UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

REAL TIME CONTROL FOR INTELLIGENT 6G NETWORKS

POL GONZÁLEZ PACHECO

Thesis supervisor: LUIS DOMINGO VELASCO ESTEBAN (Department of Computer Architecture)

Thesis co-supervisor: MARC RUIZ RAMÍREZ (Department of Computer Architecture)

Degree: Master Degree in Innovation and Research in Informatics (Computer Networks and Distributed Systems)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

27/06/2023

Acknowledgements

The research leading to these results has received funding from the H2020 B5G-OPEN (G.A. 101016663), the European Commission thought the HORIZON SNS JU DESIRE6G (G.A. 101096466) and the MICINN IBON (PID2020-114135RB-I00) projects, and from ICREA.

Abstract

The benefits of telemetry for optical networking have been shown in the literature, and several telemetry architectures have been defined. In general, telemetry data is collected from observation points in the devices and sent to a central system running besides the Software Defined Networking (SDN) controller. In this project, we try to develop a telemetry architecture that supports intelligent data aggregation and nearby data collection. Several frameworks and technologies have been explored to ensure that they fit well into the architecture's composition. A description of these different technologies is presented in this work, along with a comparison between their main features and downsides. Some intelligent techniques, aka. Algorithms have been stated and tested within architecture, showing their benefits by reducing the amount of data processed. In the design of this architecture, the main issues related to distributed systems have been faced, and some initial solutions have been proposed. In particular, several security solutions have been explored to deal with threats but also with scalability and performance issues, trying to find a balance between performance and security. Finally, two use cases are presented, showing a real implementation of the architecture that has been presented at conferences and validated within the project's development.

Contents

List of Figures

List of Tables

Acronyms

2FA Two-Factor Authentication. 34

5G Fifth-generation wireless. 1

6G Sixth-generation wireless. i–iii, 1–3, 7, 37, 44, 48, 49

ABAC Attribute-Based Access Control. 35

ACL Access Control List. 35

AE Auto Encoders. 43, 45

AI Artificial Intelligence. 2

AR/VR Augmented/Virtual Reality. 45

B5G Beyond 5G. i, 7, 8, 49

BER Bit Error Rate. 18

BFT Byzantine Fault-Torent. 35

DDoS Distributed Denial of Service. 35, 36

DLT Distributed Ledger Technology. 28, 36–39, 48, 49

DoS Denial of Service. 35

E2E End-to-End. 35, 44, 48

FT Free Text Search. 21

GMM Gaussian Mixture Models. 43

gNMI gRPC Network Management Interface. 11, 12, 15, 16

GODAI Generic mOdule for Distributed Artificial Intelligence. iii, 22–26, 28–33, 36, 40, 44, 48, 49

gRPC Google Remote Procedure Calls. 7, 11, 12, 15, 16, 36, 40–43

IDE	Integrated Development Environment. 32
IDS	Intrusion Detection system. 35
IETF	Internet Engineering Task Force. 6
INT	In-band Network Telemetry. 44
IoT	Internet of Things. 7, 9, 34
IP	Internet Protocol. 5, 28
IPS	Intrusion Prevention Systems. 35
JSON	JavaScript Object Notation. 6, 18, 24, 42
LSP	Label-Switched Path. 21
MAS	Multi-Agent System. iii, 37, 39, 44–46, 48
MB	MultiBand. 40
MIB	Management Information Base. 5, 6
ML	Machine Learning. 20, 41
NOS	Network Operating System. 32
OA	Optical Amplifiers. 17
OID	Object Identifier. 5
OSA	Optical Spectrum Analyzer. 17, 18, 21, 41
PKI	Public Key Infraestructure. 35
Pub/Sub	Publish/Subscribe. iii, 13–16, 22, 24
QAM	Quadrature Amplitude Modulation. 43
QoS	Quality of Service. 1–3
RAN	Radio Access Network. 44, 46
RBAC	Role-Based Access Control. 34
REST API	Representational State Transfer API. 23–25, 36
ROADM	Re-configurable Optical Add-drop Multiplexers. 21

RPC Remote Procedure Calls. 11

Rx Receivers. 17, 18

SDN Software Defined Networks. 20, 21, 40–42, 45, 48

SLA Service Level Agreement. 46

SMO Service Management and Orchestration. 46

SNMP Simple Network Management Protocol. iii, 5, 6, 15, 16, 32

SSL/TLS Secure Sockets Layer/Transport Layer Security. 34, 36

TAPI Transport API. 20, 40

TCP Transmission Control Protocol. 7, 15

TS Time Series. 21

Tx Transmitters. 17, 18

UAV Unmanned Aerial Vehicle. 45

UDP User Datagram Protocol. 7, 15

VNF Virtual Network Functions. 44, 45

VXLAN Virtual Extensible Local Area Network. 37, 39

Web-UI Web User Interface. iii, 25, 30, 31, 49

XML eXtensible Markup Language. 6

YANG Yet Another Next Generation. 7, 12, 16

Chapter 1

Introduction

Real-time control will play a crucial role in shaping the intelligence and capabilities of Sixth-generation wireless (6G) networks, which are set to revolutionize the way we communicate and interact with technology. As the next generation of wireless technology, 6G aims to surpass its predecessor, Fifth-generation wireless (5G), in terms of data speeds, capacity, latency, and connectivity. However, to fully realize the potential of 6G networks and enable innovative applications, sophisticated real-time control mechanisms are essential. [1] This thesis explores the significance of real-time control in intelligent 6G networks and examines its key aspects and implications for the future of communications.

- **Resource Management**: Real-time control mechanisms in intelligent 6G networks facilitate efficient resource management. With the exponential growth of data and connected devices, the allocation and optimization of network resources become paramount. Real-time control enables dynamic resource allocation based on changing network conditions, user demands, and application requirements. This ensures that resources such as bandwidth, spectrum, and computing power are allocated in an optimal manner, maximizing network efficiency and delivering a seamless user experience.

- **Latency Reduction**: One of the defining characteristics of 6G is its ultra-low latency, enabling near real-time communication. Real-time control mechanisms play a crucial role in reducing latency by optimizing network routing, minimizing packet loss, and prioritizing critical traffic. These mechanisms enable mission-critical applications such as remote surgery, autonomous vehicles, and industrial automation, where even milliseconds of latency can have significant consequences.

- **Network Slicing and QoS**: Real-time control facilitates network slicing, a technique that partitions the network into multiple virtual networks tailored to specific applications or user groups. Each network slice can have its Quality of Service (QoS) characteristics, allowing for customized service delivery based on specific requirements. Real-time control enables the dynamic creation, management, and optimization of network slices, ensuring the appropriate allocation of resources and QoS guarantees for diverse applications.

- **AI-Driven Control**: The integration of Artificial Intelligence (AI) into real-time control mechanisms further enhances the intelligence of 6G networks. AI algorithms can analyze vast amounts of data in real-time, enabling predictive and proactive decision-making. AI-driven control mechanisms can dynamically adapt to changing network conditions, predict network demands, and optimize resource allocation. By leveraging AI, real-time control in 6G networks becomes more autonomous, self-optimizing, and capable of providing personalized services to users.

- **Security and Privacy**: Real-time control also plays a vital role in ensuring the security and privacy of 6G networks. As the number of connected devices and the volume of sensitive data increases, the network must be capable of identifying and mitigating security threats in real-time. Real-time control mechanisms can monitor network traffic, detect anomalies, and initiate prompt security measures to safeguard the network and user data.

- **Energy Efficiency**: Intelligent real-time control mechanisms contribute to the energy efficiency of 6G networks. By monitoring and managing network resources in real-time, power consumption can be optimized based on demand and network conditions. Real-time control enables intelligent sleep modes, adaptive transmission power control, and efficient resource utilization, resulting in reduced energy consumption and environmental impact.

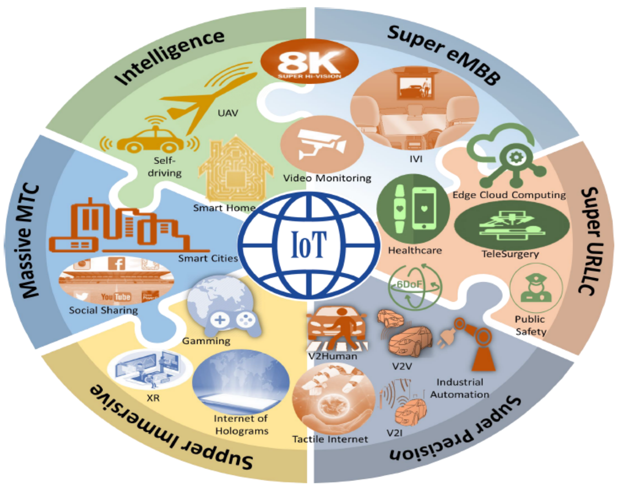


Figure 1.1: Services and use-cases for 6G [2]

Telemetry is a key element in 6G networks by enabling the collection, monitoring, and analysis of data from various network components and devices.[3] It provides real-time insights into network performance, resource utilization, and user behavior, allowing for intelligent decision-making and optimization. In the context of 6G networks, telemetry holds significant importance in several areas:

- **Network Performance Monitoring**: Telemetry allows network operators to continuously monitor and assess the performance of 6G networks. It provides real-time visibility into metrics such as latency, throughput, packet loss, and network congestion. By collecting and analyzing this data, operators can identify bottlenecks, optimize network configurations, and ensure optimal performance for various applications and services.

- **QoS Management**: Telemetry enables the measurement and monitoring of QoS parameters in 6G networks. It helps ensure that specific performance requirements, such as latency, jitter, and reliability, are met for different applications and services. By actively monitoring QoS metrics, network operators can proactively detect and address issues that may impact user experience and take corrective actions to maintain desired service levels.

- **Security and Anomaly Detection**: Telemetry plays a critical role in network security by providing real-time insights into potential threats and anomalies. By monitoring network traffic and behavior, telemetry can identify patterns indicative of security breaches, unauthorized access attempts, or abnormal activities. This early detection enables rapid response, allowing network operators to mitigate risks, safeguard sensitive data, and ensure the integrity of 6G networks.

- **Network Planning and Optimization**: Telemetry data is instrumental in network planning and optimization for 6G deployments. It provides valuable information about coverage, signal strength, and user density, helping operators identify areas for network expansion, capacity upgrades, and infrastructure optimization. By analyzing telemetry data, operators can make informed decisions regarding the placement of base stations, antenna configurations, and network densification, ensuring optimal coverage and performance.

In conclusion, telemetry plays a vital role in 6G networks by enabling real-time monitoring, analysis, and optimization. It empowers network operators to proactively manage network performance, allocate resources efficiently, detect security threats, enhance user experiences, and optimize network planning. By harnessing telemetry data, 6G networks can deliver superior performance, adapt to dynamic requirements, and support the diverse range of applications and services envisioned for the future.

In this project, we try to analyze the existing telemetry technologies in modern networks as well as the already defined architectures for telemetry in networks. With this knowledge, we will try to build a novel distributed telemetry architecture enabling real-time control in 6G networks. Furthermore, this architecture has to be flexible to be suitable for numerous types of scenarios, depending on the requirements and the available resources. For this reason, not only an architecture has to be designed, but also a framework has to be available to anyone wanting to build their own architecture based on their needs. This framework/architecture should include the new technologies available right now, such as machine learning algorithms, distributed system patterns, containerization, etc.

This is a summary of the main objectives or contributions to be achieved in this work:

- Analyze the existing telemetry technologies and their architectures.
- Analyze the needs of 6G services in future networks and the key enablers of them.
- Design, build, and test a new telemetry architecture enabling the previously studied requirements.
- Build use cases to validate this novel architecture.

The remainder of the document is organized as follows: In Chapter 2, several telemetry technologies and architectures are analyzed and compared to witness their main benefits and disadvantages. In Chapter 3, the problem statement is presented, as well as the network architecture and the proposed architecture to solve this problem. Chapter 4 focuses on the security aspects of telemetry systems, in particular the one developed in this work. In Chapter 5, two use cases where the new architecture is used are presented to showcase the main benefits and capabilities of the newly developed architecture. Finally, in Chapter 6, some conclusions and final thoughts are shared to summarize the whole work done.

Chapter 2

Background on Telemetry Systems

2.1 SNMP and Telemetry

Simple Network Management Protocol (SNMP) is an application-layer protocol that facilitates the exchange of management information between network devices and a central management system.[4] It operates on the Internet Protocol (IP) suite and allows administrators to monitor and control network devices, such as routers, switches, and servers, regardless of their geographical location.

SNMP utilizes a client-server model, where SNMP agents reside on managed devices, and SNMP managers control and monitor these agents. Agents collect and store management information, which can be accessed by managers through requests and notifications.

The SNMP architecture comprises three main components:

- **SNMP Managers**: SNMP managers are the centralized control systems responsible for monitoring and managing network devices. They send requests to SNMP agents and receive responses containing management data. Managers can retrieve information, configure device settings, and receive event notifications from agents.
- **SNMP Agents**: SNMP agents are software modules residing on managed devices, such as routers, switches, or servers. They collect and store management information, respond to SNMP manager requests, and send notifications known as traps or informs. Agents maintain a Management Information Base (MIB), which is a hierarchical database that organizes the data accessible through SNMP.
- **MIB**: The MIB is a structured collection of managed objects that represent various aspects of a network device. It organizes information into a tree-like structure, with each object having a unique identifier called an Object Identifier (OID). The MIB defines the available parameters and their corresponding data types that can be accessed through SNMP. Standard MIBs are defined

by organizations such as the Internet Engineering Task Force (IETF), while device vendors can also provide custom MIBs.

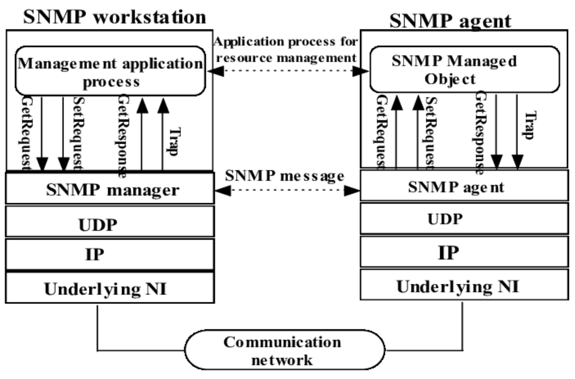


Figure 2.1: SNMP architecture [5]

SNMP operations involve the exchange of messages between SNMP managers and agents. The most common SNMP operations are:

- **Get**: The SNMP manager sends a Get request to an agent to retrieve the value of one or more specific variables in the MIB. The agent responds with a Get Response message containing the requested values.
- **Set**: The SNMP manager uses the Set operation to modify the value of one or more variables in the agent's MIB. The agent processes the request and returns a Set Response message indicating the success or failure of the operation.
- **GetNext**: The GetNext operation allows the SNMP manager to retrieve the next variable in the MIB, relative to a specified variable. This operation is useful for traversing the MIB tree and retrieving multiple variables.
- **Trap/Inform**: Agents can proactively send unsolicited notifications to managers using traps or informs. Traps are one-way notifications that do not require a response, while informs are similar but expect an acknowledgment from the manager. Traps and informs are typically sent to alert managers about critical events, such as device failures or excessive network traffic.

On the other hand, we find streaming network telemetry which uses a push model instead of the pull model used by SNMP. This push model enables continuous high resolution device operational data sent to a network management system. This method allows sending data at a higher rate and lower impact, as it does not need to create a request each time the data wants to be retrieved. Then data is configured to be streamed with a periodic cadence that could be sub-second if needed.

The data can be encoded in different formats including eXtensible Markup Language (XML), JavaScript Object Notation (JSON) or Google protocol buffers (also known as protobuf). In the case of the transport protocol it can be either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), it is frequently used with Google Remote Procedure Calls (gRPC) [6] which efficiently conveys data from one device to another. Model-driven telemetry is based on Yet Another Next Generation (YANG) models that can simplify the selection of data to stream. The definition and implementation of these models is held by the OpenConfig working group that is standardizing models to access a bunch of different network devices.

This new methodology of retrieving data also has new implications, as the data streamed from a moderately sized network can be huge. Such quantity of data may require big data storage and processing mechanisms to aggregate or extract valuable information of it. A proper configuration in terms of cadence and volume is required by network managers to not overwhelm the capabilities of the network itself.

2.2 Telemetry in B5G and 6G Networks

As the demand for faster and more reliable wireless communication continues to grow, the development of Beyond 5G (B5G) networks has emerged. These networks aim to provide unprecedented levels of connectivity and performance, enabling a wide range of applications and services. Central to the success of B5G networks is the implementation of robust telemetry architectures. Telemetry refers to the automated collection and transmission of data from remote devices or systems. In the context of B5G networks, telemetry architectures play a vital role in monitoring and managing network infrastructure, optimizing performance, and enhancing the overall user experience. These architectures leverage advanced technologies and techniques to enable real-time data gathering, analysis, and decision-making. [7]

One of the key features of telemetry architectures in B5G networks is their ability to collect data from a multitude of network elements and devices. These can include base stations, access points, edge servers, Internet of Things (IoT) devices, and even user equipment. This comprehensive data collection allows network operators and administrators to have a holistic view of the network's health, performance, and utilization. In B5G networks, telemetry architectures often rely on a combination of sensors, probes, and monitoring agents deployed throughout the network infrastructure. These components continuously gather various metrics, such as signal quality, traffic load, latency, packet loss, and energy consumption. The collected data is then aggregated and transmitted to a centralized management system for analysis and processing.

To facilitate the transmission of telemetry data, B5G networks utilize high-speed and low-latency communication technologies. These can include advanced wireless protocols like millimeter-wave (mmWave) and terahertz (THz) bands, as well as optical fiber networks. By leveraging these technologies, telemetry architectures can ensure timely and efficient data delivery, enabling near real-time monitoring and decision-making.

Moreover, telemetry architectures in B5G networks often incorporate advanced analytics and machine learning algorithms. These technologies enable the identification of patterns, anomalies, and performance trends within the collected telemetry data. By leveraging such insights, network operators can proactively detect and mitigate issues, optimize resource allocation, and deliver enhanced quality of service to end-users. Another important aspect of telemetry architectures in B5G networks is their scalability and flexibility. As B5G networks are designed to support a massive number of devices and applications, telemetry systems must be capable of handling large volumes of data and dynamically adapting to changing network conditions. This scalability allows operators to effectively monitor and manage networks with diverse requirements and traffic patterns.

2.3 Telemetry Architectures

Telemetry architectures define how to collect, process and analyze data from various sources in a systematic and efficient manner. Different telemetry architectures are designed to meet specific requirements and address different use cases.[8]

2.3.1 Centralized

A centralized telemetry architecture involves a monitoring system where telemetry data from diverse network elements is collected, processed, and analyzed in a centralized location. This architecture centralizes the telemetry data collection from base stations, routers, switches, and core network elements, ensuring a comprehensive view of the network. The collected data is transmitted to a central telemetry server, where it is stored and processed.

The centralized telemetry server employs sophisticated analytics tools and algorithms to extract valuable insights from the collected data. This allows for efficient network optimization, proactive troubleshooting, and performance monitoring. Network administrators and operators can access a unified dashboard or interface to visualize and analyze the telemetry data in real-time, enabling them to make informed decisions and take necessary actions promptly.

By utilizing a centralized telemetry architecture, network operators can gain a holistic understanding of their network's health and performance. This facilitates the identification and resolution of issues quickly, optimizes network efficiency, and aids in effective network planning and expansion. Additionally, centralized telemetry architecture enables efficient resource allocation, load balancing, and capacity management across the network, leading to improved reliability and overall user experience.

2.3.2 Distributed

A distributed telemetry architecture is a monitoring and management system that decentralizes the collection, processing, and analysis of telemetry data across multiple network elements. Unlike centralized architecture, where data is aggregated in a single location, distributed telemetry architecture distributes these tasks closer to the data sources. Telemetry data is collected from individual network devices, such as base stations, routers, switches, and core network elements, and processed locally within each device or at edge computing nodes. This approach reduces the amount of data transmitted across the network and minimizes latency.

By distributing telemetry data processing, distributed telemetry architecture enables real-time monitoring and decision-making at the edge of the network. Localized analytics tools and algorithms extract insights from the telemetry data, allowing for faster detection of anomalies and immediate response to network events. The distributed nature of the architecture also improves fault tolerance and resilience, as the failure of a single component does not disrupt the entire monitoring system. Operators and administrators can still access a centralized management system to oversee the distributed telemetry architecture. This system provides a unified view of the network, consolidating information from various distributed nodes. It allows for centralized configuration management, policy enforcement, and reporting.

A distributed telemetry architecture offers advantages such as reduced network congestion, lower latency, improved scalability, and enhanced resilience. It enables efficient monitoring and management of large-scale networks, particularly in scenarios where real-time decision-making and rapid response are critical, such as in autonomous vehicles, industrial IoT deployments, and mission-critical applications.

2.3.3 Hierarchical

A hierarchical telemetry architecture is a monitoring and management system that organizes telemetry data collection, processing, and analysis in a hierarchical structure. This architecture divides the network into multiple tiers or levels, each responsible for specific data aggregation and analysis tasks.

At the lower tier, telemetry data is collected from individual network devices such as base stations, routers, switches, and core network elements. These devices locally process and aggregate the data before transmitting it to a higher-level aggregation point or node in the hierarchy. This aggregation helps reduce the amount of data transmitted and enhances efficiency.

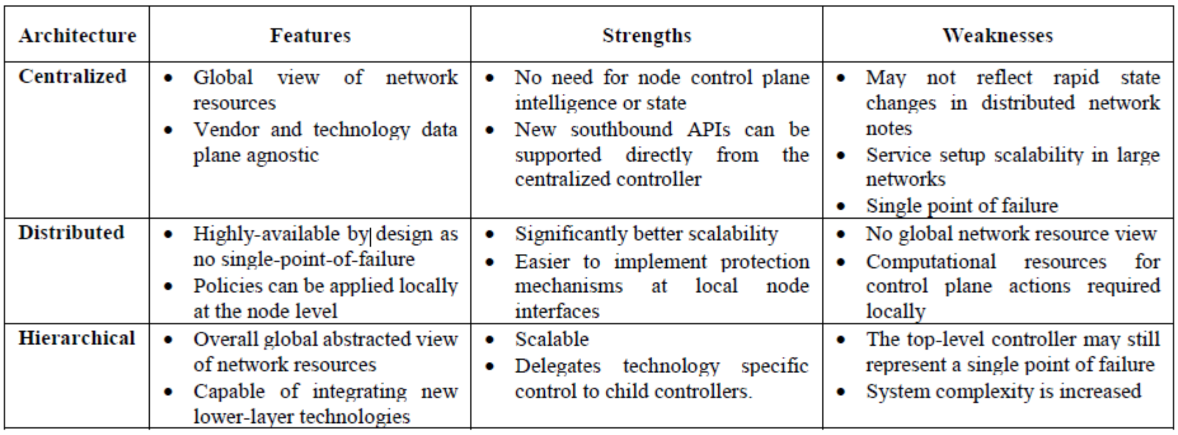
The intermediate tiers in the hierarchy further aggregate and analyze the telemetry data from lower-level nodes. They perform additional processing, filtering, and correlation to derive meaningful insights and detect anomalies or performance issues. This hierarchical approach enables a scalable and distributed telemetry infrastructure.

Finally, at the top tier, a centralized management system receives the aggregated telemetry data from all lower levels. The centralized systems employ advanced analytics tools and algorithms to generate comprehensive reports, perform in-depth analysis, and visualize the network's overall performance. Network operators and administrators can access a unified dashboard or interface to monitor and manage the entire network.

The hierarchical telemetry architecture offers several benefits, including scalability, efficient data aggregation, and optimized resource utilization. It allows for localized processing and analysis, reducing network congestion and latency. Furthermore, the hierarchical structure enables fault isolation and resilience, as issues can be detected and resolved at different levels of the architecture without impacting the entire system.

2.3.4 Comparison

The table defined in [8] summarizes the main features, strengths and weaknesses of the three previous described architectures.



Architecture	Features	Strengths	Weaknesses
Centralized	• Global view of network resources • Vendor and technology data plane agnostic	• No need for node control plane intelligence or state • New southbound APIs can be supported directly from the centralized controller	• May not reflect rapid state changes in distributed network notes • Service setup scalability in large networks • Single point of failure
Distributed	• Highly-available by design as no single-point-of-failure • Policies can be applied locally at the node level	• Significantly better scalability • Easier to implement protection mechanisms at local node interfaces	• No global network resource view • Computational resources for control plane actions required locally
Hierarchical	• Overall global abstracted view of network resources • Capable of integrating new lower-layer technologies	• Scalable • Delegates technology specific control to child controllers.	• The top-level controller may still represent a single point of failure • System complexity is increased

Table 2.1: Centralized, Distributed and Hierarchical Architectures

2.4 Communication Technologies

The choice of transport protocol depends on factors such as data volume, real-time requirements, reliability, and security considerations. It's worth noting that the choice of transport protocol may vary depending on the specific telemetry application, network requirements, and interoperability considerations. The selection of the appropriate transport protocol ensures efficient and reliable transmission of telemetry data.

2.4.1 gRPC and gNMI

gRPC and gNMI are two interconnected technologies that play significant roles in network applications, particularly in network management and communication.

gRPC, short for Google Remote Procedure Call, is an open-source Remote Procedure Calls (RPC) framework developed by Google.[6] It facilitates communication between different systems and services by defining the structure of messages and methods exchanged between them. gRPC supports multiple programming languages such as C++, Java, Python, and Go, allowing developers to create interoperable applications across different platforms. It utilizes HTTP/2 as the underlying transport protocol, which brings several advantages including high performance, multiplexing, header compression, and server-side streaming. This enables efficient and scalable communication between clients and servers. gRPC also offers bidirectional streaming, allowing both the client and server to send multiple messages asynchronously, making it well-suited for real-time communication scenarios. Additionally, gRPC provides built-in support for authentication mechanisms and load balancing strategies, ensuring secure and reliable communication between systems.

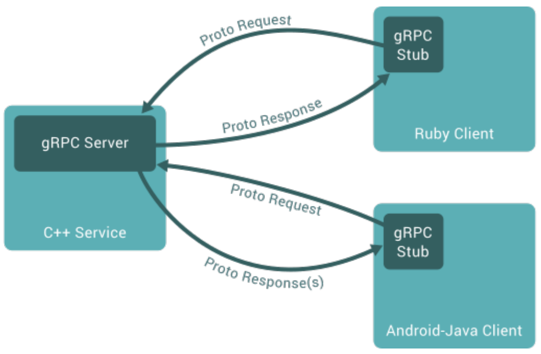


Figure 2.2: gRPC architecture [6]

gNMI, which stands for gRPC Network Management Interface, is a protocol that leverages gRPC as the transport mechanism for network management and telemetry operations.[9] It aims to simplify the management and monitoring of network devices by providing a standardized interface for accessing operational state, configuration, and telemetry data. gNMI follows a model-driven approach using the Yet Another Next Generation (YANG) modeling language, which allows for the definition of data models and schemes for network resources. This enables a standardized representation of network configurations and operational states across different devices and vendors. With gNMI, network management systems can easily retrieve and modify configuration parameters of network devices, ensuring consistency and ease of management. It also supports streaming telemetry, where network devices can continuously send operational data and statistics to management systems in real-time. This facilitates advanced monitoring, analysis, and troubleshooting capabilities for network administrators.

By combining the power of gRPC and gNMI, network applications can benefit from efficient and standardized communication between systems. gRPC provides a robust and flexible framework for inter-service communication, while gNMI enables simplified network management and telemetry operations. Together, they offer a comprehensive solution for building scalable, interoperable, and manageable network applications.

2.4.2 Apache Kafka

Apache Kafka is an open-source distributed event streaming platform that has gained significant popularity in recent years.[10] It was originally developed at LinkedIn to address the challenges of handling large-scale real-time data processing. Kafka provides a highly scalable, fault-tolerant, and durable messaging system that enables the efficient and reliable transfer of data between different systems and applications.

At its core, Kafka follows a publish-subscribe messaging model, where data is organized into topics. Producers, also known as publishers, write messages to these topics, and consumers, or subscribers, read and process these messages. Kafka ensures that messages are stored in an append-only log, retaining the order of arrival and guaranteeing durability. This log-based architecture makes Kafka an ideal choice for real-time streaming applications, where high-throughput and low-latency data processing are essential.

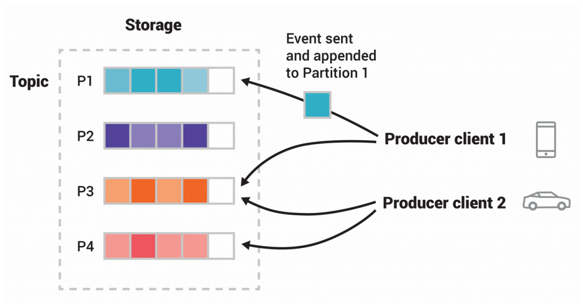


Figure 2.3: Apache Kafka messaging model [10]

One of the key features of Kafka is its ability to handle massive volumes of data. It is designed to be horizontally scalable, allowing it to handle high traffic loads and seamlessly distribute data across multiple brokers, which are the nodes in a Kafka cluster. Kafka clusters can be easily expanded or contracted to accommodate changing data demands, making it an ideal choice for dynamic and rapidly growing data environments.

Another important aspect of Kafka is its fault tolerance and resilience. Data in Kafka is replicated across multiple brokers, providing high availability and preventing data loss in the event of failures. Kafka also supports automatic leader election, ensuring that data continues to be available for consumption even if a broker goes down. This fault-tolerant design makes Kafka a robust and reliable platform for mission-critical applications.

Kafka also offers strong durability guarantees. Once data is written to a topic, it is persisted to disk, making it durable and allowing for efficient replay of messages. This durability is especially important in scenarios where data needs to be stored and retained for long periods, such as compliance and audit requirements. In addition to its core messaging capabilities, Kafka provides a rich set of features and integration options. It supports stream processing with Kafka Streams, allowing developers to build real-time applications that process and transform data streams in a scalable and fault-tolerant manner. Kafka Connect provides a framework for easily integrating Kafka with external systems, enabling seamless data pipelines between different applications and data sources.

Overall, Apache Kafka has emerged as a leading technology for building scalable, real-time data pipelines and event-driven architectures.[11] Its high throughput, fault tolerance, durability, and versatility make it well-suited for a wide range of use cases, including real-time analytics, log aggregation, data integration, and messaging systems. With a vibrant community and strong ecosystem support, Kafka continues to evolve and innovate, empowering organizations to harness the power of data and build robust, scalable, and event-driven applications.

2.4.3 Redis

Redis is an open-source, in-memory data structure store that serves as a highly efficient and versatile tool for caching, messaging, and real-time data processing.[12] It is designed for high performance and low latency, making it a popular choice for various use cases where speed and scalability are crucial. Redis utilizes key-value pairs to store data in memory, allowing for lightning-fast access and retrieval. It supports a wide range of data structures, including strings, lists, sets, sorted sets, hashes, and more, enabling developers to model complex data and perform advanced operations.

One of Redis's notable features is its Publish/Subscribe (Pub/Sub) messaging system. Redis Pub/Sub allows for asynchronous communication between different components of an application, or even separate applications altogether. It operates based on the Pub/Sub, where publishers send messages to specific channels, and subscribers receive those messages from the subscribed channels. Channels serve as communication pathways, acting as intermediaries for message broadcasting.

With Redis Pub/Sub, publishers and subscribers can decouple their interactions, creating a highly flexible and scalable architecture. Publishers can send messages to channels without needing to know who or how many subscribers exist. Subscribers, on the other hand, can subscribe to multiple channels, allowing them to receive relevant messages based on their interests. This loose coupling enables a high degree of flexibility and extensibility, making it easy to add new publishers or subscribers without impacting the existing components.

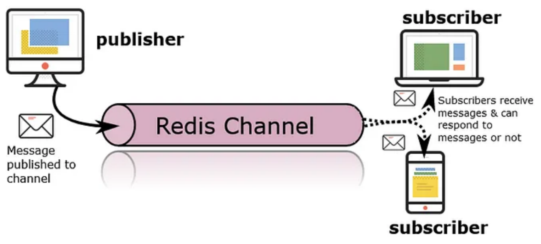

publisher
Message published to channel
Redis Channel
subscriber
Subscribers receive messages & can respond to messages or not
subscriber

Figure 2.4: Redis Pub/Sub [13]

Redis Pub/Sub supports both one-to-one and one-to-many messaging scenarios. In a one-to-one scenario, a publisher sends a message to a specific channel, and only the subscriber(s) listening to that channel will receive the message. In a one-to-many scenario, a publisher sends a message to a channel, and multiple subscribers listening to the same channel will all receive the message concurrently. This allows for efficient broadcasting of messages to a group of subscribers with minimal overhead.

In addition to the basic publish and subscribe operations, Redis Pub/Sub offers additional functionality to enhance the messaging system. This includes the ability to pattern subscribe, where subscribers can use wildcard patterns to match multiple channels for receiving messages. It also supports unsubscribing from channels, enabling dynamic subscription management based on changing requirements. Redis Pub/Sub also provides the capability to persist messages by leveraging Redis' persistence mechanisms, allowing subscribers to retrieve missed messages upon re-connection.

Overall, Redis and its Pub/Sub messaging system provide a powerful and efficient solution for building scalable and real-time applications. Its in-memory nature, extensive data structures, and the ability to handle high volumes of concurrent operations make Redis a popular choice for caching, session management, and more. With Redis Pub/Sub, developers can create loosely coupled and flexible systems, enabling efficient message broadcasting and real-time communication between various components of an application or even different applications in a distributed environment.

2.5 Comparison

In order to compare the different features and downsides of each proposed technology, a comparison is presented in terms of: purpose, communication protocols, flexibility, data types and extensibility.

2.5.1 Purpose

While SNMP and gNMI are also used for managing and configuring network devices, Apache Kafka and Redis Pub/Sub feature are completely focused on messaging offering real-time streaming communications between publishers and subscribers. In this sense, gRPC it is not just the communication framework used by gNMI, but a great option to take into consideration while building distributed systems with heterogeneous requirements. For this reason, we can clearly differentiate the technologies that could be more useful for exchanging information between different agents and which technologies could be more useful to configure network devices or get their operational status.

Said that, it is known that the actual paradigm of telemetry relies more on streaming telemetry than pull based telemetry, which is the one used in SNMP and gNMI primarily. For that, Apache Kafka and gRPC are becoming more and more relevant as most telemetry architectures are based in this second premise.

Finally, Redis is seen as multipurpose software in this specific use case, as it can be used for different purposes. His principal commitment is to be an in-memory key-based database, and it is frequently used as a cache behind databases. However, its Pub/Sub feature allows us to use Redis as message broker, keeping at the same time the other features that include. For this reason, Redis is being used widely thanks to its flexibility, performance and reduced resource requirements.

2.5.2 Communication Protocol

In general, all available options offer Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) as available transport protocol. gRPC features HTTP/2 protocol for communication enabling to set up several TCP connections at the same time, increasing the data rate the data could be sent.

2.5.3 Flexibility

In terms of flexibility, gRPC supports various programming languages and provides code generation capabilities for client and server stubs, making it easier to integrate different services. gNMI provides a standardized interface for network management, making it easier to configure and monitor network devices from different vendors. Redis and Apache Kafka both support many programming languages, and client libraries are available and documented to be used in almost any device.

2.5.4 Data Types

gRPC uses Protocol Buffers (protobufs) for defining service interfaces and data serialization. It supports structured data and message passing between services. However, a static definition of the data exchanged has to be setup and any further change implies recompiling the model. gNMI uses YANG data models for defining the structure and semantics of configuration and operational data exchanged between the network management system and devices. These YANG models can add complexity, but also offers a standard and general way to access the data conveyed by different network devices belonging to different vendors. Redis Pub/Sub and Apache Kafka support various data types, including strings, hashes, lists, sets, and more, allowing flexibility in message content.

2.5.5 Extensibility

gRPC allows for defining service methods, error handling, and custom metadata for advanced functionality. gNMI extends its capabilities by defining custom YANG data models for specific device configurations and monitoring requirements. Kafka offers an ecosystem of connectors and client libraries for integration with various data sources, sinks, and processing frameworks. Redis Pub/Sub can be extended with Redis Lua scripting and other Redis features for more advanced Pub/Sub functionalities.

2.6 Summary

In summary, SNMP is primarily used for network management and monitoring, gRPC and gNMI are focused on efficient communication and management of distributed systems and network devices, Apache Kafka is designed for real-time data pipelines and streaming applications, and Redis Pub/Sub provides a lightweight messaging system for Pub/Sub communication.

Chapter 3

Proposed Architecture

3.1 Motivation

The concept of the 5 V's in big data pertains to five essential qualities that aid in describing and comprehending the attributes of extensive and intricate datasets. These 5 V's serve as a framework for comprehending the distinctive features of big data and the difficulties linked to it. To fully utilize the immense potential of big data for their business goals, organizations must effectively tackle these aspects.

Let's provide the example in [14] of the optical core network for a national telecom operator to illustrate each of the 5 V's. We will consider a core mesh network consisting of 50 optical nodes, where the average number of connections per node is 3. In this scenario, each node is connected to every other node in the network through a single optical connection, or lightpath. Consequently, the network supports a total of 2,450 unidirectional lightpaths, requiring an equal number of Transmitters (Tx) and Receivers (Rx) to facilitate communication.

Furthermore, we assume that we can gather telemetry data from various components within the network. This includes the Tx, Rx, Optical Amplifiers (OA) located in the nodes, which compensate for filtering and fiber attenuation (totaling 300 OAs), and Optical Spectrum Analyzer (OSA) installed in each optical link throughout the network (totaling 150 OSAs).

- **Volume**: Suppose we gather measurements every second. In that case, the mentioned network produces 15.64 terabytes (TB) of data per day, which amounts to 5.58 petabytes (PB) per year. This data must be gathered, transmitted to the central telemetry system, stored in a data lake, and analyzed.
- **Velocity**: Collecting measurements at a frequency of one per second necessitates additional prerequisites concerning data collection from devices and its transfer to the centralized telemetry system. For instance, when considering optical devices that produce substantial measurements (such as OSAs

and optical receivers), high-speed data interfaces are required. For instance, OSAs would necessitate 150 kilobits per second (kb/s) interfaces, while Rx would require 640 kb/s interfaces. These speeds are based on the assumption that measurements are generated as a continuous stream of floating-point numbers. However, once collected, these measurements are usually formatted, such as into a JSON object, which increases their size. In this particular scenario, each node agent responsible for gathering local telemetry data, formatting it, and transmitting it to the centralized telemetry system would generate approximately 40 megabits per second (Mb/s). As a result, the centralized system would receive a total of 1.9 gigabits per second (Gb/s) of telemetry data.

- **Variety**: There are six defined measurements that involve combinations of individual magnitudes and related value vectors. Furthermore, unstructured data from various systems, including events, is gathered, necessitating distinct processing methods. All of these diverse data types must be promptly processed, analyzed, and interconnected. For example, by examining spectrum measurements taken from nodes along a light path and analyzing IQ constellations in the Rx, it becomes possible to identify and locate the source of a sudden increase in the Bit Error Rate (BER) observed in the Rx.

- **Veracity**: Making informed and sound choices requires having comprehensive and accurate information. Data, in order to be beneficial, must possess certain qualities, such as accuracy, freedom from errors, reliability, consistency, lack of bias, and completeness. There are various factors that can taint data, including: i) irrelevant information that distorts the data; ii) outliers that cause the dataset to deviate from its usual patterns; iii) software vulnerabilities that could lead to data manipulation; and iv) statistical data that misrepresents a specific network resource.

- **Value**: Converting telemetry data into meaningful insights is crucial to fully harness the advantages it offers for network automation. There are various ways in which operators can derive value from telemetry data, including: i) decreasing network margins; ii) automating service provisioning; iii) enhancing resource utilization and minimizing operational expenses; iv) prolonging the lifespan of network equipment; v) identifying soft-failures before they escalate into hard failures; vi) simplifying maintenance by finding root cause of failures and scheduling works; and many others.

Let us challenge some of the previous assumptions aiming at bringing requirements to the telemetry architecture:

1. Various measurements require distinct collection frequencies, which may vary or occur asynchronously. For example, Tx settings are established during connection setup or triggered by specific events, whereas the laser's temperature does not undergo rapid changes.

2. Typically, storing every measurement without significant changes is not beneficial. Nevertheless, in order to identify noteworthy variations in a specific measurement, an analysis must be conducted. This analysis should ideally take place early in the telemetry pipeline, such as at the node level, in order to minimize the amount of data transmitted to the centralized telemetry system.

3. Compression techniques, which can be either lossy or lossless, can be explored to reduce bandwidth requirements.

4. Based on the previous two problems, it is advisable to introduce decentralization in telemetry systems. It would be beneficial to conduct certain data processing and analysis at the individual nodes. However, this analysis could be coordinated by a centralized entity operating at a higher level, which would have a comprehensive perspective of the entire network.

5. It is important to ensure the accuracy of data throughout the telemetry pipeline and remove any data that appears to be contaminated. This can be done by identifying samples that deviate from the statistical trends observed in previous measurements. Such deviations may indicate outliers or anomalies. The point of detection can be either localized, such as identifying an issue with the gain of an amplifier, or centralized, where correlation with other measurements is necessary. An example of centralized detection is when spectrum measurements along a lightpath route need to be analyzed together.

6. The extraction of value from data in the telemetry pipeline should be prioritized without delay. For instance, it is preferable to identify any degradation issues directly within the network node itself rather than waiting for the centralized telemetry system to detect them from the data collected. Nevertheless, there are situations where it becomes essential to analyze and correlate data from various network nodes in order to derive meaningful insights and extract value from the data.

In conclusion, to reduce the impact of the 5 V's, intelligence can be applied along the telemetry pipeline, which needs to be extended with new elements where telemetry measurements can be processed.

3.2 Problem Statement

The automation and management of optical networks necessitate the monitoring of network devices to detect potential issues and guarantee the reliability of services, particularly optical connectivity. To achieve this, extensive telemetry data must be collected from various sources, providing detailed measurements and events. However, due to the abundance and diversity of telemetry sources, as well as the size of each data point, meeting these requirements becomes challenging without significant investments. In this study, we examine the primary drawbacks of centralized data analysis systems in telemetry architectures and propose an alternative approach: a distributed intelligence architecture.

Telemetry is a significant area of study, and as a result, there is a wealth of research available on the subject [15], uncovering its advantages in optical networking, such as network automation and failure management. Similarly to Big Data, telemetry data in optical networks consist of a compilation of data from various sources and can be characterized using the 5 V's, which represent volume, velocity, variety, veracity, and value. [14] These characteristics can be visualized as different levels of a pyramid:

1. At the base of the pyramid, volume pertains to the size and quantity of data that must be gathered and analyzed.
2. Velocity denotes the speed at which data are collected, stored, and managed. Volume and velocity together impose requirements that require careful consideration. For instance, in some cases, it may be preferable to have a limited amount of real-time data rather than a large amount of data at a slow speed.
3. Variety encompasses the diversity and range of different data types and sources.
4. Veracity is associated with the quality, accuracy, and reliability of data and data sources, and it holds the utmost importance among the 5 V's for achieving business success.
5. Value, positioned at the pinnacle of the pyramid, signifies the capacity to convert data into valuable insights.

Numerous telemetry frameworks have been described in the available literature [16]. Generally, telemetry measurements are obtained from various points of observation within network devices and transmitted to a central system that operates alongside the SDN controller. This setup establishes a telemetry pipeline that consists of two primary components: data collectors responsible for gathering measurements from observation points in devices, and a centralized telemetry system that stores and processes the received data. The underlying concept of this design is to collect and store as much data as possible, with the expectation that it can be utilized by network automation systems, such as those based on Machine Learning (ML) [17].

Simultaneously, events produced by applications and platforms such as SDN controllers and management systems can be employed to maintain consistency across systems. Rather than using traditional notifications, an event streaming mechanism is offered as an alternative. This streaming capability, separate from Transport API (TAPI) notifications, is specifically designed to handle large volumes of data and offer an enhanced operational method. Within this framework, any part of the SDN control plane can serve as a source of event telemetry, which must be transported and distributed without any modifications to other systems in the control and management planes.

3.3 The network architecture

Figure 3.1 illustrates the reference network scenario, where a SDN architecture is responsible for controlling various optical nodes, including optical transponders (TP) and Re-configurable Optical Add-drop Multiplexers (ROADM), within the data plane. It is important to note that the SDN architecture can have a hierarchical structure, consisting of different controllers such as optical line systems and parent SDN controllers. To manage the telemetry aspects, there is a centralized telemetry manager that handles the reception, processing, and storage of telemetry data in a telemetry database (telemetry DB). This database comprises two repositories: i) The measurements DB, which is a Time Series (TS) database, stores measurement data. ii) The events DB, which functions as a Free Text Search (FT) engine.

Furthermore, the telemetry data can be exported to other external systems, such as through Kafka. Some data exchange is required between the SDN control and the telemetry manager. For example, the telemetry manager needs access to the topology database (topology DB), which describes the optical network's topology, and the Label-Switched Path (LSP) database (LSP DB), which describes the optical connections. It is important to note that these databases are not shown in Figure 3.1 for simplicity.

Each individual node in the data plane is under the local management of a node agent, as depicted in Figure 3.1. The node agent is responsible for converting control messages received from the corresponding SDN controller into actions within the local node. Furthermore, the node agent contains data source adaptors, which gather measurements from observation points (labeled as M) found in optical nodes or specific optical devices such as OSAs. It also consists of a telemetry agent that handles and exports telemetry data to the telemetry manager. Additionally, events labeled as E can be gathered from applications and controllers.

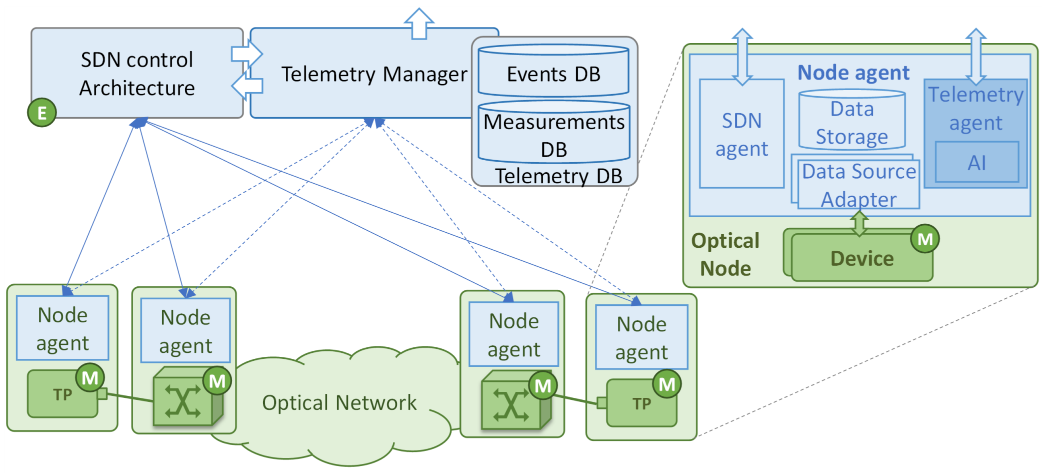


Figure 3.1: Network Architecture [18]

3.4 The GODAI framework

Generic mOdule for Distributed Artificial Intelligence (GODAI) is a framework allowing to run distributed systems and integrating intelligence in them offering a flexible and Plug-and-Play experience. This framework has been developed in Python, and it can be run in a Docker container, using any of the deployment tools available such as Kubernetes, Docker Swarm or Docker-Compose. Some of the main features that GODAI offers are:

- **Decoupled architecture**: The framework allows building hierarchical and/or distributed architectures as all the nodes can communicate with each other.
- **High modularity**: Any piece of code running in a GODAI node can be seen as a component which can be inserted or removed from the node as needed.
- **Easy to integrate**: Existing code in Python can be easily integrated in a GODAI component.
- **Flexible communications**: Internal and external communications can be reconfigured dynamically to enable or disable streaming telemetry in real-time.
- **Portability**:The nodes forming the GODAI architecture can be deployed using Docker containers and can be ported to any environment with Docker installed.
- **Lightweight**: A GODAI node needs few resources in terms of CPU and RAM (a standalone node consumes around 50 MB and 100 MB).

This framework is intended to be suitable in different scenarios where distributed computing is needed. This can include several use cases, as the trend right now is to move towards a more distributed and closer to the edge computing. With this new trend, decisions can be taken faster, and we could potentially make better decisions based on collaborative work between the different agents joining the system. This implies many applications such as federated learning, distributed telemetry architectures, multi agents systems, etc. Developers could use this framework to integrate their own algorithms and interfaces to create their own edge computing agents.

These architecture uses the different communication patters to exchange information. Firstly, we have what we call the internal communication, which is the one that is done between the different elements inside a GODAI node. This communication uses a Pub/Sub pattern where publishers push messages to a topic where subscribers will receive their messages. Secondly, there is the external communication that is done through a gRPC interface with a common schema that sends all the information as a stream of bytes, enabling interoperability and compression. This communication is done between the GODAI nodes while exchanging information that has to be processed. This framework allows running more interfaces if needed, enabling communication with a many types of external services. For instance, the Management Interface is a Representational State Transfer API (REST API) exposing methods to interact with a GODAI node to change its operational state and retrieve information about it.

In the context of the next generation of communications, such frameworks will enable new technologies that now are limited but the existing architectures and the connectivity required to run distributed computational workloads with high requirements in terms of flexibility. Also, it is mandatory to take into account the portability and integration ease to made it available to any operator willing to use this framework in their environments.

3.4.1 GODAI node

A GODAI node can be seen as the unit working piece of software of the framework, several GODAI nodes can be instantiated and connected to build distributed systems architectures. Each of these nodes is built with different microservices interconnected that are in charge of the different functionalities offered. These nodes are highly configurable and can be instantiated easily with Docker.

The main purpose of these GODAI nodes is to offer a common environment where algorithms and services can be deployed to process incoming data and to communicate with other nodes if needed. Following, there is an explanation of the microservices conforming the GODAI node.

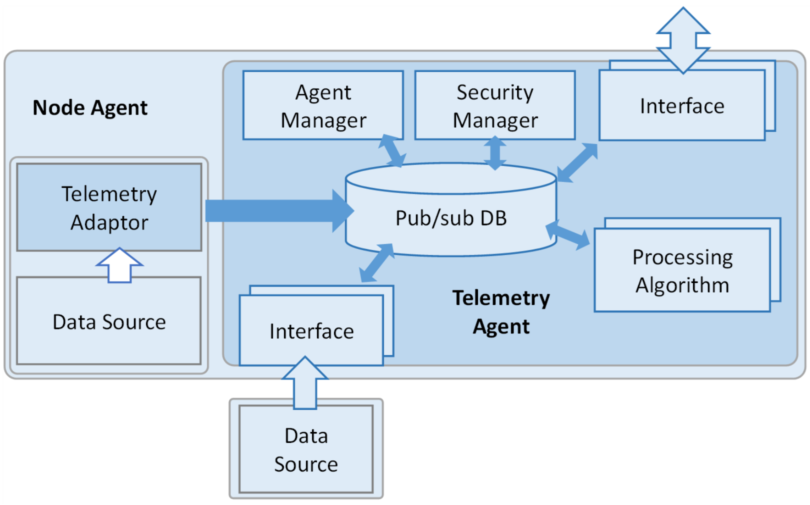


Figure 3.2: GODAI node architecture

Redis DB

One Redis DB is deployed for each GODAI node, as the main purpose of it is to enable the internal communication between the different components and services in the GODAI node. For this reason, the Pub/Sub feature included in Redis is used to expose topics where components can subscribe and where other components can send messages. In Redis Pub/Sub, there are two main entities: publishers and subscribers.

- **Publishers**: Publishers are responsible for sending messages to Redis channels. A channel is a logical grouping or topic to which messages are published. Publishers don't receive any feedback or acknowledgment about the delivery of messages. They simply publish messages to specific channels without any knowledge of who, if anyone, is listening.
- **Subscribers**: Subscribers express their interest in receiving messages from one or more channels. They subscribe to the channels they want to listen to and Redis delivers messages published to those channels to the subscribers. Subscribers can listen to multiple channels simultaneously.

It's important to note that Redis Pub/Sub does not provide durability or persistence for messages. Once a message is published and delivered to subscribers, Redis does not retain a copy of the message. If a subscriber is not actively listening at the time of message publication, the message will not be received. Redis Pub/Sub is commonly used for real-time messaging, event-driven architectures, and building scalable systems where different components need to communicate asynchronously.

Manager

The manager service is the main service of the GODAI node, which oversees the rest of the services running inside the node. This service reads a configuration file in JSON format and starts the configuration of the rest of the services and tries to deploy and configure the demanded components defined in the configuration. The configuration is done in three steps:

1. **Deployment of the Management Interface**: A REST API allowing the interaction with the GODAI node from the outside.
2. **Deployment of the Security Manager**: Service managing all the security procedures done inside the GODAI node.
3. **Deployment of the components**: Every module defined inside the configuration file is deployed and configured. This may include algorithms as well as services or interfaces.

Once the deployment and configuration is finished, the GODAI node connects to the Redis DB and subscribes to some of the available topics. This enables the internal communication between the different services and the manager. At this point the manager is ready to receive any further action that could include reconfiguration of any component, deployment of new components or receiving a signal to stop its operation. An example of a configuration file is provided in the Appendix 1.

Management Interface

The Management Interface is a REST API using the Flask framework that exposes several methods to interact with the GODAI node from the outside. The purpose of this interface is to enable communication from external systems to a GODAI node to perform an action related to the state and configuration of the different modules. The also available Web User Interface (Web-UI) interacts with this REST API in order to perform actions and to retrieve information about the operational state of the different nodes in the system. In particular, the following methods are available:

GET **/ping**
ping a GODAI node

Response application/json

200 ok

```
{
    "message": "\acrshort{godai} node running"
}

```

404 Connection error

```
{
    "message": "\acrshort{godai} node unreachable"
}

```

GET **/getComponents**
Retrieve information of the running components from a GODAI node

Response application/json

200 ok

```
{
    "components": {
        "component-1": {
            "state": "running"
        }
    }
}

```

404 Connection error

```
{
    "message": "Cannot retrieve components from the \
   acrshort{godai} node"
}

```

GET **/getKeys**
Retrieve keys from the Manager Node

Response application/json

200 ok

```
{
    "keys": {
        "godai-node-1": {
            "key": "asdlfLfYX!02343412"
        }
    }
}

```

404 Connection error

```
{
    "message": "Cannot retrieve keys from the Manager"
}

```

POST **/component/{action}**
perform an action in the selected component

Parameter

action	action to be performed in a component: includes deploy, reconfigure, start and stop
componentName	name of the component

Response application/json

200 ok

```
{
    "state": deployed,
}

```

404 error: action could not be performed

```
{
    "message": "start on component-1 failed!"
}

```

POST **/postKey**
Post a key to the Manager node

Parameter

no parameter

Body application/json

```
{
        "godai-node-1": {
            "key": "asdlfLfYX!02343412"
        }
}

```

Response application/json

200 ok

```
{
    "message": Key received successfully
}

```

More methods are still in a development phase and include the option to upload packages to a GODAI node to be executed.

Security Manager

The Security Manager is the main service in charge of the security in a GODAI node. Its main purpose is to handle the key management of the node and to receive the keys of the other nodes. With this key exchange, we can establish secure communication between the different GODAI nodes. However, this implies the existence of a central Manager Node that has to store all the keys from the rest of the nodes and distribute them. The standalone operation in a normal deployment include the following steps:

1. The GODAI nodes sends their keys to the Manager GODAI node. The Manager stores all the keys matching with the name and IP address of the node.
2. The GODAI nodes schedule every X seconds a call to the Manager to get all the keys.
3. The GODAI nodes receive the DB of keys and are able to encrypt/decrypt messages to send/receive messages from the other nodes.

Even though this practically works, it implies that one of the GODAI nodes has to perform the role of a central Manager, which can lead to performance issues and includes an overhead associated to all the messages sent to achieve a consensus. For this reason, a new procedure to secure the GODAI nodes has to be explored. In this case the use of a Distributed Ledger Technology (DLT) can suit well as we can have a distributed authority that is in charge of the key distribution and for this the Manager role is no longer needed. An initial proposal on how to achieve this has been described in [19] and it is longer explained in the security chapter.

Components

The components or modules are the computational units running inside the GODAI node. These can be described as algorithms, interfaces or services processing data or offering more capabilities to a node. In fact, we can see a module as any package/class that can be run. The existing limitations on which type of code can be run inside a GODAI node are based on the programming language. In the time when this thesis has been written, only code written in Python is compatible with a GODAI node. However, existing libraries providing bindings from other programming languages can be implemented and more of these of programming language could be used in the future.

To write a module of GODAI and avoid the complexity of setting up all the environment, a base class has been provided. This base class named "Component" takes care of all the configuration and initialization of the communication inside the GODAI node and provides functions that can be overwritten to suit the needs of the application. Here is a description of the three most important functions of the Component class:

- **run**: This function is not usually overwritten, as its purpose is to connect to the Redis DB and subscribe to a topic. Incoming data is then conveyed into the module calling the process_data function.

```python
    def run(self):
        sub = self.redis_client.pubsub(ignore_subscribe_messages=True)
        sub.subscribe(self.topic)
        self.initialization()
        self.logger.info(self.logheader + " subscribed to topic:" + self.topic)
        while True:
            message = sub.get_message()
            if message is not None and isinstance(message, dict):
                self.process_data(json.loads(message["data"]))
            time.sleep(0.001)

```

- **process_data**: The process_data function receives the data and is the entry function to the module where the actual functionality can be implemented. To send the data to any other module, the send_data can be called.

```python
    def process_data(self, data):
        msg = self.logheader + "Component is an abstract class"
        self.logger.critical(msg)
        raise GodaiException(msg)

```

- **send_data**: The send_data function receives the processed data and checks the recipients where to send the data based in already defined workflows. The data is published in the selected topics.

```python
    def send_data(self, data):
        data_json = json.dumps(data)
        if data["header"]["workflow"] in self.workflows:
            recipients = self.workflows[data["header"]["workflow"]]
            for output in recipients:
                output = self.nodeName+"/"+output
                self.logger.debug(self.logheader + "sending message to " + str(output))
                self.redis_client.publish(output, data_json)

```

As stated before, any user willing to use its own module inside a GODAI node has to inherit the Component class and overwrite the mentioned functions. At this moment this only applies to code written in Python, but further compatibility could be provided in the future. An example of the configuration of some components is provided in the Appendix 1.

Web-UI

A web interface has been developed to enhance the interactivity with the GODAI nodes and bring a graphical interface to display the operational status. This web interface has been developed using the Django framework, which a web framework written in Python that is widely used and includes many features as templates, high scalability and a fast development.

This graphical include several views with operational information regarding the GODAI nodes deployed. It also may include information about the data sources that inject data to the GODAI nodes. Several actions can be performed in this GODAI nodes as stated in the Management Interface definition done before. The main dashboard shows a summary of the elements running in the system and their relation, as well as a diagram showing graphically the relation between the data sources and the GODAI nodes deployed. A notification system is implemented to show the user any change produced in the system as well as failure management messages to inform in the case of any malfunctioning.

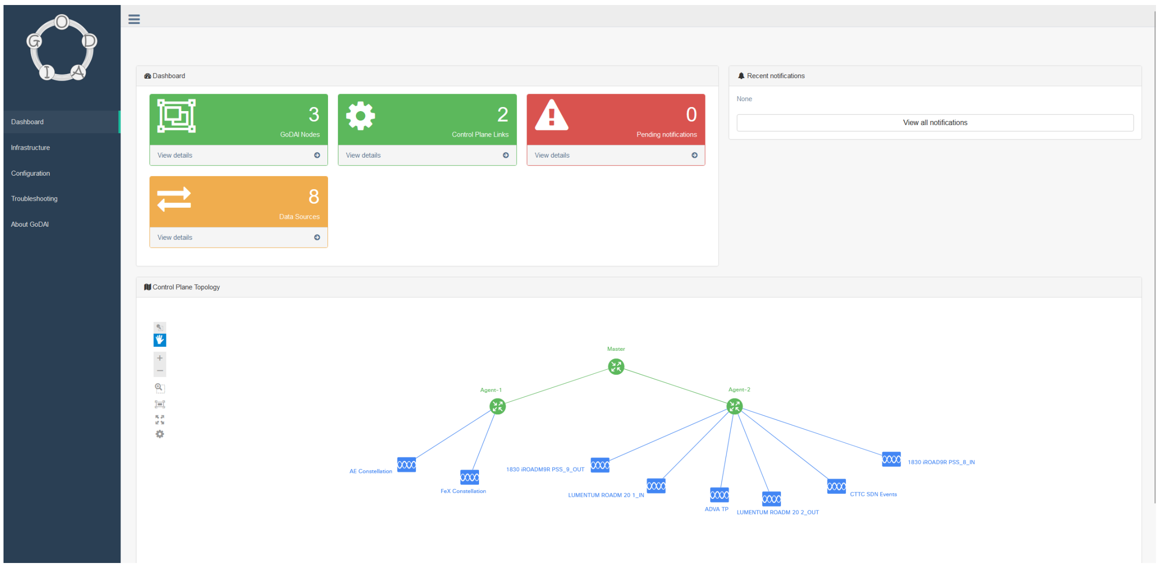
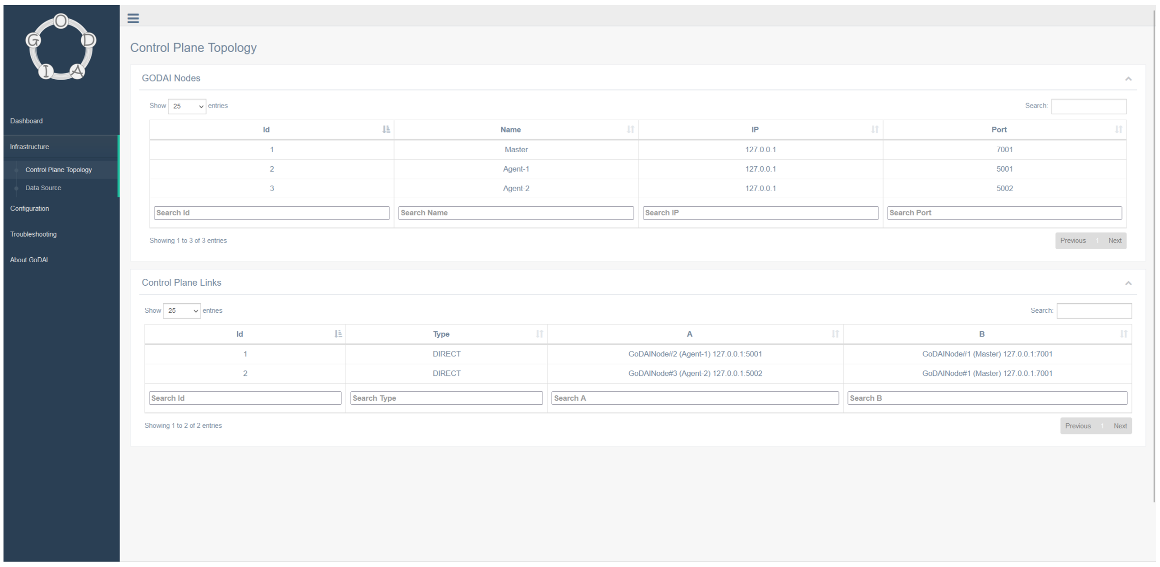


Figure 3.3: GODAI Web-UI Dashboard

The topology can be seen in detail as well as the GODAI node configuration that can edited if needed.



Control Plane Topology

GODAI Nodes

Show 25 entries Search:

Id	Name	IP	Port
1	Master	127.0.0.1	7001
2	Agent-1	127.0.0.1	5001
3	Agent-2	127.0.0.1	5002

Showing 1 to 3 of 3 entries

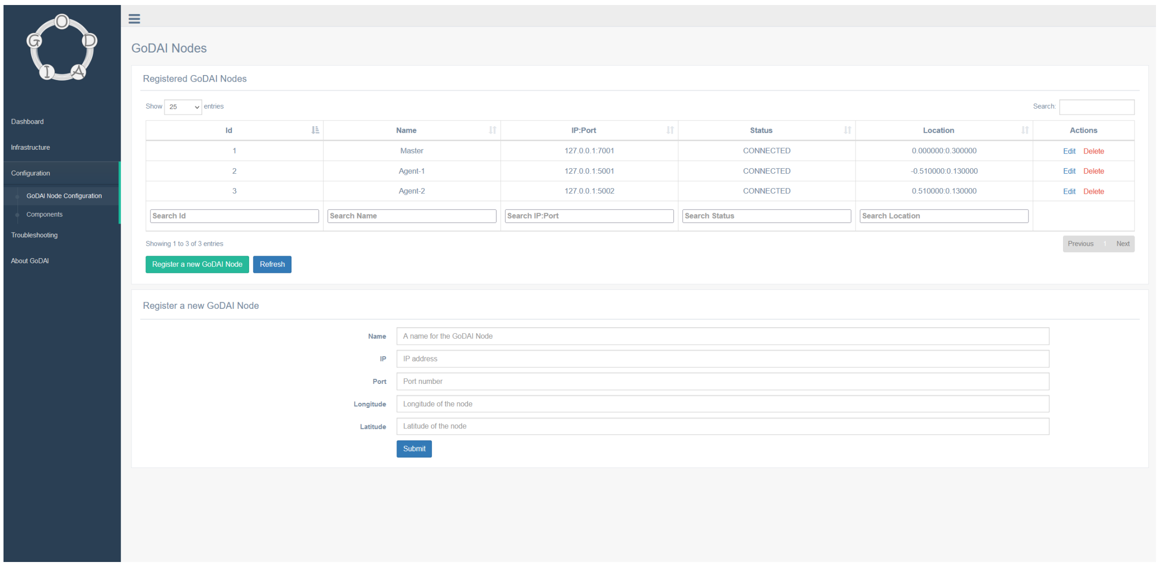
Control Plane Links

Show 25 entries Search:

Id	Type	A	B
1	DIRECT	GoDAINode#2 (Agent-1) 127.0.0.1:5001	GoDAINode#1 (Master) 127.0.0.1:7001
2	DIRECT	GoDAINode#3 (Agent-2) 127.0.0.1:5002	GoDAINode#1 (Master) 127.0.0.1:7001

Showing 1 to 2 of 2 entries

Figure 3.4: GODAI Web-UI Topology details



GoDAI Nodes

Registered GoDAI Nodes

Show 25 entries Search:

Id	Name	IP:Port	Status	Location	Actions
1	Master	127.0.0.1:7001	CONNECTED	0.000000:0.300000	Edit Delete
2	Agent-1	127.0.0.1:5001	CONNECTED	-0.510000:0.130000	Edit Delete
3	Agent-2	127.0.0.1:5002	CONNECTED	0.510000:0.130000	Edit Delete

Showing 1 to 3 of 3 entries

Register a new GoDAI Node

Name	A name for the GoDAI Node
IP	IP address
Port	Port number
Longitude	Longitude of the node
Latitude	Latitude of the node

Figure 3.5: GODAI Web-UI Nodes configuration

The web interface is still in an early stage of development, and few functionalities are available at this moment. However, as more complex scenarios and requirements appear, new functionalities have to be developed. Some of them may include: GODAI node deployment, connection establishment, data source deployment, more complex workflows' definition, among others.

3.4.2 Data Sources

In telemetry on network devices, data sources play a vital role in providing valuable insights into the performance, health, and behavior of network infrastructure. These sources encompass a multitude of components and systems within a network environment. One of the primary data sources is network equipment itself, such as routers, switches, and firewalls. These devices generate telemetry data that includes network traffic statistics, interface utilization, error rates, packet loss, and other performance metrics. They offer granular visibility into the network's behavior and help identify bottlenecks, congestion, or potential security issues.

In addition to network equipment, monitoring agents or probes deployed strategically across the network serve as another critical data source. These agents collect data on network latency, round-trip times, network topology, and protocol-level details. They continuously monitor network traffic, capture packets, and extract relevant information for analysis. These agents can also provide real-time alerts and notifications when anomalies or performance deviations are detected. Moreover, network telemetry data can be derived from Network Operating System (NOS) and management platforms. These systems aggregate data from various network devices, providing a centralized view of network health and performance. They collect and consolidate data from network device logs, SNMP traps, flow data, and other sources, offering a comprehensive picture of the network environment.

In this context, GODAI offers a general schema to allow data sources to export data easily to a GODAI node. Besides this flexibility GODAI also includes security features that control the flow of data of the different data sources. Only verified data sources are allowed to convey data to a selected GODAI node. Periodical checks are done on data sources to ensure the trustworthiness of the data conveyed.

In summary, data sources in telemetry on network devices encompass a diverse range of components, systems, and external sources. By harnessing these sources, network administrators and analysts can gain comprehensive visibility into network performance, security, and user experience, enabling them to proactively monitor, troubleshoot, and optimize network infrastructure.

3.5 Requirements

There exists two main ways to deploy a GODAI node, with different requirements and purposes. In both cases, a Redis DB must be deployed alongside to enable the communication between the internal components.

The first way is to run a GODAI node as a single script directly from the command line or using any Integrated Development Environment (IDE) that is able to run Python scripts. In this case, the only requirement is to have the Python interpreter installed, as well as needed packages within the code. Essentially, Redis and Flask are the minimum requirements to run a GODAI node without using any other external package in any of the modules. This first method is intended to be used while debugging the functionality of the GODAI node as well as the new developed and integrated modules. This can allow the developer to set breakpoints as needed to debug any issue that may occur.

The second way to run a GODAI node is inside a Docker container (among other containerization solutions) to be deployed then in an environment with Docker installed or a Kubernetes cluster. To do so, a Docker image is provided including all the required packages to be installed and containing the source code of the GODAI node code. Any additional module or file can be mounted as a directory to be then consumed by the GODAI node. This second way of deploying a GODAI node targets production environments where we may want to deploy several GODAI nodes at the same time. Having the application containerized allows us to scale up or scale down the number of GODAI nodes easily and to issue new deployments easily. In this case, the only requirements are to have installed the Docker container engine, which is installable in almost every modern operating system.

Chapter 4

Security in Telemetry Systems

4.1 General security requirements

Distributed systems are a fundamental part of modern computing, encompassing networks of interconnected computers or nodes that work together to achieve a common goal. These systems are widely used in various domains, including cloud computing, IoT, and large-scale data processing. However, the distributed nature of these systems introduces unique security challenges that must be addressed to ensure the integrity, confidentiality, and availability of data and resources. This chapter explores the importance of security in distributed systems, examines potential vulnerabilities, and presents best practices for securing these complex environments. Some of the existing security challenges in distributed systems are:

1. **Network Communication**: Distributed systems rely on network communication for data exchange among nodes. This introduces the risk of eavesdropping, data interception, and unauthorized access. Ensuring secure communication through encryption protocols, such as Secure Sockets Layer/Transport Layer Security (SSL/TLS), can mitigate these risks and protect data confidentiality.

2. **Authentication and Authorization**: In a distributed system, it is essential to establish the identity of nodes and users and grant appropriate access privileges. Robust authentication mechanisms, such as mutual authentication, Two-Factor Authentication (2FA), or digital certificates, are crucial for verifying the identities of communicating entities. Access control mechanisms, such as Role-Based Access Control (RBAC), should be implemented to authorize and restrict actions based on predefined policies.

3. **Data Integrity and Consistency**: Maintaining data integrity and consistency in distributed systems is challenging due to the potential for concurrent updates and network delays. Techniques such as cryptographic hashing, digital signatures, and consensus algorithms (e.g., Paxos or Raft) can be employed to ensure the integrity and consistency of data across distributed nodes.

4. **Fault Tolerance and Availability**: Distributed systems are designed to be fault-tolerant and provide high availability. However, malicious actors can exploit vulnerabilities to disrupt system operation, resulting in Denial of Service (DoS) attacks or compromising data availability. Implementing redundancy, load balancing, and Distributed Denial of Service (DDoS) mitigation techniques can help mitigate these risks and ensure system availability.

5. **Scalability and Trust Management**: As distributed systems grow in scale, managing trust becomes a significant challenge. Nodes may join or leave the system dynamically, making it crucial to establish trust relationships and handle trust management effectively. Public Key Infraestructure (PKI), decentralized identity management, and reputation-based systems can aid in managing trust in large-scale distributed environments.

With this security requirements [20], we can also describe the technologies and techniques ensuring that none of this security threatens can happen in distributed systems.

1. **Encryption and Secure Communication**: Implementing End-to-End (E2E) encryption using strong encryption algorithms and secure communication protocols is crucial to protect data confidentiality and integrity. Encryption should be applied not only to data in transit, but also to data at rest within distributed storage systems.

2. **Secure Access Control**: Applying granular access control policies based on the principle of least privilege helps prevent unauthorized access and reduces the attack surface. Access Control List (ACL), capabilities-based security models, and Attribute-Based Access Control (ABAC) mechanisms can be employed to enforce access control in distributed systems.

3. **Intrusion Detection and Monitoring**: Deploying Intrusion Detection system (IDS) and Intrusion Prevention Systems (IPS) within distributed systems allows for the timely detection and mitigation of security breaches. Real-time monitoring of system logs, network traffic, and application-level events can provide insights into potential security incidents.

4. **Data Replication and Backup**: Data replication across distributed nodes enhances fault tolerance and data availability. However, it is essential to ensure that replicated data remains consistent and secure. Employing secure replication techniques, such as Byzantine Fault-Torent (BFT) algorithms or erasure coding, along with regular data backups, can mitigate the risk of data loss or compromise.

5. **Secure Software Development**: The distributed nature of systems introduces additional complexities in software development. Following secure coding practices, conducting regular code reviews, and performing comprehensive security testing, including penetration testing and vulnerability assessments, are essential to identify and remediate software vulnerabilities.

Security is a vital aspect of distributed systems, given their complex and interconnected nature. By implementing encryption and secure communication, robust authentication and access control mechanisms, intrusion detection and monitoring systems, and resilient data replication and backup strategies, organizations can enhance the security posture of their distributed systems. Furthermore, adhering to secure software development practices, preparing incident response plans, and complying with industry regulations contribute to a comprehensive security framework for distributed systems, safeguarding data and ensuring the reliability and availability of these critical computing environments.

4.2 Security in GODAI

Security in GODAI is defined in all the different aspects that implies running a distributed system. Following there is a description of the different security measures applied and which type of threatens tries to avoid:

- **Secure REST API**: The REST API for managing the nodes is running a web server with SSL/TLS enabled in order to secure all the communications done through this interface. Credentials has to be provided in order to execute any action inside a GODAI node.
- **Secure broker in gRPC interface**: A security broker is deployed in each gRPC interface that is exposed in order to ensure that the receiving data is coming from trusted data sources or nodes. All data coming from not verified sources is automatically discarded, however a banning system has to be developed to avoid DDoS attacks on the interfaces. As well as the REST API the communication is encrypted using SSL/TLS.
- **Protected internal communication**: The internal communication is as well protected using the same procedures in the two methods explained before. Authentication and SSL/TLS is ensured so just the internal components of the GODAI node are available to exchange messages.

These measures ensure the secure communication between all the elements in the infrastructure. However, the certificate and key management implies having a central authority or manager in charge of this. As we want to be as distributed as possible some alternatives has been studied to avoid a central point of failure. In this sense, DLT had acquired notoriety due its features and its distributed nature.[21]

Secure distributed systems leverage distributed ledger technology, such as blockchain, to establish trust, transparency, and immutability across a network of participants. These systems operate on a decentralized architecture, eliminating the need for a central authority and relying on a consensus mechanism to validate transactions and maintain the integrity of the data. The distributed ledger serves as a shared and synchronized record of all transactions or information within the network. Each transaction is recorded in a block, which is then cryptographically linked to the previous block, creating a chain of blocks that form the blockchain. This chaining mechanism makes it exceedingly difficult for malicious actors to alter or tamper with the data since any modifications would require changing the entire chain, which would be computationally infeasible.

Secure distributed systems using distributed ledger technology find applications across various industries. In finance, blockchain-based systems enable secure and transparent peer-to-peer transactions, reducing the reliance on traditional intermediaries. In supply chain management, distributed ledgers enable traceability and provenance, helping to prevent counterfeiting and ensuring product authenticity. In healthcare, these systems enhance the security and privacy of patient data, facilitating secure sharing of medical records and streamlining processes. Overall, the integration of secure distributed systems with distributed ledger technology offers a robust and resilient infrastructure that promotes trust, security, and efficiency in diverse domains.

The work presented in [19] tries to define a series of measures to secure Multi-Agent System (MAS) in the context of real-time control of 6G services. The proposed improvements in terms of security are the following: secure execution monitoring, DLT for non-real-time secure MAS management, Virtual Extensible Local Area Network (VXLAN) for near real-time secure MAS operation. The following figure shows how these measures fits in the already described architecture.

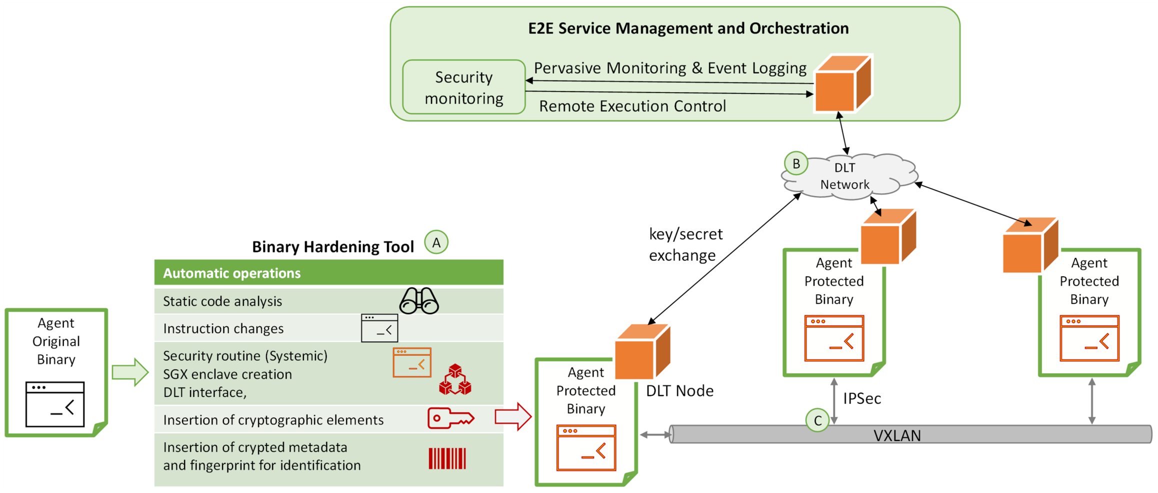


Figure 4.1: Proposed solution for Secure MAS. [19]

Secure Execution Monitoring

In order to enhance their ability to withstand different attacks and build trust in a timely manner, it is necessary to strengthen agents and closely monitor their activities. For this purpose, we propose implementing a set of initial measures based on well-established evidence, including:

1. Assessing the agents' actual performance to determine their effectiveness.
2. Verifying the authentication process of agents during their startup phase to ensure their legitimacy.
3. Conducting integrity checks on the agents' most recent execution to ensure their reliability.

These methods can also be utilized to explicitly indicate any deviations from the normal control flow of the agents. As a result, some modifications to the agents' software are required to implement these changes.

In figure 5.4(A) we can whiteness the process of loading original agents into a binary hardening tool and converting them into protected versions. These protected variants are designed to withstand attempts to compromise their confidentiality and integrity. Additionally, they are continuously monitored by connecting them to a ledger.

To ensure the overall integrity of the global agent, several criteria are evaluated. Firstly, it checks whether the agent has been tampered with through interception before being loaded. Secondly, it verifies that the agent has not been tampered with through local memory introspection during its execution. Lastly, it confirms that the agent is actively running. This last criterion helps detect denial of service attacks on the agent or its platform by monitoring resource depletion. The monitoring process provides regular updates on the integrity and activity status of the agents to an application running in the service orchestrator (labeled as B in Figure 5.4).

DLT for Non-Real-Time Secure MAS Management

We depend on a lightweight application-based DLT overlay to ensure a high level of trust among agents. This involves receiving and processing new software marks that determine trustworthiness, delivered by a binary hardening tool. The DLT reflects the health status of the agents. DLTs possess characteristics that make them well-suited for facilitating dynamic associations and exchanging keys/secrets among multiple agents. Alongside the service orchestrator, agents can dynamically join the DLT (represented as "C" in Figure 1), and smart contracts can be utilized to govern and monitor each dynamic association between agents. For effective communication, a service level agreement can be established for each association to monitor the required service level. When a newly created agent is discovered, its enrollment and retirement from the group of consensus-validated agents are initiated. Additionally, any changes to the trustworthiness elements of an agent are handled in a desynchronized manner to avoid delays in establishing communication links between agents.

VXLAN for Near Real-Time Secure MAS Operation

Using a DLT-based solution for communication between agents would introduce a delay in message exchange, which would hinder the ability to operate services in near real-time. To address this issue, we suggest combining DLT with VXLAN [22] to minimize any additional delay. The DLT exchange would be kept offline, while VXLANs would be utilized for communication in real-time among the agents (represented as D in Figure 1). VXLAN is a technology that encapsulates network traffic, allowing the creation of an overlay network on top of a physical network, thus providing a virtualized environment with a service abstraction layer. This solution offers fast deployment and the ability to create numerous networks concurrently (simultaneous segments).

Simultaneously, VXLAN raises security issues, including the potential for unauthorized devices to join multicast groups and introduce counterfeit data. Encryption protocols like IPsec offer a solution by encrypting both the content and the inner headers, mitigating the risk of rogue activities when compared to application-level encryption protocols in real-time scenarios. To minimize encryption-related delays and facilitate verification and tracking of communications by other MAS agents, our implementation incorporates the use of pre-shared secrets. However, this approach necessitates an authentication infrastructure to enable authorized agents to acquire and distribute these secrets.

We depend on DLT for this objective. Specifically, the inherent security characteristics of DLTs can be utilized to securely oversee VXLANs as communication channels among agents, once the agreement between them is reached. It should be noted that the choice of consensus mechanism only affects the initial setup phase of dynamic associations between agents and does not affect the actual exchanges between them once the associations are established.

The before mentioned techniques are still in development and they downsides are still being explored. As a future work a novel implementation of this security architecture will be deployed and tested to be compared with the previous techniques used to secure the MAS.

Chapter 5

Use Cases

In this chapter, some use cases will be presented showing already published novel architectures taking advantage of this developed framework. These two use cases show how useful and flexible GODAI can be while monitoring networking devices and processing the data extracted.

5.1 Intelligent Optical Measurement Aggregation and Streaming Event Telemetry

In this first use case, a demonstration shows how GODAI can be useful while conveying data from observation points that may include heterogeneous measurements and also being able to handle two type of measurements: events and telemetry. This demonstration was presented in the Optical Fiber Communication Conference held in San Diego this year and includes a full functional demonstration. This demonstration was proposed in the context of the research project H2020 B5G-OPEN (G.A. 101016663) that targets the design, prototyping and demonstration of a novel end-to-end integrated packet-optical transport architecture based on MultiBand (MB) optical transmission and switching networks.[23]

The information gathered from monitoring points on the devices is usually transmitted to a central system for additional analysis. While protocols designed for telemetry, such as Google Remote Procedure Calls (gRPC), can help decrease the amount of data transmitted, scalability remains a challenge due to the large volumes of measurement data that need to be collected frequently.

Moreover, applications/platforms such as SDN controllers and management systems can generate events that help maintain consistency across systems. These events can be utilized as an alternative to traditional notifications through an event streaming mechanism. Unlike TAPI notifications, this streaming capability is specifically designed to handle large-scale operations and offer an enhanced operational approach. Within the SDN control plane, any component has the potential to serve as a source of event telemetry, which must be transmitted and distributed without any alterations to other systems in the control and management planes.

In this demonstration, a telemetry architecture developed within the H2020 B5G-OPEN project is shown. This architecture supports two types of telemetry: measurements and events. For measurements, intelligent data aggregation and feature extraction techniques are applied near the data collection point to reduce the amount of data. On the other hand, event telemetry is transported seamlessly without any modifications. The demo will specifically showcase the integration of the following: i) Heterogeneous measurements obtained from an OSA collected from the Nokia Bell Labs test bed located close to Paris (France) and from commercial ADVA optical transponders (TP) in a test bed situated in the Fraunhofer HHI premises in Berlin (Germany). ii) Connection set-up and teardown events originating from an SDN controller situated at the CTTC premises near Barcelona (Spain).

The need for data availability is crucial for various network automation purposes, such as training ML models, detecting degradation and anomalies, and maintaining consistency in distributed control and management systems. To enable these functionalities, telemetry solutions play a vital role by facilitating the collection of large volumes of data and enabling on-site data processing or seamless data streaming. However, there are still uncertainties regarding the frequency of collecting telemetry data, the processing locations, and methods to reduce data volume.

Figure 5.1 presents a detailed architecture of the telemetry system, illustrating the internal structure of telemetry agents and the telemetry manager. Telemetry agents can be integrated with node agents for measurement telemetry or deployed separately for event telemetry. Internally, both telemetry agents and the telemetry manager rely on three main components: i) A manager module that configures and oversees the operation of other modules; ii) Several components encompassing algorithms for data processing, aggregation, etc., as well as interfaces like gRPC; iii) A Redis DB that facilitates communication among the different modules.

This solution offers an adaptable and dependable environment that simplifies communication and allows for the integration of new modules. The telemetry agents utilize a gRPC interface to export telemetry data to the telemetry manager, while the telemetry manager can adjust the behavior of algorithms in the agents through this interface.

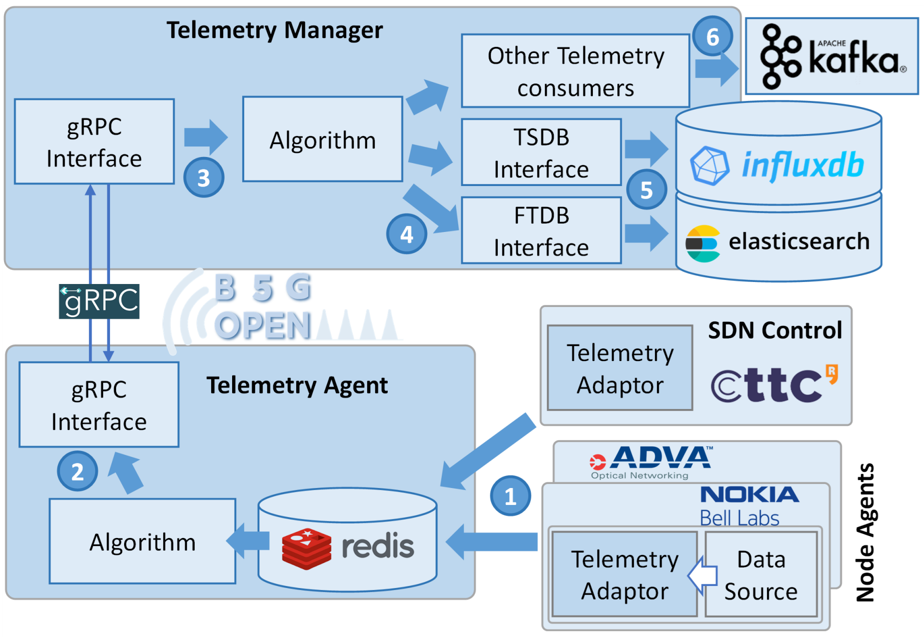


Figure 5.1: Proposed Telemetry Architecture [24]

The node agent in charge of measurement telemetry incorporates modules, known as data sources, which collect measurements from observation points within the optical nodes. To facilitate the transfer of collected data to the telemetry system, a telemetry adaptor has been created. This adaptor receives raw data from the data source and converts it into a structured JSON object. The JSON object is then published in a local Redis DB (designated as 1 in Figure 2), which serves as a boundary point. Various algorithms can subscribe to these collected measurements. For instance, let's consider a scenario where only one algorithm is subscribed, responsible for processing the measurements locally. This processing can involve three possibilities: i) no transformation of the data (null algorithm), ii) data aggregation, feature extraction, or data compression, or iii) inference for tasks such as degradation detection. The resulting data, whether transformed or unchanged, are sent to a gRPC interface module (via the Redis DB) (2), which transmits the data to the telemetry manager.

Regarding event telemetry, events generated within an SDN controller or another system are generated and injected into the telemetry agent. These events are then transparently transported through the gRPC interface to the telemetry manager. Within the telemetry manager, data received by a gRPC interface module is published in the local Redis DB so that subscribed algorithms can access it. Once processed, the output data are published in the local Redis DB (4) and can be stored in either the Measurements DB (utilizing InfluxDB) or the events DB (employing Elasticsearch) (5). Additionally, the data can be exported to external systems like Apache Kafka (6).

We now focus on introducing several techniques to greatly reduce the data volume that needs to be conveyed through the gRPC interface connecting telemetry agents to the manager. In particular, we analyze: i) data compression using autoencoders; ii) supervised feature extraction; and iii) data summarization using the arithmetic mean of a number of observations. For this example, let us assume the case where the observation point is in a TP, which gathers the received optical symbols of a m-Quadrature Amplitude Modulation (QAM) signal. The related data source then, periodically retrieves a constellation sample X (a sequence of k IQ symbols as represented in Fig. 5.2a for a 16-QAM signal) and publish it in the local Redis DB.

Let us start with the use of Auto Encoders (AE), a type of neural network with two components: the encoder, which maps input data into a lower-dimensional latent space, and the decoder, which gets data in the latent space and reconstructs the original data back. Once trained, the autoencoder takes as input 2×k values, i.e., [x1I, x1Q,... xkI, xkQ], from the received constellation sample and generates the latent space Z=[z1, ..., zL], where the size of Z is significantly lower than that of X (Fig. 5.2b). In this case, the encoder runs as an algorithm module in the telemetry agent and exchanges Z for every input sample X with the decoder running in the telemetry manager through the gRPC interface. The algorithm in the telemetry manager uses the decoder to reconstruct the constellation sample and it stores the result in the telemetry DB.

Let us now explore the concept of supervised feature extraction. In a previous study [25], Gaussian Mixture Models (GMM) [26] were utilized to describe each point in an optical constellation sample as a bivariate Gaussian distribution (Fig. 5.2c). Consequently, each point in the constellation, denoted as i, is characterized by five features: the mean position in the I and Q axes [μI, μQ], as well as the variance in I and Q, and the symmetric covariance terms representing the variations experienced by symbols belonging to point i around the mean [σI, σQ, σIQ]. Consequently, for an m-QAM signal, a total of m*5 features must be transmitted from the telemetry agent to the manager.

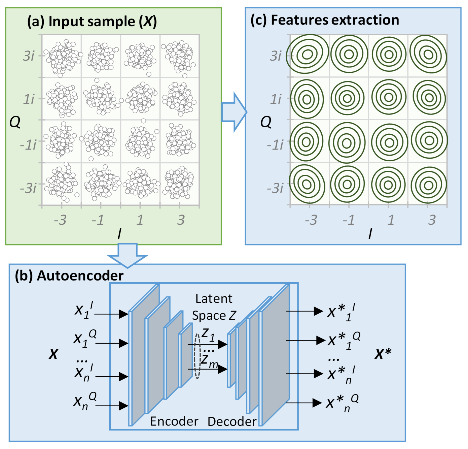


Figure 5.2: Constellation sample (a), autoencoders (b) and supervised features extraction (c) [18]

Using the aforementioned intelligent data aggregation methods, telemetry data is transferred from the observation point to the telemetry manager at the same frequency. This means that every time a new constellation sample is obtained, a subset of data representing it is created and transmitted to the telemetry manager. Assuming a high collection frequency, this approach results in a substantial volume of data being conveyed. However, under normal circumstances, this level of data transmission is generally unnecessary. Therefore, we can determine whether a representation of the new sample needs to be sent to the telemetry manager by measuring variations in the computed features. If there are no significant variations, the telemetry agent can transmit averaged values of the features at a much lower frequency, effectively reducing the volume of telemetry data being transmitted.

A video explaining in more detail the demo setup and demonstrating the main functionalities can be found in: `https://www.youtube.com/watch?v=1KYikyztsCA`

5.2 Pervasive Monitoring and Distributed Intelligence for 6G Near Real-Time Operation

The second use case involves suggesting a telemetry solution for monitoring the end-to-end delay to ensure the delay requirements for 6G services are met. It is crucial to have near real-time control and operation to meet the strict performance demands of 6G services. Rather than relying on reactive approaches, proactive network adaptation that can anticipate specific events and take proactive measures is preferred. These events include handovers causing additional delays, overloaded edge nodes leading to poor performance, dynamic steering and priority, Virtual Network Functions (VNF) activation/deactivation/replication, and slice reconfiguration. To achieve near real-time operation and control, intelligent decision-making should be located as close as possible to the data plane resources. Additionally, pervasive monitoring is required to anticipate signal degradation, queue congestions, and other issues. This paper proposes a solution that incorporates E2E In-band Network Telemetry (INT) across different network segments, such as the user equipment, Radio Access Network (RAN), transport packet, and datacenter networks. The telemetry agents and the deployed MAS incorporate the GODAI framework.

In traditional approaches that rely on centralized data lakes, cloud-based big data analytics often fail to provide timely feedback to orchestrators and controllers. However, by leveraging network telemetry through technologies like INT and postcard, it becomes possible to achieve accurate monitoring using distributed and federated agents. This requires the introduction of new entities that gather measurements from various sources such as the RAN, programmable devices, the cloud, and application entities. INT solutions enable per-packet telemetry by incorporating network state information into each packet or cloned copies, which is then conveyed through INT/postcard reports. Nonetheless, the extensive collection and processing of telemetry data pose challenges to scalability. To address this, a two-stage P4 telemetry collector was proposed in [16]. These collectors are responsible for processing and aggregating postcard telemetry reports at wire speed. However, there is a need to enhance this solution so that the collected measurements can be processed and utilized locally by node agents, enabling a high level of network awareness within the specified segment.

Encoding per-hop information on a per-packet basis results in a linear increase in the size of packet headers with each hop. This not only wastes bandwidth but can also cause packet fragmentation if the maximum transmission unit (MTU) is exceeded. To illustrate, a path trace can be obtained by including a unique hop ID in each packet. The path is formed by combining the individual hops stored in the flow packets. The reconstruction of the path occurs at a telemetry processor. While this approach necessitates storing some network state in the network nodes, it enables the collection of telemetry data without encountering scalability problems at the collectors or compromising network performance.

Telemetry measurements can be further combined by telemetry processors that aim to reduce dimensionality and decrease the data rate at the control plane. This can be achieved through various methods, such as compressing individual measurements and aggregating time series data. Both approaches utilize statistical techniques, machine learning algorithms like AE, and data stream mining. Moreover, intelligence in this context involves several aspects: i) adapting the aggregation process dynamically by determining when and how it should be performed; ii) consolidating and correlating different types of measurements; and iii) enhancing the value of measurements, for instance, by utilizing AEs to assess the relevance of metrics in the latent feature space and input in both forward and backward directions.

Distributed decision making has been suggested as a way to handle network and service operation. Its purpose is not only to reduce the workload on the SDN controller and improve scalability but also to enable near real-time control. In this approach, agent nodes are equipped with intelligent algorithms, such as reinforcement learning, which enable them to make independent decisions based on the observed conditions gathered through telemetry. Furthermore, agents can communicate with one another, even if they have varying capabilities, to form distributed control systems known as multi-agent systems (MAS). These systems collaborate to achieve a common objective, such as ensuring end-to-end delay for services despite changing conditions.

We can illustrate an example that integrates the solutions mentioned earlier in the previous section for the near real-time management of end-to-end 6G services. Figure 5.3 showcases the situation where a particular service enables communication between an Unmanned Aerial Vehicle (UAV) and a VNF that offers computational and storage capabilities for the real-time reconstruction of high-quality Augmented/Virtual Reality (AR/VR) videos. The application imposes strict demands on end-to-end and segment latency as well as jitter, necessitating ongoing monitoring of numerous stream flows. This monitoring allows for dynamic decision-making regarding routing and edge computing resource allocation to meet the promised performance requirements.

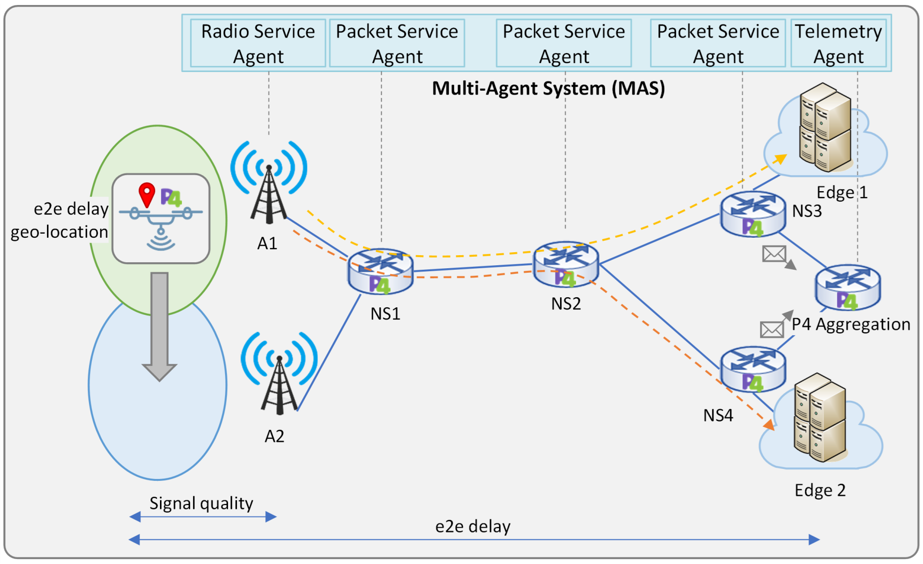


Figure 5.3: Illustrative scenario supporting en e2e 6G service. [27]

Service agents, as shown in Figure 5.4, can be comprised of the following components: i) A telemetry processor that gathers and handles telemetry data from the local node, incorporating intelligent data aggregation; ii) An inter-agent communication module that facilitates the distribution of telemetry data and the sharing of states and models among agents; iii) Technology-specific intelligence (such as RAN, packet, etc.) that enables autonomous decision-making based on both local and remote observations. Meanwhile, the Service Management and Orchestration (SMO) system offers guidelines to the MAS while allowing the MAS the flexibility to operate on the resources assigned to the service.

In Fig. 5.4., this operation is depicted, showing how agents receive resources and the maximum end-to-end delay (dmaxe2e) from the SMO for the connectivity service. To meet the required performance, each segment is initially allocated a budget delay (dmaxDi) (labeled 1 in Fig. 5.4.). Once operational, the end-to-end delay and other performance indicators are measured and shared among the agents in the MAS (2). For the radio segment, measurements and predicted metrics are utilized to adjust the resource block group adaptation with a time buffer, minimizing service Service Level Agreement (SLA) violations. However, suppose the radio segment is unable to maintain the promised delay within its domain at a certain point in time. In that case, the RAN agent communicates the new delay budget for its segment to the other service agents (3), enabling packet agents to make decisions such as altering routing to ensure the new budget delay within their domains.

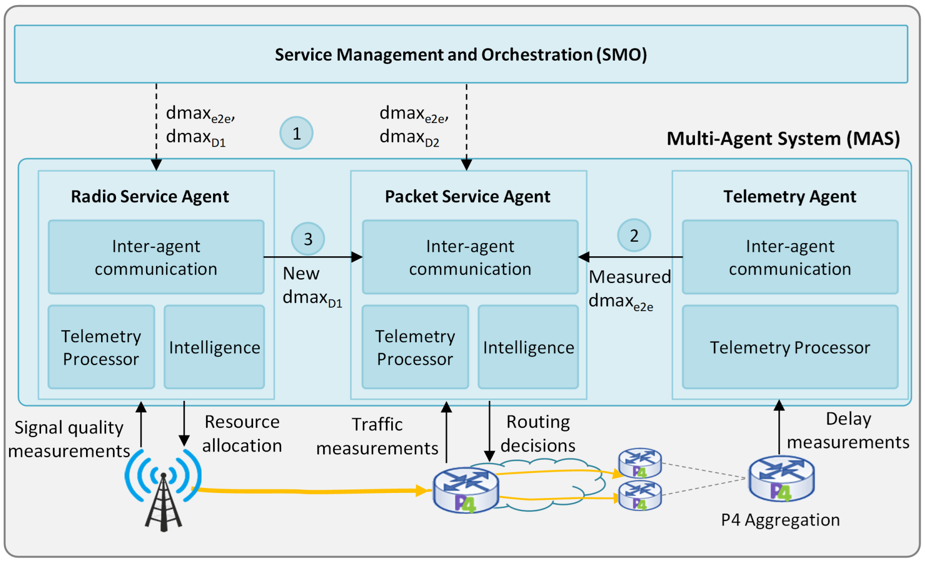


Figure 5.4: Proposed telemetry and distributed intelligence solution. [27]

This architecture was proposed and presented in the EUCNC & 6G Summit conference held in Goteborg in June 2023. This work has been developed in the context of the HORIZON SNS JU DESIRE6G (G.A. 101096466) project. The primary goal of the DESIRE6G project is to design, develop, and demonstrate a new wireless communication system that will provide near real-time autonomic networking and support extreme ultra-reliable low-latency communication (eURLLC) application requirements. In simpler terms, DESIRE6G is working to create a new network that is even faster, more reliable, and efficient than current 5G networks to meet the demands of new applications.[28]

Chapter 6

Conclusions

This work presents the GODAI architecture, a distributed system framework intended to bring intelligence to the edges in a flexible, scalable, and secure way. The state of the art is presented to take into account the many solutions already available and their capabilities. Several technologies and techniques are explored and compared to find the ones that are more suitable for the problem statement. Knowing this information, a novel architecture has been developed to address all the known issues.

The benefits of telemetry in operation and automation in networks have been widely demonstrated in the literature, and the existing need to build efficient telemetry architectures is shown. In particular, novel telemetry architectures should be able to address security, scalability, and reliability issues to enable near real-time operation in future networks.

The architecture has been defined, and all its components have been implemented and tested. A complete description of all the existing components inside GODAI is given to showcase how it works. The main capabilities and features of GODAI have been listed, as have the future work improvements that will be addressed to improve the overall operation of the architecture.

Security in GODAI is explored to show the security techniques that are already applied. DLT is presented as an improvement to be implemented in the future to solve some of the performance issues with the existing architecture.

Finally, two real use cases have been presented, showing the implementation and operation of the architecture in different scenarios. The first one focuses on building a distributed telemetry architecture that can process telemetry measurements from different data sources as well as telemetry events coming from an SDN controller. The second one presents a monitoring system in combination with a MAS that will enable near-real-time control of 6G services by monitoring and configuring the network based on E2E delay requirements.

6.1 Contributions and publications

Participation in different national and European projects developing software to support B5G and 6G future networks. Specifically, I have been involved in the following projects:

- HORIZON-HORIZON-SNS SElf-mAnaged Sustainable high-capacity Optical Networks (SEASON).
- HORIZON-SNS Deep Programmability and Secure Distributed Intelligence for Real-Time End- to-End 6G Networks (DESIRE6G).
- HORIZON-SNS PRogrammable AI-Enabled DeterminIstiC neTworking for 6G (PREDICT-6G).
- H2020 Beyond 5G - OPtical nEtwork coNtinuum (B5G-OPEN).
- MINECO AI-Powered Intent-Based Packet and Optical Transport Networks and Edge and Cloud Computing for Beyond 5G (IBON)

Several publications presented in international conferences:

- *An Intelligent Optical Telemetry Architecture* presented at **OFC 2023**.
- *Distributed Architecture Supporting Intelligent Optical Measurement Aggregation and Streaming Event Telemetry* presented at **OFC 2023**.
- *Securing a Multi-Agent System for near Real-Time Control of 6G Services* presented at **EUCNC 2023 & 6G Summit**.
- *Pervasive Monitoring and Distributed Intelligence for 6G near Real-Time Operation* presented at **EUCNC 2023 & 6G Summit**.
- *Distributed Intelligence for Pervasive Optical Network Telemetry*, under revision in IEEE/OPTICA J. Opt. Comm. and Netw., 2023.

6.2 Future Work

As this framework is in continuous development, there are several features and improvement that may arise in the future. Here there is a list of the main features to be implemented:

- Use a DLT to secure the GODAI nodes instead of having one node as a Manager of the keys.
- Improve the Web-UI to offer more functionalities and insights.
- Provide bindings to other programming languages to allow compatibility with more modules.

Appendix A

Appendix 1: Configuration files

```
{
  "manager": {
    "name": "manager",
    "type": "manager",
    "input": "input",
    "loglevel": "INFO",
    "redis": {
      "host": "localhost",
      "port": 6379
    },
    "Mng_If": {
      "host": "0.0.0.0",
      "port": 7001
    },
    "SecManager": {
      "key_size": 4096
    },
    "Agents": {
      "node-1": {"ip":  "localhost", "port":  5001},
      "node-2": {"ip":  "localhost", "port":  5002}
    }
  },
  "Components": {
  }
}
```

Listing A.1: Manager node configuration file

```
{
  "manager": {
    "name": "node-1",
    "type": "agent",
    "input": "input",
    "loglevel": "INFO",
    "redis": {
      "host": "localhost",
      "port": 6379
    },
    "Mng_If": {
      "host": "0.0.0.0",
      "port": 5001
    },
    "SecManager": {
      "key_size": 4096,
      "trusted_peers": {
      }
    },
    "Manager": {
      "manager": {"ip":  "localhost", "port":  7001}
    }
  },
  "Components": {
  }
}
```

Listing A.2: Agent node configuration file

```
  "Components": {
    "gRPC_If": {
      "class": "gRPC",
      "package": "Services.gRPC",
      "instance": {
        "workflows": {
          "AE/Evolution": [{"ip": "localhost", "port": 50051}],
          "FeX/Evolution": [{"ip": "localhost", "port": 50051}]
        }
      },
      "config": null,
      "running": true
    },
    "AE": {
      "class": "AE",
      "package": "Algorithms.AE",
      "instance": {
        "workflows": {
          "AE/Evolution": ["gRPC_If"]
        },
        "mode": "compress"
      },
      "config": null,
      "running": true
    },
    "FeX": {
      "class": "FeX_Constellation",
      "package": "Algorithms.FeX_Constellation",
      "instance": {
        "workflows": {
          "FeX/Evolution": ["Aggregator"]
        }
      },
      "config": null,
      "running": true
    },
    "Aggregator": {
      "class": "Aggregator",
      "package": "Algorithms.Aggregator",
      "instance": {
        "workflows": {
          "FeX/Evolution": ["gRPC_If"]
        }
      },
      "config": "Aggregator/FeX_Constellation.json",
      "running": true
    }
  }
```

Listing A.3: Components configuration

References

[1] Tarik Taleb et al. “White Paper on 6G Networking”. In: *6G Research Visions, No. 6* (2020).

[2] Amin Shahraki et al. “A Comprehensive Survey on 6G Networks:Applications, Core Services, Enabling Technologies, and Future Challenges”. In: (Jan. 2021).

[3] Luis Velasco et al. “Introduction to the JOCN Special Issue on Advanced Monitoring and Telemetry in Optical Networks”. In: *Journal of Optical Communications and Networking* 13 (10 Oct. 2021), AMTON1. ISSN: 1943-0620. DOI: `10.1364/JOCN.442735`.

[4] Peter Murray and Paul Stalvig. “SNMP: Simplified”. In: ().

[5] Chunjin Zhang and Shujuan Ji. “A SNMP-base broadcast storm identification method in VLAN”. In: Atlantis Press, 2013. ISBN: 978-90-78677-67-3. DOI: `10.2991/iccnce.2013.11`.

[6] *gRPC: A high performance, open source universal RPC framework*. `https://grpc.io/`.

[7] Francesco Paolucci et al. “Network Telemetry Streaming Services in SDN-Based Disaggregated Optical Networks”. In: *Journal of Lightwave Technology* 36 (15 Aug. 2018), pp. 3142–3149. ISSN: 0733-8724. DOI: `10.1109/JLT.2018.2795345`.

[8] D. King et al. “The dichotomy of distributed and centralized control: METRO-HAUL, when control planes collide for 5G networks”. In: *Optical Switching and Networking* 33 (July 2019), pp. 49–55. ISSN: 15734277. DOI: `10.1016/j.osn.2018.11.002`.

[9] Claus Töpke. *Network Programming and Automation Essentials*. Packt Publishing, 2023.

[10] *Apache Kafka*. `https://kafka.apache.org/`.

[11] Andrea Sgambelluri et al. “Reliable and scalable Kafka-based framework for optical network telemetry”. In: *Journal of Optical Communications and Networking* 13 (10 Oct. 2021), E42. ISSN: 1943-0620. DOI: `10.1364/JOCN.424639`.

[12] *Redis*. `https://redis.io/`.

[13] Ercan Erdogan. *Redis Pub/Sub with .net Core*. `https://medium.com/innoviletech/redis-pub-sub-with-net-core-758c1d3c7a98`.

[14] L. Velasco, S. Barzegar, and M. Ruiz. "Is Intelligence the Answer to Deal with the 5 V's of Telemetry Data?" In: IEEE, Mar. 2023, pp. 1–3. DOI: `10.23919/OFC49934.2023.10116324`.

[15] L. Velasco et al. "Monitoring and Data Analytics for Optical Networking: Benefits, Architectures, and Use Cases". In: *IEEE Network* 33 (6 Nov. 2019), pp. 100–108. ISSN: 0890-8044. DOI: `10.1109/MNET.2019.1800341`.

[16] F. Alhamed et al. "P4 Postcard Telemetry Collector in Packet-Optical Networks". In: IEEE, May 2022, pp. 1–3. ISBN: 978-3-903176-44-7. DOI: `10.23919/ONDM54585.2022.9782868`.

[17] N. Williams and S. Zander. *Evaluating machine learning algorithms for automated network application identification*. eng. Melbourne, VIC, 2006.

[18] Luis Velasco, Pol González, and Marc Ruiz. "An Intelligent Optical Telemetry Architecture". In: Optica Publishing Group, 2023, M3G.1. ISBN: 978-1-957171-18-0. DOI: `10.1364/OFC.2023.M3G.1`.

[19] Luis Velasco et al. "Securing Multi-Agent Systems for Near Real-Time Control of 6G Services". In: June 2023.

[20] *Distributed Systems Security*. `https://pk.org/417/notes/crypto.html`.

[21] Kiril Antevski and Carlos J. Bernardos. "Federation of 5G services using distributed ledger technologies". In: *Internet Technology Letters* 3 (6 Nov. 2020). ISSN: 2476-1508. DOI: `10.1002/itl2.193`.

[22] Mallik Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348. Aug. 2014. DOI: `10.17487/RFC7348`. URL: `https://www.rfc-editor.org/info/rfc7348`.

[23] *B5G-OPEN*. `https://www.b5g-open.eu/project/`.

[24] Pol González et al. "Distributed Architecture Supporting Intelligent Optical Measurement Aggregation and Streaming Event Telemetry". In: Optica Publishing Group, 2023, M3Z.4. ISBN: 978-1-957171-18-0. DOI: `10.1364/OFC.2023.M3Z.4`.

[25] M. Ruiz, D. Sequeira, and L. Velasco. "Deep learning-based real-time analysis of lightpath optical constellations [Invited]". In: *Journal of Optical Communications and Networking* 14 (6 June 2022), p. C70. ISSN: 1943-0620. DOI: `10.1364/JOCN.451315`.

[26] Nizar Bouguila and Wentao Fan, eds. *Mixture Models and Applications*. Springer International Publishing, 2020. ISBN: 978-3-030-23875-9. DOI: `10.1007/978-3-030-23876-6`.

[27] Luis Velasco et al. "Pervasive Monitoring and Distributed Intelligence for 6G Near Real-Time Operation". In: June 2023.

[28] *DESIRE6G*. `https://desire6g.eu/project/`.