FogROS2-SGC: A Secure and Global Connectivity Framework For Latency Sensitive Cloud Robotics

Eric Chen

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-70
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-70.html

May 9, 2024

FogROS2-SGC: A Secure and Global Connectivity Framework For Latency Sensitive Cloud Robotics

by

Kaiyuan Chen

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science, Plan II

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Kubiatowicz, Advisor
Professor Ken Goldberg, Reader

Spring 2024

The thesis of Kaiyuan Chen, titled **FogROS2-SGC: A Secure and Global Connectivity Framework For Latency Sensitive Cloud Robotics**, is approved:

Advisor _____ Date _____

Reader _____ Date _____

_____ Date _____

University of California, Berkeley

Contents

Abstract

FogROS2-SGC: A Secure and Global Connectivity Framework For Latency Sensitive Cloud Robotics

by

Kaiyuan Chen

Connectivity between robots and cloud machines is one of the most fundamental components in Cloud Robotics. We study the network infrastructure of FogROS2, an open-source Cloud Robotics platform, and propose FogROS2-SGC, a network extension of FogROS2 that can effectively connect robot systems across different physical locations, networks, and Data Distribution Services (DDS). FogROS2-SGC uses globally unique and location-independent identifiers to identify robots and cloud services. It securely and efficiently connects them and routes data between robotics components around the globe with a peer-to-peer network (Global Data Plane). FogROS2-SGC is agnostic to the ROS2 distribution and configuration, is compatible with non-ROS2 software, and seamlessly extends existing ROS2 applications without any code modification. Since memory copy and synchronization operations are expensive for memory-constrained robots, the implementation of FogROS2-SGC processes can route data without performing unnecessary copies (also known as "zero copy").

We consider latency critical cloud robotics applications that varying network conditions can cause instability and collisions. We observe such failure can be minimized in the almost universal case where there are multiple sources available for cloud servers. We extend FogROS2-SGC routing infrastructure by enabling multiple cloud robotics services to connect as the same globally unique identifiers. In the presence of multiple identical services, FogROS2-SGC dynamically identifies and transitions ("anycast") to the optimal service deployment that meets latency requirements, thus empowering robots with limited on-board computing capacity to safely and efficiently navigate dynamic, human-dense environments. The anycast is achieved through managing a state machine-based scheduler that dynamically monitors the application latency and adjusts the routing states.

We evaluate FogROS2-SGC with various simulated Cloud Robotics benchmarks (visual SLAM, motion planning, grasp planning). We show FogROS2-SGC with one connectivity case study with 4 robots and compute nodes that are 3600 km apart, and two latency sensitive case studies on collision avoidance and target tracking.

Chapter 1

Introduction

A prevailing trend indicates an increasing demand to employ resource-intensive models and algorithms for accurate robotics perception and control: popular visual perception modules include Segment Anything (SAM) [43], semantic VSLAM [77], Neural Radiance Fields (NeRF) [74], and Language Embedded Radiance Fields (LERF) [61]; intelligent motion control modules include learning hand-eye coordination with grasping [45], Model Predictive Path Integral (MPPI) [75]. The complexity of large or foundational models necessitates more than just the robot onboard computing capabilities.

Cloud [41] or Fog Robotics [29] has been proposed to utilize centralized cloud or distributed edge resources to reduce latency, enhance performance, and facilitate real-time processing and decision-making in robotic systems [22]. Cloud Robotics ([29, 34, 68]) enables robots to access external computing resources for visual perception ([62, 43, 77, 74, 42]), reinforcement learning-based intelligent motion control ([75, 17]), grasp and motion planning [68] [34], visual servoing [71], and human-robot interaction [28].

ROS2 is the defacto standard for building robotics applications. It modularizes a robotics application into *nodes*, and connects the nodes into a graph. Nodes communicate with each other through a publish-subscribe (pub/sub) system, where publisher nodes send messages to *topics*, and nodes subscribed to these topics receive these messages.

FogROS2 [35] —now an official part of the ROS2 ecosystem—outsources heavy computing tasks to on-demand hardware resources and accelerators, such as GPU, TPU, ASIC, FPGA, and high performance CPU servers. We recognize the importance of connectivity between robots and cloud machines in Cloud Robotics, and study the network infrastructure based on FogROS2 with a focus on **Global Connectivity**, **Security**, and **Support for Latency Sensitive Robotics Applications**.

1.1 Cloud Robotics Connectivity

As robots are increasingly deployed worldwide, they require mechanisms to efficiently, reliably, and securely communicate with other robots, sensors, computers, and the cloud. The applications

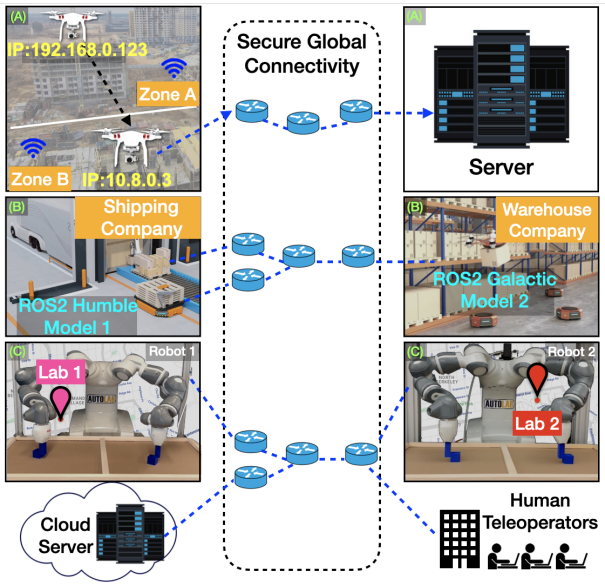


Figure 1.1: FogROS2-SGC enables Secure Global Connectivity for robots, allowing robots to communicate with other robots, computers, and the cloud through a standard ROS2 interface. With FogROS2-SGC, (A) drones navigating large construction sites can seamlessly communicate, even when their IP addresses are constantly changing due to switching Wi-Fi and cellular networks; (B) shipping and stocking robots from different corporations can securely share only the required topics necessary to facilitate the transfer of goods at a warehouse; and (C) globally distributed robots can participate in fleet learning. In experiments, we demonstrate FogROS2-SGC on Fleet-DAgger [31], a fleet learning algorithm, with 4 robot arms operating simultaneously in different locations.

are broad, from mobile robots with changing IP addresses due to traveling through different networks, to a fleet of globally distributed robots learning collaboratively. Robots connecting to the cloud, a nearby computer on a different network, or a robot halfway around the world introduce additional challenges: (1) Robots that are accessible to other systems on the public internet may be vulnerable to unauthorized connections and data breaches. (2) The heterogeneity of interconnected devices, communication protocols, and configurations causes incompatibilities that hinder integration and operation. (3) The changing network topology of mobile robots and Unmanned Aerial Vehicles (UAVs) challenges their ability to stay connected. To illustrate some of these challenges (Fig. 1.1), consider:

(A) Security and inspection drones Drones navigate a construction site and stream data to a central station that updates a dynamically-changing SLAM map [66]. As drones fly through different cellular and Wi-Fi networks, their IP addresses change, but they should remain securely connected.

(B) Coordinating heterogeneous mobile robots in a warehouse Robots belonging to different companies (e.g., shipping vs. warehouse) and of different makes and models hand off items between container and warehouse. Each robot has unique software packages and versions (e.g., operating systems and network protocols) and must communicate, but only a few selected topics are necessary for the handoff.

(C) Distributed fleet learning Robot arms at different locations pool their data and collectively

update a shared control policy. Robots unable to make progress can fall back on remote human teleoperators, using algorithms such as Fleet-DAgger [31].

To address these challenges, we present FogROS2-SGC (Secure Global Connectivity), an extension of FogROS2 that securely and reliably connects robots across different software components, network protocols, and physical locations. FogROS2-SGC enables disjoint ROS2 networks to connect to ROS2 topic interfaces named with *globally-unique* and *location-independent* identifiers. The robots using FogROS2-SGC can roam freely while staying connected because the identifiers are constant. They are 256-bit strings that are secure and anonymous to unauthorized attackers by construction —a brute-force attack would have to find a match among 10^{77} possibilities (a value close to the number of protons in the observable universe[1]). FogROS2-SGC adopts a *security-first* routing design, where only authenticated parties can connect to the robot and establish secure communication. In contrast to prior work such as SROS2 [52] and FogROS2 [35], FogROS2-SGC does not require merging distributed ROS2 networks, allowing robots to keep their ROS2 networks private and expose public topics only if explicitly configured. Providing fine-grain isolation and access control reduces the attack surface and enhances scalability.

FogROS2-SGC seamlessly integrates with ROS2 applications without code modifications via an SGC proxy. Its implementation and security policy configuration are agnostic to ROS2 distributions and their network transport middleware vendors. FogROS2-SGC is also compatible with non-ROS2 programs that interact with ROS2 components and can provide secure global connectivity to non-cloud servers and computers. Furthermore, since memory copy and synchronization operations are expensive for memory-constrained robots, the implementation of FogROS2-SGC processes can route data without performing unnecessary copies (also known as "zero copy").

1.2 Latency Sensitive Cloud Robotics

Latency-sensitive or safety-critical tasks in cloud and fog robotics may be subject to potential network failures and congestion. We recognize typical cloud robotics service providers usually provide multiple cloud services at different locations and data centers. Fig. 1.2 shows a use case of FogROS2-SGC that enables robots to connect to distributed robotic services with location-independent deployment of latency sensitive cloud robotics applications. It enables robust operation of latency sensitive applications, such as tracking or collision detection, by connecting robots to service deployments that satisfy these bounds. We present an extension to FogROS2-SGC that use these independent cloud compute servers and providers, and enable reliable latency performances by dynamically selecting the optimal service out of all available servers.

In ROS2 [48], robotics applications are built in a *location-independent* way: heterogeneous robots and modular services[2] publish to and subscribe from (pub/sub) each other as if they are running on the same machine. However, completely adhering to the ROS2 multiple-party pub-/sub communication paradigm in fog robotics falls short on the following aspects: (1) Mirrored

[1]The Eddington Number [26] (N_{Edd}) is currently estimated to be 10^{80}.

[2]*Service* refers to generic robotics application instead of the specific ROS2 service communication model. FogROS2-SGC supports both publish-subscribe and service communication models in ROS2.

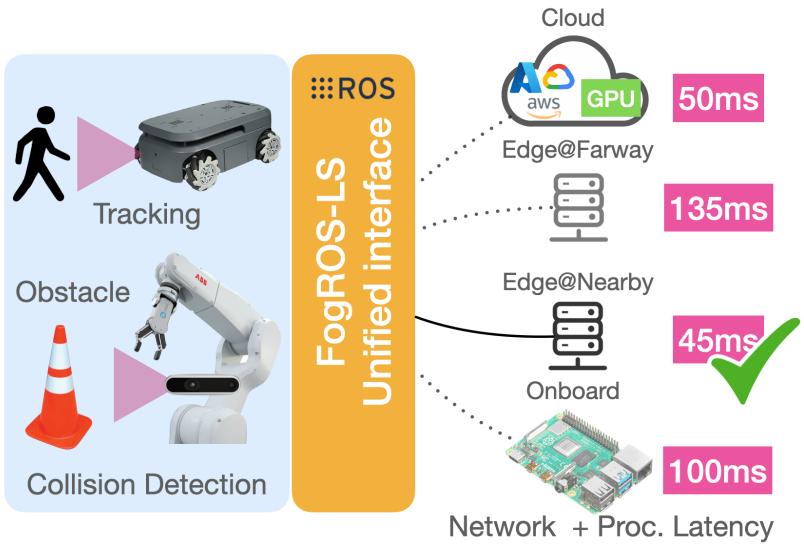


Figure 1.2: **A Sample Use Case of FogROS2-SGC** FogROS2-SGC enables the location-independent deployment of fog robotics applications, allowing robots to connect with distributed robotic services with a unified ROS2 interface. It enables robust operation of latency sensitive applications, such as tracking or collision detection, by connecting robots to service deployments that satisfy these bounds.

robotics services can be distributed to heterogeneous geographic locations and network domains. The framework needs to globally and securely discover and connect robots with those service deployments, while differentiating services hosted by other users or tasks. (2) The pub/sub paradigm leads to the request being published to all the deployments, leading to more network congestion and failure. While still adhering to ROS2 interfaces, a robot should select the one deployment that fulfills the application-specific latency bound. (3) Deployment selection should be adaptive to fluctuating application latency caused by varying network latency and hardware resource utilization, and network failures

We introduce a latency-aware extension to FogROS2-SGC that enables launching multiple instances of robotics task servers across different geographical and network domains, all identifiable by a location-independent identifier unique to the task. Robots use this identifier for global discovery and selection of service deployments based on dynamic ROS2 application latency analysis. FogROS2-SGC enforces an *anycast* invariant [14], in which each robot subscribes to exactly one service deployment. When failures or latency variations occur, FogROS2-SGC seamlessly switches to an alternative deployment that meets the latency constraints, while adhering to the anycast invariance.

1.3 System Assumptions

We consider a number of robots and cloud services at different subnetworks. Some of the subnetworks can not be directly accessible from the outside, such as local area networks behind Network

Address Translation (NAT). NAT allows multiple robots to share the same IP, but the translation is dynamic and ROS2 nodes outside cannot directly access the robots. For any pair between robot and cloud service, there is a direct and trusted path that is connected through FogROS2-SGC switches.

1.3.1 Security Model

FogROS2-SGC is secure against the following security threats:

- The robots communicate across wide-area networks with untrusted infrastructure. FogROS2-SGC guarantees that no unauthorized attacker can eavesdrop or tamper with ROS2 messages. Authorization is identified by user-configured cryptographic keys. The FogROS2-SGC prevents attackers from accessing any content in ROS2 messages and differentiates authentic robots from spoofing attackers.
- Authorized participants can deterministically derive global identifiable addresses with ROS2 topic information and cryptographic secrets. Attackers cannot reverse any information used to recover addresses or topics. FogROS2-SGC prevent attackers that know part of the ROS2 topic information from deducing the global address. For example, the attackers who know the topic name and type information cannot guess the address, because they lack the author information and security credentials of the ROS2 node.
- FogROS2-SGC connects robots without merging distributed ROS2 networks. Every robot can have an arbitrary number of private ROS2 topics and only public interfaces are shared with other authorized ROS2 networks. Other ROS2 nodes interact with these public interfaces just as they interact with a local ROS2 topic. This protects the privacy of the robot and prevents unintended messages from being shared with other disjoint networks of the system. For example, in scenario (B), a delivering robot from one company and receiving robot from another may have some proprietary topics that are kept private from each other. FogROS2-SGC isolates the topics private to each robot.

1.3.2 Features

FogROS2-SGC extends FogROS2 to address the Secure Global Connectivity (SGC) problem of securely and reliably connecting globally distributed robots, sensors, computers, and the cloud. We enumerate 10 new features to differentiate from related libraries and alternative approaches.

1. **No Application Modification** Unmodified ROS2 applications work with FogROS2-SGC and operate as though all modules reside on a single computer.
2. **Globally identifiable addresses** FogROS2-SGC enables a scalable number of ROS2 networks to publish a subset of ROS2 topics to other disjoint ROS2 networks around the globe. For example, in scenario (C) from Fig. 1.1, the robot arms are located at different geographic

locations with different local ROS2 networks. FogROS2-SGC allows remote human teleoperators to operate the robot arms as if the arms are connected to local networks. FogROS2-SGC also allows disjoint robots to publish to the same local ROS2 topics with globally unique and identifiable addresses.

3. **Location Independence Connectivity** The robotics services can be deployed at different geographic locations and network domains. FogROS2-SGC interconnects them with robots as if they were all on the same computer.

4. **Seamless and resilient connectivity to network dynamism** FogROS2-SGC adapts to the dynamic network behaviors of drones and mobile robots. FogROS2-SGC does not rely on static IP addresses to identify the robots because such addresses are usually bound to a physical location. Adding or reconnecting to robots should not restart the ROS2 node entirely, as this causes service interruptions and failures. FogROS2-SGC adapts its service selection to fluctuating network conditions and application latency.

5. **Efficient message routing** ROS2 messages are buffered in memory to be processed by FogROS2-SGC. Because robots often have memory and compute resource constraints, FogROS2-SGC is memory-efficient by reducing unnecessary message copying and memory synchronization. Experiments suggest that FogROS2-SGC reduces the network latency of a cloud-based grasp planning application by 9.42$\times$ compared to unsecured rosduct [8]-rosbridge [23].

6. **DDS-agnostic compatibility** ROS2 adopts the Data Distribution Service (DDS) as its underlying network transport middleware to marshal, unmarshal, and exchange messages. ROS2 supports different DDS implementations, such as CycloneDDS [16], FastDDS [27], and RTI Connext [3]. However, a warehouse in scenario (B) may have robots running different versions of ROS2 and DDS. FogROS2-SGC is DDS-agnostic by leveraging ROS2 abstractions and not using any DDS-specific interfaces.

1.4 Contribution

We claim the following four contributions:

1. FogROS2-SGC, an extension of FogROS2 that connects disjoint ROS2 networks by assigning public ROS2 topics with globally-unique and location-independent identifiers.

2. a location-independent routing framework that discovers and connects with ROS2 topics across different network domains

3. an Anycast-based communication extension for latency-sensitive cloud robotics applications

4. A Rust implementation of FogROS2-SGC that uses zero-copy message processing and asynchronous network operations for robots with memory and compute constraints.

We also note that the paper facilities the deployment of many robotics applications, such as vSLAM [66], motion planning [33], FleetDAgger [31], Lifelong LERF [61], grasp planning [49]. The case studies are merely to show the generality, real life applicability and performance of the proposed FogROS2-SGC. We do **not** claim the contribution of designing or implementing these algorithms.

1.5 Thesis Organization

This thesis is organized in the following way: we cover the background of ROS2, FogROS2 and other realted work in Cloud Robotics in Chapter 2, the design of FogROS2-SGC in Chapter 3, the latency sensitive extension of FogROS2-SGC in Chatper 4, and Cloud Robotics case studies in Chapter 5.

Chapter 2

Background and Related Work

2.1 Related Work

James Kaufner introduced the term 'Cloud Robotics' in 2010 [41]. Cloud and fog computing have been applied to robotic tasks such as grasp planning (Tian et al. [70], Kehoe et al. [39], and Li et al. [46]), parallelized Monte-Carlo grasp perturbation sampling (Kehoe et al. [37, 38, 40]), and motion planning (Lam et al. [44]). Chen et al. [21] and Ichnowski et al. [35]. propose frameworks for offloading computation to resources on the edge or cloud, while Ichnowski et al. [34] and Anand et al. [12] present systems that leverage serverless computing [53]. Modern computing paradigms have enabled new applications such as multi-robot interactive fleet learning (Swamy et al. [67], Hoque et al. [31]) and remote sharing of robot systems (Tanwani et al. [69], Bauer et al. [15]).

Robot Operating System (ROS) 2 [48], the successor of ROS, is the de-facto standard for developing robotics applications due to its broad availability and adaptability. In ROS 2, computational modules are abstracted into *nodes*, and they communicate with each other using a multi-party publish/subscribe paradigm through *topics*. All nodes subscribing to the same topics receive data from other nodes that publish them. Over the past decades, various attempts have been made to enable robotics applications in ROS or ROS2 to leverage Cloud or Fog computational resources. Rapyuta Mohanarajah et al. [54] is a proprietary platform for centralized management and deployment of ROS application pipelines.

Remote interactions between robots and the cloud raise security, compatibility, and connectivity challenges for robots. Virtual Private Networks (VPNs) are the most common approach for establishing secure communication between robots and the cloud for both ROS and ROS2 (e.g., Lim et al. [47]). Since establishing a VPN link between a robot and the cloud is a complex process [30], FogROS [21] and FogROS2 [35] automate the certificate generation and VPN setup. SROS2 [52] is an alternative approach to securing ROS2 communication that enforces access control of ROS2 topics. However, it requires DDS-dependent discovery mechanisms to ensure connectivity. Discovery mechanisms for DDS (such as the discovery server for FastDDS [27] and the RTI routing service for RTI Connext [3]) are vendor-specific and not compatible with other

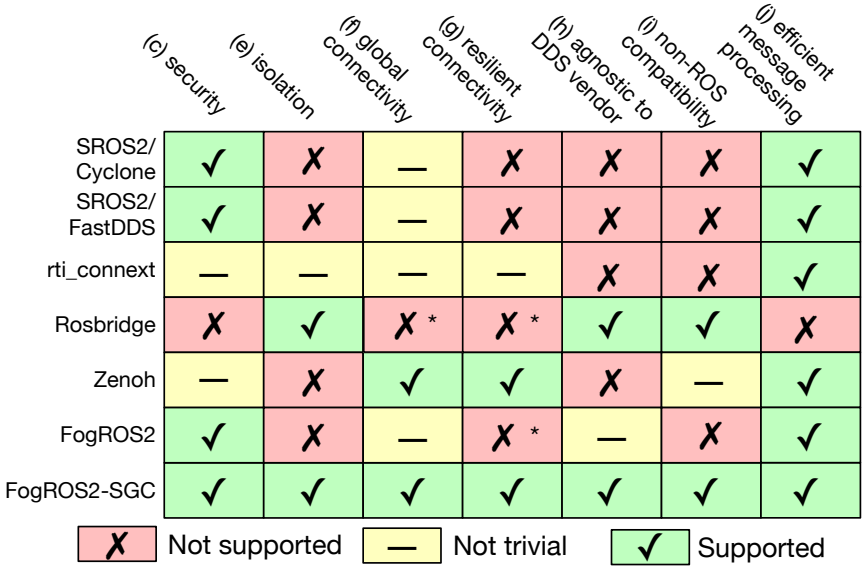


Figure 2.1: **Comparison of FogROS2-SGC with other distributed ROS2 systems.** In this table, we compare the feature support of different distributed ROS2 systems with the features. Some features can be supported but require non-trivial effort beyond changing the configuration. For example, both the routing service in rti_connext and discovery server in FastDDS/SROS2 support global connectivity but require manually modifying routing rules or setting up a point-to-point VPN when a new node joins [5]. Rosbridge and FogROS2 support only unidirectional global and resilient connectivity (marked with *), meaning that one side of the communication must have a fixed IP. In contrast, the identifier-based routing of FogROS2-SGC allows either side to have a dynamic IP address.

DDS implementations. Zenoh for ROS2 [11] is integrated with CycloneDDS to enhance peer-to-peer connectivity, but it is not compatible with other DDS implementations. ROS Remote [60] by Pereira et al. and MSA [76] by Xu et al. propose alternative protocols to unify cloud-robot communication. However, alternative protocols require modifications to ROS applications and are not compatible with ROS2. Finally, rosbridge [23] proposed by Crick et al. is widely adopted by both ROS1 and ROS2 to allow non-ROS software to interact with ROS2 nodes. It can also be used to bridge two non-compatible and remote ROS applications when used in conjunction with rosduct [8]. However, rosduct and rosbridge have significant message latency when the message size is large (e.g., images). A summary of how FogROS2-SGC differs from related work can be found in Fig. 2.1.

2.2 FogROS2

Chen et al. [21] propose FogROS, a cloud robotics framework that offloads ROS applications to the public cloud. Ichnowski et al. [35] and Chen et al. [20] extend FogROS with ten major features, including ROS2 support and major cloud service providers. FogROS2-SGC (Secure and Global Connectivity) [19][18] connects distributed ROS2 nodes through a global peer-to-peer network. In this work, we build upon the FogROS design philosophy, allowing for offloading and connect-

ing robots with ROS2 applications without any code modifications. Furthermore, we extend our study to enabling latency sensitive fog robotics applications. On the technique side, FogROS2 uses Virtual Private Network (VPN) optimized for single cloud and robot, and FogROS2-SGC lacks flexible network routing management. In contrast, our approach in FogROS2-SGC leverages the unique communication paradigm Anycast. This allows robots the flexibility to connect to one of several location-independent services and dynamically switch to meet application latency constraints.

Chapter 3

Secure and Global Connectivity

We propose FogROS2-SGC, a network extension of FogROS2 that can effectively connect robot systems across different physical locations, networks, and Data Distribution Services (DDS). See Fig. 4.1 for the system architecture. FogROS2-SGC sends messages via a globally unique identifier (Sec. 3.1). This identifier is unique to a robot and topic pair; thus, it can be used for sending and receiving messages regardless of robot location or network address (Sec. 4.1). The identifier is secure and communication is encrypted, meaning only authorized robots and nodes can access messages from its referenced topic (Sec. 3.3). To implement routing based on the identifier, FogROS2-SGC consists of two main software components—(1) a router (Sec. 4.1 and 3.3), responsible for securely routing messages between other routers and nodes, and (2) a proxy (Sec 3.4), that converts between ROS2 messages and the secure routers. As robots can be compute and memory-constrained, FogROS2-SGC provides a compute and memory-efficient implementation (Sec. 3.5).

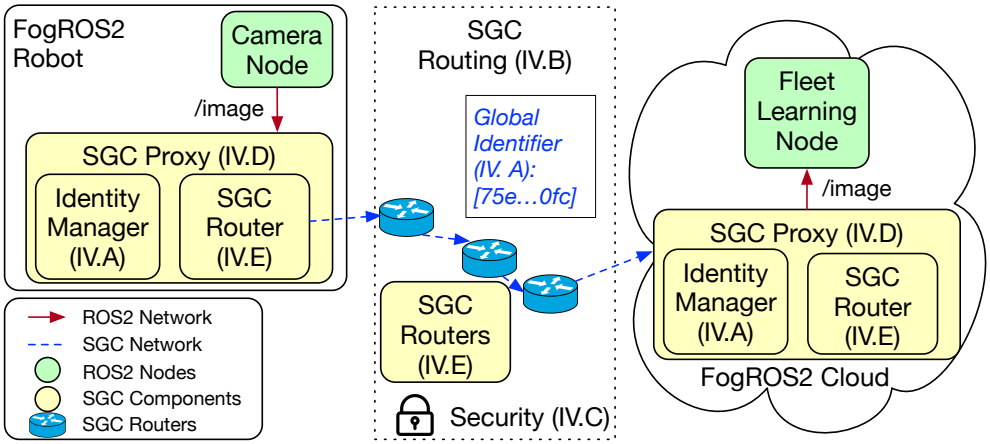


Figure 3.1: **System overview of FogROS2-SGC's architecture** showing a connection between a robot camera stream (on the ROS2 topic `/image`) and the cloud. The FogROS2-SGC assigns the ROS2 topic `/image` an anonymous, globally-unique and location-independent 256-bit identifier `[75e...0fc]` (truncated for brevity). The messages between identifiers are securely routed with the SGC router.

3.1 Global Addressability

Maintaining a globally unique identifier enables the identification of a specific robotic component across subnetworks. FogROS2-SGC uses ROS2 topics as the minimal granularity for the global identifier because a topic is an interface to ROS2 nodes, and a ROS2 node can publish or subscribe to multiple ROS2 topics at the same time. For example, a ROS2 vSLAM node in openVSLAM [66] has four ROS2 topics for camera information, video streaming, output localization, and mapping information. These ROS2 topics expose standardized interfaces with fixed message types. Users can limit the exposure of the ROS2 network by allowing only parts of the interface to be public. Partitioning public and private interfaces also enhances privacy and isolation, prevents unintended message exchanges, and reduces communication overhead.

The identifier is designed to be unique, deterministic, and location-independent. To avoid name collisions, every identifier has 256 binary bits, leading to 2^{256} possible identifiers. Instead of letting users decide, all identifiers are cryptographically derived from the metadata of the ROS2 topics by an identifier manager in the SGC proxy. The SGC proxy collects metadata such as the ROS2 node's name, author, maintainer, interface, and description from standard ROS2 interface and user configuration file. The metadata also has a unique string in case the user needs to deploy the same topic at different locations. Every topic has an associated security certificate in X.509 [57] to verify the identity of those who want to publish or subscribe to the network. All the metadata is serialized and converted into a 256-bit string using SHA-256 [4], a widely used cryptographic hashing algorithm that maps arbitrary lengths of text to almost-unique 256-bit binary strings.

Security Analysis: The hashed string is suitable for use as the globally unique identifier for the following reasons: (1) **Deterministic:** The hash is deterministic so that every party holding the same metadata can derive the same hash value and thus the same global identifier. (2) **One-way:** SHA-256 is a one-way function, so the attacker cannot deduce or reverse the original metadata from the 256-bit identifier. (3) **Avalanche effect:** A small change to the original metadata leads to a new hash value that appears unrelated to the original hash value. (4) **Large namespace:** There are 2^{256} possible identifiers and it has been proved to be computationally intractable to find two messages with the same hash. Verification of these guarantees can be found in Appel [13].

3.2 Location-Independent Routing

Although having all identifiers in the same globally-flat namespace protects the privacy of the node's identity information and physical location, the identifiers do not carry any routing information. Flipping a bit in the identifier may lead to a drastic change in its physical location, or from existent to nonexistent. Therefore, securely routing messages between flat identifiers is a challenging problem. To solve this problem, FogROS2-SGC consolidates and extends the Global Data Plane (GDP) [55], a peer-to-peer network that routes messages between location-independent identifiers. The routers are set up by the user and peer-wise connected into a routing graph; robots do not need to know other robots' addresses as long as there is a connected routing path. The routers can be any machine that has network and general compute capabilities, such as an edge computer

or a cloud server. Every router stores the mapping between the identifiers and the corresponding routing information of the identifiers in the Routing Information Base (RIB).

A joining robot or router broadcasts an *advertisement* packet that announces the existence of the identifier and the routing information to the robot. The packet format is aligned with other FogROS2-SGC packets in Fig. 3.3. Other routers store the routing information in RIB and broadcast the advertisement packet. Routing is achieved by looking up the destination routing information in the RIB and forwarding to that destination.

Fig. 3.2 illustrates a step-by-step example of a publishing and subscribing `/camera` topic with FogROS2-SGC. The figure assumes that all the connections between routers are established. This can be achieved through configuration or dynamic node discovery [6]. The steps are:

1. The robot SGC proxy P1 generates an advertisement message for the ROS2 topic `/camera` and sends it to Router 1.

2. After verifying the advertisement message, Router 1 records the advertisement in its RIB and forwards the topic information to Router 2. There can be multiple routers between Router 1 and Router 2.

3. The cloud SGC proxy P2 requests to subscribe to `/camera`, and the subscribe request is sent to Router 2.

4. The subscribe request from Router 2 is routed to Router 1 by checking the source information at Router 2's RIB. After verifying the request, Router 1's RIB records P2 as the data sink.

5. The subscribe request from Router 1 is routed to the robot by checking the source information at Router 1's RIB. If the destination is not found, the router broadcasts a query to other routers.

6. The robot's ROS2 publisher sends a ROS2 message to the proxy. The proxy forwards it to Router 1, Router 1 forwards to Router 2, and Router 2 to the cloud subscriber. At each hop, the messages are forwarded from source to sink.

3.3 Secure Communication

The security of the communication is achieved by using a secure network protocol between routers. We use Datagram Transport Layer Security (DTLS) [4] to provide communications privacy. The DTLS protocol provides secure and authenticated communication on User Datagram Protocol (UDP) and includes a built-in mechanism for dealing with lost or out-of-order packets. DTLS on UDP is well-suited for latency-critical robotics communications systems due to its lightweight nature and low overhead compared to Transmission Control Protocol (TCP). The cryptographic algorithms used to secure the ROS2 packet generation process can be found in Fig. 3.3. The message has the following security guarantees: **Confidentiality:** The ROS2 messages are encrypted with AES Encryption [25] to ensure that only parties with the correct cryptographic key can decrypt the

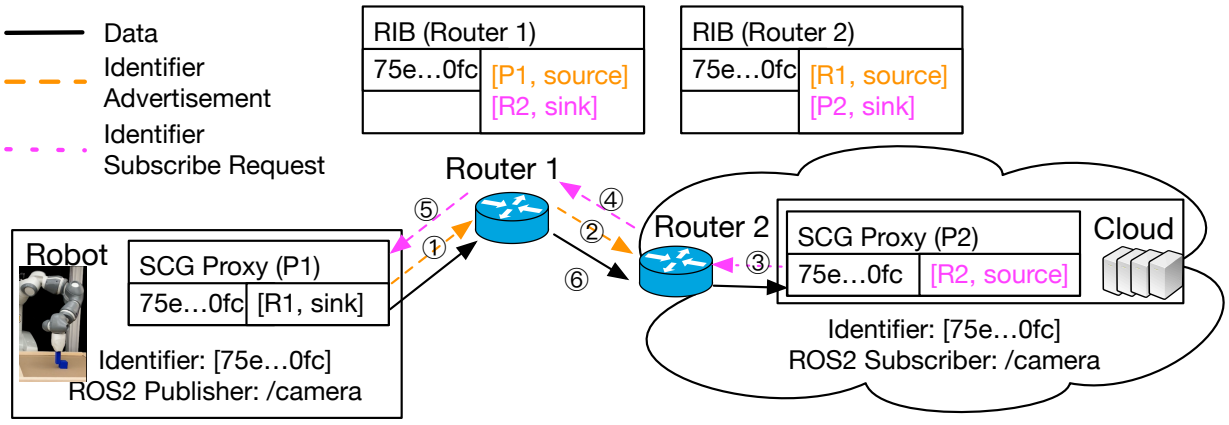


Figure 3.2: An illustration of how a routing connection is established between robot and cloud. The steps are further described in Section 4.1. (1,2) Advertisement generation and publish. (3,4,5) Subscribe request. (6) Data routing.

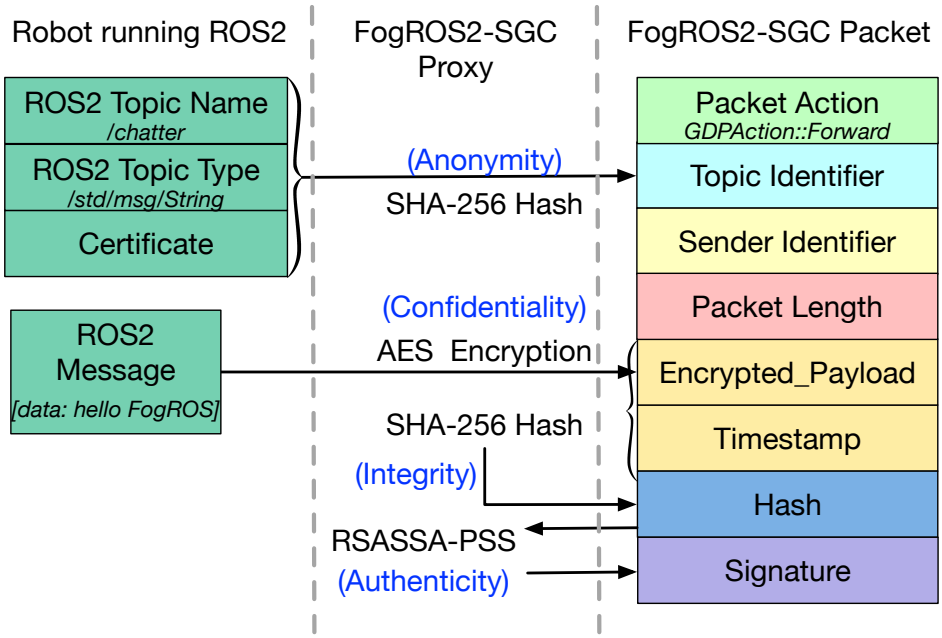


Figure 3.3: An illustration of the cryptographic tools used by SGC proxy to protect a ROS2 string message. The FogROS2-SGC Packet on the right is the message that is routed by FogROS2-SGC. The payload is encrypted to protect the confidentiality of the original ROS2 message. The encrypted data is hashed so that the receiver can verify the message is intact. The hash is signed with the sender's key so that the receiver can verify that the message comes from an authentic and authorized sender.

original ROS2 message data. **Integrity:** The encrypted message is hashed by SHA-256 [13] so the receiver or third-party auditor can easily verify that the message is intact and no other attacker has tampered with the message. **Authenticity:** The hashed message is signed by the RSASSA-PSS [9] algorithm so that receivers can verify that the message is sent from an authorized sender.

To tailor the security with the communication patterns of robotics applications, FogROS2-SGC allows flexible peering with other routers or end points. One may choose to use a dedicated DTLS connection per ROS2 topic, which is ideal for large message payload and frequent communication (e.g., video streaming). One may also choose to use a shared DTLS tunnel, where multiple ROS2 topics share the same DTLS connection. Sharing the same connection reduces the cost of secure connection management and message processing, which is ideal for small message payloads and less frequent communication.

3.4 Transparent and Compatible SGC proxy

The SGC proxy is the interface between FogROS2-SGC and the ROS2 network. In order to allow seamless integration with *any* unmodified ROS 2 application code and mainstream DDS vendors, The SGC proxy converts between ROS2 communication and FogROS2-SGC communication bidirectionally. The user first identifies ROS2 topics that they wish to publish or subscribe through a configuration file. The proxy launches a local ROS2 publisher or subscriber for the corresponding topic. New messages from the local ROS2 network are actively subscribed to by the proxy, and sent to the FogROS2-SGC network. Once the verified subscribers receive the messages, they convert them to standard ROS messages and publish to their local ROS2 network.

To allow non-ROS2 programs to communicate with ROS2 nodes, SGC proxy converts ROS2 messages to a unified JSON-based message format in transit. As a result, FogROS2-SGC can be extended to a variety of protocols such as TCP, UDP, DTLS, TLS, and gRPC. Note, however, that some of the protocols need special handling to be aligned with FogROS2-SGC. For example, gRPC requires the IP addresses of both robot and cloud for bidirectional message passing.

3.5 Compute and Memory-Efficient SGC router

FogROS2-SGC can be deployed on low-power robots under memory and compute constaints, so an efficient implementation of the routing algorithm in Section 4.1 is crucial to the overall performance of the system. Fig. 3.4 shows the architecture of the SGC router. An idiomatic workflow of the router implementation is to (1) receive data from ROS2/network, (2) decide which network connection to forward, and (3) forward data to ROS2/network. Because FogROS2-SGC needs to be extensible to heterogeneous network protocols, the router needs to maintain many simultaneous network connections, ranging from ROS2's publish/subscribe protocol to general network protocol such as DTLS.

Because low-power robots run under memory constaints, memory copying operations and synchronization operations (such as mutex) are expensive. SGC router is implemented in Rust [51]

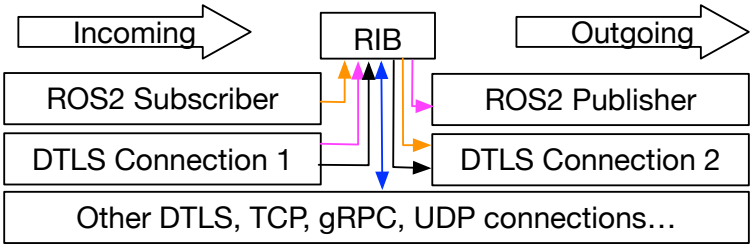


Figure 3.4: SGC router architecture. (**Orange**) Subscribe to a local ROS2 network and publish to FogROS2-SGC routing network. (**Magenta**) Receive from FogROS2-SGC routing network and publish to local ROS2 network. (**Black**) Intermediate SGC router that facilitates message routing. SGC router asynchronously reads, writes, and manages all the network connections. All the message passing (arrows) is zero-copy and does not require movement of actual messages.

to eliminate memory copying operations and the need for synchronization. Rust is a programming language that features a single ownership model: every data object has a single owner, and passing the data is moving the ownership from one variable to another. As a result, it prevents race conditions and reduces data copying by enforcing the passing of data objects by references instead of values.

3.6 Implementation

We implement FogROS2-SGC prototype in 5,000 Lines of Code in Rust. The rust component takes standard REST API interface to take the message control flow. We use r2r, a rust binding of ROS2 to acquire messages from ROS2 middleware. The system integrates ROS with additional networking capabilities (DTLS) under conditional compilation flags. It utilizes crates such as `tokio`, `serde`, `redis`, and `tracing` in Rust for asynchronous runtime, serialization, data management, and logging respectively.

The proxy implementation features **asynchronous and event-driven**, which it is built using asynchronous programming principles, leveraging Rust's `tokio` runtime.

The proxy can be partitioned in the following workflow components:

- FogROS2-SGC automates the operational mode (`pub`, `sub`, `noop`) for a topic based on the CLI output of `ros2 topic info`. It marks the topic as a remote publisher (`pub`) if no local publishers are present. It marks it as a remote subscriber (`sub`) if no local subscribers are found. It does nothing (`noop`) if both local publishers and subscribers are present.

- `ros_topic_creator`: It manages creation of topics based on their action (`pub` or `sub`). It establishes a DTLS stream for data transfer between local ROS topics and remote endpoints. For `pub`, sets up a publisher that sends data to a remote subscriber. For `sub`, sets up a subscriber that receives data from a remote publisher.

- `create_new_remote_publisher` and `create_new_remote_subscriber` These manage the creation of remote endpoints for topics designated as publishers or subscribers.

It interact with RIB (implemented with `redis`) for registering and discovering entities in a distributed system. It listen for Redis keyspace notifications to dynamically manage the list of active publishers or subscribers. It establishes DTLS streams for connecting to remote subscribers or publishers respectively.

- `ros_topic_manager` The manager coordinates the overall management of ROS topics as one single ROS2 node. It regularly checks for new topics and manages their lifecycle based on the configuration and dynamic discovery. It utilizes a Redis-based mechanism to manage and update the status of topics across multiple nodes.

The proxy and switches contains a Forward Information Base, that routes GDP packets within a network, utilizing a combination of asynchronous message passing and reactive programming to handle incoming GDP packets and routing updates.

1. **Initialization:** A hash table (`coonection_rib_table`) is initialized to keep track of the routing information for various GDP channels associated with different GDP names.
2. **Asynchronous Event Handling:** The function enters an event loop using `tokio::select!`, handling different types of messages concurrently:
   - **GDP Packet Reception:** Packets received from the `fib_rx` channel are checked against the routing table. If an entry exists, the packet is forwarded to the appropriate destinations using the `send_to_destination` function. If no entry is found, a query is sent to the RIB (Routing Information Base) to resolve the destination, and the packet may be dropped or buffered.
   - **Channel Registration:** New channels (GDPChannel objects) received from the `channel_rx` are registered in the hash table, adding new routing entries or updating existing ones.
   - **RIB Responses:** Responses from the RIB, which can indicate new routing information or updates, are handled by updating the routing table and possibly forwarding the GDP packets to the newly resolved destinations.
   - **Routing Status Updates:** Status updates received on the `stat_rs` channel prompt a refresh of the routing advertisements to all known destinations, ensuring the routing information is current.
3. **Routing Table Management:** The routing table is dynamically updated based on the network topology and routing information received from the RIB or directly via the channels.
4. **Packet Forwarding:** The `send_to_destination` function is used to forward packets to the appropriate channels based on the routing table. This includes handling send failures due to closed channels.

Chapter 4

Latency Sensitive FogROS2-SGC

In this chapter, we introduce an extension to FogROS2-SGC that enables latency-sensitive robotics applications by deploying replicated services in a location-independent manner and connecting with user-specified time constraints. We name the extension as FogROS2-LS, illustrated in Fig. 4.1, Location-independent robotics service is achieved through a secure and global routing framework (Sec. 4.1) that associates local ROS2 topics with globally-unique and secure identifiers. The same services can be replicated at different geographic locations with the same identifier, allowing robots to discover the location independent services and connect as if the service were on the same machine. FogROS2-LS uses Policy-Guided Anycast (Sec. 4.2) to address the challenge of choosing a single deployment from several instances sharing the same identifier without any ap-

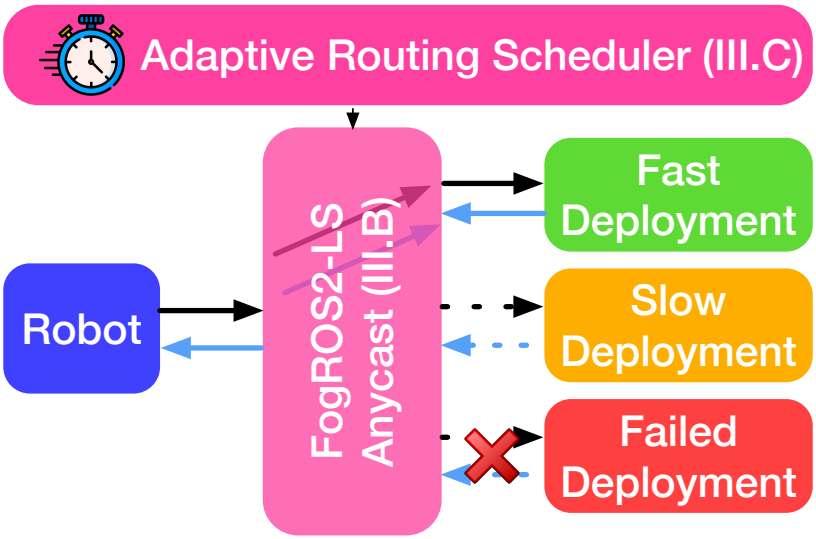


Figure 4.1: **System Diagram of FogROS2-LS components** FogROS2-LS enables latency-sensitive fog robotics applications through an adaptive scheduler that generates routing policies. The routing policy directs robots to connect with the optimal service deployment by Anycast.

plication or interface modifications of ROS2. This approach enables connection to a single service deployment based on a routing policy. This policy is dynamically generated by the Adaptive State Machine Scheduler (Sec. 4.3), which monitors and orchestrates application latency.

4.1 Location Independent Routing

Location Independent and Unique Identifiers FogROS2-LS enables multiple location independent deployments of the same service by assigning a shared globally unique identifier to all the service deployments. Robots can deterministically generate the identifier to discover and connect with the service. FogROS2-SGC details the identifier cryptographic construction process, in which an unauthorized attacker attempting a brute-force attack would have to guess the service identifier or reverse the information used to generate it among 10^{77} possibilities, a value near the number of protons in the observable universe. The original FogROS2-SGC uses this property for mobility, allowing robots to move freely across different network domains. FogROS2-LS extends FogROS2-SGC's identifier generation process so that multiple servers at various geographic locations can host the same ROS2 service by assigning their service interfaces with the same identifier. FogROS2-LS also supports identifier generation for both ROS2 publish/subscribe and service/-client communication paradigm.

Flexible Global Service Discoverability and Connectivity Given a shared identifier associated with a fog robotic service replicated across various geographical locations, robots face the challenge of globally discovering and connecting to these deployments. This task is challenging due to heterogeneous network domains, firewalls, and Network Address Translation (NAT) [72], which sometimes restrict accessibility to network addresses or ports outside of certain domains. It is further complicated by the flexibility requirement of FogROS2-LS that the robot can dynamically select the service based on the latency requirement.

FogROS2-LS achieves flexible global discoverability by maintaining a *global routing information base*, a centralized registry storing all routing data, enabling newcomers to directly connect with existing publishers and subscribers. This registry solely aids in connection establishment and maintenance without data routing. This provides two clear benefits: (1) it is sufficiently lightweight to operate on low-end servers, with UC Berkeley offering a public version; (2) it enables flexible packet control to specific endpoints by direct robot-service connections, since intermediary routers can introduce protocol complexities and inefficiencies.

FogROS2-LS enables global connectivity through proxying local ROS2 communication to WebRTC [64], a peer-to-peer network transport protocol that facilitates global connections and is commonly used for web-based video conferencing. We refer readers to [64] for its design and guarantees. In FogROS2-LS, the global routing information base maintains the WebRTC channel details of current publishers and subscribers, allowing newcomers to utilize this information to directly establish WebRTC connections with existing participants.

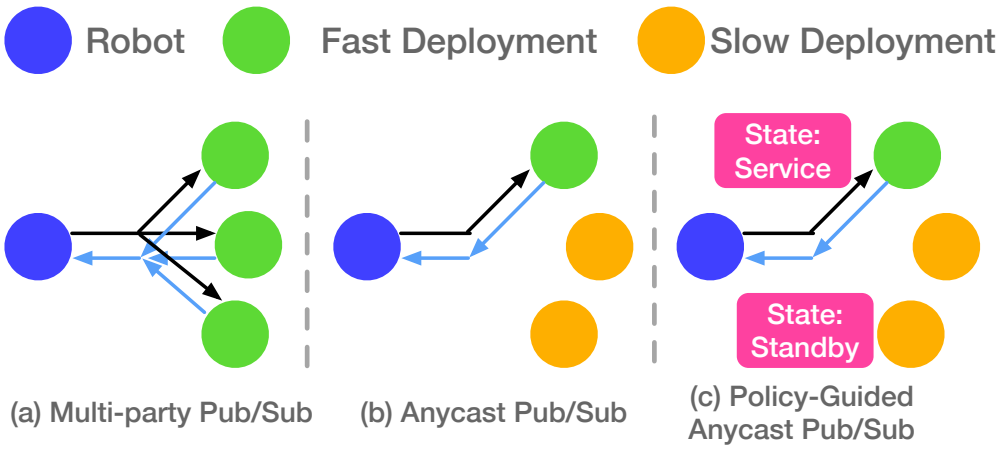


Figure 4.2: **Comparison of Publish/Subscribe and Anycast Communication Paradigms** ROS2's publish/subscribe paradigm publishes a message (e.g. a request from robots) to all existing subscribers, which wastes computational and network resources. In our context, the Anycast paradigm is more appropriate, as it forwards to only the most optimal deployment based on the policy.

4.2 Policy-Guided Anycast

Many ROS2 applications utilize the publish/subscribe paradigm in ROS2 to emulate a server and client setup, where the robot client publishes to the request topic and subscribes from the response topic, while the server subscribes from the request topic and publishes to the response topic. This emulation works if only one service deployment exists in the network. Otherwise, the robot may send requests to multiple deployments (Fig 4.2.A), wasting compute and network resources in the context of multiple deployments of the same service. Additionally, the robot could receive duplicated responses from multiple deployments.

Recognizing that robots need to communicate with only one of the services, FogROS2-LS resolves this issue by formulating *Anycast* to describe this *one-to-one-of-many* relationship: the request message only needs to be forwarded to *any* of the deployment that subscribes to the request topic, instead of all of them. Other idle service deployments in ROS2 consume negligible network and CPU resources, which can be used to handle other robots or services. However, we note that this is conceptually different from IP Anycast [14]: IP Anycast concentrates on routing packets to a destination IP address shared by multiple locations, while FogROS2-LS bridges publish/subscribe to one of many services (Fig 4.2.B).

FogROS2-LS enables Anycast by managing its global routing state: which ROS2 topics should be globally discoverable, published, or subscribed to. Anycast is achieved when a single service deployment subscribes to service request topics and publishes to service response topics, with the robot maintaining a location-independent connection exclusively with that service deployment. Given that a service may use multiple ROS2 topics, we use *state* to refer to the aggregation of the global topic publish or subscribe relationship of a robot or service deployment at a specific time. An example of the state definition can be found in Listing 1. FogROS2-LS has predefined three

states to facilitate Anycast: the *robot* state designated for the robot, the *service* state for actively managing requests, and the *standby* state which prevents the exposure of any global service topics to other machines. One can also define their own state, but additional adaptation is required for anycast invariance.

Each FogROS2-LS-integrated robot or service operates its own state machine. To switch from one service deployment to another, the scheduler reassigns the state machine, marking the old deployment as standby, and the new one as service. FogROS2-LS automates the state transitioning by tearing down the network connections of the previous state, updating the global routing information base, and establishing the connections for the new state.

We transform an intricate network routing management problem into a straightforward state machine management problem. Instead of designing networking protocols and maintaining complex routing states, we only need to keep the invariant that only one service deployment is in *service* state, while other deployments are in *standby* state (Fig 4.2.C). In FogROS2-LS, the collective states of all the state machines for the robot and service form the *policy*, which is user-initialized (Listing 1) and overseen by an adaptive scheduler (Sec. 4.3). To minimize the service interruption of state transitioning, FogROS2-LS maintains the underlying network connection from previous services, merely pausing the packet forwarding instead of completely tearing down the connections. This strategy leverages the minimal overhead involved in keeping a network connection active, allowing for the reuse of these connections when switching for those services that require continuous operation.

4.3 Adaptive Time-Bounded Policy Scheduler

FogROS2-LS ensures an application's adherence to the time constraint by monitoring the application latency, checking if the latency fulfills the bound, and dynamically adjusting its routing policy. The latency is profiled by adjusting the routing rules ahead of the deployment. The necessity for a user-defined time bound is to avoid impractical monitoring of the most optimal machine and to prevent overly frequent switching.

FogROS2-LS uses a centralized scheduler to determine which machine should be the service machine for the robot. The scheduling decision translates to the actual state machine updates that are synchronized across all available deployments for consistency and fault tolerance. The scheduler passively monitors the application latency and is triggered when the application fails to fulfill the latency bound or if the current service machine is disconnected. The scheduling decision is based on historical profiling results on both active and idle service deployments. The scheduler temporarily directs the messages to the selected service and collects latencies for a short period of time. FogROS2-LS gathers application latency profiles at bootstrapping and when none of the available machines fulfill the latency bound based on the past data.

To accommodate diverse ROS2 applications, FogROS2-LS provides three latency collection mechanisms: (1) measurement of the difference between the request and response time of ROS2 services, (2) heuristics to align the request and response topics in ROS2 pub/sub and to get the

Listing 1 FogROS2-LS State and Policy Configuration File Example This `state_definition` section illustrates how the user specifies the machine's state—such as `standby`, which the node remains idle, and `service`, which the node actively listens for input requests and outputs to the response topic. The state machine also allows specifying the parameters such as camera frame rate and resolution. User can also add their own custom states. The second half of the configuration file defines the desired latency bound for the messages from the request topic and response topic.

```
# III.B
state_definition:
  standby: # do not publish or subscribe to any topics
  service:
    topics:
      - /yolo/input: sub # subscribes to YOLO input
      - /yolo/output: pub # publish to YOLO output
  robot:
    topics:
      - /yolo/input: pub # subscribes to YOLO input
      - /yolo/output: sub # publish to YOLO output
    params:
      - /camera: rgb_module.profile:=640x480x30

initial_policy:
  turtlebot: robot
  machine_edge: service
  machine_cloud: standby

# Section III.C Adaptive Latency Monitoring
latency_bound:
  median: 0.3 # in second, max/min/median/mean/stddev
```

State Definition

Initial Policy

timing difference of the request topic and response topic, and (3) direct input of latency from a pre-defined ROS2 interface.

4.4 Implementation

Beyond FogROS2-SGC, we prototype additional 2,000 Lines of Code in Rust that takes standard REST API interface to take the message control flow. We implement a ROS2 wrapper of FogROS2-LS for developers to easily access and control the routing. The wrapper is implemented as a ROS2 module with 2000 LoC in Python. Both FogROS2-SGC and its ROS2 wrapper are open source on Github [1].

[1] https://github.com/KeplerC/fogros2-ls

4.4.1 Software-Defined Rust Switch

On top of FogROS2-SGC, FogROS2-LS implements a Rust-based API POST handler that processes incoming JSON payloads representing `ROSTopicRequest`. It sends these requests through a channel for further processing and responds with a `ROSResponse` containing the result of the operation. The server's listening port is configured through the environment variable `SGC_API_PORT`. This differentiates possible ROS2 domains if one needs to use FogROS2-LS on the same physical machine or local network.

4.4.2 Timebound Analyzer

The policy scheduler is implemented as a `Time_Bound_Analyzer` that analyzes the application latency. It is implemented as a ROS 2 node written in Python that monitors and analyzes network latency in a distributed system of ROS nodes. The node collects latency data, calculates various statistical parameters, and publishes this data for monitoring and potential adaptive control.

Upon initialization, the `Time_Bound_Analyzer` declares and retrieves necessary ROS parameters `latency_window` (time window for latency measurements) that collects latency of a certain interval, subscribes to a latency topic to receive latency measurements, gathers system information such as IP address, CPU count, and checks for GPU availability using `psutil` and `pynvml` ad finally initializes a publisher to send out profile information about the node.

Upon an incoming messages on the latency topic that contains the application latency, it manages a sliding window of latency measurements to maintain recent data within a specified time frame and automatically discards the oldest data to prevent unbounded growth of the data structure. The latency update is triggered by the `stats_timer_callback` function is invoked periodically and is responsible for aggregating recent latency measurements, computing statistical measures such as mean, median, standard deviation, and range (min/max) of latency, and publishing the computed profile to a ROS topic for external use.

Statistical measures The policy scheduler supports various statistical measurements on analysis such as `goodness_of_variance_fit`, which calculates the goodness of variance fit (GVF) and a statistic related to the classification of data into specified classes using Jenks natural breaks optimization, and `classify` that classifies individual measurements into categories based on break points.

4.4.3 Policy Scheduler

Based on the time bound analyzer, the policy scheduler manages and optimizes distributed system resources dynamically based on profiling data such as network latency and system performance metrics.

The scheduler is implemented as a ROS2 node. On initialization, the node initializes by declaring necessary ROS parameters that influence operational decisions such as the maximum number of waiting profiles and automatic resource switching capabilities. It loads configuration settings from a YAML file, establishing initial resource assignments and operational parameters for service

machines. It also subscription to profile topics and service handlers of profiling data and respond to system management commands from the time bound analyzer.

The profile is managed from various service machines is received and processed to update the internal state and performance metrics for each machine.The node monitors system timeouts and disconnects to make adjustments in resource assignments based on the latest data. Several ROS services are implemented to handle specific requests such as system profiling optimization, active profile listing, and manual resource switching based on current profiling data. Actions triggered by service calls may include state adjustments and resource reallocation to optimize system performance.

The state machine in the scheduler manages a dictionary of service states, keeping track of the operational state and performance metrics of each connected service machine. The state machine scheduler manages the logic to handle transitions of machine states (such as from standby to active) based on performance evaluations and current system needs. Performance metrics such as latency are monitored to determine the optimal machine for resource allocation. Heuristic-based decision processes evaluate if a machine's performance meets required criteria, enabling dynamic adjustments. Functionality to run parallel profiling across multiple machines is included to optimize resource allocation in real-time.

Chapter 5

Evaluation

This chapter describes the implementation effort of FogROS2-SGC and the evaluation of the existing implementation. Due to the nature of performing secure computation and communication in and between the secure enclaves, the research prototype is independent from all of the previous GDP implementations.

5.1 System Benchmark

In this section, We evaluate FogROS2-SGC on system benchmarks to show how it performs against alternative designs and on robotics benchmarks to show how robotics applications benefit from FogROS2-SGC.

5.1.1 Evaluation Setup

We use an Intel NUC with an Intel® Pentium® Silver J5005 CPU @ 1.50 GHz with a 5 Mbps network connection to act as the robot. The robot is connected with a Standard DS3 v2 cloud instance (4 vCPUs, 14 GiB memory) on Microsoft Azure. The robot is located in California (west coast of U.S.), and the cloud server is located in Virginia (east coast of U.S.).

We evaluate the performance of FogROS2-SGC's message processing latency and throughput against other distributed ROS2 systems. Messages are sent in binary with type `sensor_msgs CompressedImage` and response with string type `std_msgs/String`. We compare against the following baselines (1) **VPN**: We use Wireguard VPN [10], which is the same VPN as FogROS2 [34]. (2) **Rosbridge:** Rosbridge is the most commonly used websocket proxy that allows non-ROS code to interact with ROS code. We use Rosbridge in combination with Rosduct in the same way as FogROS [21]. (3) **Capsule:** We use Capsule, a software switch inspired by Netbricks [59], to emulate the design of FogROS2-SGC. We also implement rosduct [8] in ROS2 that converts between ROS2 and network traffic. The detailed description and implementation can be found in FogROS-G [19]. **FogROS2-SGC** uses the default DTLS network protocol. We include **FogROS2-SGC-TCP**, a variant that uses TCP instead of DTLS.

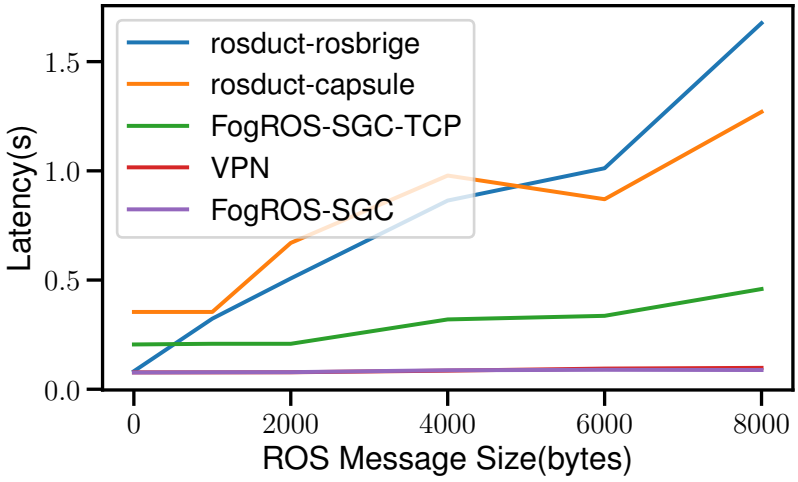


Figure 5.1: **Message round trip latency to the cloud** (lower is better). Latency is averaged over more than 50 packet window. FogROS2-SGC is 19 times faster than rosbridge baseline for 8000 byte message.

ROS2 Message Latency: We measure the Round Trip Time (RTT) between when a robot publishes a ROS2 message to the cloud and when data is received by the cloud, which echoes a short message on a separate ROS2 topic. The RTT also includes the time of parsing the messages and analyzing the latency. The result can be found in Fig. 5.1. FogROS2-SGC with DTLS has similar performance as VPN, which has 0.076s round trip latency for small messages. FogROS2-SGC is 10.2% faster than VPN for 8000 byte messages (0.088 vs 0.097). FogROS2-SGC is 19$\times$ faster than rosduct-rosbridge (0.088 vs 1.67). There are two reasons for this: (1) Rosduct is implemented in Python and provably slower than Rust. It uses blocking network operations while FogROS2-SGC uses non-blocking asynchronous network operations for sending and receiving data. (2) Rosbridge requires seralization of binary messages in JSON, which require more bytes and lead to larger messages.

ROS2 Message Throughput: Message throughput is measured by the number of messages processed per second. Different from other experiments, throughput is measured on the local area network connected with Ethernet, in order to prevent network bandwidth from being the bottleneck. Table 5.1 shows the message processing throughput. FogROS2-SGC achieves near-native throughput as ROS2 and incurs only 3% overhead due to the security and conversion to a unified message format. FogROS2-SGC has 2.1$\times$ higher throughput than rosbridge, because rosbridge requires more bytes to serialize binary strings.

Startup and Advertisement Time: In a RIB that has 10,000 routing records, the average time for publishing a name to the RIB takes 4ms and subscribing to a name from RIB takes 2ms. The average startup time from starting a program to receiving the first message takes 2.4 ms.

Protocol	Throughput (msg/second)
Original ROS	330.43
SROS2	320.17
Rosduct-Rosbridge	152.79
FogROS2-SGC-TCP	268.03
FogROS2-SGC	320.40

Table 5.1: **Message throughput evaluation of FogROS2-SGC** (higher is better). Every message is 1000 bytes. The throughput of FogROS2-SGC is near native performance while adding secure and global connectivity and 2.1 times higher than rosbridge.

	vSLAM		Grasp Planning		Motion Planning	
Scenario	fr1/xyz1	fr1/loop	raw matrix	Compressed	Apartment	Cubicle
rosduct-rosbridge	10.31	10.29	20.3	13.67	0.08	0.08
VPN	1.16	1.45	**5.7**	1.47	0.07	0.07
FogROS2-SGC-TCP	1.19	1.57	8.4	1.58	0.07	0.07
FogROS2-SGC	**1.15**	**1.42**	-	**1.45**	0.07	0.07

Table 5.2: **Network latency of FogROS2-SGC on cloud robotics applications** (lower is better) FogROS2-SGC is better than rosduct-rosbridge and VPN on vSLAM and compressed grasp planning. We conducted motion planning on other scenarios (Home, TwistyCool) and the latency is the same.

5.2 Simulated Cloud Robotics Benchmark

We evaluate the network latency of FogROS2-SGC with 3 example cloud robotics applications: SLAM with ORB-SLAM2 [56], Grasp Planning with Dex-Net [49], and Motion Planning with Motion Planning Templates (MPT) [33]. The detailed description of these benchmarks can be found in [21].

As detailed in Table 5.3, although FogROS2-SGC can scale to multiple robots and provide fine grained access control for the robots, it demonstrates even better point-to-point performance than VPN in the vSLAM and grasp planning experiments. FogROS2-SGC is 9.42 times faster than rosbridge-rosduct on compressed grasp planning images. However, FogROS2-SGC cannot reliably transmit large and uncompressed grasp planning matrices. The raw matrix after serialization is larger than 13MB. We observe a significant amount of lost and out of order messages because the default transport protocol of FogROS2-SGC is DTLS over UDP and the communication channel does not recover from lost and out of order messages. Although transmitting such large message within single ROS2 message is rare, users can choose other supported transport protocols (such as TCP, gRPC) to meet the requirement of their applications.

5.2.1 ORB-SLAM2

ORB-SLAM2 [56] is a visual simultaneous localization and mapping system that uses monocular video input. In this experiment, a Camera Node publishes a 640×480 resolution video with each

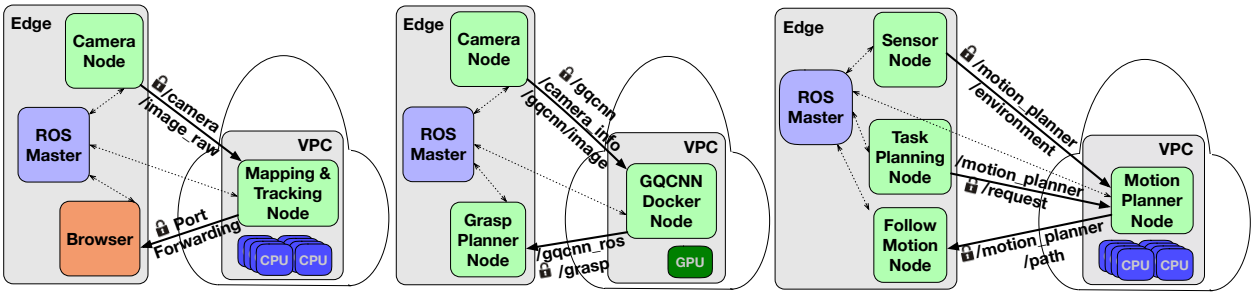


Figure 5.2: **Example FogROS2-SGC applications in experiments.** In experiments, we run 3 sample applications, each with one node accelerated by cloud computing, with the nodes and topics shown here. For brevity, we only depict the VPC-based solution here, but we also experiment with the proxy-based solution.

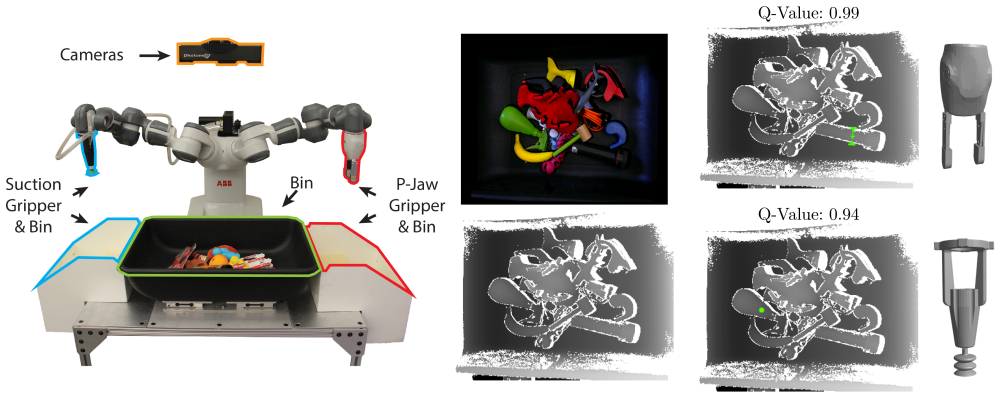


Figure 5.3: **Grasp Planning:** A robot with an overhead depth camera and either a suction or parallel jaw gripper (left) must plan a grasp on one of the objects in the bin beneath it given an RGBD image observation (middle). Examples of planned grasps (green) and their q-values are shown for both parallel jaw and suction grippers (right).

frame 48 KiB on average to the cloud (Fig. 5.2). On the cloud an ORB-SLAM2 node subscribes to the video feed [65] and computes a pointcloud map along with the current estimated location within the map, which are sent back to the robot. For more details on the ORB-SLAM2 algorithm, we refer readers to the paper and open-source code available from Mur-Artal and Tardós [56].

5.2.2 Dex-Net Grasping Service

Grasp analysis computes the contact point(s) for a robot gripper that maximize grasp reliability—the likelihood of successfully lifting the object given those contact points. To plan grasps on rigid objects in industrial bins using an overhead depth camera, we use an open-source implementation of the fully-convolutional grasp-quality convolutional neural network (FC-GQ-CNN) [63, 50]

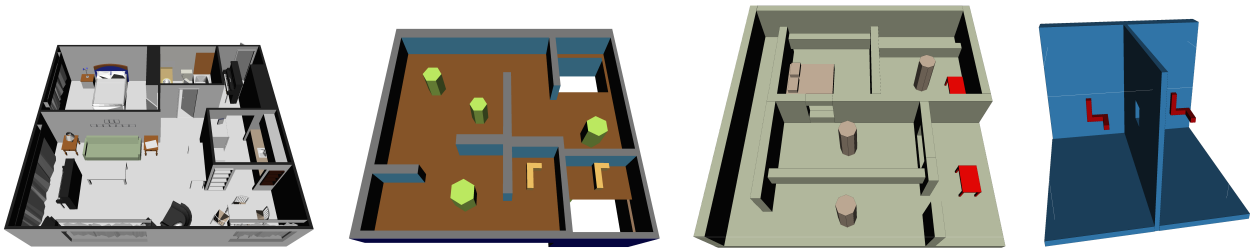


Figure 5.4: **Motion Planning Scenarios**. We run OMPL [24] motion planning problems as benchmarks. Left-to-right: Apartment, Cubicles, Home, and Twistycool. In these problems, the robot is a rigid-body object that must, through rotation and translation, find a collision-free path through the environment, from a start pose to a goal pose.

from Dex-Net[49]. We wrap FC-GQ-CNN in a ROS node and deploy it to the cloud along with pretrained neural-network weights as a Docker image. We refer the reader to Satish et al. [63] and Mahler et al. [50] for details and code for the neural network and grasping environment.

This node subscribes to 3 input topics containing a scene depth image and mask for objects to be grasped, and a message of type `sensor_msgs/CameraInfo` containing camera intrinsics. Internally, the node feeds this to FC-GQ-CNN, which outputs a grasp pose and associated estimate of grasp quality. The node wraps these outputs, along with the gripper type and coordinates in image space, into a `gqcnn_ros/GQCNNGrasp` message, and publishes it.

While the node can be run both locally or in the cloud, using cloud GPU instances as opposed to a CPU for neural-network inference can greatly reduce computation time. In either case, the node is wrapped inside of a Docker container, reducing the need for resolving dependency issues between deep-learning libraries, CUDA, OS, and ROS versions. The pretrained models in the image is intended for a setup similar to that shown in Figure 5.3; variations in camera pose, camera intrinsics, or gripper type may require retraining the underlying model for accurate predictions. We compare grasp planning times across 10 trials using both the CPU onboard the edge computer and FogROS2-SGC with the Docker images on the cloud. We also show compute times when using a compressed depth image format to transfer images instead of transferring raw images to the cloud directly. For the latter case, images are compressed and decompressed using the `republish` node from the `image_transport` ROS package [1].

5.2.3 Motion Planning

Motion planning computes a collision-free motion for a robot to get from one configuration to another. Sampling-based motion planners randomly sample configurations and connect them together into a graph, rejecting samples and motions that are in collision. These planners can be scaled with additional computing cores.

Using FogROS2-SGC, we deploy a multi-core sampling-based motion planner [32, 33] to a 96-core computer in the cloud to solve motion planning problems from the Open Motion Planning Library (OMPL) [24] (see Fig. 5.4). This planner node subscribes to topics for the collision model of the environment and motion plan requests (Fig. 5.2). When the planner node receives a message

	vSLAM		Grasp Planning		Motion Planning	
Scenario	fr1/xyz1	fr1/loop	raw matrix	Compressed	Apartment	Cubicle
rosduct-rosbridge	10.31	10.29	20.3	13.67	0.08	0.08
VPN	1.16	1.45	**5.7**	1.47	0.07	0.07
FogROS2-SGC-TCP	1.19	1.57	8.4	1.58	0.07	0.07
FogROS2-SGC	**1.15**	**1.42**	-	**1.45**	0.07	0.07

Table 5.3: **Network latency of FogROS2-SGC on cloud robotics applications** (lower is better) FogROS2-SGC is better than rosduct-rosbridge and VPN on vSLAM and compressed grasp planning. We conducted motion planning on other scenarios (Home, TwistyCool) and the latency is the same.

on any of these topics, it computes a motion plan, and then publishes it to a separate topic. For more details on the multi-core motion planner, we refer the reader to the paper and the open-source code by Ichnowski and Alterovitz [33]. To configure FogROS2-SGC to work with multi-core motion planner, we record the steps we use to setup the dependencies (e.g., FCL [58] and Nigh [36]) in a script.

We compare the planning time as the difference between publishing a motion plan request message, and receiving the plan result message, and show the results in Table 5.3. The same motion planning problem is solved in a fraction of the time on the cloud when compared to using the edge computer. However, when the network latency (between 0.3 s and 0.6 s) is longer than the motion planning computing time. For simpler planning problems, there may be little to no benefit to a cloud deployment. If the motion planner is asymptotically-optimal (finds shorter/better plans the longer it runs and with more CPU cores), then one could potentially run the motion planner for the same amount of time but get a better path using the cloud. Anand et al. [12] explored and shown the benefit of using the tradeoff between more cores and the resulting motion plan optimality.

As detailed in Table 5.3, although FogROS2-SGC can scale to multiple robots and provide fine grained access control for the robots, it demonstrates even better point-to-point performance than VPN in the vSLAM and grasp planning experiments. FogROS2-SGC is 9.42 times faster than rosbridge-rosduct on compressed grasp planning images. However, FogROS2-SGC cannot reliably transmit large and uncompressed grasp planning matrices. The raw matrix after serialization is larger than 13MB. We observe a significant amount of lost and out of order messages because the default transport protocol of FogROS2-SGC is DTLS over UDP and the communication channel does not recover from lost and out of order messages. Although transmitting such large message within single ROS2 message is rare, users can choose other supported transport protocols (such as TCP, gRPC) to meet the requirement of their applications.

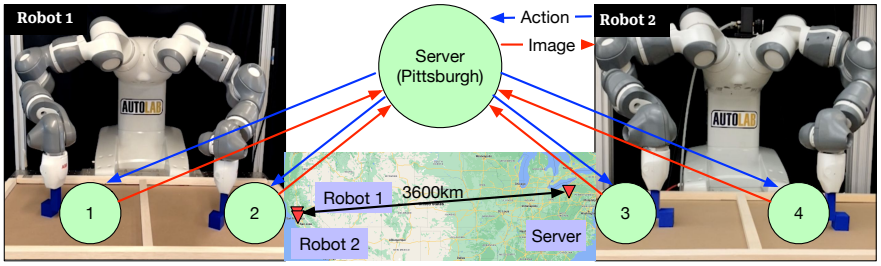


Figure 5.5: **The experiment setup of Fleet-DAgger**. Two ABB YuMi robots located in two separate buildings in Berkeley utilize computation from a server located in Pittsburgh for an image based block pushing task.

5.3 Cloud Robotics Case Studies

5.3.1 Connectivity Case Study: Fleet-DAgger

We apply FogROS2-SGC to the control of a fleet of 4 physical robot arms, an increasingly relevant setting in robotics and the third motivating example in Fig. 1.1. We use the physical experiment setup from Fleet-DAgger [31], where each robot simultaneously performs an image-based block-pushing task (see Fig. 1.1C). The task is to repeatedly push a cube to a goal region randomly generated in the image, where a new goal is sampled from the reachable workspace upon reaching the previous goal. The 4 workspaces have an identical setup (but different block positions and goals) to enable the aggregation of each robot's data into a shared dataset and training of a single shared policy on this dataset, as is typical in fleet learning [31]. When autonomous control is unreliable, the robots fall back on and learn from remote human teleoperation, where global connectivity can dramatically increase the number of available humans. The arms belong to two bimanual ABB YuMi robots in two different labs about 1 km apart with separate local area networks. To test global connectivity, compute is off-loaded to a separate node in a third local area network at Carnegie Mellon University 3600 km away, where the robot nodes send images of the current state and receive actions to execute.

In a previous implementation, Hoque et al. [31] use Secure Shell (SSH) and Secure File Transfer Protocol (SFTP) to communicate between robots and the centralized compute node and Python multiprocessing to enable simultaneous execution. This approach requires storing all SSH credentials at a single node (a security concern), writing image data to the file system of all nodes at every timestep, complex asynchronous programming, and restricting all node locations to within the university campus firewall. To mitigate these issues, we (1) re-implement the communication system with ROS2 and (2) seamlessly connect all nodes with FogROS2-SGC with TCP by modifying only a single configuration text file on each node. Relative to the previous implementation, the FogROS2-SGC implementation reduces communication time by 64% (Table 5.4), where communication time includes image transmission latency and synchronization across all arms but not machine learning or arm motion. FogROS2-SGC also reduces communication time by 33% relative to the initial implementation even when the robots are in Berkeley, CA and the server is moved to Pittsburgh, PA. Note that the SSH method does not work between Berkeley and Pittsburgh due to university network firewalls [2]. A diagram of the system architecture is in Fig. 5.5.

Communication System	Server Location	Communication Time (s)
SSH + SFTP	Berkeley, CA	0.86
	Pittsburgh, PA	-
FogROS2-SGC	Berkeley, CA	0.31
	Pittsburgh, PA	0.58

Table 5.4: **Communication time of SSH+SFTP and FogROS2-SGC** (lower is better). FogROS2-SGC with TCP reduces the communication time per experiment step (i.e., one simultaneous action on the 4 arms) by 64% when compared to SSH+SFTP, and has 33% lower communication time than SSH+SFTP in Berkeley even if the server is moved to Pittsburgh. SSH does not work if the server is in Pittsburgh due to a university firewall restriction.

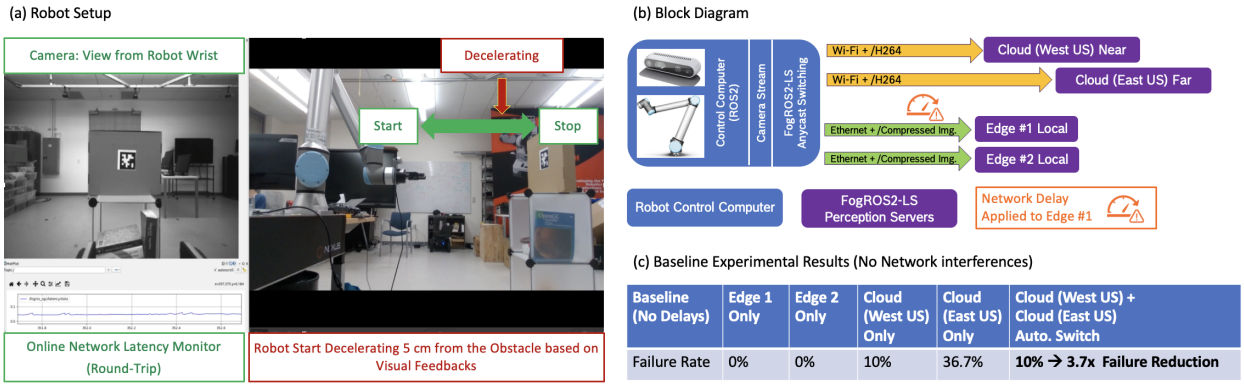
Delay added to Edge 1 Failure Rate (%)	Edge 1 Only	Edge 1 + Edge 2	+ Cloud(W)	+ Cloud(E)	+ Cloud(W) + Cloud(E)
x ms Applied to 100% Packets	56.7%	0.0%	10.0%	30.0%	**6.7%**
(Failure Reduction)		(∞)	(5.7x)	(1.9x)	**(8.5x)**
x ms Applied to 50% Packets	26.7%	0.0%	**6.7%**	53.3%	10.0%
(Failure Reduction)		(∞)	**(4.0x)**	(0.5x)	(2.7x)

Table 5.5: **When local network was congested, FogROS2-SGC reduced failures:** Upon application of random network delays on Edge #1, FogROS2-SGC automatically re-route to alternative services, i.e. Edge 2, Cloud(W), Cloud(E). Such automatic switching functionality to an alternative, location independent service(s) improves robustness of the system and reduced failure rate (lower the better) of the overall system. FogROS2-SGC matches performance of the best individual machine, reducing the failure rate by up to 8.5 times by selecting the best available machine compared to single edge service system under heavy network congestion. We also showed that FogROS2-SGC enabled flexible yet reliable access to open cloud services via Anycast to reduce failure rate.

5.3.2 Latency-Sensitive Collision Avoidance

We demonstrated location-independent FogROS2-LS services with two latency-sensitive physical robotics tasks: (A) high-speed collision avoidance (Fig. 5.6) and (B) continuous object following (Fig. 5.7). Both tasks required continuous visual feedback from the robots and time-sensitive controls back to the robots. The first task required a single time-sensitive command while the second required continuous commands. To offload visual perception to both edge and cloud servers, we streamed online camera feeds from these robots to off-board servers via FogROS2-LS for continuous QR Code pose estimation (AprilTag [73]). 6D pose of the target and robotics control signals were then sent back, also via FogROS2-LS, to complete the feedback loop. FogROS2-LS continuously monitored the round trip time of each frame of pose estimation on all available edge and cloud servers connected to the operating robot during run time.

When the current service failed to return 6D poses to the robot within the designated time, FogROS2-LS autonomously switched to another operational service. This approach enhances system robustness, both by averting potential failures, such as in collision avoidance scenarios, and by facilitating recovery in instances of continuous object tracking.



(c) Baseline Experimental Results (No Network interferences)

Baseline (No Delays)	Edge 1 Only	Edge 2 Only	Cloud (West US) Only	Cloud (East US) Only	Cloud (West US) + Cloud (East US) Auto. Switch
Failure Rate	0%	0%	10%	36.7%	**10% → 3.7x Failure Reduction**

Figure 5.6: **High Speed Collision Avoidance Evaluation** (a) The robot arm approached the target with maximum speed and decelerates at 5cm away from the target based on the visual detection of the QR Code. Delayed estimation leads to failure by collision. (b) Architecture of multiple off-board perception servers both on the Edge and in the Cloud (c) Baseline experimental results (without network delays) showing that FogROS2-LS reduces failure by 3.7x when two Cloud servers were available to support a location independent service, compared to the case of a single, far away Cloud (E) service.

Setup Fig. 5.6 shows the physical setup of the collision avoidance evaluation. The Universal Robots arm (UR10e), with an Intel RealSense D435i mounted on the wrist, advanced towards the QR Code at maximum speed (measured at 1.3 m/sec). Meanwhile, the camera sent a monocular video stream to the Edge or Cloud at 640x480 resolution, maintaining a steady QR Code 6D pose estimation at 90 Hz. Due to bandwidth limitations, we streamed the video to the cloud with H.264 compression. We streamed Video to a local edge server via series of compressed images rather than H.264, as edge network has better bandwidth. The robot arm underwent maximum deceleration if robot arm recognized that it was 5 cm away from the target. It is considered failure if the off-board robot command, in response to the visual detection, failed to reach the robot in time, causing the robot arm to collide with the target that holds the QR code.

We used FogROS2-LS to connect the robot controller to various Edge or Cloud servers. We isolated the ROS nodes on the robot controller from Edge server's using ROS2 domains to avoid cross interference. All the experiments were conducted 30 times. Prior to each run, FogROS2-LS profiled networks for up to 100 ms per machine. It then selected the machine with the lowest round-trip latency that can fullfill the configured time-bound. It then pauses for 500 ms to allow H.264 stabilization before each robotics trial begin.

Results To benchmark, we calculated the failures rate over the 30 trials for baseline (Fig. 5.6 (c)) and local network congestion (Table 5.5). We showed that the location-independent service built with FogROS2-LS reduced the failure rate of the collision avoidance robot task, because FogROS2-LS can choose the best available service automatically via Anycast.

FogROS2-LS can reduce the failure rate by up to 8.5 times. When 100% network latency is applied and the cloud is available, FogROS2-LS chooses Cloud for almost all the trials, and the overall performance is close to the performance of the Cloud. Notably, FogROS2-LS can perform better than any available standalone machines by selecting the available machine with best latency

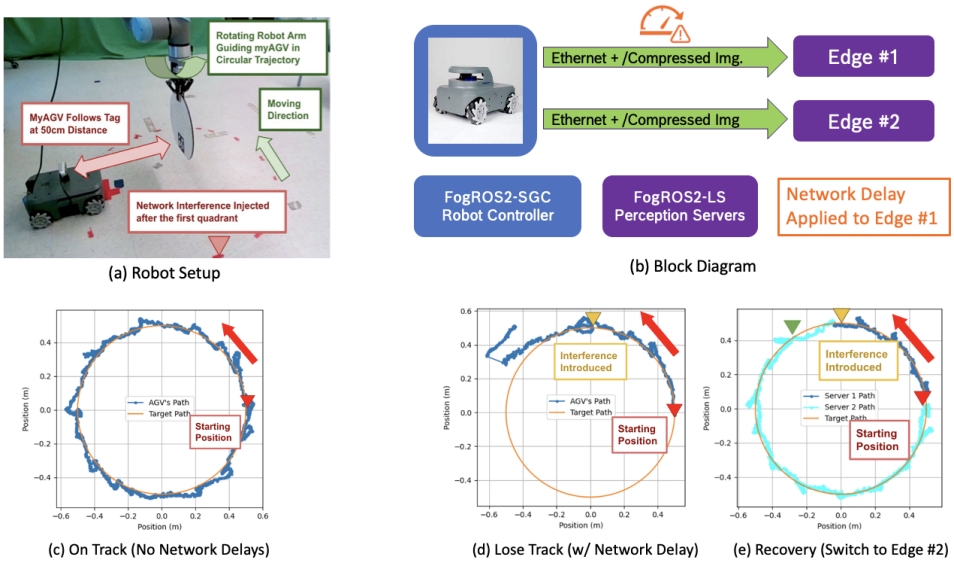


Figure 5.7: **Position of Mobile Robot Relative to a Circular Path**. (a) & (b) Setup & Architecture; (c), the mobile robot successfully follows the circular path when no network inference was introduced; (d), a 50 ms network delay was introduced after the initial quadrant, causing mobile robot to immediately deviate from the path; (e) showcases FogROS2-LS's ability to recover from the failure encountered by transitioning to another Edge server.

at a given moment. It can achieve an even lower failure rate of up to 1.5 times lower than that of any single machine available.

In all the series of experiments, one failure case occurred–when 50% latency was applied to Edge 1 and FogROS2-LS offload perception to Cloud US East. The failure rate of the combined Cloud-Edge was worse than that of the original Edge 1 machine. This outcome was caused by imprecise network profiling due to the following reasons: (1) substantial network variation for Edge and Cloud is not practical for optimization; (2) while the FogROS2-LS network profiler operates under the assumption of uncorrelated request latencies, the H.264 compression alternates between transmitting a complete image frame and sending only the difference from the previous frame, so request latencies can be correlated in time.

5.3.3 Continuous Target Following

Setup We used MyAGV [7] for a continuous target following experiment. This fast-moving mobile vehicle has Mecanum omnidirectional wheels, a Raspberry Pi 4, and an Intel RealSense D435i camera. During the experiment, it tracked the QR code on the UR10e robot arm such that the robot aimed to stay 0.5 meters away in the normal direction of the tag. The wrist of UR10e maintains a rotation speed of 0.075 rad/s, guiding mobile robot in a circular path. In the meantime, the Raspberry Pi runs FogROS2-LS to stream the RGB video to the Edge server at 424x240 resolution at 30 Hz with an Ethernet cable. The position of MyAGV w.r.t. the start is estimated as $R(\theta)x$, where $R(\theta)$ is a 2x2 rotation matrix with angle θ, θ is the angle reading of the UR10's wrist joint,

and x is the detected 2x1 position of MyAGV's camera w.r.t. the QR code. In the evaluation, we use two identical Edge servers connecting to MyAGV via FogROS2-LS, one executes QR Code detection, while a backup Edge server remains on standby. While the MyAGV progresses past the first quadrant (90 degrees) of the target circluar path, we introduce a 50ms network latency, causing a tracking failure and deviation from the trajectory. A successful FogROS2-LS switch should facilitate tracking recovery, allowing the system to resume object tracking by leveraging the backup Edge server.

Results Fig. 5.7 shows the the results of target following with FogROS2-LS. In the presence of 50 ms network latency after the initial quadrant, MyAGV deviates from the path. FogROS2-LS is able to recover from the failure encountered in (e) by transitioning to another Edge server.

Chapter 6

Current and Future Work

This thesis serves as a checkpoint to the network infrastructure design and implementation of FogROS2-SGC. In order to realize the federated vision that everyone may contribute his or her own resources and enable secure and efficient computation and communication, we recognize future works from the following aspects: (1) scalability, (2) Quality-of-Service, (3) cellular network integration.

6.1 Quality of Service with FogROS2-FT

Cloud services can suffer occasional downtime, and connectivity between the robot and cloud are prone to variations in network Quality-of-Service (QoS). We are working on FogROS2-FT (Fault Tolerant) to mitigate these issues by introducing a multi-cloud extension that automatically replicates independent stateless robotic services, routes requests to these replicas, and directs the first response back. With replication, robots can still benefit from cloud computations even when a cloud service provider is down or there is low QoS. Additionally, many cloud computing providers offer low-cost "spot" computing instances that may shutdown unpredictably. Normally, these low-cost instances would be inappropriate for cloud robotics, but the fault tolerance nature of FogROS2-FT allows them to be used reliably.

6.2 Cellular Network Integration

We consider it as future work to integrate FogROS2-SGC with cellular 5G network. We leverage redundant network interfaces available to robots, such as Wi-Fi and 5G, to improve the reliability of the cloud robotics. This can further improve the reliabilty of mobile robots that may stream data to the cloud with home or enterprise Wi-Fi network, while use 5G network to maintain a reliable and constant network to the cloud. For example, if the robot is in a current network with high network latency variance, the connectivity framework automatically switch to 5G network or duplicate the message for both network interfaces.

6.3 Scalability

The current FogROS2-SGC is designed for the connection among a small number of robots and cloud services that are mutually connected. It relies on a centralized signaling server to facilitate the connection. We consider it as future work to extend the centralized signaling server to a distributed storage for storing persistent connections at different sub-networks.

Bibliography

[1] image_transport. wiki.ros.org/image_transport.

[2] UC Berkeley Minimum Security Standard. https://security.berkeley.edu/policy/minimum-security-standards-networked-devices-mssnd.

[3] RTI Connext DDS. https://www.rti.com/products.

[4] Signature and hash algorithms for TLS and DTLS. https://www.ibm.com/docs/en/zos/2.5.0?topic=support-signature-hash-algorithms.

[5] Scalable Distributed Robot Fleet With Fast DDS Discovery Server. https://husarnet.com/blog/ros2-dds-discovery-server. Accessed: 2023-03-1.

[6] Multicast DNS RFC 6762. https://www.rfc-editor.org/rfc/rfc6762.html.

[7] My AGV Mobile Robot. https://shop.elephantrobotics.com/products/myagv.

[8] rosduct. https://github.com/uts-magic-lab/rosduct.

[9] RSASSA-PSS RFC 4056. https://www.rfc-editor.org/rfc/rfc4056.

[10] wireguard VPN. https://www.wireguard.com/.

[11] Integrating ROS2 with Eclipse zenoh. https://zenoh.io/blog/2021-04-28-ros2-integration/. Accessed: 2021-02-15.

[12] Raghav Anand, Jeffrey Ichnowski, Chenggang Wu, Joseph M Hellerstein, Joseph E Gonzalez, and Ken Goldberg. Serverless multi-query motion planning for fog robotics. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2021.

[13] Andrew W Appel. Verification of a cryptographic primitive: Sha-256. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 37(2):1–31, 2015.

[14] Hitesh Ballani and Paul Francis. Towards a global ip anycast service. *ACM SIGCOMM Computer Communication Review*, 35(4):301–312, 2005.

[15] Stefan Bauer, Felix Widmaier, Manuel Wüthrich, Niklas Funk, Julen Urain De Jesus, Jan Peters, Joe Watson, Claire Chen, Krishnan Srinivasan, Junwu Zhang, Jeffrey Zhang, Matthew R. Walter, Rishabh Madan, Charles B. Schaff, Takahiro Maeda, Takuma Yoneda, Denis Yarats, Arthur Allshire, Ethan K. Gordon, Tapomayukh Bhattacharjee, Siddhartha S. Srinivasa, Animesh Garg, Annika Buchholz, Sebastian Stark, Thomas Steinbrenner, Joel Akpo, Shruti Joshi, Vaibhav Agrawal, and Bernhard Schölkopf. A robot cluster for reproducible research in dexterous manipulation. *arXiv preprint arXiv:2109.10957*, 2021.

[16] Erik Boasson, Angelo Corsaro, and Hans van t Hag. Eclipse Cyclone DDS. `https://projects.eclipse.org/projects/iot.cyclonedds`.

[17] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[18] Kaiyuan Chen, Ryan Hoque, Karthik Dharmarajan, Edith LLontop, Simeon Adebola, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. FogROS2-SGC: A ROS2 Cloud Robotics Platform for Secure Global Connectivity. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2035–2042. IEEE, 2021.

[19] Kaiyuan Chen, Jiachen Yuan, Nikhil Jha, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. FogROS G: Enabling secure, connected and mobile fog robotics with global addressability. *arXiv preprint arXiv:2210.11691*, 2022.

[20] Kaiyuan Chen, Kush Hari, Rohil Khare, Charlotte Le, Trinity Chung, Jaimyn Drake, Simeon Adebloa, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. FogROS2-Config: A toolkit for choosing server configuration for cloud robotics. *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[21] Kaiyuan Eric Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. FogROS: An adaptive framework for automating fog robotics deployment. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2035–2042. IEEE, 2021.

[22] S Chinchali, A Sharma, J Harrison, A Elhafsi, D Kang, E Pergament, E Cidon, S Katti, and M Pavone. Network offloading policies for cloud robotics: a learning-based approach. arxiv e-prints. *arXiv preprint arXiv:1902.05703*, 2019.

[23] C. Crick, G. Jay, S. Osentoski, and O. C. Jenkins. ROS and Rosbridge: Roboticists out of the loop. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 493–494, 2012. doi: 10.1145/2157689.2157846.

[24] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. URL `http://ompl.kavrakilab.org`.

[25] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[26] Arthur Stanley Eddington. *The mathematical theory of relativity*. The University Press, 1923.

[27] eProsima. Fast DDS. `https://www.eprosima.com/index.php/products-all/eprosima-fast-dds`.

[28] Siva Leela Krishna Chand Gudi, Suman Ojha, Benjamin Johnston, Jesse Clark, and Mary-Anne Williams. Fog robotics for efficient, fluent and robust human-robot interaction. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–5. IEEE, 2018.

[29] SLK Chand Gudi et al. Fog robotics: An introduction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[30] Sami Salama Hussen Hajjaj and Khairul Saleh Mohamed Sahari. Establishing remote networks for ROS applications via port forwarding: A detailed tutorial. *International Journal of Advanced Robotic Systems*, 14(3):1729881417703355, 2017.

[31] Ryan Hoque, Lawrence Yunliang Chen, Satvik Sharma, Karthik Dharmarajan, Brijen Thananjeyan, Pieter Abbeel, and Ken Goldberg. Fleet-DAgger: Interactive robot fleet learning with scalable human supervision. *Conference on Robot Learning (CoRL)*, 2022.

[32] Jeffrey Ichnowski and Ron Alterovitz. Scalable multicore motion planning using lock-free concurrency. *IEEE Trans. Robotics*, 30(5):1123–1136, 2014.

[33] Jeffrey Ichnowski and Ron Alterovitz. Motion planning templates: A motion planning framework for robots with low-power CPUs. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019.

[34] Jeffrey Ichnowski, William Lee, Victor Murta, Samuel Paradis, Ron Alterovitz, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. Fog robotics algorithms for distributed motion planning using lambda serverless computing. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4232–4238, 2020.

[35] Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Vıctor Mayoral-Vilches, Hugo Zhan, Derek Xu, Ramtin Ghassemi, John Kubiatowicz, et al. FogROS2: An adaptive and extensible platform for cloud and fog robotics using ros 2. *arXiv preprint arXiv:2205.09778*, 2022.

[36] Ichnowski Jeffrey and Alterovitz Ron. Concurrent nearest-neighbor searching for parallel sampling-based motion planning in SO(3), SE(3), and Euclidean spaces. In *Workshop on the Algorithmic Foundation of Robotics (WAFR)*. Springer, 2018.

[37] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip. In *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*, pages 1106–1113, 2012.

[38] Ben Kehoe, Dmitry Berenson, and Goldberg Ken. Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 576–583, 2012.

[39] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-based robot grasping with the google object recognition engine. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4263–4270, 2013.

[40] Ben Kehoe, Deepak Warrier, Sachin Patil, and Ken Goldberg. Cloud-based grasp analysis and planning for toleranced parts using parallelized Monte Carlo sampling. *IEEE Trans. Automation Science and Engineering*, 12(2):455–470, 2014.

[41] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Trans. Automation Science and Engineering*, 12(2):398–409, 2015.

[42] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. *arXiv preprint arXiv:2303.09553*, 2023.

[43] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[44] Miu-Ling Lam and Kit-Yung Lam. Path planning as a service PPaaS: Cloud-based robotic path planning. In *Proc. IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pages 1839–1844, 2014.

[45] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[46] Pusong Li, Bill DeRose, Jeffrey Mahler, Juan Aparicio Ojea, Ajay Kumar Tanwani, and Ken Goldberg. Dex-Net as a service (DNaaS): A cloud-based robust robot grasp planning system. In *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*, pages 1420–1427, 2018.

[47] Jia Zhi Lim and Danny Wee-Kiat Ng. Cloud based implementation of ROS through VPN. In *Int. Conf. on Smart Computing & Communications (ICSCC)*, pages 1–5. IEEE, 2019.

[48] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66): eabm6074, 2022.

[49] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-Net 2.0: Deep learning to plan robust grasps

with synthetic point clouds and analytic grasp metrics. In *Proc. Robotics: Science and Systems (RSS)*, 2017.

[50] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019.

[51] Nicholas D Matsakis and Felix S Klock II. The rust language. In *ACM SIGAda Ada Letters*, volume 34, pages 103–104. ACM, 2014.

[52] Vıctor Mayoral-Vilches, Ruffin White, Gianluca Caiazza, and Mikael Arguedas. SROS2: Usable cyber security tools for ros 2. *arXiv e-prints*, pages arXiv–2208, 2022.

[53] Garrett McGrath and Paul R Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410. IEEE, 2017.

[54] Gajamohan Mohanarajah, Dominique Hunziker, Raffaello D'Andrea, and Markus Waibel. Rapyuta: A cloud robotics platform. *IEEE Trans. Automation Science and Engineering*, 12 (2):481–493, 2014.

[55] Nitesh Mor, Richard Pratt, Eric Allman, Kenneth Lutz, and John Kubiatowicz. Global data plane: A federated vision for secure data in edge computing. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1652–1663. IEEE, 2019.

[56] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robotics*, 33(5):1255–1262, 2017.

[57] Michael Myers, Carlisle Adams, Dave Solo, and David Kemp. Internet x. 509 certificate request message format. Technical report, 1999.

[58] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 3859–3866, 2012.

[59] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. {NetBricks}: Taking the v out of {NFV}. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 203–216, 2016.

[60] Alyson Benoni Matias Pereira, Ricardo Emerson Julio, and Guilherme Sousa Bastos. ROS-Remote: Using ROS on cloud to access robots remotely. In *Robot Operating System (ROS)*, pages 569–605. Springer, 2019.

[61] Adam Rashid, Chung Min Kim, Justin Kerr, Letian Fu, Kush Hari, Ayah Ahmad, Kaiyuan Chen, Huang Huan, Marcus Gualtieri, Michael Wang, Christian Juette, Nan Tian, Liu Ren, and Ken Goldberg. Lifelong lerf: Local 3d semantic inventory monitoring using fogros2. *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[62] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[63] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks. *IEEE Robotics & Automation Letters*, 2019.

[64] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. Webrtc technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 1006–1009. IEEE, 2015.

[65] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2012.

[66] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A versatile visual slam framework. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, pages 2292–2295, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6. doi: 10.1145/3343031.3350539. URL `http://doi.acm.org/10.1145/3343031.3350539`.

[67] Gokul Swamy, Siddharth Reddy, Sergey Levine, and Anca D Dragan. Scaled autonomy: Enabling human operators to control robot fleets. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5942–5948. IEEE, 2020.

[68] Ajay Kumar Tanwani, Nitesh Mor, John Kubiatowicz, Joseph E Gonzalez, and Ken Goldberg. A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4559–4566. IEEE, 2019.

[69] Ajay Kumar Tanwani, Raghav Anand, Joseph E Gonzalez, and Ken Goldberg. RILaaS: Robot inference and learning as a service. *IEEE Robotics & Automation Letters*, 5(3):4423–4430, 2020.

[70] Nan Tian, Matthew Matl, Jeffrey Mahler, Yu Xiang Zhou, Samantha Staszak, Christopher Correa, Steven Zheng, Qiang Li, Robert Zhang, and Ken Goldberg. A cloud robot system using the dexterity network and Berkeley robotics and automation as a service (BRASS). In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1615–1622, 2017.

[71] Nan Tian, Ajay Kummar Tanwani, Jinfa Chen, Mas Ma, Robert Zhang, Bill Huang, Ken Goldberg, and Somayeh Sojoudi. A fog robotic system for dynamic visual servoing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1982–1988. IEEE, 2019.

[72] George Tsirtsis and Pyda Srisuresh. Network address translation-protocol translation (nat-pt). Technical report, 2000.

[73] John Wang and Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016.

[74] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021.

[75] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[76] Binhuai Xu and Jing Bian. A cloud robotic application platform design based on the microservices architecture. In *Int. Conf. on Control, Robotics and Intelligent System*, pages 13–18, 2020.

[77] Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. Ds-slam: A semantic visual slam towards dynamic environments. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1168–1174. IEEE, 2018.