



Technical Standards for a Multi-Agency Drone Detection Common Operating Picture (COP)

1. Introduction

A **drone detection Common Operating Picture (COP)** is a shared system where multiple agencies and sensor systems contribute to one unified view of the airspace. This document defines the technical standards and protocols for a national, multi-agency COP that can ingest and normalize data from many different drone/sUAS detection sensors (radar, RF, cameras, etc.), regardless of vendor. The goal is to ensure **interoperability** (everything works together), **security**, and **real-time performance** in a system that may involve numerous stakeholders.

1.1 Purpose and Scope

The purpose of these standards is to guide the design and implementation of a **multi-vendor, multi-sensor drone detection network**. By adhering to open standards, agencies can avoid vendor lock-in and ensure new sensor types can be added via “plug-in” adapters rather than a complete redesign of the system. This document covers network communication, time synchronization, sensor data formats, security, data fusion, and system management standards required for the COP. It also highlights *why* certain standards or protocols are chosen (including alternatives where relevant), so that even readers with basic IT knowledge (e.g. familiarity with IP networks, APIs, JSON) can understand the rationale.

Core Objectives of the COP system:

- **Unified Situational Awareness:** All drone detection feeds (radar tracks, video feeds, RF detections, etc.) are combined into a single coherent picture in real-time. An operator shouldn't need to check separate systems for each sensor type.
- **Modular Integration:** The system uses an **adapter model** – each sensor type either natively speaks an open standard or is connected via a software gateway that translates its proprietary output into the standardized format. This ensures new sensors can be integrated by writing a new adapter, without changing the core system.
- **Multi-Agency Data Sharing:** The COP supports role-based or attribute-based views so that different agencies (e.g. military, law enforcement, aviation authorities) can access the data they need while sensitive information is protected. The design follows a “need-to-know” principle using data labeling and access control.
- **Real-Time Performance:** Detections and alerts must propagate through the system with minimal latency. The network and processing pipeline are designed for sub-second updates, using real-time protocols for critical data.
- **Auditability and Accountability:** Every action (user login, sensor data injection, track update, etc.) is logged. The system maintains data provenance (origin of each track or detection) and an audit trail for later analysis or legal evidence.
- **Future-Proof and Open:** The architecture follows **Modular Open Systems Approach (MOSA)** principles and the emerging **Sensor Open Systems Architecture (SOSA)** guidelines. This means preferring open, widely adopted standards over proprietary ones, ensuring backward compatibility where possible, and designing for future expansion (e.g. new sensor technologies or evolving drone threats).

1.2 Architecture Approach: Open Standards and Adapter Pattern

Open Standards First: Wherever possible, the system uses open, published standards (e.g. IEEE, IETF, ASTM, NATO STANAG, etc.) for interfaces and data formats. Open standards ensure that multiple vendors can implement them, fostering competition and innovation. They also allow different subsystems to interoperate without custom integration work.

“Core vs Adapter” Model: The COP’s **core** (central data bus and fusion engine) operates only on these standard formats – it does **not** directly implement every vendor’s proprietary protocol. If a sensor cannot output an approved standard format, it must connect via an **edge adapter** or gateway. The adapter is a software module or small server that **converts** the vendor’s proprietary data into the COP’s canonical format (and vice versa if needed for control messages). In essence, the adapter acts as a translator between the sensor and the open standard data bus.

- *Example:* Suppose a radar vendor has a custom binary protocol for its tracks. Instead of modifying the COP to understand it, we deploy an adapter (possibly provided by the vendor or a third-party) that reads the radar’s output and repackages the data into the chosen standard (like ASTERIX format for radar tracks). The COP core then only deals with ASTERIX, unaware of the proprietary format behind the scenes.

Benefits: This approach prevents vendor lock-in – you can replace or add sensors by just adding/adapting the relevant adapter. It also isolates any proprietary quirks to the edges of the system, keeping the central system clean and focused on the unified picture. All internal data on the COP message bus will be in standardized, documented formats.

1.3 Summary of Key Points (Introduction)

- The COP system integrates **multiple sensor types and agencies** into one unified situational awareness display.
- **Open standards** are used for all data exchange to maximize interoperability and future-proofing.
- An **adapter architecture** allows proprietary sensors to join the system by translating their data to the standard formats, rather than changing the core system for each vendor.
- The design emphasizes real-time data flow, strong security (zero-trust principles), and comprehensive logging for auditing.
- By following these guidelines, agencies can share drone detection data seamlessly while retaining control over sensitive information.

2. Network Infrastructure and Data Transport

This section describes how the sensors and the COP core communicate over networks. It covers IP networking choices, how we handle high-volume data like video or radar feeds, and how we ensure timely and reliable delivery (even over WAN links). The network must support real-time streaming of sensor data to possibly many users, without being overwhelmed or introducing dangerous delays.

2.1 IP Network Foundation (IPv4/IPv6 and Segmentation)

Standard: Use a dual-stack IPv4/IPv6 network with proper VLAN/IP segmentation for security.

Rationale: The system should support both IPv4 and IPv6 from the start. IPv4 is still prevalent (and many existing systems use it), but IPv6 provides a vastly larger address space which will be important as

the number of sensors grows. Running dual-stack ensures compatibility during the long transition period to IPv6. It is likely that future devices and mobile networks (5G/6G) will prefer IPv6, so we include it now rather than retrofit later. Moreover, many government networks mandate IPv6 compatibility.

Network Segmentation: All sensors and users should not sit on one flat network. Instead, use VLANs or separate subnets to isolate different categories of traffic. For example, one segment for radar sensors, one for cameras, one for RF sensors, and perhaps separate segments for each agency's user access. This limits the scope of broadcast traffic and contains any potential compromise (a breach in one sensor's subnet shouldn't automatically give access to others).

- By default, **routing between segments is disabled**. Only through explicitly configured firewall rules or routers are specific data flows allowed (principle of least privilege). For instance, radar sensor VLAN can send data to the fusion server, but not directly to the camera VLAN, etc.
- Segmentation also allows applying different security policies. One segment might carry very sensitive data (military sensors) and thus have stricter monitoring or encryption rules.

Implementation Notes:

- All new network hardware (switches, routers) purchased must support IPv6 alongside IPv4. The addressing plan should allocate IPv6 subnets for each VLAN.
- Ensure devices have IP address configurations for both families. Test that all protocols (multicast, PTP, etc.) work on IPv6 as well as IPv4.
- Use private IPv4 ranges and unique local IPv6 addresses for internal sensor nets; NAT or interconnection to broader networks can be handled at the boundary if needed.
- Document the network architecture: which VLANs exist, what IP ranges and what purpose (e.g. "VLAN10: Radar sensors, 192.168.10.0/24 and fd00:10::/64"). This helps coordination between agencies.
- Deploy firewall controls at the intersections of these segments. Only allow necessary ports/protocols (e.g. allow PTP timing messages, allow the specific telemetry streams, but block unrelated traffic).

2.2 Multicast Distribution for Sensor Data

Many sensor data types (video feeds, radar target reports) will be distributed from one source to many consumers (e.g. one camera feed to many operator stations). IP **multicast** is ideal for this one-to-many delivery. We adopt **Source-Specific Multicast (SSM)** as the multicast model.

Standard: Source-Specific Multicast (SSM) using PIM-SSM routing with IGMPv3 (for IPv4) and MLDv2 (for IPv6). SSM restricts multicast traffic to a specific source.

Why SSM (and not the older Any-Source Multicast)? In traditional *Any-Source Multicast* (ASM), anyone can send to a multicast group address, and the network delivers all senders to the group. This is flexible but has a big drawback: it's not secure, as any rogue device could inject packets into the group. Also, the network must figure out all sources sending to the group, which adds complexity. With **SSM**, a receiver joins not just an address (group) but a **specific source**. Only packets from that source are forwarded. This makes the multicast delivery **much more secure and controlled** – you only get what you explicitly asked for. It also reduces network load and routing state, since the network doesn't have to maintain *(G) trees for all sources, only (S,G) for each source*. In short, SSM "improves security by limiting the source"** and ensures a rogue sensor can't impersonate another.

Implementation:

- **IPv4 SSM range:** Use 232.0.0.0/8 for SSM group addresses (this range is reserved for SSM). For example, a particular radar video stream might be on 232.5.5.5 with source 10.1.2.3 (only that radar's IP can send to that group).
- **IPv6 SSM range:** Use FF3x::/32. In practice, IPv6 defines FF3x::/96 for SSM but recommends treating any FF3x::/32 as SSM-capable. We will allocate specific FF3e:... addresses for our streams.
- Switches/routers must support IGMPv3 and MLDv2. This is crucial: IGMPv3/MLDv2 allow receivers to signal the specific source they want. Older IGMP versions can't do that. All receiving hosts (like operator consoles or fusion servers) need to run IGMPv3/MLDv2 as well.
- **PIM-SSM:** The routing protocol that will carry multicast between IP subnets is Protocol Independent Multicast in SSM mode. Configure PIM-SSM on routers with appropriate RP (rendezvous point) configuration if needed (in pure SSM, RPs are not used like in ASM).
- **IGMP Snooping:** For local LANs, enable IGMP snooping on switches to prevent flooding multicast traffic to ports that don't need it. Snooping listens to IGMP join messages and only forwards multicast to interested ports, saving bandwidth.

Alternative (Unicast Distribution): In some deployments, multicast might be disallowed or not feasible (some cloud networks, or if traversing the public internet). In that case, an alternative is to use a **brokered unicast** distribution – for example, each sensor sends to a message broker (like an MQTT or AMQP server), and consumers subscribe to topics. The broker then replicates data to each consumer via unicast. While this works, it **doesn't scale as well** (the broker can become a bottleneck, and you send duplicate streams to each client, increasing bandwidth usage). It also introduces a bit more latency (store-and-forward). Thus, unicast broker is only a backup plan if multicast truly cannot be used.

2.3 Quality of Service and Time-Sensitive Networking (TSN)

Not all data on the network is equal. A critical alarm (like "Drone detected 1 km away!") is far more urgent than, say, a routine telemetry update or a video frame that comes 100ms later. We implement **Quality of Service (QoS)** to prioritize urgent and real-time traffic. Specifically, we use **DiffServ (DSCP markings)** on IP packets, combined with **Time-Sensitive Networking (TSN)** features at layer 2 for ultra-critical traffic scheduling.

Standard: Differentiated Services Code Point (DSCP) markings, with network devices mapping these to priority queues. Optionally, **IEEE 802.1Qbv Time-Aware Shaper** in switches for strict time scheduling of critical flows (if using TSN-capable equipment).

We define a DSCP **mapping table** for various traffic types in the COP system:

Traffic Type	DSCP Value & Name	Behavior	Notes
Sensor control & alerts (e.g. DDS/RTPS control messages coordinating sensors, immediate threat alerts)	46 (EF – Expedited Forwarding)	Highest priority, low latency	Goal < 50 ms latency end-to-end. This class should preempt others.
Primary video streams (Electro-optical/Infrared feeds)	34 (AF41 – Assured Forwarding)	High priority, guaranteed bandwidth	Important but a few dropped frames are tolerable vs delay.

Traffic Type	DSCP Value & Name	Behavior	Notes
Radar track reports / plots	34 (AF41) or 26 (AF31)	High priority (similar to video)	Use AF41 for high-criticality radar feeds, AF31 for less critical.
Audio/Acoustic sensor streams	44 (VA – Voice Admit)	Low-jitter, low-latency	Similar to voice traffic – needs consistent timing for triangulation of sound.
Telemetry & heartbeat messages (non-bursty status updates, command acknowledgments)	45 (NQB – Non Queue Building)	Low latency, minimal bandwidth	NQB is a newer RFC class for traffic that doesn't flood queues; ensures quick delivery of low-rate messages.
Network management (routing protocols, PTP sync, etc.)	48 (CS6 – Class Selector 6)	Reserved for network control	Only network control devices use this (e.g. BGP, PTP timing messages), to ensure the network stability.

Explanation: We use Expedited Forwarding (EF) for the absolutely mission-critical coordination messages – these packets should essentially always jump to the front of the queue (within reason, to avoid starving others). AF (Assured Forwarding) classes provide a way to have high priority but still fair queued service; AF4x is high, AF3x slightly lower. We put video and radar in AF4x or AF3x because while important, we can tolerate some packet loss (e.g. one missed video frame) but not huge delays. Audio gets a special voice-like treatment because timing consistency affects its usefulness (spatial audio processing needs evenly spaced packets). Telemetry (like periodic health pings or logs) gets NQB – this is a newer concept that isolates “small bursts” so they don't get stuck behind big flows; NQB traffic shouldn't build large queues. Management traffic (like the switches exchanging PTP or routing info) is given a high class (CS6) but since it's only used by the infrastructure, it remains a small portion.

Switch/Router QoS Configuration: All network devices must be configured to recognize these DSCP values and put them into appropriate hardware queues with defined priorities or bandwidth guarantees. For instance, EF might go into a strict priority queue (always serviced first), AF classes into weighted fair queues, etc. The exact queuing mechanism can vary by device (some might use 8 hardware queues aligned with IEEE 802.1p user priorities). The main point is consistency – ensure every device from sensor to operator station honors the DSCP and treats EF as highest priority, etc.

Time-Sensitive Networking (802.1Qbv): In especially critical links (maybe between sensor hubs and fusion center), if we have TSN-capable switches, we can use **time-aware shaping**. 802.1Qbv lets us schedule certain queues to transmit only in specific time windows (a cycle). For example, we could dedicate a 1 ms window every 5 ms where only EF traffic is sent and all other queues are gated off – this guarantees bounded latency for that class because it won't ever wait more than the cycle time. TSN features are optional but recommended for segments that carry a lot of real-time data (like high-speed radar video combined with other traffic). Essentially TSN can provide deterministic latency and no jitter by carefully orchestrating transmissions.

Why QoS is critical: Imagine a 4K camera feed flooding the network – if we had no QoS, an urgent radar alert packet could be delayed or dropped behind hundreds of video packets. By marking and

prioritizing, the alert zooms through while the video packets might be buffered slightly. This ensures that a burst of less important data (like a big video frame or a bulk file transfer) cannot drown out the critical messages. In a multi-agency scenario, there might be many consumers; QoS helps maintain order and performance.

Implementation Requirements:

- **End devices mark packets:** The sensor adapters and other sources of traffic should set the DSCP field on their outgoing IP packets according to the table. (Most OS/network stacks allow setting DSCP via socket options or in application.)
- **Network honors DSCP:** Turn on DiffServ QoS on all switches/routers. Define mapping from DSCP to egress queue and bandwidth share. Make sure trust boundaries are considered (if a device could be compromised and mark everything EF, do we trust all devices? Possibly on sensor VLANs we trust markings; on user VLAN maybe rewrite or police).
- **Policing:** It can be wise to police traffic entering at a high DSCP – e.g. ensure no device sends an excessive rate of EF traffic that could starve others. For instance, limit EF to say 5% of link unless explicitly allowed, since EF should only be small control messages.
- **TSN hardware:** If using TSN, configure gate schedules. Ensure all participating devices have a common time base (which ties into PTP timing, discussed later). One could, for example, schedule a 125 µs window every 1 ms for high-priority sensor data, etc. This is an advanced step and requires careful planning.

2.4 Wide-Area Network (WAN) Streaming and Backhaul Resilience

In many cases, the sensors and the COP users might not be co-located; data might have to travel over a WAN or internet (for example, a remote border radar sending data to a central command center). WAN links can be unreliable or have packet loss (especially wireless/4G/5G links). We need a transport protocol for streaming video and other data reliably over such links.

Standard: Use a reliable UDP-based streaming protocol such as **RIST (Reliable Internet Stream Transport, VSF TR-06)** or **SRT (Secure Reliable Transport)** for video and other critical streams over WAN. Both provide similar mechanisms to recover lost packets.

Rationale: Raw UDP is lightweight and real-time, but if a packet is lost in transit, it's gone (no recovery). Traditional TCP would recover lost data but often at the cost of large delays (it will stop and retransmit, causing video to stutter or lag significantly). RIST and SRT are designed for live video streaming: they primarily send UDP packets but have an **ARQ (Automatic Repeat reQuest)** mechanism at the application layer – meaning if a packet is lost, the receiver notices and asks for a resend, and the protocol quickly sends that missing packet again. This way, minor packet losses can be corrected without impacting the whole stream. They also support features like forward error correction (sending redundant data to preemptively cover losses) and packet bundling.

RIST vs SRT: Both are good and are interoperable only with themselves (i.e., both sides must use the same protocol). RIST is an open specification from the Video Services Forum; SRT was originally by Haivision and open-sourced. For a multi-point distribution (one sender, many receivers) such as broadcasting a video feed, **RIST “Main Profile”** has built-in support for multipoint, and is often favored in broadcast industry. **SRT** is widely adopted for point-to-point contribution links. Either can be used; we can choose based on what vendors support: - If the use case involves **many receivers** (like sending one sensor feed to 10 agencies), RIST may be slightly preferable. - If it's mostly **one-to-one** links or existing systems already support SRT, then SRT is fine.

Both protocols include built-in encryption (typically using DTLS or AES) to secure the streams, which is important if going over the public internet.

Bonding and Redundancy: The system should support using multiple WAN links in parallel (e.g., two cellular modems, or a fiber plus a 4G backup). RIST and SRT can both work with external bonding solutions or some vendor implementations support bonding multiple sockets. The idea is to send duplicate or split traffic across two paths so that if one path drops a packet, the other might still deliver it. Another approach is hitless failover: send two identical streams over different networks (this can be done at a higher level or using SMPTE 2022-7 seamless protection switching). **IEEE 802.1CB Frame Replication and Elimination (FRER)**, discussed next, can also apply over a WAN if both entry and exit points support it (duplicates frames over multiple paths and discards the extras).

Performance Goal: The switching between a primary and backup path (if one goes down) should be **under 500 ms** so that operators may not even notice more than perhaps a momentary hiccup. RIST/SRT can buffer and adjust to network conditions to some extent to hide brief outages.

2.5 Redundant Path Networking (Zero Packet Loss Concept)

For the most critical data flows, especially within a local or metro area network, we employ **redundant simultaneous paths** to achieve **near-zero packet loss** even if a link fails. This is where **IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER)** comes in.

Standard: IEEE 802.1CB (FRER) – the sender (or an edge device) replicates each packet and sends it via two (or more) independent network paths to the destination. The destination (or an intermediate merge point) receives duplicates and discards the extra copy, keeping just one.

Rationale: In a mission-critical environment, we cannot afford to lose data due to a single link or switch failure. For example, if a radar track update is lost just as a drone is approaching a protected zone, that could be a serious issue. FRER provides *hitless failover* – if one path drops a packet, the other path still delivers it, so the end system sees **zero packet loss**. This is like having two couriers deliver the same message via different routes; as long as at least one gets through, you're covered. It's more efficient than application-layer retries for small, constant streams because it doesn't wait for a timeout; the data is already delivered on the alternate path in near-real-time.

Implementation:

- Typically, you'll have two separate network paths (could be two switches and cable infrastructures, possibly two network interface cards on a sensor, etc.). They need to be disjoint enough that one failure doesn't take out both.
- The source (could be the sensor's adapter or the first switch) appends a sequence number to each frame and sends it out both paths.
- The destination (or a merge node near it) sees frames from both paths. It uses the sequence numbers to identify duplicates: it will accept the first arrival of a given sequence and drop the second copy. If one path is slightly slower, it still will drop the duplicate when it arrives because it already got that sequence via the faster path.
- Network equipment or end nodes must support 802.1CB for this to work. If end nodes don't, an alternative is using two parallel streams at the application and merging at the app level, but the standard provides a uniform way at the Ethernet layer.

Note: This doubles the bandwidth usage for those streams (since every packet is sent twice), so we usually reserve this for the absolutely essential traffic or when network conditions are bad. However, in

LAN environments 802.1CB can ensure not only protection against failures but also against random packet drops due to congestion – since if one path experiences congestion and drops packets, the other path might still deliver them. The result is highly reliable delivery.

Summary of Network Standards:

- We use **IPv4/IPv6 dual stack**, segmented networks to ensure scalability and security.
- **Multicast via SSM** for efficient one-to-many data distribution, which prevents unauthorized data injection and reduces network load.
- **QoS (DSCP and TSN)** to prioritize critical sensor data and guarantee timely arrival (using EF for urgent alerts, etc.).
- **Reliable streaming protocols (RIST/SRT)** to handle lossy WAN links, with ARQ to recover lost packets and encryption for security.
- **Redundant paths (802.1CB FRER)** to eliminate single points of failure, delivering zero-loss performance for key data by duplicating packets over independent routes.

2.x Summary of Key Points (Network & Transport)

- **IPv4/IPv6 Dual Stack:** Ensures current and future compatibility; use VLANs to isolate sensor networks for security.
- **Source-Specific Multicast (SSM):** Only deliver multicast data from known sources – improving security and efficiency for distributing sensor feeds.
- **QoS Differentiation:** Mark and prioritize traffic (alerts vs video vs telemetry) so that urgent messages always get through quickly, even if bandwidth is constrained.
- **Time-Sensitive Networking:** Where possible, use time-aware traffic shaping for deterministic delivery of critical flows.
- **Reliable Streaming over WAN:** Use protocols like RIST or SRT over unreliable networks to recover lost packets on the fly, instead of using TCP which adds latency.
- **Redundant Paths (FRER):** For critical links, send duplicate packets via multiple paths to achieve no packet loss even if a link fails – improving reliability dramatically.

3. Timing and Synchronization

Time synchronization is the heartbeat of a multi-sensor system. If you have a radar detecting something at time X and a camera capturing video at time Y, you need to know how X and Y relate – are they the same moment? A few hundred milliseconds off? Accurate timestamps allow us to **fuse sensor data correctly**, correlate tracks, and even do things like triangulate a signal from multiple sensors. We therefore require a high-precision time sync across the network, much more accurate than everyday NTP.

3.1 Precision Time Protocol (PTP) – Primary Time Sync Standard

Standard: IEEE 1588-2019 Precision Time Protocol (PTPv2.1), with support for hardware timestamping and a hierarchical master-slave clock architecture (grandmaster, boundary clocks, etc.).

Rationale: PTP can synchronize clocks in a network to within microseconds or even nanoseconds under the right conditions [1](#) [2](#). By contrast, standard NTP (Network Time Protocol) typically gets only to a few milliseconds accuracy on a LAN. Why do we need such precision? Consider sensor fusion: if a drone is flying fast, even a 50 ms timestamp error could mean tens of meters of difference in perceived position. With microsecond sync, that error is negligible. For applications like correlating a radar blip with an RF signal hit, you want timestamps as close as possible so you know those events are the same

target. Also, if multiple cameras or an array of acoustic sensors are used, tight sync allows coherent processing (like beamforming or 3D localization).

PTP was designed for exactly these kinds of systems (industrial control, measurement, telecommunications) where time sync is crucial. It works by sending timing packets over the network and adjusting clocks based on measured delays. Unlike GPS clock sync (which could also give good time), PTP doesn't require every node to have a satellite receiver, and it works indoors. We prefer PTP over GPS for local synchronization to avoid dependency on GPS signals (which might be jammed during a hostile drone incident).

Implementation Approach:

- **Grandmaster Clocks:** We will have one or two **grandmaster** time sources. Typically, a grandmaster will get time from GPS or a national time standard, and then distribute it via PTP. Redundancy is key: if one fails, another can take over (PTP's Best Master Clock Algorithm (BMCA) handles this election automatically).
- **Boundary Clocks:** All network switches connecting the sensors should ideally be PTP **boundary clocks** or at least transparent clocks. A boundary clock participates in PTP: it acts like a slave to the grandmaster on one side and a master to devices on its other ports. This means each network segment gets a fresh timing relay, reducing the error accumulation. A transparent clock, on the other hand, doesn't act as master/slave but just measures its own delay on the timing packets and writes that info into them (so recipients can compensate).
- **Accuracy Goal:** End-to-end time error of ≤ 1 microsecond across the system (for local campus or metro networks). This is achievable with hardware timestamping and proper configuration. Each sensor node's clock should not deviate more than a microsecond from the grandmaster.
- **Profile:** For local (LAN) segments, we can use the default PTP profile or possibly the Power or Enterprise profile. For wide-area (WAN) distribution of time (if the system has to sync across large distances or through routers), we might use the **ITU-T Telecom Profile G.8275.1** which is designed for moving time over telecom networks with full timing support.
- **Hardware Requirements:** All critical sensors and switches **must have hardware PTP support** (e.g. NICs that can timestamp PTP packets in hardware, switches with built-in PTP functions). This dramatically improves accuracy by eliminating software jitter. Grandmasters should use high-quality oscillators (OCXO or Rubidium) to hold time if GPS is lost (holdover).

In short, PTP will give the system a shared sense of time as if all devices had their clocks wired together.

3.2 Securing the Time Synchronization (PTP Security)

Time sync is so critical that if an attacker could manipulate it, they could wreak havoc (imagine falsifying timestamps to confuse the system, or making sensors look out-of-sync). Therefore, we must secure PTP. Historically, PTP had little security, but the latest standard **IEEE 1588d-2022/2023** introduced a comprehensive security mechanism.

Standard: Implement PTP Security as per IEEE 1588d-2022 "four-prong" approach, which includes:

- **Prong A: Authentication TLV** – Each PTP message can carry an Authentication TLV (Type-Length-Value) that holds a cryptographic signature or Message Authentication Code (MAC) to ensure the message came from a legitimate source and wasn't tampered with. Essentially, only someone with the correct keys can sign the timing packets, preventing a malicious device from introducing a false grandmaster or bogus timing info.

- **Prong B: Transport Security** – Run PTP inside a secure transport like MACsec or TLS. In our case, since we are encrypting links at layer 2 (see Security section), PTP messages on those links are protected by MACsec encryption anyway. Alternatively, one could encapsulate PTP in DTLS or use other secure channels. The idea is to prevent eavesdropping or injection at the transport layer.
- **Prong C: Architecture Hardening** – This means design the network such that it's hard for an attacker to get in the middle. For example, use **path delay** measurements to detect anomalies (transparent clocks can help ensure no unexpected delay jumps), and configure boundary clocks to **only accept timing from approved upstream ports** (so if someone plugs a "rogue grandmaster" into some access port, the boundary clock will ignore it because that port is not authorized for master). Essentially, lock down who can be a GM and who can't through config.
- **Prong D: Monitoring and Anomaly Detection** – Continuously watch the performance of PTP. If normally all sensors have offset <1 µs and suddenly one sensor's clock is 50 µs off, raise an alert – maybe someone is messing with it or its GPS reference failed. Monitor metrics like offset from master, delay, BMCA events (e.g. if the grandmaster changes unexpectedly, log it). There are tools and YANG models for PTP to help with monitoring.

In practice, implementing all this means enabling the authentication extension on devices that support it, distributing keys or certificates for PTP message signing, and ensuring the network's MACsec is turned on (which we plan to do anyway). Some current devices may not support 1588d yet, in which case we rely on network-layer encryption and tight ACLs (e.g. only allow PTP from known MAC/IP addresses).

Note: NTS4PTP (*Network Time Security for PTP*) is being discussed (similar to secure NTP's NTS), but it's not standardized yet, so we won't use experimental schemes like that in production.

3.3 High-Accuracy Timing Option: gPTP (802.1AS) for TSN Domains

For most purposes, standard PTP as above suffices (accuracy in microseconds). However, in some specialized cases (like if we have a cluster of sensors needing sub-microsecond sync for sensor fusion, or if using Time-Sensitive Networking features that demand tight sync), we might employ **IEEE 802.1AS (gPTP)**.

Standard (Optional): IEEE 802.1AS-2020 (Generalized Precision Time Protocol) – essentially PTP tuned for audio/video and TSN networks. It can achieve synchronization in the order of tens of nanoseconds in a LAN.

When to use gPTP: If we have a TSN segment – for example, an array of radars or acoustic sensors that need to capture measurements at exactly the same time or if we are doing something like "coordinated multi-sensor transmissions". gPTP is built into the Ethernet TSN standards and is often the default in AVB/TSN-capable switches.

One scenario: Suppose we have multiple short-range radars around an airport and we want to aggregate their raw video in a time-aligned way to do composite tracking. gPTP can ensure each radar's frame start is aligned within nanoseconds. Another example is aligning video camera frames with an RF sensor sampling clock for advanced detection algorithms.

Differences: 802.1AS is essentially a profile of PTP: it runs at Layer 2 (Ethernet directly, not IP), doesn't allow arbitrary topologies (every device must participate, no "non-participant" nodes), and doesn't allow NTP-like modes. It's very strict to reduce variability.

We include this mainly to note that if any part of the network is a **TSN island**, then within that island, 802.1AS should be used (devices like TSN switches and controllers will handle it). At the boundary to the rest of the system, a time gateway can translate between 802.1AS and standard PTP.

3.4 Summary of Key Points (Timing & Sync)

- **Precision Time Protocol (PTP):** All devices sync clocks to within ~1 microsecond using PTP (IEEE 1588). This high precision is critical for correlating multi-sensor data (compared to only millisecond accuracy of NTP).
- **PTP Implementation:** Deploy redundant **grandmaster** clocks (GPS disciplined), use boundary/transparent clocks in network gear for best accuracy. Hardware timestamping is essential for microsecond precision.
- **Security of Timing:** Protect the timing system with authentication on PTP messages and encrypted links. Only authorized devices can act as time sources. Continuous monitoring will detect any time anomalies (to catch spoofing or failure).
- **Optional gPTP for TSN:** In segments requiring extreme sync (nanosecond range), use IEEE 802.1AS (the AVB/TSN profile of PTP) to tightly coordinate sensor nodes.
- **Bottom line:** A secure, highly accurate common clock across the network allows the COP to trust the timestamps from all sensors, which is foundational for reliable sensor fusion.

4. Sensor Interface Standards and Data Formats

Now we dive into the data that the sensors produce and how it's formatted for the COP. Each sensor type (radar, radio frequency, cameras, etc.) has unique data, but we need a common language for each so that the COP can ingest it. This section specifies the **canonical data formats** for sensor outputs. If a sensor can't natively speak these, an adapter must convert it. We also explain *why* each standard was chosen – in general, we pick formats that are widely adopted in industry or military use, to ensure multi-vendor support.

(Throughout this section, we'll mention some example vendors or systems that use these standards – this is just to illustrate feasibility, not to endorse any particular product.)

4.1 Radar Sensor Interfaces

Radars are a primary sensor for drone detection. They produce two main kinds of data: **tracks** (or plots) and **radar video**. Tracks are processed outputs (location and trajectory of a detected object), whereas radar video is the raw or semi-raw radar returns (often used for display or further processing). We have standards for each:

4.1.1 Radar Track Data – ASTERIX Category 062

Standard: **EUROCONTROL ASTERIX CAT-062** (latest edition, e.g. 1.21 from 2025) for system track messages.

What it is: ASTERIX stands for "All Purpose Structured Eurocontrol Radar Information Exchange". It's a family of standards (categories) for different surveillance data. Category 062 is specifically defined for **system tracks** – i.e., processed target information such as you'd see on an air traffic control screen (track number, position, speed, etc. of an aircraft or drone). ASTERIX is a binary message format that is highly efficient and standardized across many radar systems.

Rationale: ASTERIX is **widely used internationally** for radar data exchange, both in civil and military contexts. Many radar vendors natively support outputting ASTERIX because it's a de facto requirement to interface with larger systems. By choosing ASTERIX Cat-062, we ensure that adding a new radar likely just means enabling its ASTERIX output or writing a fairly straightforward converter. The format includes all necessary fields for drone tracking (and more). It can carry 2D or 3D coordinates, track quality, identification friend-or-foe info, etc. It's also designed to be compact so that even hundreds of tracks per second can be sent without overwhelming the network.

Key Data Fields in Cat-062: A track message typically includes: - Data Source Identifier (which radar or system produced it – using SAC/SIC codes), - Track Number (unique ID for the track), - Time of Last Update (precise timestamp, which we ensure is PTP-synced), - Track Position (in lat/long or x/y, plus altitude if 3D), - Track Velocity (speed and heading), - Track Status (like if it's tentative, confirmed, what type of target it might be if known, etc.), - Possibly target classification (some systems can mark a track as "drone", "bird", "unknown" if the radar can tell).

These fields map to ASTERIX data items defined in the standard. For example, ASTERIX has items like I062/080 (track status), I062/105 (track position in WGS84), etc.

Requirements/Guidelines: - The adapter or radar should avoid sending messages that are bigger than the network MTU. ASTERIX is usually fine because a single track record is quite small (tens of bytes). However, ASTERIX messages can be concatenated (multiple records in one packet). We should ensure the packing doesn't exceed UDP packet size that fits in Ethernet (~1500 bytes). If needed, limit how many records per packet. - Use UTC timestamps that are derived from the PTP synchronized clock so that all tracks from different radars are time-consistent. - If possible, include track quality metrics (like a covariance matrix or uncertainty if the radar provides it). That helps the fusion system later (some ASTERIX versions allow optional fields for accuracy). - We will designate unique SAC/SIC codes for each radar in the network (these are like station addresses in ASTERIX) so that the source of each track can be identified within the message.

Alternative / Supplement: STANAG 4676 (NATO standard for track data exchange) is an alternative especially if our system is to interoperate with NATO systems. STANAG 4676 is an extensive track format (often in XML or binary STANAG format) that includes detailed info like track histories, covariance, data source, etc. It is actually more complex than ASTERIX. ASTERIX is a bit simpler and widely implemented. We may support exporting fused tracks in STANAG 4676 as well (discussed later in fusion section). But for ingesting radar sensor tracks, ASTERIX Cat-062 is usually the path of least resistance given how many radars support it.

4.1.2 Radar "Video" (Sensor Raw Plot Data) – ASTERIX Category 240

Standard: EUROCONTROL ASTERIX CAT-240 for radar plot/video data.

What it is: Cat-240 is defined for exchanging radar video (the stream of range-doppler returns or plots prior to tracking). In air traffic control, this is used to send the actual radar video to a display or to a processing unit. It can include a sequence of bins (range cells) for each antenna rotation, etc. Essentially, it's the closest standardized form to raw radar returns.

Rationale: Why include radar video? In a drone defense system, sometimes the central C2 might want to display the radar hits on a map (like raw hits, especially if tracks are not yet confirmed) or use the raw data for custom processing (like detecting a very low slow drone that the radar's own tracker might have filtered out). ASTERIX Cat-240 allows multi-vendor radars to output their video in a common format. Many modern radars have this ability, as it's used in advanced ATC and surveillance systems. If a

vendor doesn't support Cat-240, an adapter might not be trivial (because raw video formats can vary), but at least Cat-240 gives a target format to convert to.

Key Points for Implementation: - **Bandwidth:** Radar video can be high-bandwidth, especially high resolution radars. By sending via multicast, we ensure it only goes where needed (e.g. to an analysis console or recording device). We need to consider network capacity (might be tens of Mbps per radar). - Use UDP multicast for Cat-240 streams (since it's essentially streaming data tied to radar rotation or timeline). Ensure the MTU is sufficient; if not, radars often allow adjusting how much data per packet to avoid fragmentation. - PTP synchronization is important if combining data from multiple radars or correlating with other sensors. Each video frame or plot batch should have a timestamp (which Cat-240 supports) aligned to PTP time. - Not all deployments will use radar video; if the focus is on track-level fusion, we might disable this. But we include the standard to keep the option open.

Alternative: If a radar uses the UK **SAPIENT** standard (which is more of a whole sensor API, not just data format), it may output plots/tracks differently. SAPIENT has message types for detections and tracks, which could supplement or replace ASTERIX in some contexts. However, to keep things unified, we'd likely translate SAPIENT messages from a radar into ASTERIX Cat-240 (for plots) and Cat-062 (for tracks) at the adapter level, unless we decide to natively support SAPIENT in the COP (which is another option if multiple sensors are SAPIENT). For now, ASTERIX is the baseline.

4.2 RF (Radio Frequency) Sensors and SDRs

RF sensors detect the radio signals emitted by drones (like control links, video downlinks, telemetry). These can be spectrum scanners, direction finders, or communication intercept devices. Many such sensors are built on Software-Defined Radios (SDRs). We need a standard way to transport their data (which can include raw IQ samples, spectrograms, or detected signal metadata).

4.2.1 Live RF Signal Data – VITA 49 (VRT)

Standard: ANSI/VITA 49.2 (VITA Radio Transport or VRT) for streaming RF data and metadata.

What it is: VITA 49 is a standard packet format for digitized RF signals. It allows streaming of IF/RF samples along with context metadata in a well-defined way. A VITA 49 stream consists of **Signal Data Packets** (carrying sample payloads) and **Context Packets** (carrying metadata about those samples, such as frequency, sample rate, timestamp, geolocation of the sensor, etc.).

Rationale: The RF detection world has coalesced around VITA 49 for sensor interoperability – it's used in many military SIGINT systems and by SDR manufacturers. If an RF sensor (like a spectrum analyzer or drone-detector RF unit) outputs VITA 49, the COP could potentially directly ingest it or at least an adapter can parse it. The key advantage is that it's **timely and structured**. Each burst of signal data can be timestamped (with PTP time) so we know exactly when that signal was received. The context allows sensors to report what frequency they're tuned to, what gain, etc., in real time. This is crucial for a multi-sensor system to make sense of RF hits (for example, to geolocate a drone's controller, you might have two RF sensors sending bearing and signal strength – the context tells you the frequencies and times so you can correlate them).

Implementation Considerations: - **Timestamping:** Every RF data packet must include a precise timestamp, derived from PTP. For example, if an SDR captures IQ samples, it might timestamp the start of the frame of samples. This allows fusion of RF with radar (like comparing the time a radar saw something and the time an RF emission was received). - **Context metadata:** Use context packets to announce things like center frequency, bandwidth, sensor position (if applicable). VITA 49.2 has fields

for GPS location of the sensor, which is useful if sensors are mobile. - **Fragmentation:** VITA49 packets can be large if carrying many samples; ensure they fit network MTUs or use a protocol that fragments/reassembles at a lower layer if needed. Usually, it's good to keep them small enough to avoid IP fragmentation. - **Usage:** If an RF sensor is doing realtime analysis, it might not send continuous raw IQ (which could be huge), but maybe short bursts when something is detected. Alternatively, it might send periodic spectrums (like a waterfall update). VITA 49 can handle both streaming and bursty data.

Example: A drone RF detector might send a context packet indicating "I'm tuned at 2.4 GHz now" and then send signal packets containing snippets of the signal it caught, with timestamps. The COP or a specialized process could decode these or at least log them for evidence. If needed, multiple sensors' VITA 49 streams could be combined or compared.

4.2.2 Recorded RF Data – SigMF Format

Standard: **SigMF (Signal Metadata Format)** for storing captured RF recordings and their metadata (for offline analysis or evidence).

What it is: SigMF isn't a streaming protocol, but a recording **file format** specification. It pairs raw recorded IQ data (e.g. in a binary file) with a JSON-formatted metadata file describing the recording (what the sample rate was, when it was taken, what each field represents, etc.). It's extremely useful for sharing or archiving RF data because anyone can read the metadata and interpret the samples correctly.

Rationale: If the system records RF snippets (say, the 5 seconds of signal around a drone detection event), using SigMF ensures those recordings can be shared between agencies or analyzed later without ambiguity. It's an open standard embraced by the SDR community and avoids proprietary recording formats. The JSON metadata can include the same type of info as VITA 49 context: center frequency, sample rate, start time, and even annotations (like "this segment corresponds to drone controller signal").

Implementation: - Adapters or sensor controllers should output recordings in SigMF for any long-term storage. This might be done by converting a live VITA 49 stream that's captured into a file: take the binary IQ, save to a file, and generate a .sigmf-meta JSON with all relevant details. - Include as much info as possible in the metadata (time, freq, location of sensor, etc.) so that months later one can understand what was captured. - The COP could provide a library or tool to generate SigMF from live data when an operator requests a recording or when an alert triggers automatic recording.

In summary, **VITA 49** covers real-time transport of RF data within the system, and **SigMF** covers how we store it for later use.

4.3 Electro-Optical/Infrared (EO/IR) Full-Motion Video

Cameras (daylight EO or thermal IR) are common for visually identifying drones. They produce video streams, often with metadata about where the camera is pointing. We need to standardize both the video encoding and the accompanying metadata so that any camera feed can be used in the COP and augmented with map overlays, etc.

Standard: **STANAG 4609** (NATO Digital Motion Imagery Standard) with **MISB KLV metadata** (MISB standards ST 0601, ST 0603, ST 0903, etc.).

What it is: STANAG 4609 defines how to package video and metadata together for interoperability ³. In practice, it means: - Video is encoded in a common format (typically H.264 or H.265 codec) and wrapped in an **MPEG-2 Transport Stream**. - Metadata about the video (like sensor location, time, orientation, and possibly detected targets) is embedded as **KLV (Key-Length-Value)** packets in the stream, aligned with the frames. - The Motion Imagery Standards Board (MISB) has specific standards for what metadata to include and how: - **MISB ST 0601** "UAS Datalink Local Set" – this is the primary set of metadata fields for UAV video, including things like latitude, longitude, altitude of the platform, sensor azimuth/elevation, etc. Essentially the basics to geo-reference the video ⁴ ⁵. - **MISB ST 0603/0604** – these cover precision timestamps in the metadata (how to timestamp each frame or each metadata packet to exact microsecond or better using PTP time). - **MISB ST 0903** – defines **Video Moving Target Indicator (VMTI)** metadata, which is how you tag moving objects in the video (the pixel coordinates of a detected moving object, etc.). If a camera or an analytic software finds a moving target (like a drone or bird in the video), it can put that info into the KLV for that frame. - There are others (ST 0806 for Remote Video Terminal, etc., but the above are key for our case).

Rationale: STANAG 4609 with MISB metadata is **the industry standard for military ISR video** ³. Many border security or law enforcement aerial systems also adopt it for compatibility. The huge benefit is **interoperability**: a video feed with this standard can be displayed on any STANAG-compliant viewer with overlays showing the camera footprint on a map, etc. Without standardization, each camera vendor might have their own way of sending location or might not send any metadata at all, making it hard to use the video for anything beyond eyeballing. By insisting on this, we ensure, for example, that if a camera is tracking a drone, the system knows exactly what patch of ground the camera is looking at and can fuse that with other sensors (like correlate a radar track with the camera view).

Implementation: - **Video Encoding:** H.264 (AVC) or H.265 (HEVC) are acceptable; they balance quality with bandwidth. The standard allows both. H.265 might save bandwidth for high-res feeds but H.264 is more widely supported by legacy systems. - **Container:** Use MPEG-2 Transport Stream (TS) over UDP (or over RTP). This is a common way to packetize video and KLV. Essentially, video frames are broken into TS packets and metadata KLV is also inserted as TS packets with a different PID. - **Timing:** Synchronize video with system time via SMPTE 2059-2 (which basically aligns video timing with PTP). MISB ST 0603/0604 ensure each frame's metadata includes a timestamp (e.g. UNIX epoch or PTP epoch). If everything is PTP synced, the video frames can be stamped to real-world time within <1 ms accuracy. This helps when reviewing footage or correlating with other events. - **Mandatory Metadata:** At the very least, we require the 0601 fields for: - Platform Latitude, Longitude, Altitude, - Sensor Relative Azimuth, Elevation, and Field of View, - UTC Time Stamp (from GPS or PTP), - perhaps platform heading, speed. These allow us to draw on the map where the camera is and what it's looking at (e.g. draw a cone or footprint). - **Optional Metadata:** If available, include things like range to target if the camera has a laser rangefinder, or VMTI if video analytics are running (that could even be automated by an AI detecting moving targets and adding KLV markers). - **Network Transport:** For local networks, this can be multicast UDP TS. For sending over WAN, we might encapsulate it in RTP or use the RIST/SRT mentioned earlier to ensure reliability.

By using this standard, any agency's video analyst tool that conforms to MISB can ingest the feed, and if we record the feed, it's recorded with all the geospatial context, which is crucial for evidentiary or analysis value.

Alternative Transports: If multicast isn't possible or if we need more reliability on IP networks, there are protocols like **RIST** or **SRT** (as earlier) which work well with MPEG-TS streams. They typically just treat the TS as payload. Also, **WebRTC** or other streaming tech could be considered for specific user access, but internally, we stick to one standard to avoid complexity.

4.4 Acoustic Sensors (Audio)

Acoustic arrays or microphones can detect the sound of drones (the buzz of rotors). These produce audio streams and sometimes processed detections (like acoustic direction of arrival). While acoustic data is less standard than radar or video, we can use pro-audio networking standards to transmit microphone data if needed.

Standard: **RTP Audio with AES67 profile** (which is an interoperability mode for low-latency audio over IP).

Rationale: **AES67** is an audio networking standard widely used in broadcast for carrying synchronized, uncompressed audio streams between devices in real-time. It is built on RTP (Real-time Transport Protocol) and works with PTP (it typically uses PTP for clock sync, just like our system does). By using AES67, we can have microphones or acoustic sensors send their audio to the COP with very low latency and jitter, allowing the possibility of triangulating sound or just recording it.

- AES67 specifies 48 kHz, 24-bit audio streams in RTP with typically 1ms or so packets – we can use that as is, or use slightly compressed if needed (though it's generally uncompressed PCM).
- The key is that AES67-enabled devices will synchronize their sampling clocks via PTP (often using the same PTP as everything else). This means multiple acoustic sensors capturing the same drone sound can have their audio samples aligned in time, which is critical for techniques like cross-correlation to find the time difference of arrival (and thus angle/distance).
- Many professional microphone systems or acoustic arrays might not natively output AES67, but an adapter or a small hardware interface can convert analog or USB audio to an IP stream.

Use in COP: - If we deploy acoustic sensors, each sensor (or its adapter) would send an RTP stream labeled with its sensor ID. We might not forward all raw audio to every user (due to bandwidth and limited need), but it could go to a central processor that tries to detect drone acoustic signatures or compute bearings. - Alternatively, acoustic sensors might themselves do the processing and just send detections (bearing and maybe an audio snippet). In that case, just sending the processed info (e.g. via some JSON message) might suffice. But having raw audio available could help in validating detections or recording evidence (e.g., the sound of the drone).

Integration: - Use PTP (the same system-wide PTP) to sync the audio sampling. AES67 mandates use of PTPv2 (IEEE 1588-2008 with appropriate profile). - Ensure QoS: mark audio packets with DSCP 44 (Voice-Admit, as we did in QoS table) to give them appropriate priority and to avoid jitter. - If sending over a WAN, consider compressing (maybe Opus codec in RTP) because uncompressed audio is about 1.5 Mbps per channel. But for a few sensors on a LAN, that's fine.

In summary, while acoustic is a smaller part of the system, using an open audio-over-IP standard like AES67 ensures if we have multi-vendor microphone arrays, we can integrate them similarly to how studios integrate microphones and speakers over IP.

4.5 Drone Remote ID

Modern drones are required in many jurisdictions to broadcast a **Remote ID** – basically a digital license plate that includes the drone's ID, location, and so on. There are two flavors: broadcast (directly via WiFi/Bluetooth) and network (via internet UAS Service Suppliers). Our COP should ingest any available Remote ID info to augment the picture (e.g., show friendly or known drones, or detect unidentified ones).

4.5.1 Data Format - ASTM F3411-22a

Standard: ASTM F3411-22a – this is the international standard specification for UAS Remote ID and tracking, which has been adopted by regulators (FAA in the US, EASA in EU, etc.). It defines both the **Broadcast Remote ID** messages and the **Network Remote ID** schema.

- **Broadcast Remote ID (BRID):** Drones broadcast via Bluetooth or Wi-Fi (Wi-Fi NAN or Bluetooth 4/5 depending on region) a message packet that includes drone ID (like a registration number or UUID), drone position, altitude, velocity, heading, and possibly pilot location if available, plus an emergency status. These messages are short (a few dozen bytes) and are sent periodically (e.g., ~1 Hz or more).
- **Network Remote ID (NET-RID):** Drones (or operators) can also send ID info to a network service (like a USS – UAS Service Supplier) which then shares it with authorized users. This is more for air traffic management. The ASTM standard covers how that data is structured as well (usually JSON over APIs, but we might not directly deal with network RID unless we integrate with a UTM system).

Rationale: By supporting ASTM F3411, we ensure compatibility with the mandated drone identification signals. If a friendly or compliant drone is in the air, we can capture its ID and track from these broadcasts. Many commercial drone detection systems are adding Remote ID receivers for this reason. Also, if a drone doesn't have Remote ID in an area where it should, that itself is a flag (could be a rogue or older drone).

Implementation: - We need **Remote ID receivers** or the ability for some sensors (perhaps RF sensors or dedicated Bluetooth/Wi-Fi sniffers) to pick up these broadcasts. The adapter for that receiver will decode the ASTM messages. - The data then can be fed into the COP: essentially treat Remote ID info as another sensor feed. It provides a track (with GPS-based location from the drone's own GNSS) and an ID (which might correspond to a registry entry of who owns it). - ASTM F3411 defines the message fields and formatting (for broadcast, it's a specific byte sequence). Adapters should parse that into a common format for the COP, maybe even into the same track format (e.g., produce an ASTERIX track for a Remote ID source, with the drone's self-reported position). - Ensure this data is tagged as coming from "Remote ID" and perhaps its confidence might be high (since it's the drone itself reporting, if it's genuine).

One caveat: Remote ID can be **spoofed** – a nefarious drone could send a false ID or location. That's where the next part (DRIP) comes in.

4.5.2 Authentication and Trust – IETF DRIP (Drone Remote ID Protocol)

Standard: IETF DRIP (Drone Remote ID Protocol) as specified in RFC 9434 (architecture) and RFC 9575 (DRIP Entity Tag authentication formats).

What it is: DRIP is a set of protocols to make Remote ID messages trustworthy. It basically defines how to use cryptographic identifiers (called **DRIP Entity Tags (DET)**) which are based on HHITs – Hierarchical Host Identity Tags) for drones. The idea is that each drone has a public/private key pair; the Remote ID contains a cryptographic digest or signature proving the message comes from that drone, and a third-party (like our system) can verify it using a registry.

In simpler terms, DRIP adds a digital signature to the Remote ID broadcast so you can detect if someone is faking a drone's ID. Without DRIP, someone could pretend to be a different drone by just

copying their ID. With DRIP, that fake would not have the right key to sign the message, so you'd know it's not authentic.

Rationale: As agencies using this system, we want to be able to **trust** Remote ID info. For example, if a drone says "I am drone #123 owned by Acme Corp, at location X", we want to know if that's verifiably true or potentially spoofed. DRIP provides the tools for authenticity. It's an evolving standard, but given that our system is intended to be forward-looking and secure, we should implement support for it from the start.

Implementation: - We need to incorporate a **DRIP verifier** in the Remote ID adapter. When a Remote ID broadcast is received, if it includes the DRIP authentication payload (a signature or message authentication code), our system will: - Extract the DET (which is basically the drone's cryptographic ID, often derived from its public key). - Check this against known assignments (DRIP will have registry infrastructure – e.g., the IANA DRIP registry or CAA (Civil Aviation Authority) registries that bind a DET to a real drone registration). - Validate the signature on the message using the corresponding public key for that DET. - Result: mark the track as "Trusted" if it checks out, or "Unverified/Unknown" if not. - Also, implement policy: e.g., if a drone's ID is not authenticated via DRIP, maybe raise an alert or at least treat with suspicion. Conversely, if it *is* authenticated and known friendly (say police drone), flag it as friendly in the interface. - We'll need to keep updated with the DRIP protocols as they mature, but RFC 9575 provides the initial set of authentication messages/fields.

Integration with Remote ID ingestion: - If multiple RAAs (Registration Authority Authorities) exist (like different countries or agencies issuing credentials), the system policy can include which ones to trust. For instance, accept DRIP identifiers from FAA or EASA registries, etc., but maybe ignore ones from unknown sources. - The COP could incorporate a "*trust score*" for tracks: a Remote ID track with a valid DRIP signature might get a "green" label as verified. One with no signature or a bad one could get a "red" label as possibly spoofed.

In summary, **ASTM F3411** gives us the data format to get drone self-ID info, and **DRIP** gives us the means to verify that information's authenticity. Together, they help distinguish cooperative drones from potentially malicious ones.

4.6 Summary of Key Points (Sensor Interfaces)

- **Radar Tracks:** Use **ASTERIX Cat-062** for radar target tracks – a widely adopted format carrying position/velocity/ID for targets. This ensures radar outputs from different vendors look the same to the fusion system.
- **Radar Video:** Use **ASTERIX Cat-240** if raw radar hits are shared, enabling a common way to visualize or further process radar returns.
- **RF Signals:** Use **VITA 49 (VRT)** to stream digitized RF data with timestamps and metadata, making integration of SDR-based sensors easier. For recordings, use **SigMF** (JSON metadata + binary samples) to preserve information for analysis and evidence.
- **EO/IR Video:** Use **STANAG 4609** with **MISB KLV metadata** to ensure full-motion video feeds are geo-tagged and time-synced 3 4. This standard lets us overlay camera views on the map and share video across agencies seamlessly.
- **Acoustic Sensors:** Stream audio via **RTP with AES67** profile for synchronized, low-latency audio. This allows using multiple acoustic sensors together for drone sound detection.
- **Remote ID:** Ingest drone self-identification using **ASTM F3411** format (broadcast Bluetooth/Wi-Fi messages), and apply **IETF DRIP** verification (RFC 9434/9575) to authenticate those messages. This helps distinguish genuine drone IDs from spoofed signals.

By enforcing these standards, the COP creates a *common language* for sensor data. Adapters on the edge translate any proprietary protocols into these, so inside the system everything is consistent and can be processed and fused without custom handling for each vendor.

5. Security and Access Control

Security is absolutely critical in a multi-agency system that could be targeted by adversaries (who might be flying the unauthorized drones!). We must ensure that only authorized devices and users access the system, that communications are encrypted to prevent eavesdropping or injection, and that data is shared on a need-to-know basis. We also need to maintain an audit trail for accountability.

This section covers device identity, network access control, encryption, and data access policies.

5.1 Device Identity and Network Admission Control (802.1X & 802.1AR)

Every sensor or component that plugs into the network should prove its identity before it's trusted. We don't want someone unplugging a sensor and plugging in a rogue laptop feeding fake data, or a malicious sensor impersonating another.

Standard: IEEE 802.1X (port-based network access control) with **EAP-TLS** authentication, using device credentials based on **IEEE 802.1AR** certificates (Device ID, a.k.a. IDevID/LDevID).

Rationale: 802.1X is commonly used for authenticating clients on wired and wireless networks (often in enterprises for user laptops). Here we apply it to sensor devices: when a device connects to an Ethernet switch port (or Wi-Fi), it doesn't get network access until it presents valid credentials (like a user would on Wi-Fi). Using EAP-TLS means the device presents a digital certificate to authenticate. **802.1AR** defines how devices have unique cryptographic identities (an **Initial Device ID** from the manufacturer, and Local Device IDs that we can issue). This ensures **strong, hardware-bound identity** – the private key lives on the device (possibly in a secure element).

Implementation: - Each sensor or adapter gets a certificate (X.509) that identifies it (including perhaps what type or which organization it belongs to). Manufacturers may provide an IDevID installed at factory (signed by manufacturer CA). We can either trust those or more likely issue our own certificates (LDevIDs) from a national PKI for all sensors. - Network switches and controllers will have a Radius or similar authentication server. When a device connects, the 802.1X protocol triggers; the device's adapter (supplicant) provides the cert, proves possession of the key, and the server checks if that cert is trusted (signed by our CA, not expired, not revoked). - Only if authenticated does the switch port open up and assign the appropriate VLAN etc. If not, the device can be put in a quarantine VLAN or just no access. - We will configure different authorization rules depending on certificate attributes. For example, if a device presents a cert that marks it as a "Radar sensor type, Agency X", the NAC system might put it in the Radar VLAN and give it certain network privileges. This is part of the zero trust approach – even after auth, limit what it can do. - **Monitor authentication attempts:** If some unknown device tries to connect, that should generate an alert (someone plugging in something unauthorized).

Analogy: This is like a **bouncer at the door** of a club – each device has an ID badge (certificate). If it's not on the guest list (trusted CA/signature), it doesn't get in. Even if someone clones an IP or tries to mimic a device, without the private key for the cert, they can't complete the auth. This greatly reduces risk of unauthorized equipment on our networks.

Scenario: Imagine a surveillance camera on a pole – if an attacker physically disconnects it and plugs in a laptop, with 802.1X the laptop won’t get any network connectivity (it won’t have the camera’s certificate). Even if they try to spoof the camera’s IP, the switch won’t forward traffic. This buys us security at the first entry point.

5.2 Link-Layer Encryption (MACsec)

Even with authentication, someone might tap the network cables or intercept wireless links. We need to encrypt data in transit. At the very least between sites, but ideally even within a site to prevent insider threats or signal interception.

Standard: IEEE 802.1AE MACsec (Media Access Control Security) with 802.1X MKA (Key Agreement) for distributing encryption keys.

Rationale: MACsec provides encryption at the data link layer (Layer 2). It essentially encrypts every Ethernet frame on a link (except the lowest-level protocol bits) using symmetric keys, and provides integrity (ensuring frames aren’t modified) ⁶. The benefit of MACsec over, say, IPsec in our scenario: - It’s **transparent** to the network and applications – the sensors don’t need to know about it; the Ethernet PHY or switch encrypts/decrypts frames. - It protects *all* traffic on that link: including ARP, PTP, multicast, etc., which IPsec (layer 3) wouldn’t unless we separately handle those. - Modern hardware can do MACsec at line rate (10Gbps, 100Gbps) with no performance hit, whereas IPsec often has overhead and can be tougher to scale on high-speed links. - Simplicity in a managed network: easier to configure on switch ports than managing IPsec tunnels for each device.

Implementation: - **Where:** We should apply MACsec on all critical wired links: certainly on trunk links between switches (to prevent intercept on those), and optionally down to sensor connections if the sensor NICs support it. If sensors don’t, we can at least do “hop-by-hop” MACsec on switch interlinks and maybe use IPsec on the device if needed. Many modern devices, especially industrial ones, are starting to support MACsec though. - **Key management:** Use 802.1X MKA (which can piggyback on the same 802.1X session we established for auth). Essentially, once the device is authenticated, the 802.1X session can generate a symmetric key for that link and configure both sides to encrypt with it (this is all automated by protocols). - **Crypto:** Require a strong cipher suite like GCM-AES-256 for encryption. MACsec supports AES-GCM which provides both confidentiality and integrity. Rotate keys regularly (802.1X can rekey periodically, say every 8 or 24 hours) to limit exposure. - **Network device support:** Ensure switches chosen have MACsec capability on the ports (many enterprise switches do on uplinks at least). Also, point-to-point wireless bridges or any WAN equipment ideally should support it too, or else we might use IPsec tunnels there.

The result is that even if someone taps a fiber or a cable, they see only encrypted gibberish. It also prevents spoofing at layer 2 (frames from unauthenticated devices won’t be accepted because they can’t encrypt with the correct key). MACsec basically extends our trust boundary to exactly the devices we allow – anything outside sees nothing useful.

5.3 Data Access Control and Confidentiality Labels (ABAC)

At a higher layer, once data is in the COP, we need to control who can see what. Multiple agencies will use this system, and some data might be sensitive. For example, the military may contribute a radar feed that they consider classified – they want the fusion to use it, but perhaps a local law enforcement user should only get a generalized result (like the drone’s position) and not see the radar’s location or raw data.

Standard: Attribute-Based Access Control (ABAC) coupled with **data sensitivity labeling**, e.g., using **NATO STANAG 4774/4778** for confidentiality metadata.

Rationale: Traditional role-based access (RBAC) might be too coarse – ABAC allows us to create policies based on attributes of the *user* (agency, clearance, role) and attributes of the *data* (classification level, source agency, data type). Each piece of data (like a track or a video frame) can carry a label saying what classification or restrictions it has (e.g. “Secret//REL TO NATO” or “Law Enforcement Sensitive”). The system can then automatically enforce who can see it or if it needs to be sanitized for certain users.

NATO STANAG 4774 defines an XML format for confidentiality labels – essentially a structured way to tag data with classification and caveats. STANAG 4778 defines how to bind those labels to the data (so they travel with it). By using these or similar standards (e.g. US CAPCO labels), we can implement fine-grained control.

Example Scenario: A military radar track comes in labeled “NATO Confidential, SensorGeoLocation=Secret”. The policy might allow law enforcement users to see the track’s existence (that a drone is there) but not the sensor origin. The system could thus display the drone on the map for them but hide or anonymize the radar source. Meanwhile, a military user with proper clearance sees everything.

Implementation: - **Tagging:** Every data object in the system (track, detection, video stream, alert) should have metadata tags for classification and any special handling. This could be done at the message level (e.g. including a label field in a track message format) or at the database level. We might categorize data like “Unclassified, Protected, Secret” etc., along with flags for “Releasable to X”. - **Policy Engine:** Implement an ABAC engine that checks user attributes (perhaps from an identity management system – e.g. user’s org, clearance level, role like “analyst” or “commander”) against the data labels. This can be done using standards like the OASIS XACML or just custom logic. If a user isn’t cleared for certain data, the system will either filter it out or downgrade it. - **Downgrade/Sanitization:** In some cases, instead of outright blocking data, we can **downgrade** it. For example, show the track but with reduced precision (maybe round the coordinates or delay it a few seconds) so that classified sensor capabilities aren’t revealed. Another example is blurring parts of a video for users not authorized to see raw footage of certain areas. - **Audit:** All access decisions should be logged – e.g. if a user tried to view something and was denied by policy, or when a classified piece of data was released to someone (even in downgraded form), that’s recorded.

By using ABAC and labeling, we ensure multi-agency sharing happens safely. We can confidently ingest data from a high-classification source, and the system will ensure that only the appropriate views are given to others. This also helps with compliance to laws/policies like privacy laws – e.g., tag data containing personal info and restrict access to it.

(For implementation we might integrate with existing solutions or standards: e.g., in the US context, NIEM or CJIS policies for law enforcement data, etc., but the principle is the same.)

5.4 Logging, Audit, and System Monitoring

Security isn’t complete without oversight – we need robust logging of system events for both security monitoring and later audit.

Standard: RFC 5424 Syslog format for log messages, sent over **TLS (RFC 5425)** to a central log server (Security Information and Event Management system, SIEM). Use structured logging with standard fields.

Rationale: A standardized logging format means logs from different components (Windows servers, Linux servers, network devices, applications) can all be aggregated and understood in one place. RFC 5424 defines a modern syslog format that includes timestamps, severity, process info, etc., and allows structured data fields (for example, an audit event can have fields for user, action, object). RFC 5425 mandates using TLS to encrypt the log transport, so that sensitive log data isn't intercepted or altered in transit.

Implementation: - Every component of the COP (adapters, core fusion engine, databases, network switches, etc.) should send logs to the central system. They should include: - Security events: login attempts (successful or failed), 802.1X auth results, data access denials by ABAC, etc. - System events: sensor connected/disconnected, process started, errors, etc. - User actions: if an operator manually reclassifies a track or overrides something, log it. - Use a dedicated log aggregation server (or cluster) that listens on port 6514 (the default for syslog over TLS). All agents use mutual TLS (the server has a cert and clients have certs to authenticate as well, so only authorized systems send logs). - Follow formatting guidelines: e.g., include an *Event ID* or *Message ID* for correlation, include relevant object info. We may adopt or reference **Common Event Format (CEF)** or **JSON logging** for application logs but ensure it's encapsulated in the syslog envelope. - **Tamper-evident storage:** Logs should be stored in a way that they can't be easily modified without detection. This could mean append-only storage or using a blockchain-like ledger for important audit logs, or simply restricting access strongly. Regular backups of logs and possibly sending to two separate systems (to prevent one system from being tampered). - Ensure timestamps in logs are in sync (again, thanks to PTP, all systems should log in the same time frame).

Compliance: Logging supports after-action investigations and compliance with any legal requirements (for example, being able to show who accessed what information, or to trace the sequence of events in an incident). It's also key for real-time security monitoring – a SIEM can raise alerts if, say, an unauthorized device keeps trying to connect (by analyzing 802.1X failure logs) or if a user from agency A attempts to access data labeled for agency B (ABAC denial logs).

5.5 Summary of Key Points (Security)

- **Strong Device Authentication:** Every device must authenticate via 802.1X before joining the network, using certificates (802.1AR DevID). This stops rogue hardware from plugging in without notice.
- **Encryption Everywhere:** All sensitive links use **MACsec** encryption at layer 2, ensuring that even if communications are intercepted, the data remains confidential and untampered ⁶. Additionally, higher-layer encryption (TLS) is used for any API or control channels.
- **Fine-Grained Data Access:** Using **ABAC** with data labels (e.g. STANAG 4774) allows us to dynamically restrict or sanitize information based on user attributes. Each track or video feed can carry a classification, and the system will enforce who can see what.
- **Audit Trails:** All actions and security events are logged in standardized format (syslog RFC 5424 over TLS). The system maintains an **immutable audit trail** so that any misuse or incident can be investigated after the fact. Logs are time-synced and centrally collected for correlation.
- **Continuous Monitoring:** The security infrastructure should be monitored – e.g., alerts on failed logins, unauthorized network access attempts, time sync anomalies (from Section 3), etc., to catch attacks or malfunctions in real time (NIST SP 800-115 guidance on test scenarios is followed for periodic security testing).

- **In essence**, a zero-trust approach is adopted: verify every device and user, assume the network is hostile (hence encryption), and limit access to only what's needed, while keeping a close eye on everything through logging.

6. Common Operating Picture (Data Fusion & Presentation)

Now that we have all these sensor inputs and a secure network to carry them, the COP's core job is to fuse the data and present a coherent picture to users. This involves correlating tracks from multiple sensors (data fusion), managing multiple targets, and allowing users to see what they need to see. We'll cover the standards for track fusion output and the approach to sharing the fused picture across agencies.

6.1 Multi-Sensor Track Fusion – STANAG 4676

When multiple sensors observe the same drone, the system should fuse those observations into one "track" so the drone is represented once on the display, with the best combined estimate of its position and trajectory. Also, if one sensor drops tracking, another might still have it – fusion ensures continuity. Conversely, if sensors have different tracks that turn out to be the same drone, fusion merges them; if one sensor's single track splits into two objects, the system must handle that too.

Standard: **NATO STANAG 4676** (Ed.3 or Ed.4) for *Track Data Exchange*. Additionally, NATO's AEDP-12 provides implementation guidelines for STANAG 4676.

What it is: STANAG 4676 defines a very comprehensive data model for tracks. It's often encoded in XML or sometimes binary (there's a JSON representation too). It includes not only position and kinematics but also:

- Track histories (past positions, update times),
- Covariance (uncertainty ellipses of the track state),
- Track source attribution (which sensor reports contributed to this track),
- Track status (e.g., tentative, confirmed, dropped),
- Ability to handle track splitting and merging events,
- Metadata like identity (if identified as friend/foe), etc.

It's basically designed for multi-sensor fusion in defense systems, where you might have radar, IR, etc. all contributing.

Rationale: STANAG 4676 is a high-level fusion standard that can carry the richness of data we want for a COP. We might not expose all of it to every user (some might get a simpler view), but using it internally or as an export format ensures we can interoperate with other high-level systems (like NATO C2, or even just recording the complete fused track info for analysis).

However, implementing full STANAG 4676 is non-trivial. At minimum, we can use it as a reference for what information to maintain about tracks. The key pieces to implement:

- Each fused track has a unique ID and maintains a list of contributing sensor track IDs (this is track **lineage** or provenance). So we know "Track F123 is composed of Radar1's track 56 and RF sensor A's detection X, etc."
- It stores a **state vector** (position, velocity, maybe acceleration) with an **uncertainty covariance matrix**. This is important for sensor fusion algorithms (like Kalman filters).
- We log **track events**: track initiated, track merged with another, track split, track dropped. STANAG 4676 defines how to encode those events.

Implementation Approach:

- The fusion engine will receive track reports (e.g. ASTERIX Cat-062) from various sensors. It will perform correlation (likely by comparing positions, velocities, etc., with gating thresholds) to decide if a new report is a new track or associated with an existing track.
- If associated, it updates that track's state (possibly using a filter to smooth/estimate).
- If new, it creates a new track

object. - If a track hasn't been updated by any sensor for a while, it might be considered "lost" and eventually dropped (but keep the history for a time). - If two tracks later seem to be the same object, they get merged – assign one ID as primary, the other is ended. If one track splits (say one sensor thought it was one but now it appears as two separate objects), handle that accordingly. - All these actions are recorded and can be output (or at least available) in the STANAG 4676 format.

Even if we don't literally output an XML file of STANAG 4676 for every update (that would be too heavy real-time), we design the data structures following it so we **could** generate a STANAG 4676 report or share with systems that require it. Perhaps the COP could provide a periodic "track picture snapshot" in 4676 format for sharing to external command systems.

Quality and Metrics: STANAG 4676 encourages providing quality metrics – e.g. a track quality score, or sensor confidence. We should incorporate that: e.g., how confident is the fusion that this track is real vs false alarm, or what's the positional error estimate. These help users prioritize threats.

Note: Some simpler systems might not need the full complexity; we can hide complexity from users (show them one icon per track). But under the hood, having the rigorous data helps in multi-agency investigations (who tracked it? with what accuracy? etc.).

6.2 Role-Based/User-Based Data Views (Access Control in COP)

(This ties into the security ABAC above, but here we discuss it from the user/COP perspective.)

Different users of the COP will have different needs. A military operator might see a very detailed screen (with sensor locations, raw data overlays), while a local law enforcement user might see just the drone tracks and perhaps only in their region. The system should provide **role-based or attribute-based filtering of the COP view**.

Standard/Concept: Attribute-Based Access Control (ABAC) – as described, we tag data and enforce policies. Here specifically: - Use user roles/attributes to filter the data fed to their client application. - Possibly utilize OGC standards for map/feature services if needed to disseminate info (for instance, if sharing to web-based mapping clients, etc., one could use standards like OGC API – Features, etc., but with filtering).

Practical Implementation: - The COP server can maintain multiple "publish/subscribe" channels or queries. Users subscribe to the feeds they're allowed. For example, one feed might be "All tracks unfiltered" (only available to top-level operators), another might be "Tracks in Region West, anonymized" for another agency. - When a user logs in, their JWT or session will carry roles (e.g. via claims like Agency=X, Level=Secret, etc.). The backend uses that to determine which subscription or which filter to apply. - **Data Tagging in COP Database:** Each track or detection could have a field like `classification` and `releasability`. The system could implement queries like "select all tracks where classification <= user_clearance or releasability includes user_agency". - **Views:** The COP UI might have layers that are enabled/disabled based on permissions. E.g., sensor layer (showing sensor locations) only for those allowed; raw video feed windows only for some; whereas everyone sees the basic map with tracks.

Example: A Border Patrol user might log in and see tracks and maybe video feeds from cameras in their area, but they might not even know a military radar 100km away is feeding those tracks (they just see the track). Meanwhile, an Air Force user sees everything including that radar's location and coverage. The system automatically does this; users don't have to manually select.

Audit angle: Every time data is hidden or shown due to policy, it could be logged. For instance, if an intel analyst tries to click on a track to see its detailed history and that includes classified sensor info, if they are not cleared, the system might refuse and log that attempt.

Collaboration: Users with higher privilege might be able to “downgrade” a piece of data manually if needed (e.g., decide to share a track with an allied nation – the system should allow marking it as such, which updates the label and then it flows to those users). That process would be controlled and logged.

6.3 User Interface and API Standards (Informative)

(This section is not strongly requested by the prompt, but for completeness, we mention how the COP might present data or allow integration.)

Mapping and Visualization: Likely use standard digital map data (like WMTS/XYZ tiles for map background). For overlays, possibly use something like GeoJSON or KML if exporting tracks to other systems. Not a hard requirement, but the system’s front-end could use web standards so that it’s easily accessible (e.g., a web-based UI for broad access, using HTTPS and JSON REST or WebSockets).

COP Data API: We might provide a REST or DDS-based API to query the common picture for those who can’t use the standard UI. For example, an integration with an airport’s system might pull a feed of current tracks via a REST endpoint returning JSON tracks (with appropriate filter). Using something like **OpenAPI/Swagger** documented JSON API or an **OMG DDS (Data Distribution Service) interface** are options. However, detailing that is beyond scope here, just acknowledging we would adhere to standard API practices.

6.4 Summary of Key Points (COP & Fusion)

- **Track Fusion:** The COP fuses sensor inputs into single tracks per drone using advanced tracking algorithms. We structure fused track data following **STANAG 4676**, which means we keep track histories, sensor provenance, and quality metrics for each track (who saw it, how confident) in a standardized way.
- **Unified Picture:** At any time, users see one integrated picture, not a disjointed feed per sensor. If three sensors see the same drone, the system shows one track with combined information (e.g., perhaps labeled by multiple sources). This avoids confusion and clutter.
- **Information Control:** Building on security, the COP only shows data appropriate to the user. Through **ABAC policies**, sensitive details (like sensor positions or high-classification tracks) are withheld or generalized for users without clearance. This ensures wide sharing without compromising secrets.
- **Collaboration:** The system maintains data lineage and labels, so any track shared across agencies carries its context (origin, classification). If an agency adds identification info (say, “this is our authorized drone #XYZ”), that can be tagged to the track and seen by others if permitted.
- **Interoperability:** By using standard track formats (4676, ASTERIX) and possibly providing open APIs, the COP can exchange data with other systems (e.g., feed into an airport ATM system, or accept tasking from an external C2). It isn’t a silo; it’s an open-standards-based hub of drone air picture information.

7. System Management and Maintenance

Finally, keeping such a system running smoothly requires standardized approaches to configuration and monitoring. This ensures that multi-vendor components can be managed in a uniform way, and we can maintain the system efficiently.

7.1 Configuration Management – NETCONF/YANG

As we have many network devices and possibly sensor gateways, we want an automated, transactional way to configure them, rather than manually via CLI or web GUIs.

Standard: **NETCONF (RFC 6241)** for configuration, using **YANG models** for the data schema (RFC 6020/7950).

Rationale: NETCONF is a protocol specifically made to manage network and device configurations. It's transaction-oriented (you can stage changes and commit or rollback), and it works over secure channels (SSH or TLS). YANG is the modeling language that defines the configuration data structures in a standardized way. Many modern network devices (switches, routers) and even some sensors or software provide YANG models for their configuration and support NETCONF or its cousin RESTCONF.

Using NETCONF/YANG means: - We can script and automate the bringing up of the system or changes to it. For example, we could have a script to onboard a new sensor: allocate IPs, add it to appropriate network segments, push initial config via NETCONF (like telling the switch to expect a device with a certain certificate on that port). - With YANG models, we ensure we validate config parameters properly. The vendor supplies YANG for their device (like a switch's VLAN config model or a camera's streaming config model), so the management system can catch if an invalid value is set. - We can also retrieve configuration or state in a structured way to backup or audit device configurations.

Implementation: - Use a centralized configuration manager (could be an open-source tool or a commercial one) that speaks NETCONF/RESTCONF to devices. - Maintain an inventory of devices and their capabilities (which YANG models they support). - Use common YANG models where possible, e.g., **OpenConfig** models. OpenConfig is an initiative that provides vendor-neutral YANG models for common things (interfaces, BGP, system clocks, etc.). If our devices support OpenConfig, we can manage different brands in one consistent way. - For sensors or applications, if they don't support NETCONF, we might use other means (like Ansible scripts or vendor APIs). But as a goal, any new component should have a programmable interface, preferably modeled in YANG/NETCONF, to allow integration into this management scheme.

Alternative: RESTCONF (RFC 8040) – this is essentially HTTP/REST access to YANG-modeled data, which some systems prefer if they already use web APIs. It's simpler in some ways but less feature-rich than NETCONF (no built-in locking or candidate config concept). However, for systems that are more software than hardware, RESTCONF could be easier to implement and use.

Either way, the key is *model-driven management* vs ad-hoc. It reduces configuration errors (the system can validate and also do partial commits, etc.) and makes it easier to manage many devices.

7.2 Telemetry and Monitoring – gNMI and Streaming Telemetry

Instead of the old approach of polling devices for status (like SNMP polling which is slow, periodic, and doesn't scale well), we employ modern **streaming telemetry**: devices continuously push measured data to a collector.

Standard: **gNMI (gRPC Network Management Interface)** with **OpenConfig telemetry models** for streaming device data.

Rationale: gNMI is a Google-developed open standard (now in OpenConfig) that uses gRPC (a high-performance RPC framework over HTTP/2) to subscribe to data on a device. The device sends updates either on a schedule or whenever values change. This is far more efficient and near-real-time compared to polling every X seconds. It also uses the same YANG models, meaning the data we get is structured according to the model (e.g., we get a JSON or protobuf that matches the YANG fields).

What to monitor: - **Network devices:** port statuses, bandwidth usage, QoS stats, PTP stats (offset from master, etc.), MACsec status (encryption up or not), CPU/memory of devices, temperature, etc. - **Sensors/adapters:** operational status, CPU load, any internal error counts (like a radar might expose error flags via SNMP or an API; we'd convert that to telemetry). - **Applications/servers:** Could use other telemetry like Prometheus metrics or logs, but we can encapsulate some in gNMI if applicable.

Implementation: - Use a telemetry collector system that can handle gNMI inputs. There are open-source ones (like Telegraf with plugins, or specific OpenConfig collectors). - Define subscription queries for each device. For example, subscribe to interface statistics on all switches (on-change or periodic 10s), subscribe to PTP clock class and offset on grandmaster (on-change), subscribe to sensor heartbeat statuses (maybe via an adapter that exposes those as YANG values). - Ensure the network can handle telemetry load – with many devices sending frequent updates, plan capacity (but gNMI is binary and efficient). - Security: gNMI runs over TLS typically, with client certs for auth or other auth means. We will secure it similar to NETCONF (the devices and collector trust each other's certs).

Outcome: This will enable near real-time dashboards of system health. E.g., you could see a graph of link latency, packet loss, etc., instantly notice if a sensor's PTP sync is drifting (which might indicate a tampering or failure), or catch high CPU on a video server before it fails.

It also helps with scaling: adding more devices doesn't exponentially increase polling load because each just streams its own data.

7.3 Testing and Conformance

(Combining acceptance testing from original doc Section 9 as it fits ensuring the system meets standards.)

After building the system, we must test it against these standards to ensure everything works as intended.

Key tests include: - **Timing Validation:** Using PTP monitoring tools or PTP YANG models, verify that all devices are within the microsecond sync budget. Intentionally fail the primary grandmaster and ensure failover to backup occurs within 1 second (the BMCA should re-elect quickly). Test PTP security by trying to introduce a rogue grandmaster (it should be ignored/blocked). - **Multicast and QoS Validation:** Simulate a sudden load (like flood video) and confirm that an EF-marked test message still gets through <50ms. Do IGMP join/leave storms to ensure membership to multicast changes propagate within ~1 second (so when a user subscribes or unsubscribes data, they get it promptly). Verify SSM filters by attempting to send from a non-authorized source to a group – ensure no receivers get that (the network should drop it). - **Redundancy/Failover:** For FRER, maybe break one of the dual paths and confirm that the stream continues with zero packet loss (tools exist to verify if packets were lost or out-of-order). For WAN, failover a link and measure recovery time of stream (should be <500ms with our config). - **Security Penetration Testing:** Following **NIST SP 800-115** or similar, perform authorized penetration tests: e.g., try to plug in an unauthorized device and see if 802.1X stops it; try to sniff traffic and see if it's encrypted (should be unreadable); attempt a replay or MITM on PTP (should be prevented by auth or detected by monitoring), attempt to access data as a low-privileged user (the ABAC should

prevent it). - **Interoperability tests:** Bring a sensor of each supported type in a lab, feed known patterns, ensure the COP ingests them correctly, forming tracks, etc. E.g., feed a simulated ASTERIX Cat-062 from Vendor A radar and Vendor B radar and see that they fuse if they represent same object. - **Throughput and Load:** Ensure the system can handle the anticipated scale (e.g., X number of sensors, Y number of simultaneous tracks, Z users viewing). If needed, tune performance (increase network bandwidth or optimize code). Use tools or recorded data to simulate heavy load.

All these should be documented in an acceptance test plan. The system is only as good as it performs under stress, so these standards come with numbers (like latency thresholds, failover times) that we must meet. Where things fall short in testing, we iterate (maybe adjust configurations or upgrade hardware).

7.4 Summary of Key Points (Management & Maintenance)

- **Automated Config Management:** Use **NETCONF/YANG** to configure network gear and possibly sensors, enabling consistent, repeatable setups and easy integration of new devices.
- **Streaming Telemetry:** Use **gNMI/OpenConfig** for real-time monitoring of system health. This gives immediate visibility into issues (like sync loss, high utilization) instead of waiting for periodic SNMP polls.
- **Unified Monitoring:** All components report to a central system, so admins have one pane of glass for the status of networks, sensors, servers, etc., with alerts for any anomaly.
- **Regular Testing:** Establish a routine of testing failovers and security (penetration tests) to ensure the system remains resilient and secure as updates or changes occur. Compliance tests (e.g. does each interface actually enforce the QoS and VLAN rules) should be part of commissioning new deployments.
- **Documentation and Change Control:** Every device's intended config is modeled (YANG) and version-controlled. Changes go through a change management process, reducing the risk of misconfiguration.
- **Overall,** this systematic approach to management reduces downtime, ensures the system scales, and that the complex multi-vendor environment can be controlled without chaos.

8. Challenges, “Blind Spots”, and Mitigations

Finally, it's worth acknowledging some known challenges (or “blind spots”) in a multi-sensor drone detection COP and how our standards and design mitigate them:

Challenge 1: Acoustic False Alarms (Noise vs. Drone Sounds)

Issue: Acoustic sensors can trigger on benign noises (e.g., lawnmowers, helicopters, even loud vehicles), which could clutter the system with false drone alerts. In urban or noisy environments, microphones might frequently misclassify sounds.

Mitigations: Rely on acoustic sensors primarily to **cue** other sensors. For example, if an acoustic sensor hears a possible drone, the system can point a camera there (if PTZ cameras are integrated) for visual confirmation, rather than immediately declaring a drone. We give acoustic detections a lower confidence level unless corroborated. Our fusion engine can be tuned to not promote an acoustic-only track to a full alert unless persistent or multi-sensor confirmed. Also, using machine learning on acoustic signatures and continuously updating it can reduce false positives (those algorithms can be part of the sensor's processing). In COP policy, treat acoustic detections as advisory unless combined with another modality.

Challenge 2: Remote ID Spoofing or Absence

Issue: A sophisticated enemy could transmit fake Remote ID signals (e.g., to mislead us about a drone's

identity or location), or conversely, a drone might not broadcast Remote ID at all (either because it's rogue/old or deliberately disabled).

Mitigations: We implemented **DRIP authentication** – this will flag unauthentic Remote ID messages. If a Remote ID doesn't check out, the system can label that track as "Unverified Drone" and perhaps elevate its threat level (since law-abiding drones should have valid ID). We also do not rely on Remote ID as the sole detection; it's supplementary. If a drone has no Remote ID but our radars/RF pick it up, we still track it and alert. Essentially, Remote ID becomes one more sensor – a cooperative one. Lack of Remote ID where expected is itself an alert. All Remote ID data is logged, so any spoofing attempt (with bad signature) is recorded and can be analyzed by authorities (maybe to trace the source of the spoof signal if possible).

Challenge 3: Latency Stacking and Tracking Fast Drones

Issue: Even though each subsystem is "real-time", small delays add up: sensor processing (e.g. camera encoding) might add 200 ms, network transit another 50 ms, display/render another 100 ms. A fast drone (say moving 20 m/s) could move several meters in that time, meaning the picture on screen lags reality. If too sluggish, that undermines interception or visualization.

Mitigations: We aim to minimize each part: using efficient network (multicast, no unnecessary hops), QoS to cut queue delays, and fast processing (GPU decoding for video, etc.). More formally, we use **PTP timestamps** for each detection, so even if there is a slight lag, the system knows the detection time. The fusion engine can then **extrapolate** the track position to current time using last known velocity – effectively predicting where the drone is "now". For example, if our track data is 0.5 seconds old, and drone velocity is known, we project it forward on the display. This keeps the displayed position closer to real. Additionally, we enforce that any critical alert (like entering a restricted zone) is raised as soon as *any* sensor sees it, even if others are lagging. The system also allows "time alignment" analyses offline (so investigators know exactly where everything was at a given true time). Our adherence to PTP and timestamping everywhere is the core enabler to manage latency.

Challenge 4: Privacy and Legal Constraints

Issue: The system might collect surveillance data over civilian areas (high-res cameras, etc.). There are privacy laws (like GDPR in Europe) and civil liberties concerns. Agencies need to ensure they aren't storing or sharing data beyond their legal authority.

Mitigations: Built-in **data labeling and policy enforcement** addresses some of this – e.g., video feeds might be labeled as containing personally identifiable information (PII), and policy might restrict who can access full resolution. We can implement **privacy masking** on video: essentially, certain areas or contexts in the video can be masked (blurred or blacked out) unless a validated threat is present. For instance, the system might continuously record raw video but only allow review of segments where a drone was detected, and purge other segments after a short time. All data retention follows the least necessary principle: logs and tracks are kept only as long as needed for response or legal requirements. Also, sharing of data between agencies goes through the ABAC checks – e.g., law enforcement might require a warrant to access full archives, etc., which can be enforced by the system governance. In short, the technical architecture gives the flexibility to implement strict privacy rules and audit their compliance (because we log all access).

Challenge 5: Interoperability vs. Proprietary Temptation

Issue: A vendor might offer a shiny new sensor with great capabilities but it uses a proprietary interface that doesn't fit our standards. There may be pressure to integrate it quickly (especially if it promises improved detection), potentially undermining the open standards approach if done haphazardly.

Mitigations: The governance of the system should hold the line that any new sensor must come with an **adapter or open interface**. If a vendor doesn't support, say, ASTERIX, the contract could require them to develop an ASTERIX output or provide an SDK so we can write an adapter. The architecture is designed to accommodate new stuff via plugins – we will schedule proper development for that rather

than quick-and-dirty integration that breaks the model. Over time, as our standards-based approach gains traction, vendors will likely include support out-of-the-box (because they know it's needed to sell to us). This is more of a program management point, but vital to maintaining the system's health.

9. Conclusion

In this document, we specified a comprehensive set of technical standards and best practices for a multi-vendor, multi-agency drone detection Common Operating Picture system. By adhering to open standards at every interface – from network protocols to data formats and security mechanisms – the system achieves a high degree of interoperability, flexibility, and trustworthiness. We have built in the ability to evolve (add new sensors, adopt new threat response techniques) without overhauling the whole system, thanks to the modular, adapter-based design and forward-looking choices like IPv6, PTP, and standard data models.

To summarize the high-level benefits of this standards-based approach:

- **Interoperability:** Any compliant sensor or subsystem can plug in, enabling agencies to choose the best sensors on the market and share data seamlessly.
- **Security:** A layered defense from physical network access up through application data access ensures that only authorized participants are on the network and only authorized eyes see sensitive data. Cryptographic verification (for both devices and data like Remote ID) adds trust in a potentially adversarial environment.
- **Real-Time Efficacy:** Utilizing multicast, QoS, and time synchronization means the system delivers information with minimal delay and aligns data in time for accurate fusion. When seconds matter (or even microseconds for sensor alignment), the chosen standards provide the performance needed.
- **Scalability and Resilience:** The network design using TSN and FRER means the system can scale to many sensors and users without collapsing under bandwidth or losing data on failures. Redundancy and reliable transport ensure continuity of the picture even amid network issues.
- **Maintainability:** Standard management protocols and telemetry allow the whole system to be monitored and controlled in a uniform way, reducing the burden on operators and administrators. Issues can be quickly identified and addressed, and upgrades or expansions can be rolled out systematically.
- **Accountability and Audit:** Through comprehensive logging and standardized record-keeping (for tracks, for user actions), the system creates a forensic trail. This not only helps in security but also in evaluating system performance and outcomes after events (lessons learned).
- **Future-Proofing:** The system is designed with the future in mind – more drones, more sensors (like perhaps drone mitigation systems), integration into broader airspace management (UTM). By using flexible, widely supported standards (many of which are still evolving with backward compatibility), we position the COP to adapt rather than become obsolete. For instance, as new Remote ID or communication standards emerge, they can be integrated via adapters if they comply with general principles.

In conclusion, this standards-based COP architecture offers a path to **unify situational awareness** for drone threats across agencies while maintaining robust security and operational integrity. It serves as a technical blueprint that vendors can build against and agencies can trust to interoperate. By being detailed and rigorous in the specifications – yet also providing rationale – we ensure everyone (from engineers to stakeholders) understands not just *what* to do, but *why*. This shared understanding is key to a successful implementation and long-term sustainability of the system.

References (Informative):

- IEEE 1588-2019 Standard for Precision Time Protocol (PTP) and IEEE 1588d-2022 Security Amendment.
- IEEE 802 series (802.1X, 802.1AR for identity; 802.1AE MACsec; 802.1Qbv, Qbu, Qci for TSN; 802.1CB FRER).
- EUROCONTROL ASTERIX standards (Cat-062, Cat-240, etc.) for surveillance data exchange.
- NATO STANAG standards (4609 for video, 4676 for tracks, 4774/4778 for security labels).
- IETF RFCs: RFC 8321, 8655 for DetNet (deterministic networking concepts), RFC 3376/4607 for IGMPv3/SSM, RFC 5424/5425 for Syslog over TLS, RFC 9434/9575 for DRIP Remote ID protocols, and others relevant to QoS (DiffServ, NQB).
- ASTM F3411-22a for UAS Remote ID.
- MISB (Motion Imagery Standards Board) standards ST 0601, 0603, 0903 for video metadata.
- OpenConfig and related management models for device telemetry and configuration.

(Note: The vendor examples given in the initial references are omitted here, as the focus is on standards. However, multiple vendors have demonstrated support for these standards, indicating a healthy ecosystem for procurement.)

1 2 Precision Time Protocol (PTP) Explained

<https://networklessons.com/ip-services/precision-time-protocol-ptp-explained>

3 4 5 STANAG 4609 – ISR Video – ImpleoTV

<https://impleotv.com/2025/03/11/stanag-4609-isr-video/>

6 MACsec: a different solution to encrypt network traffic

<https://developers.redhat.com/blog/2016/10/14/macsec-a-different-solution-to-encrypt-network-traffic>