TABLE III
CATEGORIZATION OF ROBOT SOFTWARE ARCHITECTURE BASED ON COMMUNICATION PROTOCOL

Communication paradigm	Description	Examples	Features	Application examples
Client–server	In this paradigm, one component (the client) requests services or data from another component (the server). It is a one-to-one communication model.	Robot operating systems (ROS), remote desktop protocol, file transfer protocol, simple mail transfer protocol.	Centralized control and data sharing, suitable for applications with a single point of control.	Remote robotic control over a network, teleoperation systems.
Publisher–subscriber (Pub–Sub) ([13])	This paradigm uses a publish-subscribe model where one or more publishers send messages or data to one or more subscribers. It is a one-to-many communication model.	ROS, MQTT, advanced message queuing protocol, Apache Kafka.	Decoupled communication, scalable and efficient for broadcasting data, supports real-time updates.	Robot sensor data distribution, event-driven robotics systems.
Peer–to–peer (P2P) ([14])	In a peer-to-peer network, all components are equal and can communicate directly with each other. There is no centralized server.	BitTorrent, Skype, Bitcoin.	Distributed control and decision-making, robust and fault-tolerant, suitable for decentralized systems.	Multirobot coordination and collaboration, swarm robotics.
Master– slave ([15])	This paradigm involves a single master or coordinator component (the master) that controls and communicates with multiple subordinate components (the slaves).	SETI	Hierarchical control, efficient resource allocation, well-suited for coordinated tasks.	Multi-robot assembly lines, collaborative robots (cobots).
Broadcast ([16])	In a broadcast communication model, a message or data is transmitted to all components in the network. It is a one-to-all communication model.	UML-RT, address resolution protocol.	Simplicity and efficiency in data distribution, useful for broadcasting commands or status updates.	Broadcasting control commands to multiple robots, notification systems in robot fleets.
Event–driven ([17])	This paradigm focuses on reacting to events or triggers. Components subscribe to specific events and respond when those events occur.	Apache Kafka, AWS Lambda, Apache Storm.	Responsive and event-triggered behavior, supports asynchronous communication.	Robot behaviors triggered by environmental events, reactive control systems.
Multicast ([18])	Multicast communication allows a message or data to be sent to a selected group of components, often specified by an IP multicast address.	multiparty audio/video conferencing.	Efficient data distribution to specific groups, reduces network load compared to broadcasting.	Coordinated behavior in subgroups of a robot team, networked robot simulations.
Point–to–point ([19])	Data is exchanged directly between two specific components or robots.	LAN cable, stovepipes.	Direct communication, simplicity.	Information exchange between farm vehicles and sensors.
Request–response ([20])	One component sends a request, and another component responds with the requested information or action.	Sending a spreadsheet to the printer.	Simple request handling, suitable for remote control.	Remote monitoring and control of irrigation systems.
Message queues ([21])	Messages are sent to a queue, and other components can retrieve them, providing asynchronous and decoupled communication.	Apache Kafka, RabbitMQ, and LavinMQ.	Task scheduling, event handling.	Task management and coordination in precision agriculture.

layers, allowing the robot to perform complex behaviors in a modular fashion.

5) Centralized Architecture Versus Distributed Architecture: Centralized architecture organizes the robotic system around a central processing unit, while distributed architecture allows multiple processing units to work together to achieve a common goal. Example, client–server architecture (centralized architecture) is a centralized architecture where a single server provides resources or services to multiple clients. Clients request resources or services from the server, which then responds with the requested information. This architecture is often used in web applications or database systems; peer-to-peer architecture (distributed architecture) is a distributed architecture [12] where multiple nodes communicate directly with each other without the need for a central server. Each node can act as both a client and a server, making it more fault-tolerant and scalable than a centralized architecture. This architecture is often used in file-sharing or messaging systems.

B. Categorization Based on Communication Strategies

Robotic software architectural paradigms can be categorized based on communication protocols that facilitate interactions between different components of a robotic system. Table III, categorizes some common robotic software paradigms based on communication protocols, accompanied by their descriptions, features, and examples of applications.

These communication paradigms play a crucial role in enhancing agricultural processes, from precision farming and crop monitoring to pest control and automation of various tasks. The choice of communication paradigm depends on the specific needs and challenges of agricultural robotics applications.