the old broker to proactively inform the new broker to acquire ownership of the topic, which hides this latency for communication with ZooKeeper (Step 6) from the clients, thus reducing client downtimes. We define the duration between the Load Manager sending "Release Topic" command to old broker (1) and receiving the "Topic Release Complete" response (5) as the *per-topic migration time*. This metric represents the elapsed time incurred in the migration workflow by the Load Manager for a single topic.

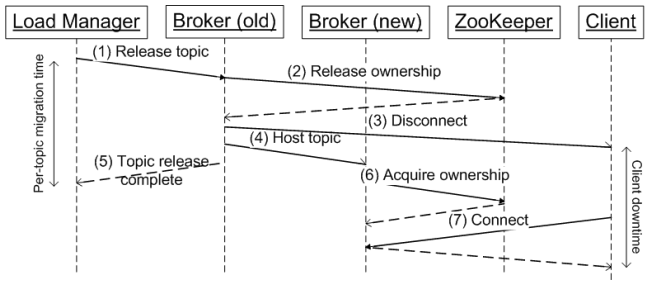


Figure 2: Sequence diagram of **ePulsar**'s topic migration workflow. The time duration for which a client remains disconnected from a broker is termed as *client downtime*.

Concurrent Topic Migration. ePulsar's Load Manager employs a pool of threads to execute topic migration. As shown in Fig. 2, topic ownership in ZooKeeper is updated by the individual brokers themselves. This strategy allows multiple concurrent migrations to proceed, relying on ZooKeeper for ensuring consistent updates to topic ownership. Further, this strategy also allows the Load Manager to carry out concurrent migration of topics to chosen brokers using the thread pool. On the other hand, the Load Manager in the baseline Pulsar carries out topic migration sequentially.

4 Implementation

We implement **ePulsar** by integrating the architectural components presented in Section 3 into the control plane of Apache Pulsar version 2.2.1. As we mentioned in Section 3.2, for fine-grained monitoring we enforce the maximum number of topics in a bundle to be equal to 1, and split any bundle immediately that is assigned more than one topic. Keeping the bundle concept in tact allows **ePulsar** to leverage Pulsar's bundle-oriented monitoring and load management system. We maintain per-topic latency constraints in ZooKeeper and allow application developers to specify them through a command-line utility. The periodicity of processing monitoring data in the Load Manager is 5 seconds.

Deployment Configuration of ePulsar. Broker and bookie nodes are hosted both at edge sites for serving low-latency applications as well as in a remote datacenter (i.e., cloud). The Topic Manager role is assigned to a broker through leader election, although we restrict the role to be assigned to a cloud broker because of better reliability of cloud resources. The ZooKeeper instance is co-resident in the cloud. Clients connect to the pub-sub infrastructure through various access media, e.g., cellular (4G LTE), WiFi or wired networks.

Network Coordinate P2P System. We use Serf [19] as the network coordinate agent as described in Section 3.2. Serf agent uses a gossip protocol to discover peers in the cluster, and newly joined nodes fetch information about seed nodes from a central database. Member nodes of a Serf cluster communicate with each other as per the enhanced Vivaldi NC protocol, which has been shown to converge to stable coordinates after 60 minutes of the NC cluster being up [23].

Integrating NC P2P System with ePulsar. Each of **ePulsar**'s entities - brokers and clients - run an instance of the Serf agent that form a P2P cluster. We designate brokers in the cloud as seed nodes of the NC cluster. **ePulsar**'s clients can be static or mobile depending on the application use-case.

Through experimentation, we have found that the Vivaldi NC protocol does not provide stable latency estimates when mobile clients are nodes in the NC P2P cluster. However, such mobile devices invariably connect to the Internet via a nearby gateway node e.g., local breakout [25] for clients running on a 4G/LTE network. We assume the presence of a lightweight network coordinate proxy (NC Proxy) running on such gateway nodes, serving as the source of network coordinate information for the mobile clients connected to that gateway node. Thus the NC of a mobile client defaults to that of the gateway node that it is currently connected to. Since all the data plane actions of a mobile client goes through its associated gateway node (which serves as the NC Proxy), the client-proxy latency is a constant factor in the end-to-end delay, and is accounted for by adding it to the *height* parameter of the proxy's NC[1].

The entities in **ePulsar** running NC's Serf agents (broker, static clients, and NC proxies) are stable and have a much longer uptime (on the order of days) compared to the 60 minute convergence time of the NC cluster. The client library queries its current network coordinate (or that of its NC proxy) periodically every 5 seconds and reports it to the broker hosting its topic. Similarly, the broker queries its current network coordinate periodically every 5 seconds.

5 Performance Evaluations

We evaluate **ePulsar** to validate the following hypotheses.

(1) Inter-node latency estimation using **ePulsar**'s network coordinate protocol has high accuracy and imposes low overhead on participating nodes (Section 5.3).

[1]Height component of Vivaldi's NC accounts for constant latency faced by clients due to their access network.

(2) Per-topic aggregation of client NCs as centroids does not result in selecting brokers that violate end-to-end (E2E) latency constraints (Section 5.5), while providing considerable savings in monitoring overhead (Section 5.6).
(3) Agility-oriented optimizations in the control-plane result in reduction of migration overheads (client downtime and per-topic migration time) over Pulsar (Section 5.7).
(4) **ePulsar** is able to meet E2E latency constraints for exemplar applications (Section 5.8).

We verify the above hypotheses using two main methods: (1) Microbenchmarks that analyze different parts of **ePulsar**'s architecture in isolation. (2) End-to-end evaluations of multiple application scenarios consisting of realistic infrastructure topology and client workload.

5.1 Evaluation Scenarios

We wish to evaluate **ePulsar** under realistic infrastructure and subscription patterns. For this purpose, we consider the following evaluation scenarios from which we design microbenchmarks and end-to-end experiments.

5.1.1 Unmanned Aerial Vehicle Swarms

A swarm consists of multiple drones that move together for accomplishing a given task. The swarm contains a leader drone and the rest are followers. Each swarm follows a Random Waypoint mobility model [36].

Subscription Pattern. The leader drone sends movement commands to the followers through a topic called *follow_leader*. The followers communicate information extracted from onboard sensors to the leader via a topic *sensor_data*. The E2E pub-sub latency constraint is set at 40 ms.

Infrastructure. We consider a city-wide cellular network equipped with edge resources, where UAVs use LTE as the communication medium. We assume that the city is divided into multiple *Mobile Edge Computing (MEC)* zones, each with a single edge site. The locations of the edge sites is determined via k-means clustering on the cell tower locations [35] of Atlanta [7]. The edge sites communicate with each other via a city-level switch, with inter-site RTT of 30 ms. Each edge site hosts a broker and an NC proxy. Each client is directly connected to the edge site corresponding to its current location based on k-means clustering. Since clients are mobile, they query NC from the respective sites they are directly connected to. The broker running the Load Manager component and the ZooKeeper instance is hosted in the cloud with a one-way latency of 40 ms to any edge site.

5.1.2 Massively Multiplayer Online Gaming

The MMOG scenario comprises multiple players joining a game session from multiple cities across the USA. We consider a distributed game server deployment with each city hosting a game server. Each game server serves clients in the same city, and uses pub-sub middleware to exchange game-state updates with other servers. Client avatar interactions are modeled based on Destiny 2 [5], wherein avatars form groups (uniformly sampled) and play with/against each other. Each avatar has an exponentially distributed lifetime for being present in the current group, after which it would join another uniformly selected group.

Subscription Pattern. The subscription pattern is object-based [6] wherein each avatar a is associated with a topic T_a to which the game server serving a's client pushes information about any action taken by that avatar. The game server subscribes to the topics of all avatars in its current group to receive updates about their gameplay actions. The E2E pub-sub latency threshold is set to 100ms.

Infrastructure. We assume a multi-city infrastructure in the US (similar to Google's Edge network [30]) with each city consisting of a game server and a pub-sub broker. Inter-city latencies are modeled based on the WonderNetwork dataset [32]. Latency between broker and game server in the same city is set to 5 milliseconds. The broker hosting the Load Manager component and the ZooKeeper instance is located in the city of New York, which is also a part of the topology.

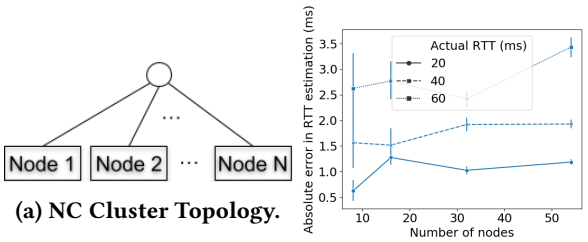
5.2 Evaluation Platform

The evaluation scenarios described in Section 5.1 pose the following requirements to be satisfied by the evaluation platform: (i) support a heterogeneous network topology, (ii) allow emulation of unmodified software components (Pulsar entities and clients), and (iii) emulate device mobility. To satisfy these requirements, we use the *Containernet* [29] evaluation platform, which has also been used by previous edge computing research [15, 33]. Containernet uses Docker containers as hosts (allowing use of unmodified software entities) in network topologies emulated using Open vSwitch. We set custom latencies on the network links using the Linux tool *tc* (to support heterogeneous topologies), and remove/create network links on the fly (to emulate device mobility).

The emulated infrastructure is deployed on an Ubuntu 16.04 VM with 48 CPU cores and 64 GB RAM. We use Docker's resource reservation to allocate dedicated resources to each container and minimize performance interference.

5.3 Latency Estimation via Network Coordinates

Network coordinates offer a more scalable way of measuring pairwise network latencies compared to direct per-pair measurements. The intent in this subsection is to validate the first hypothesis, namely, the efficacy of the NC protocol which is used in **ePulsar**. For this purpose, we consider a simple yet representative topology of an NC cluster as shown in Fig. 3a. Nodes of the NC cluster are connected to a central switch. All the nodes have the same link latency to the switch (which is varied in the experiments using the Linux *tc* tool).



(a) NC Cluster Topology.

(b) RTT estimation error.

Figure 3: Accuracy of the enhanced Vivaldi NC protocol using a simple topology shown in Fig. 3a. The link latency is controlled using the Linux *tc* tool.

Accuracy. We study the impact of two control variables on RTT estimation accuracy of NC protocol: (1) number of nodes in the NC cluster, and (2) the actual RTT setting between the nodes. Metric of interest is per-pair error in RTT estimation. Fig. 3 shows that the error remains relatively constant and low (< 3.5 ms) with increasing number of nodes. The error increases marginally with higher actual RTT but remains small (< 6 %) in relation to the actual RTT setting. There could be transient errors in latency estimation as a result of unpredictable vagaries in the WAN which are out of the control of the NC protocol. The optimizations to the Vivaldi algorithm [23, 24] safeguard the NC protocol from transient noise in RTT measurements between NC Serf agents. **ePulsar**'s latency estimation accuracy is a function of the enhanced Vivaldi NC protocol used by the Serf [19] agent which **ePulsar** adopts as is. Transient errors in latency estimation manifest in **ePulsar** as temporarily sub-optimal broker selection that could violate E2E latency requirements of the pub-sub clients. However, due to the transient nature of these errors, the broker selection will self-correct itself.

Overhead of Running Agent. We measured the CPU and memory usage of the NC agent with two control knobs: number of participating nodes in the protocol (varied from 2-128), and the frequency of querying the agent for its coordinate (varied from every second to every 5 seconds). The CPU utilization for all the above configurations is less than one percent while memory requirement is less than 15 MB.

This set of experiments and results confirms our first hypothesis that using a decentralized network coordinate protocol to estimate network latencies between entities is accurate. At the same time, it incurs low resource overhead.

5.4 Profiling ePulsar's Data-Plane

In this section we describe the methodology for determining the traffic parameters and thresholds for determining the broker overload (as mentioned in Section 3.4.2). We do so by profiling the data-plane of **ePulsar** against the workload generated by OpenMessaging benchmark[27]. Our evaluation setup comprises 1 broker, 1 bookie and 2 client nodes each running on a separate virtual machine; each with 8 CPUs and 16 GB RAM. We focus on persistent topics for this evaluation, as they exert more load on the data-plane. We measure the E2E latency against a wide range of workloads by changing the following traffic parameters: number of topics, number of producers and consumers per topic, message size, and per-topic messages rate. From the collected data, we found that the aggregate output message rate (msgs/sec) on a broker is strongly correlated with *p95* E2E latency, and rates higher than 1500K messages per second leads to significant deterioration of processing latency on the broker. Therefore, in the E2E evaluations, we use this threshold as a filter to determine broker overload in the broker selection policy.

5.5 Broker Selection Policy Assessment

Earlier (in Section 3.4.1) we proposed a topic partitioning policy for satisfying the latency constraints of topics. In this section, we evaluate the effectiveness of the policy to meet its objective for realistic infrastructure topologies and client subscription patterns. We compare the proposed policy against the following two baselines.

- **AllPairs**. Same as **ePulsar**, but instead of clients' NC centroids, **AllPairs** takes the NC of each individual client and computes the expected E2E latency for each producer-consumer pair. A broker is chosen only if the worst-case E2E latency falls below the threshold.
- **Pulsar**. As mentioned in Section 2.3, Pulsar offers well-developed data-plane semantics which are appropriate for the target applications for the edge. Therefore, we choose Pulsar as the other baseline. Pulsar uses consistent hashing to compute the hash for a topic name. The output space of the hash function is divided among all brokers uniformly. The topic is assigned to the broker inside whose partition the topic's hash falls.

5.5.1 Methodology

For each representative application scenario mentioned in Section 5.1, we generate infrastructure topologies with varying amount of geo-distribution. For the inter-city MMOG topology we vary the number of cities, while for the UAV swarm topology we vary the number of MEC zones in the metropolitan area. For each such topology, we first emulate the infrastructure of the given topology using Containernet. After allowing the NC agents in brokers and clients to stabilize for 10 minutes, we query each agent's coordinate. The querying is done once per topology. Using the coordinates of all nodes in the topology, we can then estimate the E2E delay for any producer-consumer pair of a topic given the location of the clients and the broker hosting that topic. Based on the specific application scenario's subscription pattern, we determine the clients for each topic and place them on the nodes of the generated topology. The coordinates of the

producers and consumers for each topic serve as the input to the broker selection policy. We analyze the result of selection policy in terms of the E2E latency and violation ratio. Violation ratio represents the fraction of producer-consumer pairs for whom the latency threshold is violated. In the experiments, we consider 1000 different random permutations of client placement and topic subscriptions. The intent is to have a large coverage of possibilities wherein clients could be located in different geographical areas and/or could be subscribing to different sets of topics.

5.5.2 MMOG Scenario

We vary the number of cities in the topology and distribute a total of 1024 MMOG clients across all cities with equal probability (following the strategy suggested in Deng, et al. [11]). We randomly assign the clients into 128 groups to simulate group formation during the session. Clients' avatars follow the subscription pattern described in Section 5.1.2.

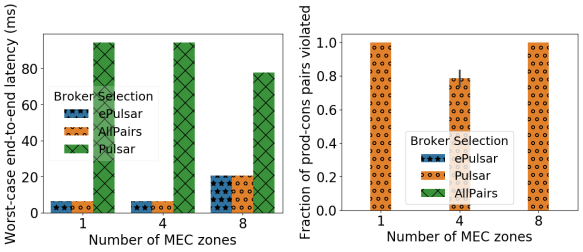
(a) Worst-case E2E latency **(b) Violation Ratio**

Figure 4: Analysis of broker selection policy for MMOG application scenario.

Fig. 4a shows the worst-case E2E latency over all producer-consumer pairs for all topics in the evaluation for the different broker placement policies. Pulsar's consistent hashing is consistently worse than the two latency-aware placement strategies (AllPairs and **ePulsar**'s policy). **ePulsar** offers a worst-case latency similar to AllPairs. In Fig. 4b, the violation ratio for each broker selection policy is shown. Pulsar's consistent hashing causes a high number of violations due to latency-agnostic topic partitioning, whereas AllPairs and **ePulsar** offer similar levels of violation.

5.5.3 UAV Swarm Scenario

We vary the number of MEC zones in the simulated metro area and distribute 16 UAV swarms in the city. Each swarm comprises 8 UAVs, with one of them serving as the leader. UAVs follow the subscription pattern described in Section 5.1.1. Figs. 5a and 5b show the worst-case E2E latency and violation ratio over all producer-consumer pairs. Since **ePulsar** performs latency-aware topic partitioning, the worst-case latency remains under the threshold, resulting in no violations even when the number of MEC zones is increased.



(a) Worst-case E2E latency **(b) Violation Ratio.**

Figure 5: Analysis of broker selection policy for UAV swarm application scenario.

The results in this subsection, for both application scenarios, validate the first part of the second hypothesis that the loss of information by aggregating clients' network coordinates as centroids does not result in poor broker selection with respect to meeting E2E latency constraints.

5.6 Monitoring Overhead Reduction

We evaluate the savings in monitoring traffic by aggregating per-topic client NCs at the serving broker before reporting them to ZooKeeper. This traffic is sent continuously through the WAN and impacts scalability of the system, hence we consider aggregate monitoring traffic rate as the metric-of-interest. We focus our evaluation on a single broker hosting topics with multiple clients - as the behavior is independent of other brokers. We vary the number of topics hosted on the broker and the number of clients connected to each topic.

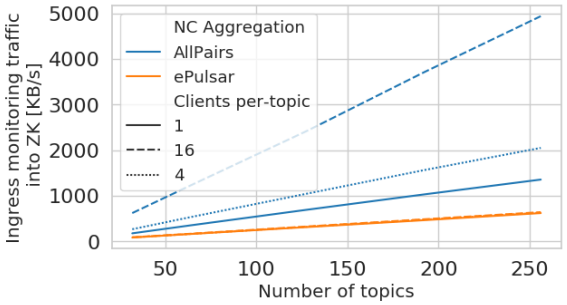
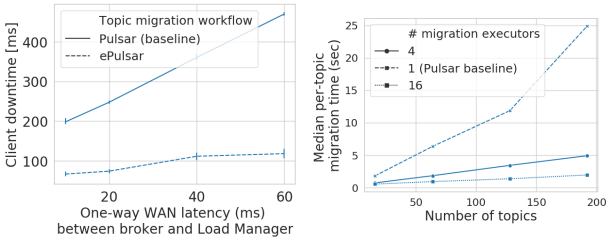


Figure 6: Monitoring traffic rate under varying number of topics and clients per topic. **ePulsar**'s NC aggregation results in considerable savings over naive AllPairs.

Fig. 6 shows the data rate of monitoring traffic sent to ZooKeeper. An increasing number of topics results in higher data rate. The rate of increase is higher without centroid aggregation (AllPairs policy) and also with more clients per topic. **ePulsar**'s aggregation, however, causes data rate to be independent of the number of clients - since a constant amount of data is sent to ZooKeeper per topic. These results validate the second part of the second hypothesis that aggregating clients' network coordinates as centroids results in reducing the monitoring overhead.

5.7 Reduction of Migration Overhead

Here we present the improvements due to **ePulsar**'s optimizations to Pulsar's topic migration workflow. **ePulsar** improves agility of the control-plane by improving coordination between the Load Manager and the brokers involved in a migration. Higher concurrency for executing topic migrations is another contribution that augments the control plane agility. We choose two metrics of interest: (1) client downtime during topic migration, i.e., time between disconnection from the old broker and establishment of connection to the new broker, and (2) per-topic migration time to the new broker. We vary the number of topics that are concurrently migrated, the one-way WAN latency between brokers at the edge and the Load Manager, and the number of migration executor threads in the Load Manager. Our test setup comprises 2 brokers and multiple client containers running the client processes. For this experiment, we use a custom Load Manager implementation that periodically triggers the migration of all topics in the system from the first to the second broker and back and so on. Each topic has only 1 client so that we can focus on the worst-case client downtime.



(a) Client downtime when 128 topics are simultaneously migrated using 16 migration executors.

(b) Median per-topic migration time (one-way WAN latency between broker and Load Manager = 40ms).

Figure 7: Comparison of topic migration overheads for baseline Pulsar and **ePulsar**. Note that only **ePulsar** benefits from multiple migration executor threads.

Fig. 7a shows that **ePulsar**'s client downtime is significantly lower than that of baseline Pulsar when 128 topics are concurrently migrated using 16 migration executor threads. The optimized migration workflow of **ePulsar** (Fig. 2) is the primary reason for this performance gain. Multiple rounds of communication with the remote Load Manager and ZooKeeper through the WAN is the cause for the inflation of the client downtime for the baseline. The higher the WAN latency between the broker and the Load Manager the higher the gain for **ePulsar**. Fig. 7b shows the per-topic migration time with varying number of topics and different settings of migration executor threads. The one-way WAN latency from the broker to the Load Manager is set at 40 ms for this experiment. **ePulsar** benefits from the concurrency in migration execution to reduce the per-topic migration time, while the baseline Pulsar does topic migration sequentially (Section 3.6).

The results in this subsection validate the third hypothesis regarding the agility of **ePulsar** to reduce the completion time and client downtimes during topic migration.

5.8 End-to-End Evaluations

In this section, we evaluate **ePulsar**'s ability to respect E2E latency constraints of the exemplar applications (Section 2.2), and validate the fourth hypothesis. The metric we use for the evaluation is E2E pub-sub latency – i.e., the elapsed time between a client publishing on a topic and the receipt of the published message by all the clients subscribing to that topic.

5.8.1 UAV Swarms Scenario

We consider the infrastructure topology described in Section 5.1.1 with 4 MEC zones and emulate using Containernet. We emulate each UAV swarm as an independent container where the mobility of all of the members of the swarm are identical. In the emulated network topology, each zone consists of a network switch to which the broker and the NC proxy connect. We create a link between the swarm's container and the switch corresponding to the swarm's current MEC zone. When a swarm moves into a new zone, the link to the previous zone's switch is removed and a link to the new zone's switch is created. We emulate 8 independent swarms, each with 8 UAVs, following the Random Waypoint mobility model in the city at a relatively high speed of 50 meters/sec[2]. Both leader and followers generate 200 msgs/sec each of size 1 KB [37]. We perform this experiment for 10 minutes.

We show the E2E latency of a single representative topic from each swarm in Fig. 8[3]. For each swarm. the E2E latency remains under the latency threshold (40 ms) for most of the experiment duration. Transient violations of latency threshold occur when a swarm moves into a different MEC zone than the one currently hosting the swarm's topics. **ePulsar**'s monitoring module detects such violations and triggers migration of the swarm's topics to the broker at the new MEC zone, after which the E2E latency returns back under the latency threshold.

5.8.2 MMOG Scenario

We emulate a realistic instance of the MMOG scenario on an infrastructure topology consisting of 5 cities in USA, as described in Section 5.1.2. Each city contains 1 broker node and multiple edge sites each running an NC proxy and a game server (the game servers are stationary clients to **ePulsar**). 64 MMOG mobile clients are uniformly distributed among the

[2]We use such a high speed to trigger several mobility-driven topic migrations during the experiment.

[3]To avoid cluttering the figure, we do not show all the topics of each swarm since their behavior is identical.

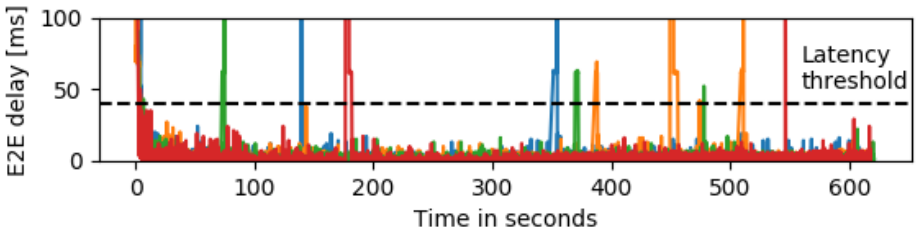


Figure 8: E2E latencies experienced by the 8 independent drone swarms for their respective representative topic over time. Latency violations (transient spikes) are observed when a swarm moves from one MEC zone to another. For a brief period, the latency remains higher than the threshold. **ePulsar**'s topic migration brings the latency back under the threshold.

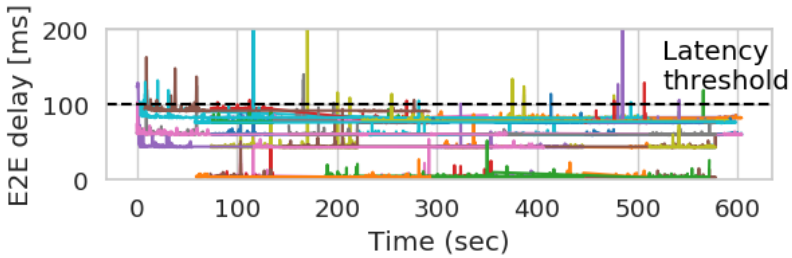


Figure 9: E2E Area-of-Interest (AoI) latencies for one representative client avatar in the MMOG scenario. Each line represents the delay experienced in receiving updates made by individual avatars that are in the representative client's AoI, and runs for as long as the avatars are in the AoI of the representative client.

cities and they form groups at random. To model dynamism, the emulated MMOG mobile clients leave their current group after a period T, and join another group at random. Each client samples T from an independent uniform distribution between 30 and 60 seconds. We set the per-avatar update message size to 998 bytes and the message rate to 300 messages/sec [38]. We perform this experiment for 10 minutes.

Fig. 9 shows the E2E Area-of-Interest (AoI) latency experienced by one particular MMOG mobile client. During its gameplay, a number of different avatars enter the AoI of the given client's avatar, and we denote the delay in receiving their action updates by an individual line. Transient spikes in E2E latency are observed when the change in subscriptions to a topic causes the migration of the topic to a better broker. **ePulsar** is able to consistently provide E2E latency below the threshold of 100ms even with frequent AoI changes.

6 Conclusion and Future Work

We presented a control-plane architecture for edge-centric pub-sub systems and integrated it into an open-source cloud-centric pub-sub system Apache Pulsar. The resulting system, **ePulsar**, performs latency-aware topic partitioning and supports agile reconfiguration in the event of E2E latency violation. For scalable inter-node latency estimation, **ePulsar** incorporates an enhanced Vivaldi network coordinates protocol. **ePulsar** performs continuous monitoring to detect E2E latency violations, which triggers topic repartitioning and migration of topics to new brokers. The control flow of **ePulsar** is optimized to reduce client downtime and topic migration overheads. Microbenchmarks and end-to-end evaluations show the agility of **ePulsar** relative to the baseline Pulsar. While the design principles of **ePulsar** have been demonstrated using Pulsar, the principles are general and can be applied to any edge-centric topic-based pub-sub system.

Avenues for future work include reducing the dependence on centralized components in the control-plane of **ePulsar**, specifically the Load Manager and the ZooKeeper, thus increasing **ePulsar**'s scalability in a geo-distributed setting. Distributed management of topics in a pub-sub system has been explored by Dedousis, et al. [10], and similar techniques can be incorporated into **ePulsar**. Prior art has explored the use of Raft consensus and Serf gossip protocol to replace ZooKeeper in Kafka [13], and such ideas can be explored for adoption in **ePulsar** as well.

7 Acknowledgments

We would like to thank our shepherd, Dr. Eve Schooler, and all anonymous reviewers for their insightful feedback and suggestions, which substantially improved the content and presentation of this paper. This work was funded in part by NSF CPS-1446801, NSF CNS-1909346, and a gift from Microsoft Corp.

References

[1] Sherif Abdelwahab and Bechir Hamdaoui. Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms. *CoRR*, abs/1610.00620, 2016.

[2] Apache. Pulsar architecture overview : Apache bookkeeper. https://pulsar.apache.org/docs/en/concepts-architecture-overview/#apache-bookkeeper. Accessed: 2021-01-20.

[3] Marios Assiotis and Velin Tzanov. A distributed architecture for mmorpg. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, pages 4–es, 2006.

[4] Sabur Baidya, Zoheb Shaikh, and Marco Levorato. Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 37–45, 2018.

[5] Bungie. Destiny 2 new player guide. https://www.bungie.net/en/Guide/Destiny2#entry3. Accessed: 2021-01-20.

[6] César Cañas, Kaiwen Zhang, Bettina Kemme, Jörg Kienzle, and Hans-Arno Jacobsen. Publish/subscribe network designs for multiplayer games. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, page 241–252, New York, NY, USA, 2014. Association for Computing Machinery.

[7] CellMapper. Cellmapper. https://www.cellmapper.net/. Accessed: 2021-01-20.

[8] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6. IEEE, 2012.

[9] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004.

[10] Dimitris Dedousis, Nikos Zacheilas, and Vana Kalogeraki. On the fly load balancing to address hot topics in topic-based pub/sub systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 76–86. IEEE, 2018.

[11] Y. Deng, Y. Li, R. Seet, X. Tang, and W. Cai. The server allocation problem for session-based multiplayer cloud gaming. *IEEE Transactions on Multimedia*, 20(5):1233–1245, 2018.

[12] Benoit Donnet, Bamba Gueye, and Mohamed Ali Kaafar. A survey on network coordinates systems, design, and security. *IEEE Communications Surveys & Tutorials*, 12(4):488–503, 2010.

[13] DZone. Why disintegration of apache zookeeper from kafka is in the pipeline. https://dzone.com/articles/why-disintegration-of-apache-zookeeper-from-kafka. Accessed: 2021-01-20.

[14] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003.

[15] Claudio Fiandrino, Alejandro Blanco Pizarro, Pablo Jiménez Mateo, Carlos Andrés Ramiro, Norbert Ludant, and Joerg Widmer. openleon: An end-to-end emulation platform from the edge data center to the mobile user. *Computer Communications*, 148:17–26, 2019.

[16] J. Gascon-Samson, F. Garcia, B. Kemme, and J. Kienzle. Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 486–496, June 2015.

[17] J. Gascon-Samson, J. Kienzle, and B. Kemme. Multipub: Latency and cost-aware global-scale cloud publish/subscribe. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2075–2082, 2017.

[18] Xinjie Guan, Xili Wan, Feng Ye, and Baek-Young Choi. Handover minimized service region partition for mobile edge computing in wireless metropolitan area networks. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–6. IEEE, 2018.

[19] HashiCorp. Serf : Network coordinates. https://www.serf.io/docs/internals/coordinates.html. Accessed: 2021-01-18.

[20] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8, 2010.

[21] Vapor IO. Kinetic edge, edge colocation and interconnection - vapor io. https://www.vapor.io/. Accessed: 2021-01-21.

[22] Shweta Khare, Hongyang Sun, Kaiwen Zhang, Julien Gascon-Samson, Aniruddha Gokhale, Xenofon Koutsoukos, and Hamzah Abdelaziz. Scalable edge computing for low latency data dissemination in topic-based publish/subscribe. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 214–227. IEEE, 2018.

[23] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, page 22, USA, 2007. USENIX Association.

[24] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of euclidean embedding for host-based network coordinate systems. *IEEE/ACM Transactions on Networking*, 18(1):27–40, 2009.

[25] Seung-Que Lee and Jin-up Kim. Local breakout of mobile access network traffic by mobile edge computing. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 741–743. IEEE, 2016.

[26] EDGE MICRO. Edgemicro, a single source for edge colocation worldwide. https://www.edgemicro.com/. Accessed: 2021-01-21.

[27] OpenMessaging. Apache pulsar benchmarks. http://openmessaging.cloud/docs/benchmarks/pulsar/. Accessed: 2021-01-20.

[28] Larry Peterson, Ali Al-Shabibi, Tom Anshutz, Scott Baker, Andy Bavier, Saurav Das, Jonathan Hart, Guru Palukar, and William Snow. Central office re-architected as a data center. *IEEE Communications Magazine*, 54(10):96–101, 2016.

[29] Manuel Peuster. Containernet. https://containernet.github.io/. Accessed: 2021-01-21.

[30] Jared N Plumb and Ryan Stutsman. Exploiting google's edge network for massively multiplayer online games. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–8. IEEE, 2018.

[31] Thomas Rausch, Stefan Nastic, and Schahram Dustdar. Emma: Distributed qos-aware mqtt middleware for edge computing applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–197. IEEE, 2018.

[32] Paul Reinheimer. https://wondernetwork.com/. Accessed: 2021-01-21.

[33] Christian Esteve Rothenberg, Danny A Lachos Perez, Nathan F Saraiva de Sousa, Raphael V Rosa, Raza Ul Mustafa, Md Tariqul Islam, and Pedro Henrique Gomes. Intent-based control loop for dash video service assurance using ml-based edge qoe estimation. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 353–355. IEEE, 2020.

[34] Anam Tahir, Jari Böling, Mohammad-Hashem Haghbayan, Hannu T Toivonen, and Juha Plosila. Swarms of unmanned aerial vehicles—a survey. *Journal of Industrial Information Integration*, 16:100106, 2019.

[35] Shangguang Wang, Yali Zhao, Jinlinag Xu, Jie Yuan, and Ching-Hsien Hsu. Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:160–168, 2019.

[36] Wikipedia. Random waypoint model. https://en.wikipedia.org/wiki/Random_waypoint_model/. Accessed: 2021-01-15.

[37] Guang Yang, Xingqin Lin, Yan Li, Hang Cui, Min Xu, Dan Wu, Henrik Rydén, and Sakib Bin Redhwan. A telecom perspective on the internet of drones: From lte-advanced to 5g. *arXiv preprint arXiv:1803.11048*, 2018.

[38] Kaiwen Zhang, Cesar Canas, Bettina Kemme, Jörg Kienzle, and Hansarno Jacobsen. Publish/subscribe network designs for multiplayer games. 12 2014.