EdgeRDV: A Framework for Edge Workload Management at Scale

Gloire Rubambiza
Cornell University
Ithaca, NY, U.S.A
gloire@cs.cornell.edu

Braulio Dumba
IBM Research
Yorktown Heights, NY, U.S.A
braulio.dumba@ibm.com

Andrew J. Anderson
IBM Research
Yorktown Heights, NY, U.S.A
Andrew.Anderson@ibm.com

Hakim Weatherspoon
Cornell University
Ithaca, NY, U.S.A
hweather@cs.cornell.edu

***Abstract*—Edge computing is a distributed computing paradigm that moves data-intensive applications and services (e.g., AI) closer to the data source. The rapid growth of edge endpoints connected to the Internet today poses several challenges in scalable application life cycle management. That is, managing data and workloads on several thousand, up to millions of edge endpoints, challenged by limited connectivity, resource constraints, network and edge endpoint failures. In this work, we present EdgeRDV, a new edge abstraction that builds on the idea of rendezvous nodes to manage edge workloads at scale. The EdgeRDV architecture is comprised of a central cloud management endpoint (or cloud hub), a central gateway for each edge site (or edge hub), redundant gateways (or rendezvous nodes), and edge endpoints. Beyond its scalable architecture, EdgeRDV presents new techniques and algorithms that address single points of failures and provide adjustable levels of resilience and cost-effectiveness in edge network deployments. We conducted preliminary experiments to evaluate EdgeRDV, through simulations, and our results show that EdgeRDV requires one to three orders of magnitude fewer intermediate nodes compared to relay structures, can gracefully adapt to failures, and requires a constant number of messages during failure recovery in edge sites with up to 667K+ edge endpoints.**

***Index Terms*—edge computing, scalability, rendezvous, binary trees, IoT.**

I. Introduction

Edge computing is a distributed computing paradigm that moves compute and storage away from centralized points. In fact, edge computing distributes applications, data, and services geographically closer to the edge endpoints that consume these services. The roots of edge computing reach back to the 1980's when Prodigy [1], [2], [3], an early online service, implemented content caching near the edge to provide better and faster service for users. Content delivery networks (CDNs) [4], [5] such as Akamai extended this concept to the Internet to accelerate web performance and delivery of contents. Edge computing generalizes and extends these concepts.

The rapid growth in the number of edge endpoints (e.g., computers, mobile devices, smart appliances, kubernetes (k8s) clusters, and other Internet of Things (IoT) devices.) connected to the edge of the Internet today has increased significantly due to advances in AI, computing, and storage. Such rapid growth has also accelerated the amount of data generated at the edge, for example: studies show that around 10% of enterprise-generated data is created outside a traditional centralized data center or cloud, and this figure will reach 75% by 2025 [6]. Additionally, the data generated from IoT-connected edge endpoints is estimated to grow to 73.1 ZB by 2025 [7]. As the volume and velocity of data increases at the edges, so too does the number of applications deployed at the edge and the need to address some of the shortcomings of cloud computing. For instance, it is expensive and inefficient to transfer data generated at the edge to a cloud or data center for processing. It is also expensive and inefficient to transfer data back to the edge from the cloud or data center. Moving data through and across boundaries in a network topology (ingress, egress) consumes capacity and adds latency to the transmission process. Furthermore, data processed in the cloud lacks local contextual information, such as precise user location, local network conditions, or information about users' mobility behavior [8].

To address these challenges, advances in edge computing are focused on collecting, processing, and enabling insightful decisions, for a variety of industry use cases (e.g., manufacturing, remote exploration, smart homes, supply chains), at or near where data is collected. When data is processed and analyzed at/near its collection point at the edge, communications between the edge and the cloud is reduced to such things as delivery of application and configurations, reporting, and data summaries. Hence, data can be processed in a fast and timely manner, while reducing network consumption, to meet the requirements of modern applications (e.g., fast response time, data privacy/sovereignty and security[1], etc.).

To manage the life-cycle (e.g., deploy, update, and retire) of edge applications and to bring value to the large amount of data generated at the edge today, many edge applications are deployed in a hub and spoke model (e.g., IBM MVI [9], Open Horizon [10], etc.). The hub is the central control plane for management of edge workloads deployed in a number of spoke locations. A spoke is where data is generated or locally aggregated for processing. A telecommunications access point, an assembly line, or a retail branch are good examples of places where spokes are deployed and operated. Deploying

Work done when first author was an intern at the IBM T. J. Watson Research Center.

[1] Analyzing data where it originates limits the risk of a security breach. Edge computing also introduces other security challenges but its discussion is out of the scope of this paper

and operating an application at the edge raises several challenges, including scalable application life cycle management. Additional challenges are due, among other things, to limited connectivity, network and edge endpoint failures, constrained compute and storage resources. In particular, how can we deploy and manage applications deployed in edge endpoints across various locations, using a single control point or 'single pane of glass'? Further, how can we ensure timely application status updates in the face of limited network bandwidth, planned disconnections, and connection failures at the edge? How can we handle the scale, diversity, and density of edge endpoints in so many edge sites[2]? Managing at scale is the key issue at the edge as the number of edge endpoints located across a multitude of sites can reach several thousand (e.g., quick service restaurant: McDonald's drive-thru order processing [11] and k8s edge clusters at Chick-fil-A [12]) up to a 100K+ (e.g., number of vehicles in a software defined vehicle network and total number of appliances, devices, and sensors in smart factory across several sites in industry 4.0 [13], [14]).

To address this gap, several frameworks [15], [16], [17], [10] have been proposed to mitigate some of the limitations associated with edge application lifecycle management. However, these methods have scalability limitations, some are susceptible to single points of failure and/or require significant bandwidth for edge application management. CDNs can be used to deploy edge applications at scale but CDNs alone are not optimal to minimize data transfer and network bandwidth between cloud and edge as well as minimizing the response time for application deployment. Furthermore, using CDNs alone introduces other challenges, for example: *how do we efficiently propagate application status from edge to cloud?* (more discussion in Sect. IV).

To address these gaps, we present a new edge abstraction that builds on the idea of rendezvous nodes. We demonstrate that our approach (1) can be optimized to accommodate different levels of fault tolerance and costs, (2) avoids single points of failures through rendezvous node replication, and (3) requires one to three orders of magnitude fewer intermediate nodes and up to 11% fewer messages as compared with relay structures, in networks of up to 667K+ edge endpoints.

The contributions of this paper are as follows:

- We propose EdgeRDV, a scalable architecture for edge workload management comprised of a cloud-based management endpoint (cloud hub), a central controller for each site (edge hub), redundant gateways for each edge endpoint (rendezvous nodes), and edge endpoints.
- We conduct preliminary experiments to evaluate EdgeRDV's scale and resilience in edge sites up to 667K+ edge endpoints, and we show that our framework can be optimized for cost and resilience.
- We discuss how techniques in EdgeRDV are broadly applicable to distributed edge settings, including scalable infrastructure bootstrapping, adjustable network resilience, and efficient resource usage.

The remainder of the paper is organized as follows. Section II discusses our framework and methodology. Section III presents our experiments and discusses our experimental results. Section IV discusses the implications of our work. Section V discusses related works. We conclude and discuss future work in Section VI.

II. Methodology

In this section, we present our framework, capable of managing workloads across thousands of edge sites and serving up to a 100K+ edge endpoints. We achieve this by minimizing bandwidth and avoiding single points of failure. Figure 1 shows the architecture of our framework. It comprises of a *cloud hub*, an *edge hub* for each edge site, redundant *rendezvous nodes* and *edge endpoints*[3]. Next, we describe each of these architectural elements and their interactions in more detail. We conclude this section by providing a detailed discussion on the rendezvous node's selection algorithm and on the network bootstrapping process.

A. Architecture

1) Cloud Hub: the cloud hub serves as a central control and management plane for multiple edge sites. It provides a global view of the edge infrastructure. It is where site operators or admins configure or push manifests, expressing desired state, that include a list of objects that must be deployed to edge endpoints at a target edge site. The cloud hub also collects aggregated status from the objects running at all edge endpoints. Hence, the cloud hub can effectively manage objects/states of edge endpoints spread across hundreds/thousands of edge sites by communicating with edge hub computing resources deployed at every site.

2) Edge Hub: it is the control point for each edge site, serving thousands of edge endpoints. The edge hub receives metadata from the cloud hub to pull and cache appropriate objects (e.g., AI model) from object stores. Importantly, the caching reduces the overhead for the cloud hub because edge endpoints at the lower layers of the architecture do not communicate directly with it. While the edge hub provides scalable operations at each edge site, it also represents a single point of failure in case of network outages. We addressed this limitation using redundant rendezvous nodes, which are described next.

3) Rendezvous Nodes: (or RDV nodes for short), represent a replicated buffer zone between the edge hub and edge endpoint. RDV nodes pull and cache objects from the edge hub. Further, RDV nodes aggregate status for all edge endpoints under their management. RDV nodes provide three benefits. First, they reduce the communication overhead at the edge endpoints by limiting edge endpoint-to-edge hub messages during the network bootstrapping phase (see II-B3). Second, they preserve valuable edge hub bandwidth by avoiding relayed requests for each edge endpoint to the cloud hub and/or object store. Third, RDV node redundancy overcomes the

[2] A physical location with several edges nodes such as a manufacturing site

[3] Recall that, in this paper, we use the term "edge endpoint" to refer to an edge computing node such as an edge cluster or edge devices such as cameras

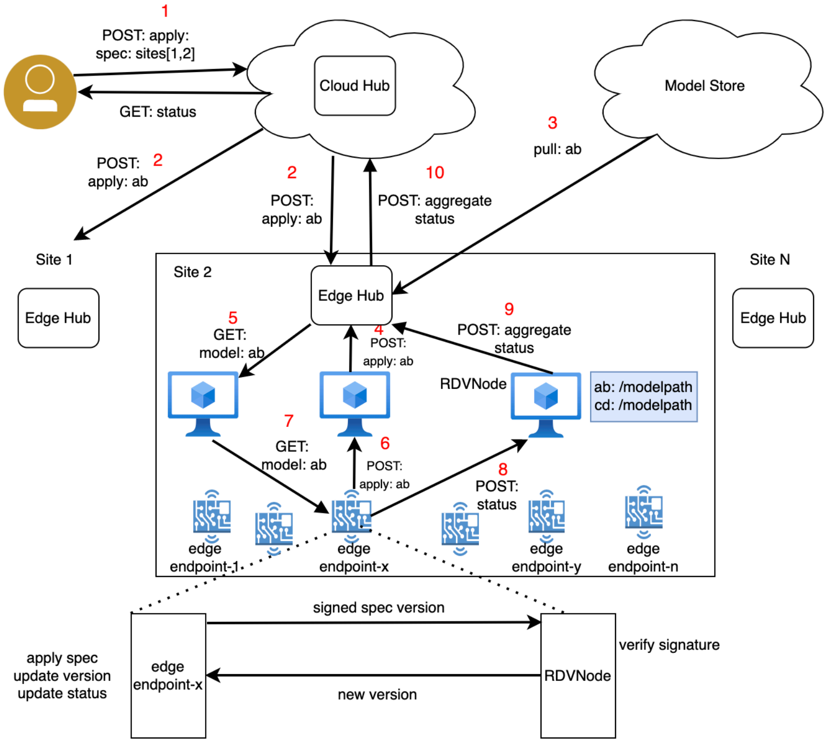


Fig. 1. The EdgeRDV architecture: comprised of a cloud-based management endpoint (cloud hub), a central controller for each edge site (edge hub), redundant gateways (rendezvous nodes), and edge endpoints - illustrating how a workload (an AI model as a use-case) is deployed from the cloud to edge endpoints

single point of failure within an edge site (i.e., the edge hub). To further improve resilience, we will investigate the trade-offs of selecting a RDV node to become an edge hub via a leader election algorithm in future work.

4) Edge Endpoints: The edge endpoints receive metadata with updates from RDV nodes, apply the updates, and continuously report their state. Edge endpoints only report the aggregate status of all the objects or workloads under their management, therefore minimizing the state transported to the cloud. Note that, in our framework, edge endpoints and RDV nodes are located in the same network, and their communication modality represents one implementation of peer-to-peer communication. In some scenarios an edge endpoint can also function as a RDV node.

B. RDV Selection and Network Bootstrapping

As discussed above, the introduction of RDV nodes improves the scale and resilience of our architecture. Edge endpoints in an edge site only communicate with RDV nodes to pull objects and report status. RDVs are assigned to nodes during the network bootstrap process. However, as the number of edge endpoints associated with an edge site increases, we want to minimize the overhead of each RDV node and keep the assignment of RDV nodes to edge endpoints balanced. To minimize overhead we need to address the following challenges: a) placement of RDV nodes in a network (II-B1); b) selection and recommendation of RDV nodes for each edge site (II-B2) and c) tractable and fault-tolerant assignment of edge endpoints to RDV nodes (II-B3). In the following discussion, we present our design and solutions to overcome these challenges:

To assign RDV nodes to edge endpoints, we constructed a virtual id (or VID) space using a Kademlia-like [18], [19] virtual binary tree during the network bootstrapping phase (similar to the approach in [20]). In this tree, the edge endpoints are represented by the leaf nodes and all the intermediary nodes are logical nodes and candidates for RDV node selection (different from the approach in [20]). In this virtual tree, the VID of a node is the L-bit long binary string along the path from the root to the corresponding node (see Fig. 2). Like in [20], L denotes the number of bits used to represent the VID space. Hence, the logical distance between a pair of nodes in the VID space is L minus the length of the longest common prefix for the pair. For example, suppose node A with VID = 00000 and Node B with VID = 10000. Then the logical distance(γ) between nodes A and B is 5 (L=5; γ = 5-0 = 5).

This VID space can also be used to embed the physical connectivity or proximity between edge endpoints [20]: if two edge endpoints are close in the VID space, then they are also close in the physical topology. To build the VID space we use the top-down and centralized algorithm proposed in [20]. This algorithm is suitable when the initial edge site or network topology is known a priori. Next, we describe our methodology to select RDV nodes, how we assign these nodes to edge endpoints, and our network bootstrapping process.

1) RDV Node Placement: Recall RDV nodes are responsible for the delivery of objects and metadata to edge endpoints. Hence, the right placement of RDV nodes is a prerequisite to answer availability and fault tolerance questions in an edge site. We considered two approaches (see Fig. 2) to find the optimal minimum latency placement of RDV nodes at scale.

Let M denote the depth[4] of VID space (i.e., edge endpoints are leaf nodes located at depth M as discussed above). Let N denote a depth in the VID space, where $1 <= N < M$, and S is the set of 2^N bit strings that can be assigned to RDV nodes. The first approach operates as follows. Suppose there are *at most* D available nodes that can be assigned as RDV nodes, where $D < 2^N$. The first (or **minimalist**) approach assigns the RDV nodes VIDs by using a constant X, where $X = \frac{(2^N)}{D}$, to choose eligible VIDs in S starting from index 0. We call X an *additive index*. For example, in Figure 2, N = 3, D = 3. Therefore, the additive index is 2 (i.e.,$\frac{2^3}{3}$), the indices are (0, 2, 4), and the corresponding VIDs are illustrated in the figure. Note that the minimalist approach prioritizes the *cost*[5] of bootstrapping an edge network by *mapping edge endpoints constraints into the VID space.*

In contrast, the second (or **optimist**) approach assigns RDV node VIDs by choosing an arbitrary depth N where all VIDs in S are eligible for assignment. For example, in Figure 2, N = 2. Therefore, the network must have 4 (i.e., 2^2) RDV nodes. Note that the optimist approach prioritizes *resilience* by *mapping as many VIDs from the VID space as possible, assuming no edge endpoint constraints*. By resilience, we mean keeping a balanced load among all RDV nodes in case of a single RDV failure.

To explore these two approaches, we conducted a theoretical edge-endpoint-to-RDV-node assignments and RDV node failure simulations. Specifically, by placing various quantities of RDV nodes at different binary tree depths and reassigning edge endpoints in a scenario where their primary RDV node fails. Figure 3 demonstrates that, regardless of the tree depth, the minimal approach skews the RDV-node-to-edge endpoint redistribution (i.e., lower resilience) in case of a single RDV node failure, in exchange for lower cost. On the other hand, the optimist approach maintains network balance (i.e., higher resilience) during redistribution while allowing a variable tree depth to be chosen for the initial RDV node VID assignments (i.e., variable cost). In other words, the optimist approach maintains both *cost* and *resilience flexibility*. Therefore, the RDV placement method used throughout this work relies on the optimist approach.

2) RDV Node Selection and Recommendation: To support 100K+ edge endpoints, the number of RDV nodes must scale gracefully with the number of edge endpoints while maintaining desirable levels of resilience. To achieve that, we developed an RDV node recommendation algorithm that takes into consideration cost and resilience trade-offs: given an edge site's desired number of associated edge endpoints and a resilience level (expressed as a percentage of total edge endpoints coverage per RDV node), algorithm 1 outputs the following: 1) binary tree depth; 2) number of RDV nodes, and 3) RDV node level placement in the tree. Using this algorithm, an edge site administrator can adjust the initial number of RDV nodes to match a desired level of resilience and cost given an edge network site.

Distance	Ranking	RDV
3	1	00 * **
4	2	01 * **
5	3	10 * **
5	4	11 * **

TABLE I
EDGE ENDPOINT RDV RANKING TABLE - OPTIMIST APPROACH FOR RDV VID ASSIGNMENT

Algorithm 1 RDV Recommendation Algorithm

```
def estimate_rdv(total_endpts, desired_covg_pct):
  subtree_endpts = total_endpts / 2
  max_depth = get_tree_depth(total_endpts)
  starting_depth = 1
  rec_node_lvl = rec_estimate_rdv(subtree_endpts, total_endpts, starting_depth, max_depth, desired_covg_pct)
  return max_depth, pow(2, rec_node_lvl), rec_node_lvl

def rec_estimate_rdv(subtree_endpts, total_endpts, depth, max_depth, desired_covg_pct):

  if depth == max_depth then
    return depth - 1
  end if

  coverage_percent = (subtree_endpts/total_endpts) * 100
  if coverage_percent < desired_covg_pct then
    return depth
  end if

  subtree_endpts = subtree_endpts / 2
  depth += 1
  return rec_estimate_rdv(subtree_endpts, total_endpts, depth, max_depth, desired_covg_pct)
```

3) Network Bootstrapping: We use a centralized approach for edge endpoint ID-VID mapping and assume that all edge endpoints in an edge site are on the same network. Then, during network bootstrapping we map edge endpoint IDs to VIDs of leaf nodes in our binary tree (recall that only the leaf nodes in the tree represent edge endpoints). Thus, the VID of an edge endpoint is the *L*-bit long binary string along the path from the root to the corresponding leaf node. This ID-to-IP mapping represents a routing layer that enables edge endpoints' communication with RDV nodes.

[4] The number of levels in the virtual binary tree

[5] Here we measure cost by number of RDV nodes required for an edge site

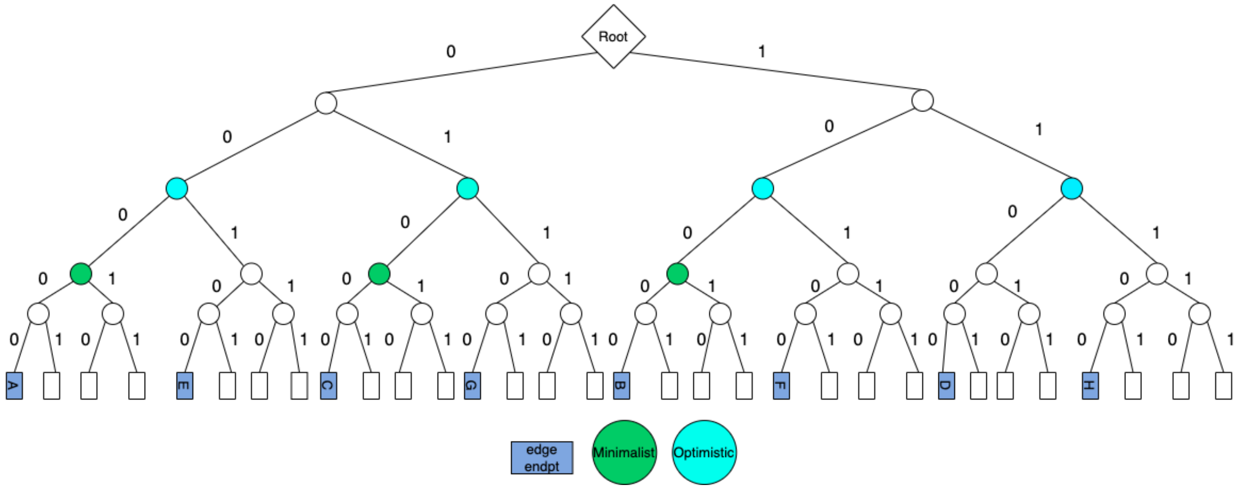


Fig. 2. The minimalist and optimist approaches to RDV node placement in the vid space - the white nodes (vertices) and white boxes represent the unused vid's

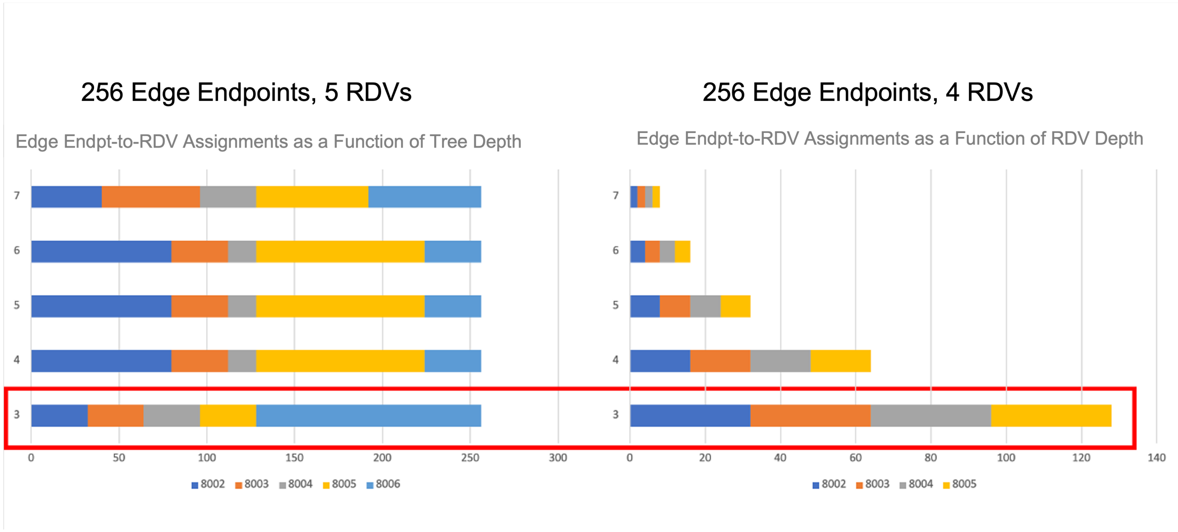


Fig. 3. Simulation results for minimalist (left) and optimist (right) RDV node placement approaches

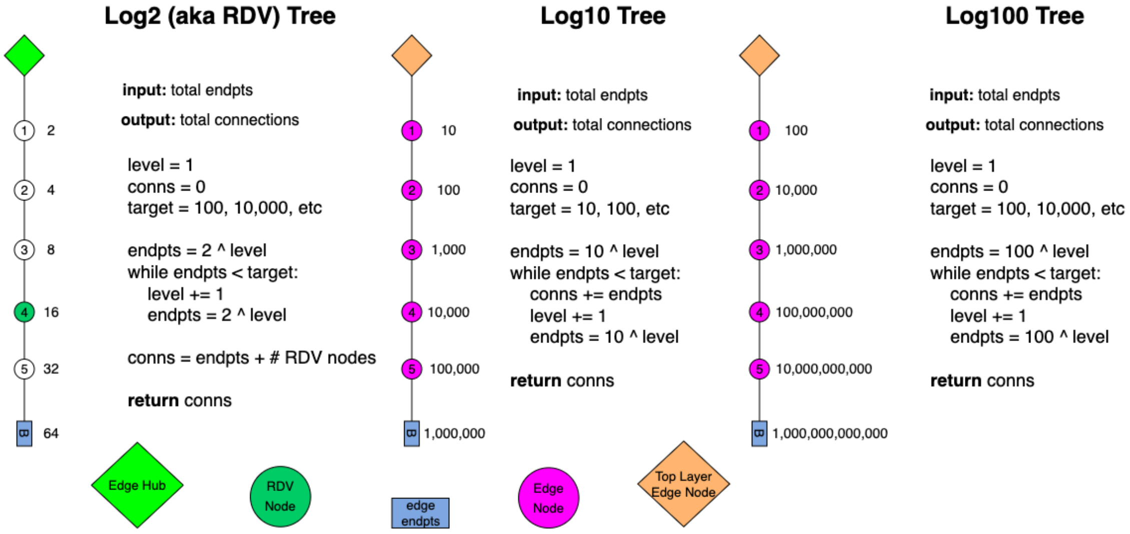
Algorithm 2 Binary Tree Depth Recommendation Algorithm

```
def get_tree_depth(total_endpts):
    tree_depth = 1
    current_depth_endpts = pow(2, tree_depth)
    while current_depth_endpts <= total_endpts do
        if current_depth_endpts == total_endpts then
            break
        end if

        tree_depth += 1
        current_depth_endpts = pow(2, tree_depth)
    end while
    return tree_depth
```

The network bootstrapping process is as follows: edge endpoints contact the edge hub to get their assigned VIDs and the list of (pre-assigned) VIDs of RDV nodes. Next, edge endpoints compute logical distances (γ) to each RDV node [19], [21] and build an RDV ranking table (see Table I). Then, the edge point locates its ranking table and selects the "closest" RDV node to register for receiving updates, pulling objects, and reporting status. In this scheme, the edge hub maintains the list of VIDs for all reachable RDV nodes in the network.

The above network bootstrapping process has the following two advantages: a) the edge hub only needs to maintain the ID/VID mapping for all assigned VIDs; b) the edge hub offloads the RDV node ranking computation to the edge endpoints. Hence, the network bootstrapping can easily scale to thousands of edge endpoints with minimal overhead attributed to the edge hub.



```
input: total endpts
output: total connections

level = 1
conns = 0
target = 100, 10,000, etc

endpts = 2 ^ level
while endpts < target:
    level += 1
    endpts = 2 ^ level

conns = endpts + # RDV nodes

return conns
```

```
input: total endpts
output: total connections

level = 1
conns = 0
target = 10, 100, etc

endpts = 10 ^ level
while endpts < target:
    conns += endpts
    level += 1
    endpts = 10 ^ level

return conns
```

```
input: total endpts
output: total connections

level = 1
conns = 0
target = 100, 10,000, etc

endpts = 100 ^ level
while endpts < target:
    conns += endpts
    level += 1
    endpts = 100 ^ level

return conns
```

Fig. 4. Proxy representations of tree flavors

III. Analysis/Evaluation

We have conducted several experiments to evaluate our proposed framework. More specifically, as illustrated in Figure 4, we compare our EdgeRDV method (i.e., log2 based tree) to other relay-based methods (i.e., log10, log20 and log100 trees). In this section, we describe three sets of experiments. In the first experiment, we compared the number of intermediate nodes for two tree flavors, and our methodology with various RDV node coverage. In the second experiment, we analyze failure adaptation by analyzing the number of messages required to update all edge endpoints during failure events. In the third experiment, we analyze the number of messages required to pull the latest configurations in a worst-case scenario (i.e., all edge endpoints previously lost the path to the cloud).

A. Experimental Setup

We have developed our customized, in-house simulator for EdgeRDV to enable an extensive edge endpoint simulation on large network topologies. We implemented the aforementioned architecture elements of EdgeRDV as RESTful modules. These modules can be readily deployed as containers on a variety of physical devices or virtual machines. In our experiments, we evaluate EdgeRDV by running its components as container processes on the same host (Docker Linux containers using the Ubuntu 20.04 base image). Note that we analyzed scale up to 667,712 edge endpoints. For the RDV node placement/selection/recommendation (II-B) analysis, unless otherwise noted, we used an RDV node coverage of 10%. By coverage we mean the number of edge endpoints that an RDV node can support.

B. Minimizing Intermediate Nodes

In tree-based edge endpoint deployments, the intermediate nodes form the middle layer responsible for transporting state and workloads to/from root nodes to leaf nodes (edge endpoints) – see Fig. 4. The larger is the number of intermediate nodes the higher is the number of single points of failure and the deployment cost for the edge site infrastructure. Therefore, minimizing the number of intermediate nodes is crucial to minimizing cost and increasing scale for the edge infrastructure.

In this experiment, we compared the number of intermediate nodes for three tree flavors and our method with various RDV node coverage (i.e., 10% and 20%). We used random sampling to select buckets for edges sites, from 5,000 up to 1,000,000 edge endpoints per site. Figure 5 shows that, depending on the chosen RDV node coverage, the RDV-based approach requires **one to three orders of magnitude** fewer intermediate nodes. Note that our method can adjust the number of intermediate nodes (aka RDV nodes) depending on the desired coverage per edge site. Therefore, our method provides a parameter to optimize the trade-off between maximizing the coverage per RDV node and increasing the resilience of the infrastructure associated with higher cost.

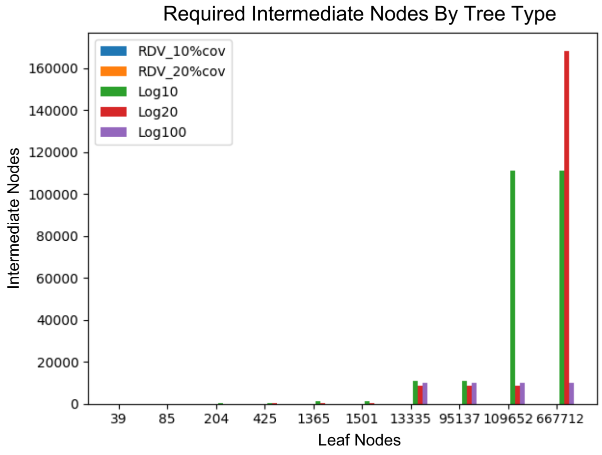


Fig. 5. A comparative analysis of intermediate nodes by tree type

C. Adapting to Failures

Relay methods such as Azure IoT propagate messages to edge endpoints in a relay fashion [22]. In this experiment, we analyzed failure adaptation through the number of messages required to update all edge endpoints of failure events (i.e., loss of connectivity at top layer nodes) in an RDV-based tree (i.e., RDV detection) and a log10 tree (i.e., relay detection). Figure 6 shows that, compared to a relay method, the RDV method scales gracefully with the number of edge endpoints. Specifically, the RDV method requires **up to 11% fewer messages** depending on the number of edge endpoints. The reduction in messages is related to the need for no more than two hops for failure notifications to get propagated to edge endpoints instead of n-hops[6] found in other relay methods such as Azure IoT. Furthermore, a failure signal at the root node in a relay structure could be delayed if any of the intermediate nodes, responsible for managing some edge endpoints, have also failed. In contrast, edge endpoints in an RDV-based network can rely on back-up RDV nodes (i.e., the next highest ranked RDV node) in the case that their primary RDV node fails.

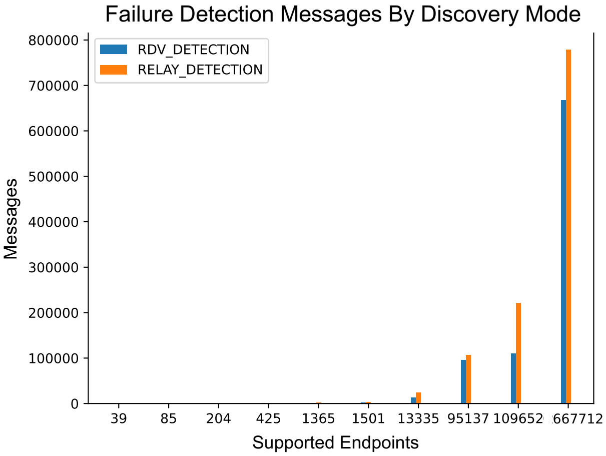


Fig. 6. Relay and RDV failure detection message costs

D. Configuration Updates

Loss of connectivity implies edge endpoints do not receive timely updates from the cloud hub. Thus, after failure recovery, edge endpoints must send messages to receive and apply the latest configurations. Depending on the tree structure, this failure recovery phase may challenge the available bandwidth within a given edge site if all edge endpoints and intermediate nodes access the cloud gateway (edge hub) at the same time. In this experiment, we analyze the number of messages required to pull the latest configurations in a worst-case scenario whereby all edge endpoints previously lost their path to the cloud. In particular, we conducted an experiment comparing the per edge endpoint messages in an RDV-based tree (i.e., RDV pull) and a log10 tree (i.e., relay pull). Figure 7 shows that the RDV method scales **constantly** with an increasing number of edge endpoints. This is because it takes no more than two hops for an edge endpoint to request and receive the latest configuration update(s). In contrast, the number of update messages grows linearly with the number of layers in a relay structure. Further, as a corollary, each intermediate layer in a relay structure introduces an increasingly larger flash crowd problem during configuration updates whereby numerous edge endpoints access an intermediate node at the same time.

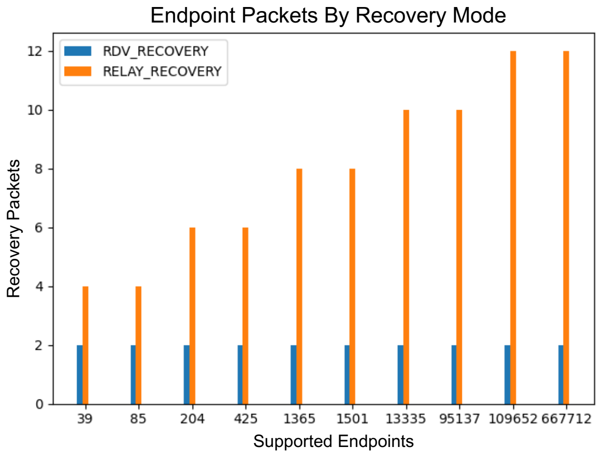


Fig. 7. Relay and RDV failure recovery message costs

IV. Discussion/Implications

Edge computing is intended to reduce network bandwidth consumption by processing and acting on data as close to its source as possible. Furthermore, it is imperative to manage edge workloads despite node and network failures. Our methodology demonstrates techniques of achieving these goals across thousands of sites representing millions of edge endpoints – state of the art methods only support thousands to tens of thousands of edge endpoints [23], [24]. In this section, we discuss the implications of this work on distributed edge systems.

The architecture presented here is intended to scale to thousands of edge sites, representing millions of edge endpoints, by implementing efficient techniques across each edge site. Figure 8 shows how we can horizontally scale our framework to support one million edge endpoints deployed across several edge sites with minimal overhead to the cloud management plane (cloud hub). This is due to the number of connections to the cloud hub only growing linearly with the number of edge sites; independently of the number of edge endpoints per edge site. Additionally, each site only downloads and caches application manifests that are relevant to capability-specific edge endpoints.

The practical challenges addressed by our methodology (Section II-B) further cement the rendezvous method as a general approach for exploring cost and resilience trade-offs

[6] n corresponds to the number of levels in the tree

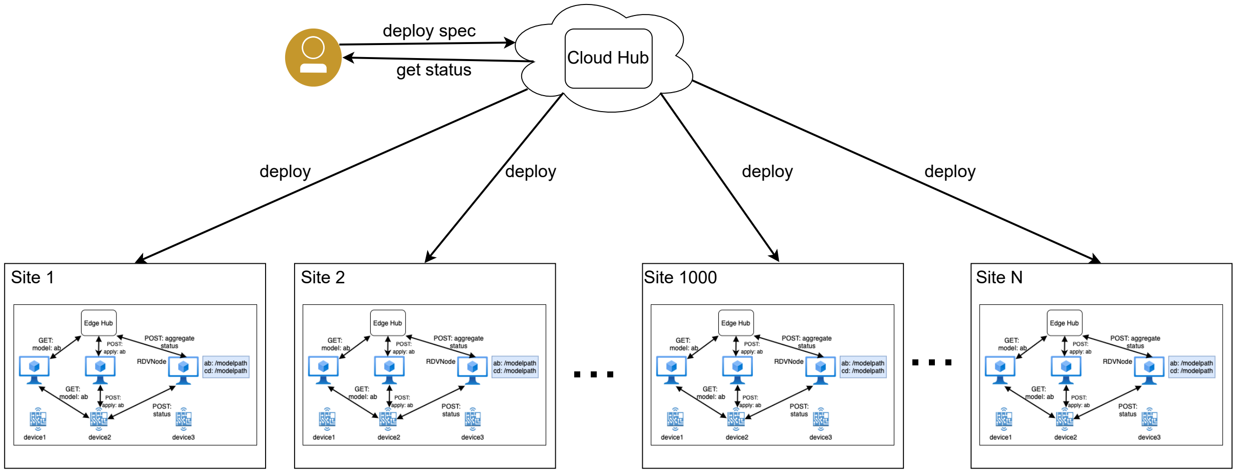


Fig. 8. Horizontal scaling EdgeRDV across N edge sites to support one million plus edge endpoints with *minimal* overhead to the cloud hub

in network deployments. In particular, we showcase how the rendezvous method is applicable for edge contexts where the network structure is known a priori. Unlike other tree structures, the binary tree presents properties that are easier to represent and exploit. This includes network slicing at each level to increase/decrease the resilience level by exactly 50%.

The core design principles that this work adheres to are: increasing scalability, reducing bandwidth consumption, and eliminating single points of failure. Our analysis indicates that the proposed techniques result in requiring significantly fewer intermediate nodes and dramatically reducing failure detection/recovery messages (Section III). In particular, these results showcase how our techniques would scale at different points in the application life cycle management workflow. That is, initial infrastructure bootstrapping (by providing appropriate resilience and cost adjustments), graceful failure detection (by providing multiple detection paths), and low latency failure recovery (by reducing intermediate layers). Further, the use of logical distances implies an inversely proportional relationship between edge endpoints and their assigned RDV nodes. This means that the higher the logical distance between an edge endpoint and an RDV node, the less impact a single RDV node's failure will have. A similar result is observed in [25]. Admittedly, this notion relies on latency between nodes, within an edge site, remaining negligible (i.e., not reflected in the logical distances). Note that a similar notion is difficult to replicate in other relay structures such as Azure IoT [15].

Robustness is often defined as the ability of a network to continue to function when it is subjected to failures. Our proposed architecture is resilient to failure. In the scenario of cloud hub failure, the edge hub will continue to deliver the latest objects or metadata it received to edge endpoints via the RDV nodes. In the case of failure of the edge hub, the RDV nodes will continue to deliver objects and metadata to edge endpoints. Lastly, the failure of a single RDV node does not create any overhead, as edge endpoints can easily switch to other replicas of the RDV nodes. Therefore, our architecture provides multi-layer resilience to failure.

Furthermore, our proposed solution provides some advantages over using CDN to deliver application workloads to constrained edge endpoints: a) **minimize data transfer**: in edge computing, edge applications need to report status back to the central hub to provide a global view of the edge infrastructure. In our approach, RDV nodes collect and aggregate the status from the edge endpoints, hence, minimize the amount of data transferred over the network when compared with CDN – where every edge endpoint would need to report a status back directly to the cloud hub; b) **fast response time**: our approach provides faster response time when compared to CDN - as the edge endpoints pull objects or metadata from RDV nodes located in the same network rather than from CDN edge servers several hops away, or in other networks; c) **network bandwidth minimization**: in our approach only a single edge hub receives metadata from the cloud hub independent of the number of edge endpoints associated with an edge site. In contrast, with CDN, every edge endpoint will need to pull content from the closest CDN edge server which may overload the edge server as the number of edge endpoints increase. Furthermore, the network bandwidth consumption will be directly proportional to the number of edge endpoints in an edge site. However, our solution can also be deployed in conjunction with a CDN infrastructure to further improve resilience; for example, a CDN can be used as the transport layer between the cloud hub and the edge hub.

V. Related Works

A. Managing Edge Operations in Resource-Constrained Environments

The challenge of managing data/workloads on resource-constrained edge endpoints has been explored across different contexts (e.g., community cellular networks [27], wildlife tracking [28], drone image analytics [29]). We describe lessons drawn from these other contexts to inform our methodology. The authors in [27] tackled the challenge of scalably managing

Platform	Module Management	Cloud Comms	Protocols	Device-to-device Comms?
Azure IoT Edge [15]	IoT Agent	IoT Hub	MQTT, AMQP	No
Google IoT Core [17]	IoT Core	IoT Core	MQTT, HTTP	No
AWS IoT Core [16]	IoT Core	IoT Core	MQTT, HTTPS, LORAWAN	Yes
IBM EAM [26]	Edge Agent	Edge Service	REST API	No

TABLE II
CAPABILITY COMPARISON FOR INDUSTRIAL IOT PLATFORMS FROM MAJOR CLOUD PROVIDERS

independent cellular networks. Community cellular networks (CCNs) are intended to be deployed for the people and run by the people. That is, they offer phone services and applications across multiple sites while still tapping into the resources such as phone numbers from incumbent Mobile Network Operators (MNOS). The key approach in [27] involved operating multiple CCNs under a single Community Cellular Manager (CCM) controller, which in turn is managed by MNOs for other aspects such as secure bootstrapping and resource usage reports. Like CCM, we aim to operate each edge site as a single entity that can deploy specific workloads and scale up depending on the number of supported nodes (i.e., RDV nodes, edge endpoints). Still, each edge site would be plugged into a central control plane (i.e., edge hub and cloud hubs) spanning multiple edge sites.

Another challenge in edge contexts is operating millions of edge endpoints, at scale, with resource and network bandwidth limitations. ZebraNet [28] is a classical example that involves tracking wildlife migration patterns using battery-powered collar tags on Zebras. One of the technical challenges addressed in ZebraNet involves communication and data collection from sporadically reachable and resource-constrained nodes. The ZebraNet architecture relied on peer-to-peer data swaps to retrieve data from unreachable nodes. That is, faced with limited storage and network bandwidth, nodes prioritize connections and data from more recently connected peers through dynamic ranking during data swap phases. In this context, we leverage a similar approach by ranking RDV nodes in terms of logical distance from a given edge endpoint. This stands in contrast to the "flood protocol" in ZebraNet where nodes waste network resources by broadcasting or flooding data packets to all nearby nodes.

B. Industrial IoT Management Platforms

Industrial IoT platforms are developed by major cloud providers [15]–[17], [26]. The platforms are widely deployed to support diverse use cases including transportation [17], smart homes [16], manufacturing [15], [26], etc. While the targeted use cases are different, they share the architectural goal of deploying and managing workloads on edge endpoints (Table II). Workload manifests (or metadata) are routinely deployed to specific edge endpoints or groups of edge endpoints based on labels or tags. We leverage a similar approach where manifests can be sent to specific edge sites from the cloud hub. We further extend this approach to each edge site. In other words, edge hubs and RDV nodes may only download relevant objects/workloads to reduce load on constrained storage and CPU resources.

Each industrial IoT deployment typically accesses cloud resources through a gateway or "top layer" node [22]. Thus, "lower layer" edge endpoints rely on the gateway for outbound requests to the cloud. In most implementations, the edge endpoints' logical connections are pooled into a single physical connection at the cloud-connected top layer node. This reduces the bandwidth overhead on top layer nodes and the cloud management endpoints. In our approach, a single physical connection is established by each site's edge hub to the cloud hub to reduce load on the latter. Note that, for existing platforms, the connection pooling is both a benefit and a forcing function of the chosen upstream protocols such as MQTT or AMQP [17], [30]. In contrast, we implement the architectural elements using HTTP because the edge endpoints do not communicate directly with the edge hub.

Industrial IoT workloads are deployed as "modules" or "models" on the edge endpoints [15], [26]. Among the major platforms, Azure IoT enables edge-endpoint-to-module and module-to-module communication, which enables disconnected operations for situations when cloud connectivity is temporarily lost [30]. Note that the platforms rarely enable edge-endpoint-to-edge-endpoint communication, which could reduce the number of edge endpoint requests to the cloud through the top layer node. Granted, AWS IoT is capable of edge-endpoint-to-edge-endpoint communication through a publish/subscribe (pub/sub) interface [31]. The pub/sub is brokered through its IoT Core (i.e., top layer) node. This approach faces two limitations. First, the pub/sub interface becomes a bottleneck as the number of edge endpoints grows. Secondly, although it serves as an important channel for edge-endpoint-to-edge-endpoint messages, the broker represents a single point of failure during disconnected operations. In contrast, our approach optionally distinguishes the capacities of edge endpoints and RDV nodes, which enables edge-endpoint-to-edge-endpoint communication with built-in redundancy to the top layer node.

As described above, industrial IoT networks operate with 'top layer' (or 'parent') and 'lower layer' (or 'child') edge endpoints [32], in line with the ANSI/ISA-95 standard [33]. This alludes to tree-like network structures. However, a layered tree structure, with necessary intermediate "parent" nodes, may not scale efficiently because the number of intermediate nodes grows *linearly* with the number of edge endpoints. In fact, Azure IoT recommends not scaling a network beyond five levels and no more than 100 edge endpoints per IoT node [23].

In contrast, we present an architecture whose number of intermediate nodes grows by a *logarithmic* factor with the number of edge endpoints to be supported. In fact, the number of intermediate nodes can be adjusted to a desired level of resilience for an arbitrary number of edge endpoints.

C. Content Delivery Networks (CDNs)

CDNs are the result of significant efforts in exploring content delivery at scale [4], [5]. CDN advances and optimizations are motivated by major challenges inherent to the Internet's original design and evolution such as inefficient routing protocols, requirements for increased scale, and unreliable networks. [5]. The overarching goal of CDNs is to reduce the number and latency of round-trip times (RTTs) between content origin servers and "edge" servers. We draw many lessons from decades of CDN experience in our architecture.

The original Akamai CDN aimed to solve the flash crowd problem, "in which request load overwhelms some aspect of the site's infrastructure, such as the front-end web server, network equipment, or allocated bandwidth" [4, p. 50]. Akamai devised a system to serve content from a variable number of servers closer to the network edge. We draw an analogy whereby thousands of edge endpoints simultaneously access cloud resources with requests. State-of-the-art IoT platforms have resolved this issue by bundling 'logical' edge endpoint connections into a single physical cloud connection [30]. However, as a network scales to millions of edge endpoints across thousands of edge sites, this multiplexing effect still embodies the flash crowd problem and a single point of failure, especially in edge sites with a single cloud gateway. Like CDNs, we avoid network overload by redirecting edge endpoint requests to redundant, and potentially more powerful, nodes (the equivalent of CDN edge servers). Going beyond the CDN approach, within each edge site, we present a layered architecture with a parameter to control the desired level of resilience to spikes and failures through multiple, ranked RDV nodes representing each edge endpoint.

CDN edge servers evolved from challenges related to hosting content to offering computation for running workloads closer to the end user [5]. Like CDNs, our goal is to reliably deliver state/workloads from model stores to potentially thousands of sites/servers. Additionally, we assume a level of control over the edge endpoints and networks where the workloads run. Here, we provide even more granular redundancy and lower latency at each site through RDV node replication. Similar to CDNs, our proposed methodology reduces the number of RTTs between the end users (i.e., edge endpoints) and origin servers (i.e., model stores) through effective, multilevel caching at the edge hubs and RDV nodes.

Another important component of CDNs is the mapping system that associates user requests to the closest live edge server [5]. The mapping system relies on a detailed view of the CDN network - enabled by granular monitoring of RTTs between CDN overlay networks, edge servers, and the broader Internet. In other words, the edge servers provide an important view into their loads, download times, and other metrics. Thus, the mapping system operates a fail-over system in case the closest edge server to a user is no longer available. Note that a mapping system requires a significant amount of network traffic for sustainable operations. In contrast to a mapping system, we move the fail-over mechanism to the edge endpoints. Specifically, the edge endpoints are individually tasked with ranking backup RDV nodes in case their primary RDV node fails.

VI. Conclusion

Edge computing is intended to reduce network latency by bringing applications as close to their data source(s) as possible - away from centralized cloud data centers. Managing application lifecycles across thousands of edge sites, supporting hundreds of thousands to millions of edge endpoints, poses many challenges –including disconnected operations, edge endpoint failures, etc. Existing hub-and-spoke frameworks attempt to address these shortcomings. However, these frameworks have scalability limitations and are still subject to single points of failure and significant network bandwidth consumption. To bridge this gap, we presented an alternative framework, EdgeRDV, that builds on the idea of RDV nodes. EdgeRDV is a scalable architecture comprised of a cloud-based management endpoint (cloud hub), a central controller for each edge site (edge hub), redundant gateways (RDV nodes), and edge endpoints. Our EdgeRDV methodology presents novel techniques and algorithms to optimize infrastructure costs and resilience settings.

Through simulations, we conducted preliminary experiments to evaluate EdgeRDV's scale and resilience in edges sites up to a 667K+ edge endpoints. Our experiments demonstrate that EdgeRDV requires one to three orders of magnitude fewer intermediate nodes, can adapt to failures with up to 11% fewer network messages compared to relay structures, and avoids the flash crowd problem in application configuration updates during failure recovery. We discuss how the techniques employed by EdgeRDV are broadly applicable to distributed edge settings, including scalable infrastructure bootstrapping, adjustable network resilience, and efficient resource usage. As a future work, we plan to implement EdgeRDV in hardware and showcase its benefits in terms of scalability, network bandwidth, data transfer, and RTT optimization.

Acknowledgment

The authors would like to thank Cheuk Lam, Paolo Dettori, Franco Stellari, Robert Filepp and the anonymous reviewers for their constructive feedback and interesting discussions during the manuscript's preparation.

References

[1] R. Filepp, M. L. Gordon, A. W. Bidwell, A. M. Wolf, F. C. Young, D. Tiemann, K. H. Appleman, and S. Meo, "Method for storing data in an interactive computer network," Aug. 15 1995, uS Patent 5,442,771.

[2] R. Filepp, M. L. Gordon, A. W. Bidwell, F. C. Young, A. M. Wolf, S. Meo, D. Tiemann, R. D. Cohen, M. Bellar, K. H. Appleman, L. Abrahams, and M. J. Silfen, "Reception system for an interactive computer network and method of operation," US Patent US5 347 632A, Sep., 1994.

[3] R. Filepp, M. L. Gordon, A. W. Bidwell, F. C. Young, A. M. Wolf, S. Meo, D. Tiemann, L. Abrahams, M. J. Silfen, A. R. Dalsass, F. M. Lee, and K. H. Appleman, "Interactive computer network and method of operation," US Patent US5 594 910A, Jan., 1997.

[4] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, Sep. 2002.

[5] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[6] Gartner, "What edge computing means for infrastructure and operations leaders," https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders, Oct. 2018.

[7] Forbes, "Digital devices took over our lives in 2020: Here's how to stay secure," https://www.forbes.com/sites/forbestechcouncil/2021/04/15/digital-devices-took-over-our-lives-in-2020-heres-how-to-stay-secure/?sh=43c7ff951ed2, Apr. 2021.

[8] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[9] IBM, "Ibm maximo visual inspection," https://www.ibm.com/products/maximo/visual-inspection, Feb. 2023.

[10] L. Edge, "Open horizon," https://www.lfedge.org/projects/openhorizon/, Feb. 2023.

[11] Forbes, "Ibm research rolls out a comprehensive ai and platform-based edge research strategy anchored by enterprise use cases and partnerships," https://www.forbes.com/sites/moorinsights/2022/08/08/ibm-research-rolls-out-a-comprehensive-ai-and-ml-edge-research-strategy-anchored-by-enterprise-partnerships-and-use-cases/?sh=6c2fd8da13ed, Aug. 2022.

[12] C. fil A, "Edge computing at chick-fil-a," https://medium.com/chick-fil-atech/edge-computing-at-chick-fil-a-2621f4b5a969, Jul. 2018.

[13] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, Aug. 2014.

[14] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-Defined Industrial Internet of Things in the Context of Industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, Oct. 2016.

[15] A. Microsoft, "Azure IoT – Internet of Things Platform — Microsoft Azure," https://azure.microsoft.com/en-us/solutions/iot/, Jan. 2023.

[16] A. W. Services, "AWS IoT Core," https://aws.amazon.com/iot-core/, Jan. 2023.

[17] Google, "Cloud IoT Core," https://cloud.google.com/iot-core, Jan. 2023.

[18] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers*. Springer, 2002, pp. 53–65.

[19] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. New York, NY, USA: Association for Computing Machinery, Aug. 2003, pp. 381–394.

[20] S. Jain, Y. Chen, Z.-L. Zhang, and S. Jain, "VIRO: A scalable, robust and namespace independent virtual Id routing for future networks," in *2011 Proceedings IEEE INFOCOM*, Apr. 2011, pp. 2381–2389.

[21] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer, 2002, pp. 53–65.

[22] Microsoft, "How to create nested Azure IoT Edge device hierarchies," https://learn.microsoft.com/en-us/azure/iot-edge/how-to-connect-downstream-iot-edge-device, Jan. 2023.

[23] ——, "Limits and restrictions - Azure IoT Edge," https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-limits-and-restrictions, Jan. 2023.

[24] Google, "Quotas and limits — Cloud IoT Core Documentation — Google Cloud," https://cloud.google.com/iot/quotas, Feb. 2023.

[25] B. Dumba, G. Sun, H. Mekky, and Z.-L. Zhang, "Experience in Implementing & Deploying a Non-IP Routing Protocol VIRO in GENI," in *2014 IEEE 22nd International Conference on Network Protocols*, Oct. 2014, pp. 533–539.

[26] IBM, "IBM Edge Application Manager - Overview," https://www.ibm.com/cloud/edge-application-manager, Jul. 2022.

[27] S. Hasan, M. C. Barela, M. Johnson, E. Brewer, and K. Heimerl, "Scaling Community Cellular Networks with CommunityCellularManager," in *16th ${$USENIX$}$ Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 735–750.

[28] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X. New York, NY, USA: Association for Computing Machinery, Oct. 2002, pp. 96–107.

[29] S. Jha, Y. Li, S. Noghabi, V. Ranganathan, P. Kumar, A. Nelson, M. Toelle, S. Sinha, R. Chandra, and A. Badam, "Visage: Enabling timely analytics for drone imagery," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 789–803.

[30] Microsoft, "Learn how the runtime manages devices - Azure IoT Edge," https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime, Jan. 2023.

[31] A. W. Services, "Connecting devices to AWS IoT - AWS IoT Core," https://docs.aws.amazon.com/iot/latest/developerguide/iot-connect-devices.html, Jan. 2023.

[32] Microsoft, "Tutorial - Create a hierarchy of IoT Edge devices - Azure IoT Edge for Linux on Windows," https://learn.microsoft.com/en-us/azure/iot-edge/tutorial-nested-iot-edge-for-linux-on-windows, Jan. 2023.

[33] Wikipedia, "ANSI/ISA-95," *Wikipedia*, Jul. 2021.