Chapter Outline

Key Communication Protocols and Development Interfaces

Chapter 9 Multi-Vehicle Communication and Intelligent Networking

9.1 Networking Fundamentals

9.2 Hardware and Link Mechanisms

9.3 Software Layers and Protocols

9.4 Channel and Interference Modeling

9.5 RflySim Networking and Simulation

9.6 Protocols and Development Interfaces

9.7 Experiments and Validation

9.8 Exercises and Classroom Discussion

4 Units in This Section

9.6.1 MAVLink Protocol

Textbook: Section 9.2.2.1

Micro Air Vehicle Link (MAVLink)

9.6.2 DDS Middleware

Textbook: Section 9.2.2.2

Data Distribution Service (DDS)

9.6.3 OLSR Routing Protocol

Textbook: Section 9.2.2.3

Optimized Link State Routing (OLSR)

9.6.4 Common Interfaces and Custom Development

Textbook: Section 9.2.3

9.6 Protocols and Development Interfaces

9.6.1 MAVLink Protocol: Roles of the Communication Protocols

Message Structure, Endpoint Matching, and Routing Serve Different Roles

Question	Mechanism in This Section	Direct Result
How does a receiver interpret bytes?	MAVLink message encoding	A parseable message frame
Which programs receive which data?	DDS publish/subscribe; quality of service (QoS)	Matched Writer and Reader
Next hop for an Internet Protocol (IP) packet?	OLSR proactive route maintenance	Local destination → next-hop table

Architecture determines the combination; these mechanisms are not a required serial chain.

9.6.1 MAVLink Protocol: Design Goals and Versions

MAVLink Carries Typed Messages in Compact Frames

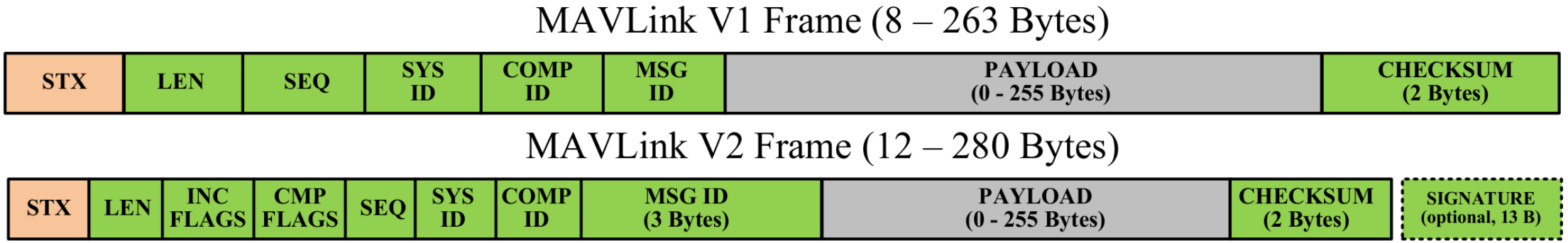


Figure 9.9, detail: v1 and v2 frame structures

Message mechanism: MsgID identifies the field structure. Generated libraries encode, parse, and compute the cyclic redundancy check (CRC).

Version changes: v2 extends message IDs and fields, supports signing and trailing-zero truncation. Check capabilities when communicating with older nodes.

9.6.1 MAVLink Protocol: Transmission Flow

Figure 9.9 Shows Encoding, Transmission, and Parsing

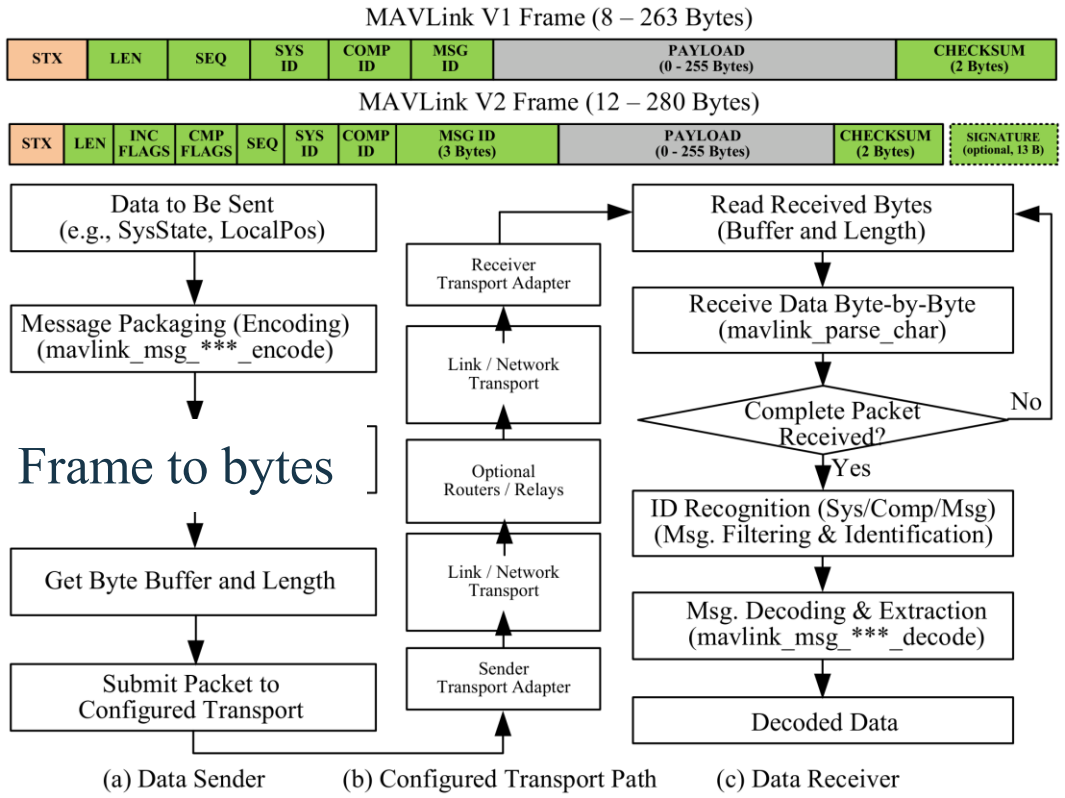


Figure 9.9 MAVLink Data Frames and Transmission Flow

Left: structure → encode → frame bytes → send over the actual link.

Right: bytes → complete, valid frame → decode by message ID.

DDS is optional; OLSR applies to the relevant multi-hop IP network.
UDP: User Datagram Protocol.
TCP: Transmission Control Protocol.
CAN: Controller Area Network.
UART: Universal Asynchronous Receiver/Transmitter.

9.6.1 MAVLink Protocol: Field Table, Offsets 0–4

The Fixed Header Identifies Version, Length, and Sequence

Byte Offset	Field / Bytes	Meaning and Handling
0	Magic / 1	0xFD identifies MAVLink v2
1	Length / 1	Actual payload length n, in bytes; range 0–255
2	IncompatFlags / 1	Reject frames with unknown incompatibility flags
3	CompatFlags / 1	Unknown compatibility flags may be ignored
4	Seq / 1	Wraps through 0–255; helps analyze loss and reordering

n denotes payload length; offsets identify byte positions.

9.6.1 MAVLink Protocol: Field Table, Offsets 5–9

System ID, Component ID, and Message Type Are Distinct

Byte Offset	Field / Bytes	What It Identifies
5	SysID / 1	Sending system, such as vehicle 1
6	CompID / 1	Sending component, such as an autopilot or payload pod
7–9	MsgID / 3	Message type; sent low byte, middle byte, then high byte

Example: MsgID=32 means LOCAL_POSITION_NED, not vehicle 32.
Header source IDs, payload target IDs, IP addresses, and UDP ports mean different things.

9.6.1 MAVLink Protocol: Field Table, Variable-Length Fields

Payload Length N Determines CRC and Signature Offsets

Byte Offset	Field / Bytes	Explanation
$10 \ldots (9+n)$	Payload / n	From the message definition; absent when n=0
$(n+10) \ldots (n+11)$	CRC / 2	Frame checksum incorporating CRC_EXTRA
$(n+12) \ldots (n+24)$	Signature / optional 13	Appended when the signing flag is set

CRC: error detection, not retransmission or authentication. Signing: authentication and integrity, not encryption.
Custom messages need compatible Extensible Markup Language (XML) definitions and generated libraries at both ends.

9.6.1 MAVLink Protocol: Frame-Length Example

Actual Payload and Signing Determine Frame Length

$$L_{v2} = 10 + n + 2 + 13s, \quad s \in \{0,1\}, \quad n = 28$$

L_{v2}: frame bytes; n: payload bytes; $s = 0$: unsigned; $s = 1$: signed.

Part	Offsets in This Example, Inclusive	Length
Fixed header	0–9	10B
Payload	10–37	28B
CRC / optional signature	38–39 / 40–52	2B / 13B

Result: 40 B unsigned, 53 B signed; excludes UDP/IP and link-layer overhead.

9.6.1 MAVLink Protocol: Encoding and Parsing Interfaces

Generate Frame Bytes; Read Fields Only After Successful Parsing

```
mavlink_msg_heartbeat_encode(
    sysid, compid, &msg, &hb);
frame_len =
    mavlink_msg_to_send_buffer(
        buf, &msg);
send_bytes(buf, frame_len);
```

```
if (mavlink_parse_char(
    chan, byte, &msg, &status)) {
  if (msg.msgid == 0)
    mavlink_msg_heartbeat_decode(
        &msg, &hb);
}
```

Initialize hb, msg, buf, and status using generated-library types; frame_len is the entire frame length in bytes.

send_bytes is a custom send function. These fragments also need source filtering, error handling, and application logic after valid frames.

9.6.1 MAVLink Protocol: Position-Message Mapping

The Installed Interface Decodes Position into a State Object

```
if msg.get_type() == "LOCAL_POSITION_NED":
    self.uavPosNED[0] = msg.x
    self.uavPosNED[1] = msg.y
    self.uavPosNED[2] = msg.z
    self.netForwardData()
```

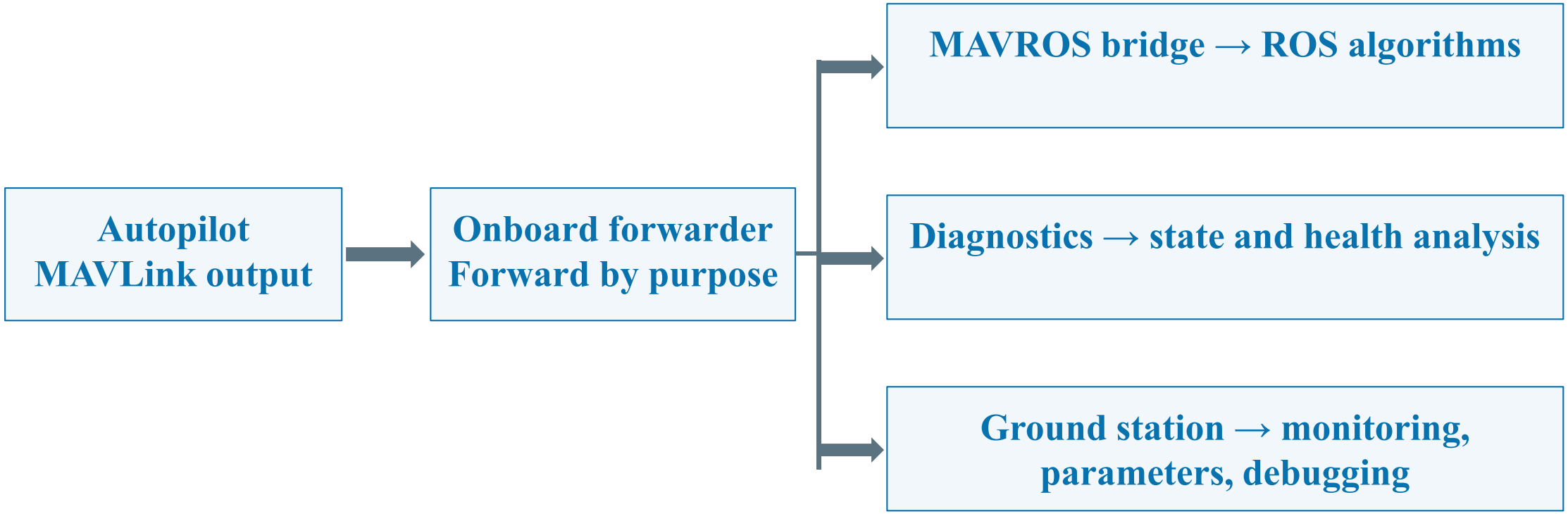
Input: a parsed MAVLink message with type ID 32.

Output: position enters uavPosNED, then triggers forwarding.

Source: getMavMsg in RflySimSDK/ctrl/PX4MavCtrlV4.py.
The full branch also updates velocity, globally referenced position, and events; only position and forwarding are shown.
North–east–down (NED), m: msg.x north, msg.y east, msg.z down; all from the local origin.

9.6.1 MAVLink Protocol: Forwarding to Multiple Nodes

A MAVLink Stream Can Serve Several Consumers in Parallel



MAVROS bridges MAVLink to the Robot Operating System (ROS). Configure messages, rates, and permissions per task. Bridging, filtering, and duplicate traffic affect timeliness.

Chapter Outline

Key Communication Protocols and Development Interfaces

Chapter 9 Multi-Vehicle Communication and Intelligent Networking

9.1 Networking Fundamentals

9.2 Hardware and Link Mechanisms

9.3 Software Layers and Protocols

9.4 Channel and Interference Modeling

9.5 RflySim Networking and Simulation

9.6 Protocols and Development Interfaces

9.7 Experiments and Validation

9.8 Exercises and Classroom Discussion

4 Units in This Section

9.6.1 MAVLink Protocol

Textbook: Section 9.2.2.1

9.6.2 DDS Middleware

Textbook: Section 9.2.2.2

9.6.3 OLSR Routing Protocol

Textbook: Section 9.2.2.3

9.6.4 Common Interfaces and Custom Development

Textbook: Section 9.2.3

9.6.2 DDS Middleware: The DCPS Conceptual Model

DDS Links Producers and Consumers by Topic and Data Type

Data Distribution Service (DDS); Data-Centric Publish-Subscribe (DCPS).



Domain and participant: a Domain is a logical communication scope. DomainParticipant manages local entities and their lifetimes.

Sample path: Writer writes a sample → matched Readers receive it. A Topic is not a central message queue.

9.6.2 DDS Middleware: Discovery and Matching

A Shared Domain Is Only One Requirement for Matching

Participant discovery
Domain / reachability

→

Endpoint matching
Topic, type, QoS

→

Sample delivery
Writer → Reader

Setup or Observation	Establishes	Does Not Establish
Same Domain	Same logical domain	Correct topic, type, or permissions
New-match callback	Endpoints have matched	Application has received a fresh, valid sample
Valid sample received	Data arrived and can be parsed	All latency and control requirements are met

9.6.2 DDS Middleware: Quality-of-Service Policies

QoS Policies Govern Reliability, History, and Time Semantics

Policy	Behavior Controlled	State-Stream Check
Reliability	BEST_EFFORT / RELIABLE	Loss tolerance versus retransmission overhead
History / ResourceLimits	KEEP_LAST depth / KEEP_ALL with resource limits	Avoid unbounded backlog; retain the history needed
Deadline / Lifespan	Update-interval monitoring / sample expiration	Detect stopped updates / exclude expired samples
LatencyBudget	Intended communication latency allowance	Does not replace end-to-end measurement or scheduling

Quality of service (QoS) must be compatible at both ends. The application must also check timestamps and task deadlines.

9.6.2 DDS Middleware: A Freshness Example

Reliably Retaining Old Data Can Still Delay Control

$$f_{\text{pub}} = 50\ \text{Hz}, \quad f_{\text{use}} = 10\ \text{Hz}, \quad \Delta Q = 5 - 1 = 4$$

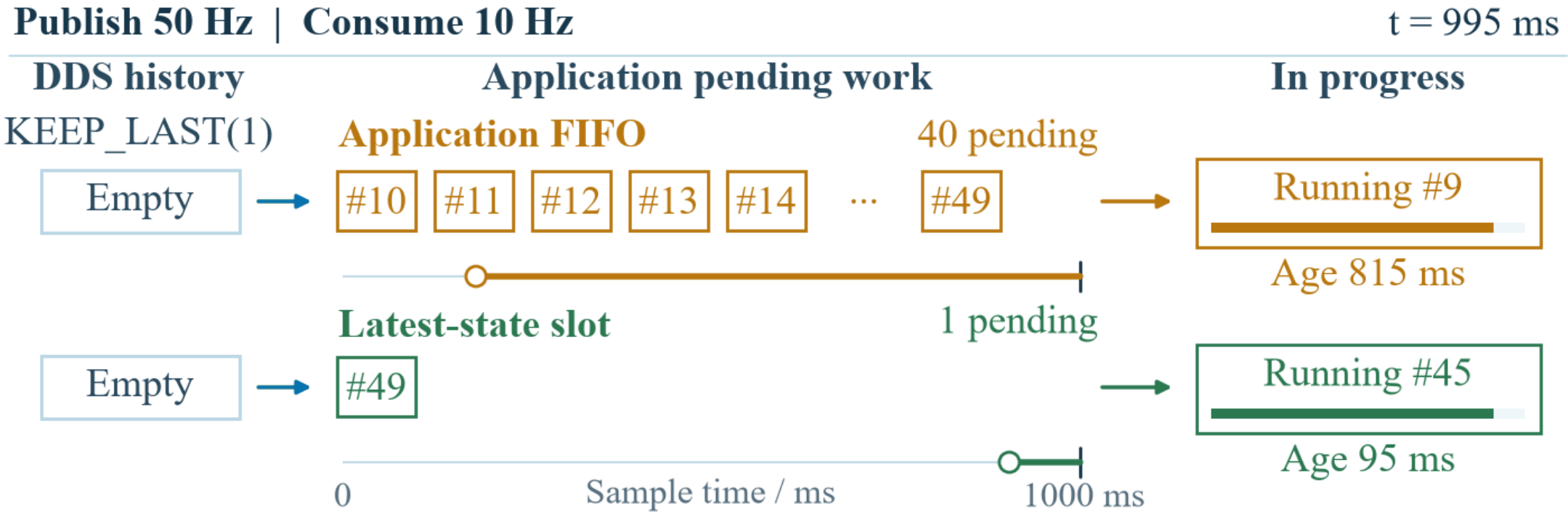
f_{pub}: publish rate; f_{use}: consumption rate. One per action; all arrive, none discarded. First in, first out (FIFO): 5 arrive, 1 processed per 0.1 s; ΔQ counts the added pending samples.

Application Need	Possible Design	Still to Verify
Latest vehicle state	KEEP_LAST(1); read the latest sample	Data age, discard policy, continuity
Events that must be retained	Reliable delivery and bounded persistent records	Processing capacity, resource limits, recovery mechanism

KEEP_LAST(1) bounds history depth. It cannot remove an old task already running; manage application queues too.

9.6.2 DDS Middleware: Consumption and State Age

Application Queues Change Freshness at the Same History Depth



Open circles: sampling times of running tasks; vertical line: current time

KEEP_LAST(1) limits per-instance history; manage application queues and active tasks.

9.6.2 DDS Middleware: Open-Source Implementation and Transport

Fast DDS Maps DDS Entities to RTPS Endpoints



RTPS (Real-Time Publish-Subscribe) defines endpoint discovery and data exchange.
RTPSParticipant organizes endpoints.
RTPSWriter and RTPSReader exchange changes below DDS.

Compatibility: sharing a standard does not make every version and configuration interoperable. Discovery, types, and QoS must be compatible.

Latency budget: check whether the current version implements the policy, then measure network, cache, and application-processing delays.

9.6.2 DDS Middleware: DCPS Entities in the Example

HelloWorld Registers a Type Before Creating a Topic

```
data_type = HelloWorld.HelloWorldPubSubType()
data_type.setName("HelloWorldDataType")
support = fastdds.TypeSupport(data_type)
participant.register_type(support)
topic = participant.create_topic(
    "myTopic", data_type.getName(), qos)
```

Type: generated PubSubType provides type and serialization support.

Topic: name and type jointly determine potential matches.

This excerpt omits self prefixes and renames local variables; fastdds, HelloWorld, and participant are initialized.
Next, create Writer/Reader entities, register listeners, and write or read samples.

9.6.2 DDS Middleware: Minimal Entry Point and Troubleshooting

Endpoint Matches Precede Checks of Sample Content and Timeliness

[installation directory]/RflySimAPIs/9.RflySimComm/
0.ApiExps/e2.DDSDepsTestExps/Demo/
Parent folder: launch Python38Run.bat twice; cd Demo in both. Start subscriber first.

```
python -u HelloWorldExample.py -p subscriber -d 0
python -u HelloWorldExample.py -p publisher -d 0
```

Options: -d 0 sets domain 0; publisher exits after 10 messages. Press Enter to stop subscriber.
Troubleshoot: dependencies → Domain / network reachability → Topic / type / QoS → matching → reception.

Expected: both report matched; receiver prints Received Hello World : 0 through 9 in order.
Validate: check read return value, info.valid_data, and arrival times. A live process alone is insufficient.

Chapter Outline

Key Communication Protocols and Development Interfaces

Chapter 9 Multi-Vehicle Communication and Intelligent Networking

9.1 Networking Fundamentals

9.2 Hardware and Link Mechanisms

9.3 Software Layers and Protocols

9.4 Channel and Interference Modeling

9.5 RflySim Networking and Simulation

9.6 Protocols and Development Interfaces

9.7 Experiments and Validation

9.8 Exercises and Classroom Discussion

4 Units in This Section

9.6.1 MAVLink Protocol

Textbook: Section 9.2.2.1

9.6.2 DDS Middleware

Textbook: Section 9.2.2.2

9.6.3 OLSR Routing Protocol

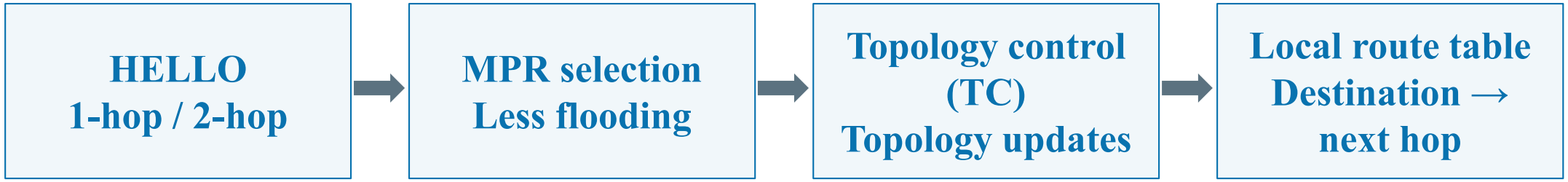
Textbook: Section 9.2.2.3

9.6.4 Common Interfaces and Custom Development

Textbook: Section 9.2.3

9.6.3 OLSR Routing Protocol: Neighbor Discovery and Overall Flow

OLSR Maintains Routes Proactively; HELLO Probes Local Neighbors



HELLO: exchanged only within one hop. Distinguish one-way and symmetric links; the strict two-hop set excludes self and one-hop neighbors.

MPR means Multipoint Relay; it does not stand for multipath routing.

Maintenance cost: shorter update intervals track change more closely and increase control traffic. Expiration and convergence still take time.

9.6.3 OLSR Routing Protocol: The MPR Coverage Objective

Equation 9.5 Covers Strict Two-Hop Neighbors with Few Relays

$$\min|\mathrm{MPR}(i)| \quad \text{s.t.} \quad \bigcup_{j\in\mathrm{MPR}(i)} R(j) \supseteq N_2(i)$$

Eq. (9.5)

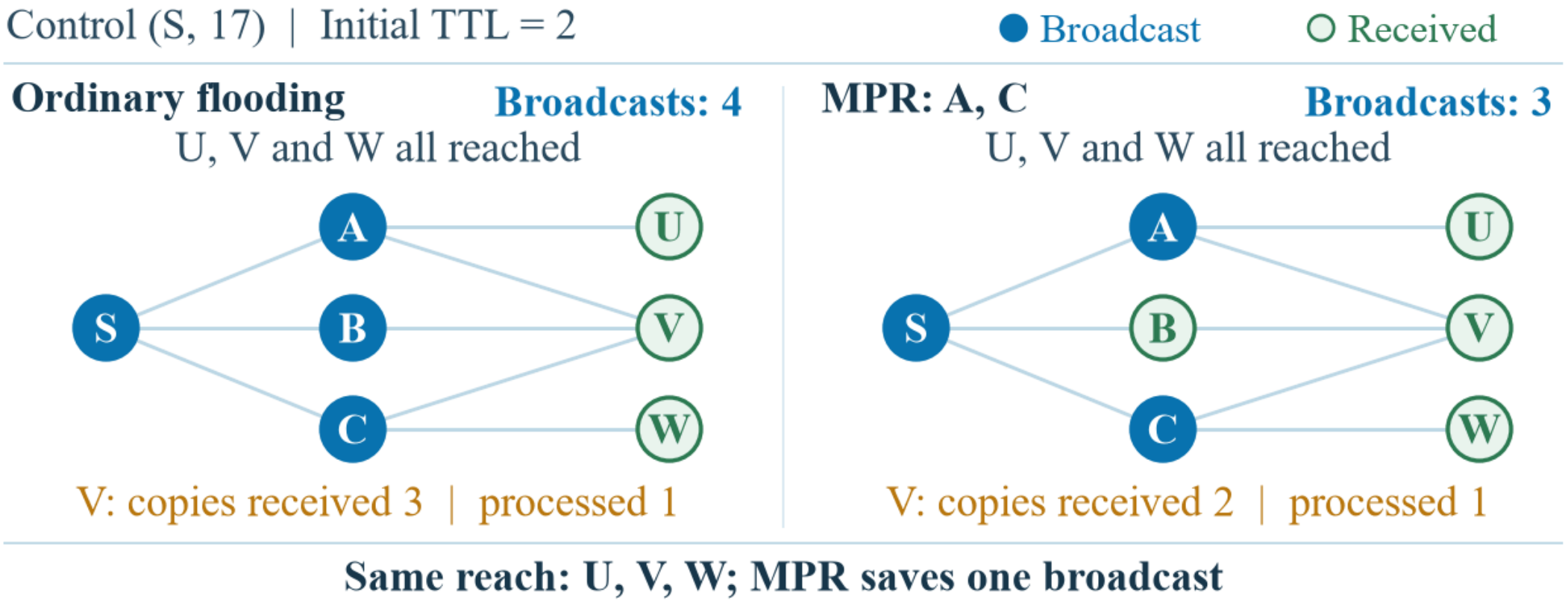
Sets: i is this node; $\mathrm{MPR}(i)$ is its selected set of symmetric one-hop relays; $|\cdot|$ counts nodes. $R(j)$ is the strict two-hop set covered by neighbor j; $N_2(i)$ is this node's full strict two-hop neighbor set.

Coverage: the union of selected coverage sets must include all strict two-hop neighbors; the goal is fewer redundant broadcasts.

Heuristic: select mandatory candidates first, then cover more of the uncovered set. Global optimality is not guaranteed.

9.6.3 OLSR Routing Protocol: MPR Coverage and Flooding

MPR Reduces Redundant Broadcasts for the Same two-hop Coverage



With time to live (TTL)=2: ordinary flooding uses 4 broadcasts, MPR uses 3; both detect duplicates.

9.6.3 OLSR Routing Protocol: Set-Cover Calculation

The Coverage Table Shows Each MPR Selection Step

Candidate or Step	Coverage Set	Uncovered Set / Reasoning
A / B / C	{U,V} / {V} / {V,W}	Initially: {U,V,W}
Select A	{U,V}	Remaining: {W}; only A covers U
Select C	{V,W}	None remaining; only C covers W
Minimum count in this example	One candidate covers at most 2	At least 2 needed; {A,C} attains this bound

Assume equal willingness and that all candidates are eligible. Each node computes its own MPR set locally.

9.6.3 OLSR Routing Protocol: TC and Next Hops

TC Advertises Topology; Routing Tables Direct Data

Stage	Exchange / Calculation	Result
A selects B as MPR	A announces the relationship in HELLO	B knows A is one of its selectors
B generates TC	Advertises at least the neighbors that selected B	Remote nodes learn the needed topology
Each node computes routes	Local neighbor tables and TC topology information	Destination, next hop, distance
Application-data forwarding	Send using the local IP routing table	Reach destination via reachable next hops

The MPR set, MPR selector set, and application-data routing table are three distinct objects.

9.6.3 OLSR Routing Protocol: Expected Transmission Count (ETX)

Equation 9.6 Converts Delivery Probabilities into Expected Attempts

$$\mathrm{ETX}(u,v) = \frac{1}{d_f d_r} \qquad \text{Eq. (9.6)}$$

d_f: forward delivery probability; d_r: reverse acknowledgment delivery probability; each in (0,1].

ETX is dimensionless, under an independent, stationary repeated-attempt model.

$$\mathrm{ETX}_{\mathrm{direct}} = \frac{1}{0.4 \times 0.8} = 3.125$$

u and v are link endpoints; subscript direct denotes this example's direct link.

Zero delivery probability makes a link unusable. Low ETX alone need not give minimum delay or maximum throughput.

9.6.3 OLSR Routing Protocol: The Shortest-Path Model

Equation 9.7 Minimizes the Chosen Sum of Edge Costs

$$d(s,v) = \min_{p \in \mathcal{P}(s,v)} \sum_{(x,y) \in p} w(x,y) \qquad \text{Eq. (9.7)}$$

s, v: start and end nodes; p: a candidate path; (x, y): an edge on that path.

$\mathcal{P}(s,v)$: candidate paths; $w(x,y)$: nonnegative edge weight; $d(s,v)$: minimum total cost.

Candidate Path from A to C	Hops	ETX Sum
A→C; success probabilities 0.4 / 0.8	1	3.125
A→B→C; each link 0.9 / 0.9	2	$2/(0.9 \times 0.9) \approx 2.469$

Illustrative inputs: ETX selects A→B→C; hop count selects A→C.

9.6.3 OLSR Routing Protocol: Complete Three-Node Calculation

Dijkstra Updates Costs and Next Hops Step by Step

Step	Settled Nodes	Cost to B	Cost to C / Next Hop
Initialize	A (cost 0)	1.235	3.125 / C
Settle B; relax edges	A, B	1.235	2.469 / B
Settle C	A, B, C	1.235	2.469 / B

Each round settles the unsettled node with the lowest cost, then updates neighbors through any cheaper path via it (relaxation).
Final routes at A: destination B → next hop B; destination C → next hop B.
Basic OLSR uses hop count. An ETX extension requires separate link measurement, advertisement, and route selection.

9.6.3 OLSR Routing Protocol: Implementation and Extensions

Basic OLSR and Quality or Prediction Extensions Need Separate Validation

```
case 1:
    list.Add(olsr, 100);
    break;
```

Source: rt=1 selects OlsrHelper in RflyNet.cc. Custom-module dependencies mean direct compilation is not assured.

Basic mechanisms: HELLO, MPR, TC, and hop-count routing. ETX is not included automatically.

Extension Direction	Information or Structure Added	Main Cost
Quality / prediction	Delivery ratio / position and motion model	Measurement error; overhead; prediction error
Clustering / hybrid	Inter-cluster organization / proactive–reactive switching	Maintenance / switching complexity

Chapter Outline

Key Communication Protocols and Development Interfaces

Chapter 9 Multi-Vehicle Communication and Intelligent Networking

9.1 Networking Fundamentals

9.2 Hardware and Link Mechanisms

9.3 Software Layers and Protocols

9.4 Channel and Interference Modeling

9.5 RflySim Networking and Simulation

9.6 Protocols and Development Interfaces

9.7 Experiments and Validation

9.8 Exercises and Classroom Discussion

4 Units in This Section

9.6.1 MAVLink Protocol

Textbook: Section 9.2.2.1

9.6.2 DDS Middleware

Textbook: Section 9.2.2.2

9.6.3 OLSR Routing Protocol

Textbook: Section 9.2.2.3

9.6.4 Common Interfaces and Custom Development

Textbook: Section 9.2.3

9.6.4 Common Interfaces and Custom Development: Point-to-Point Interfaces

Destination Vehicle IDs and UDP Ports Are Different Parameters

Interface Call	Parameter Meaning	Actual Result Here
enUavForward(2)	Target vehicle ID	224.0.0.10:60002
enNetForward([60002], ip)	Destination ports and IP address	ip:60002
StartNetRecOwn()	Own CopterID=2	Listen on 60002
StartNetRec(60002)	Receive port	Bind directly to 60002

NetSimAPI(mav) can forward on state-update events; NetSimAPI(1) only assigns source ID 1 and does not collect state automatically.

9.6.4 Common Interfaces and Custom Development: Minimal Point-to-Point Calls

Receive First, Then Forward Vehicle 1 State to Vehicle 2

Sender: vehicle 1

```
mav = PX4MavCtrl.PX4MavCtrler(1)
net = NetSimAPIV4.NetSimAPI(mav)
mav.InitMavLoop()
net.enUavForward(2)
```

Receiver: vehicle 2

```
net = NetSimAPIV4.NetSimAPI(2)
net.StartNetRecOwn()
for _ in range(100):
    if "1" in net.UavDict: break
    time.sleep(0.1)
uav1 = net.UavDict.get("1")
```

Imports: time, PX4MavCtrlV4 as PX4MavCtrl, NetSimAPIV4; use two processes.
Start SITLRun2MavlinkFull.bat; after initialization run UAV2Ctrl.py, then UAV1Ctrl.py.
uav1 set: first packet; still unset at ~10 s: timeout. Ongoing control needs timestamp checks.

Installed example: 9.RflySimComm/0.ApiExps/e1.RflyNetAPIExps/1.P2PSendData

9.6.4 Common Interfaces and Custom Development: State-Packet Parsing

Source ID, Time, and Coordinates Have Distinct Roles

Part	Fields or Format	Interpretation
32 B header	iiiiQd; source ID, mode, starting ID 64-bit mask, send wall-clock time	Identification, filtering, and timing statistics
104 B state	d6f9d; state time, attitude Velocity, GPS origin, two positions	Angles: rad; velocity: m/s NED position: m; GPS lat./lon.: degrees
Receive containers	UavDict['1']; UavData[0]	Dictionary: source ID; array: order of first arrival

12345678 only identifies format. Cross-platform formats must define byte order, version, and length. GPS: Global Positioning System.

9.6.4 Common Interfaces and Custom Development: Broadcast and Vehicle Filtering

Shared-port Distribution and Application Filtering Act at Different Points



Shared distribution: enUavForward(0) and StartNetRec() use the default shared port 60000.

Physical destinations: PortList determines where sendto sends. Multiple destination ports produce multiple packet copies.

Logical filtering: source-ID filtering occurs in the application. A destination mask constrains distribution only if the forwarder interprets it.

9.6.4 Common Interfaces and Custom Development: On-Demand State Requests

Request Lists Name Sources; Send Lists Name Destinations



Side or Action	List or Operation	Important Boundary
Requester	StartReqUavData([1])	Current loop requests at about 1 Hz
Responder	netAddUavSendList([2])	Retains destination IDs for later forwarding
Cancel and exit	Clear local list or stop requests	Does not automatically remove the remote destination

9.6.4 Common Interfaces and Custom Development: Custom-Data Forwarding

Custom Payloads Need a Byte Format and Checks Before Unpacking

```
payload = struct.pack("<Bff", 1, 2.5, -1.0)
net.netForwardBuf(payload)
# Receiver: after the new-packet event
buf = bytes(net.bufData)
net.bufEvent.clear()
if len(buf) == 9:
    ver, a, c = struct.unpack("<Bff", buf)
    if ver == 1:
        use_values(a, c)
```

Format: import struct; little-endian, 1 B version, two 4 B floats.

Enable enNetForward at sender; StartNetRecOwn at receiver.

Application: validate control-value ranges before use_values applies the data.

Unpacked result: ver=1, a=2.5, c=-1.0. The outer header still uses format iiiiQd.

9.6.4 Common Interfaces and Custom Development: Timing and Lifecycle

First Reception, Fresh Data, and Clean Shutdown Are Different States

Check	Common Mistake	Correct Criterion
Freshness	Keep using an existing dictionary entry	Sample time advances; degrade or exit on timeout
Cross-machine delay	Receive minus send wall time is network delay	Use a common time base; estimate synchronization error
Resource shutdown	A stop flag ends everything immediately	Wake reception; end threads and close sockets

Application programming interface (API)
[installation directory]/RflySimAPIs/9.RflySimComm/API.pdf

Exercises and Discussion

Trace Each Delivery Failure to Its Underlying Mechanism

1. MAVLink: after a CRC passes, can we conclude that a control command executed? What evidence is still missing?

2. DDS: both ends share a Domain but no samples arrive. Which configurations and states should be checked next?

3. OLSR: C is not an MPR selected by A. Can C still send and forward application packets?

4. ETX: the two-hop cost 2.469 is below the direct cost 3.125. Does this prove lower latency?

Key Answers

Frame Validity, Matching, Reachability, and Execution Need Separate Evidence

MAVLink: check source and target, message semantics, the command's acknowledgment mechanism, and actual response. A valid CRC does not prove execution.

DDS: network reachability → Domain → Topic / type / QoS → matching → valid sample → sampling time versus use time.

OLSR: C remains a network node. MPR governs the relevant control flooding; application unicast uses each node's routing table.

ETX: this example selects two hops. Comparing latency also requires per-hop rate, retransmission time, contention, and queue statistics.

Section Summary

Mechanisms Explain Results; Experiments Test Assumptions

Message encoding: identify source ID, message ID, payload length, and checksum. Serialization alone does not ensure reliable delivery.

Data distribution: domain, topic, type, and QoS jointly determine matching. Complete delivery does not guarantee sufficient freshness.

Multi-hop routing: HELLO → MPR → TC → routing table. Distinguish basic hop-count routing from the ETX extension.

Next experiments: fix the communication task; record transmissions, arrivals, data age, and motion response to explain performance differences.