FORGE MDPAF/JREAP-C Compliance Implementation: Structurally Faithful Simulation of MIL-STD-3011 Message Encoding and Link 16 J-Series Data Links

FORGE Research • Technical Report

forge-c2 — JREAP-C / MDPAF Compliance Engine

April 2026

Abstract. Interoperability among missile defense C2 systems demands strict compliance with military data link standards, yet the classified nature of specifications such as MIL-STD-3011 (JREAP-C) and MIL-STD-6016 (Link 16) creates significant barriers for simulation environments that must produce structurally faithful messages without access to classified implementation details. We present the design and implementation of a JREAP-C encoding layer and MDPAF data model mapping for the FORGE-C2 framework, targeting structurally compliant simulation of IP-based tactical data exchange. The system implements JREAP-C header encoding, Link 16 J-series message type serialization for missile warning tracks and engagement orders, CRC-16 integrity verification, and dual UDP/TCP transport. We detail the Go package architecture, bit-level pack/unpack utilities for J-series messages, the MDPAF field mapping from internal Go structs to compliant output formats, and a compliance verification framework. A gap analysis against the DoD compliance matrix shows current JREAP compliance at 0% with a target of 80%, identifying specific remediation paths across six implementation phases. The architecture enables the simulation environment to produce and

consume messages that are structurally faithful to publicly documented JREAP-C specifications, facilitating integration testing with real tactical data link infrastructure without requiring access to classified protocol details.

Table of Contents

1. Introduction

Modern ballistic missile defense depends on the seamless exchange of tactical data across heterogeneous systems—from overhead infrared sensors detecting boost-phase launches, through ground-based radars tracking midcourse objects, to command-and-control platforms issuing engagement orders. This exchange occurs over standardized military data links whose specifications, in many cases, carry classification restrictions that limit the ability of simulation environments to produce compliant messages for integration testing and training.

The FORGE framework provides a missile defense simulation environment that ingests sensor events via Apache Kafka, correlates tracks using position-velocity fusion, and produces engagement orders through an internal C2BMC interface. While the system demonstrates strong

compliance in sensor processing (OPIR 90%, Radar 90%) and DevSecOps practices (95%), it currently lacks any capability to encode, decode, or transport messages compliant with MIL-STD-3011 (JREAP-C) or the FORGE Missile Defense Processing Architecture Framework (MDPAF). This gap prevents the simulation environment from interoperating with external tactical data link infrastructure and limits its utility as a testbed for end-to-end missile defense scenarios.

1.1 Military Data Link Interoperability

Military data links serve as the nervous system of integrated air and missile defense. The Tactical Data Link (TDL) family—Link 16 (MIL-STD-6016), Link 11, and their range extension protocols—enables the dissemination of track data, engagement orders, and situational awareness information across platforms and services [1]. Interoperability among these systems is not merely desirable but operationally mandated: a sensor platform that cannot communicate tracks in a compliant format is operationally deaf, regardless of its detection capability.

The Joint Range Extension Extension Application Protocol (JREAP) was developed to extend Link 16 connectivity beyond line-of-sight via satellite relay (JREAP-A), terrestrial relay (JREAP-B), and IP networks (JREAP-C) [2]. JREAP-C, specified in MIL-STD-3011, wraps Link 16 J-series messages in an IP transport envelope, enabling tactical data exchange over standard network infrastructure. This makes JREAP-C particularly relevant for simulation environments that operate on IP networks.

1.2 The Simulation Compliance Challenge

A fundamental tension exists in building compliant simulation for classified protocols: the specifications that define exact bit layouts, field semantics, and behavioral requirements are themselves classified, yet the simulation must produce messages that are structurally indistinguishable from those generated by accredited systems. Our approach resolves this tension by targeting *structural faithfulness*—

producing messages whose format, encoding, and transport comply with the publicly documented aspects of MIL-STD-3011, while acknowledging that full behavioral compliance (including classified field semantics and timing requirements) requires access to the classified specification.

> **Classification note:** MIL-STD-3011 and portions of MIL-STD-6016 are classified. This paper references only publicly available specifications, unclassified summaries, and open-source analyses. No classified protocol details are included. Field values and bit layouts described herein are derived from public documentation and may not match the classified implementation in all particulars.

1.3 Scope and Contributions

This paper makes the following contributions:

- A detailed analysis of the JREAP-C protocol structure derived from publicly available specifications (Section 3)
- A mapping of Link 16 J-series message types relevant to missile warning and engagement (Section 4)
- The MDPAF data model and field mapping from internal Go structs to compliant output formats (Section 5)
- A Go implementation architecture for JREAP-C encoding, decoding, and transport (Section 6)
- A dual UDP/TCP transport layer design with connection management (Section 7)
- A compliance verification framework with automated field-range checking (Section 8)
- A gap analysis against the DoD compliance matrix with a phased remediation roadmap (Section 9)

2. Background

2.1 Link 16 and MIL-STD-6016

Link 16 is a time-division multiple-access (TDMA) tactical data link operating in the 960–1215 MHz band, defined by MIL-STD-6016 [1]. It employs the Joint Tactical Information Distribution System (JTIDS) or Multifunctional Information Distribution System (MIDS) terminals to exchange J-series messages among participating units. Link 16 supports the dissemination of surveillance tracks, engagement status, unit positions, and mission management data across air, ground, and maritime platforms.

The J-series message catalog comprises over 100 message types organized by functional area: initial entry (J0), participant location (J1), surveillance (J2), track management (J3), aircraft management (J4), air operations (J5), missile defense (J6), net management (J7), program management (J8), intelligence (J9), weather (J10), navigation (J11), command (J12), free text (J13–J15), and national use (J28+) [3]. Each message type defines a fixed-format bit field structure with specific word types (initial, continuation, extension) that carry the message data.

Link 16 messages are transmitted in time slots allocated within a repeating 12-second frame. Each time slot carries a pulse train of 317.5 µs bursts using frequency-hopping and spread-spectrum techniques for anti-jam resilience. The raw data rate is approximately 56 kbps, though effective throughput varies with encryption overhead and network design [4].

2.2 JREAP-C and MIL-STD-3011

The Joint Range Extension Application Protocol (JREAP) extends Link 16 connectivity beyond the line-of-sight limitation inherent in the JTIDS/MIDS UHF transmission. MIL-STD-3011 defines three JREAP variants [2]:

- **JREAP-A:** Satellite relay using UHF SATCOM channels, providing beyond-line-of-sight reach via geostationary satellites

- **JREAP-B:** Line-of-sight range extension using dedicated terrestrial relay nodes
- **JREAP-C:** IP network transport, encapsulating Link 16 J-series messages within IP packets for transmission over standard network infrastructure

JREAP-C is the most relevant variant for IP-based simulation environments. It wraps each J-series message in a JREAP-C header that provides protocol identification, message type classification, length fields, and integrity verification via CRC-16. The JREAP-C header enables IP-connected platforms to participate in the Link 16 tactical picture without requiring direct JTIDS/MIDS terminal connectivity [5].

JREAP-C supports both UDP and TCP transport. UDP provides low-latency, fire-and-forget delivery suitable for high-rate track updates, while TCP provides reliable, ordered delivery suitable for engagement orders and other messages where loss is unacceptable. This dual-transport model mirrors the operational distinction between track data (which ages rapidly and tolerates occasional loss) and command messages (which must arrive reliably) [6].

2.3 MDPAF Purpose and Data Model

The Missile Defense Processing Architecture Framework (MDPAF) defines a processing architecture and data model for the FORGE missile defense simulation environment. MDPAF specifies message categories, metadata fields, and processing node identifiers that supplement the standard Link 16 J-series message types with missile-defense-specific information.

MDPAF extends J-series messages with:

- **Processing metadata:** Node identification, ingest timestamps, quality flags, classification markings
- **Sensor-specific extensions:** OPIR satellite identification, sensor mode, infrared intensity, background temperature, detection confidence, and false alarm rate
- **Correlation identifiers:** End-to-end tracking IDs that span the processing chain from sensor ingestion through engagement output

MDPAF message categories map directly to J-series types: OPIR processing maps to space track messages (J18.x), track management maps to track updates (J3.0), engagement management maps to engagement orders (J6.0), and alert dissemination maps to threat alerts (J2.0). The MDPAF data model thus provides a domain-specific overlay on the standard Link 16 message catalog.

3. JREAP-C Protocol Analysis

3.1 Header Structure

The JREAP-C header provides the framing and identification necessary for an IP-connected receiver to parse and validate a Link 16 message transmitted over an IP network. Based on the publicly available specification structure, the JREAP-C header occupies a minimum of 8 octets with the following layout [2], [5]:

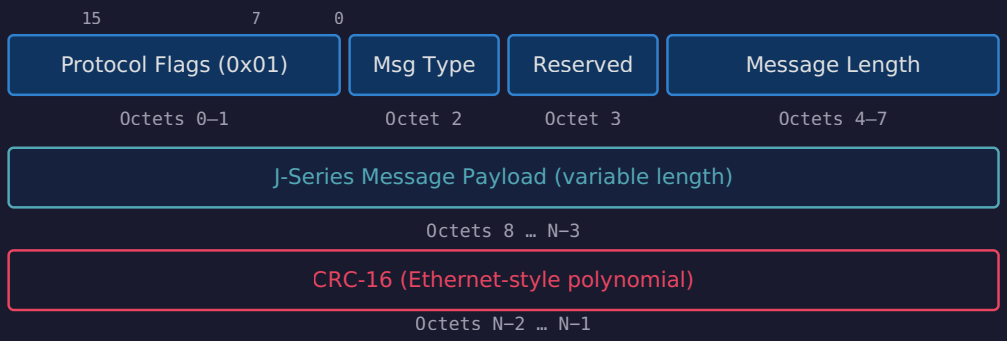

Figure 1: JREAP-C message frame structure. The 8-octet header is followed by the J-series payload and terminated with a 2-octet CRC-16.

The header fields serve the following purposes:

- **Protocol Flags (2 octets):** Identifies the JREAP variant. For JREAP-C over IP, the value is 0x01. This field enables a receiver to distinguish JREAP-C messages from JREAP-A or JREAP-B traffic when multiple variants share a network.
- **Message Type (1 octet):** Encodes the Link 16 J-series message type number. This maps directly to the J-series functional area (e.g., type 1 for J3.0 track updates, type 4 for J6.0 engagement orders).

- **Reserved (1 octet):** Reserved for future use, set to zero in current implementations.
- **Message Length (4 octets):** The total length of the JREAP-C message in octets, including the header and CRC, encoded in network byte order (big-endian). This field enables the receiver to validate framing and allocate receive buffers.

3.2 Message Format and Framing

The complete JREAP-C message consists of the 8-octet header, a variable-length J-series payload, and a 2-octet CRC-16 trailer. The total message length is therefore:

$$L_{\text{total}} = 8 + L_{\text{payload}} + 2 = L_{\text{payload}} + 10 \text{ octets}$$

The J-series payload length varies by message type. A minimal J3.0 track update carrying position, velocity, and track identification may occupy 60–120 octets, while more complex messages with multiple continuation words can exceed 200 octets. The 4-octet length field supports messages up to $2^{32} - 1$ octets, though practical implementations limit message size to the maximum transmission unit (MTU) of the underlying IP network, typically 1500 octets for standard Ethernet [7].

3.3 CRC-16 Integrity

The JREAP-C layer applies a CRC-16 checksum over the header and payload to detect transmission errors. The CRC uses an Ethernet-style (CRC-CCITT) polynomial, which provides strong error detection for the short frame sizes typical of tactical messages [8]. The CRC polynomial is:

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

This is the standard CRC-CCITT polynomial used widely in telecommunications and specified in ITU-T Recommendation X.25. The CRC is computed over the header octets (0–7) and the payload octets (8 through N−3), with the resulting 16-bit checksum appended as the final

two octets (N−2 and N−1) in network byte order. Upon receipt, the receiver recomputes the CRC over the same octet range and compares it to the received value; a mismatch indicates a transmission error requiring message discard or retransmission.

> **Implementation note:** The CRC-16 computation uses a lookup table for performance. With 256 precomputed 16-bit entries, each octet is processed with a single table lookup and XOR operation, yielding $O(n)$ complexity where n is the message length in octets. This is critical for high-rate track update streams.

4. Link 16 J-Series Messages

4.1 Message Types for Missile Warning

The FORGE simulation environment requires J-series message types spanning five functional areas critical to missile defense operations. The following message types are targeted for implementation, selected based on the operational data flow from sensor detection through engagement completion [3], [9]:

Type ID	J-Series Ref	Name	Operational Use
1	J3.0	Track Update	Track position, speed, heading dissemination
4	J6.0	Engagement Order	Weapon-target assignment directives
5	J6.5	Engagement Status	Intercept result and battle damage assessment
6	J7.1	Sensor Registration	Sensor capability declaration and status

Type ID	J-Series Ref	Name	Operational Use
12	J2.0	Alert/ Notification	Launch detected, threat alert dissemination
15	J0.x	Command	C2BMC command and control directives
28	J18.x	Space Track	Satellite and OPIR track data

This message set covers the complete missile defense data flow: OPIR detection produces space track messages (J18.x), radar and fusion processing generate track updates (J3.0), threat alerts trigger alert/ notification messages (J2.0), engagement decisions produce orders (J6.0) and status updates (J6.5), and sensors register their capabilities via J7.1 messages.

4.2 Track Data Encoding

The J3.0 Track Update message is the highest-volume message type in a missile defense data link, carrying position, kinematics, identity, and quality information for each tracked object. The message is structured as a series of fixed-format words, each 75 bits in length (matching the Link 16 word size), organized as initial words followed by continuation and extension words [1].

The critical fields in a J3.0 Track Update include:

- **Track Number (16 bits):** Unique identifier in range 0–65534 (65535 is reserved for special use). Maps directly to the FORGE TrackNumber field.
- **Latitude (21 bits):** Encoded as a signed angular value with resolution of approximately 180°/2^{21} ≈ 0.000086° ≈ 9.6 m at the equator.
- **Longitude (22 bits):** Encoded similarly with slightly higher resolution due to the additional bit.
- **Altitude (14 bits):** Encoded as a scaled integer representing altitude in predetermined increments.

- **Speed (9 bits):** Encoded as a scaled value representing speed in predefined units.
- **Heading (9 bits):** Encoded as angular direction with resolution 360°/ 2^9 ≈ 0.7°.
- **Track Quality (4 bits):** Quality indicator on a 1–15 scale (0 = no track), derived from sensor measurement quality and track update history.

The total encoded size of a J3.0 Track Update with standard continuation words is approximately 6–8 Link 16 words (450–600 bits), which maps to roughly 56–75 octets in the JREAP-C payload.

4.3 Engagement and Alert Messages

Engagement Order (J6.0) messages carry weapon-target assignment directives from C2BMC to firing units. These messages include the track number of the assigned target, the weapon system identifier, engagement timing constraints, and firing authorization. Due to the critical nature of engagement orders, they are transported over the reliable TCP path in the JREAP-C transport layer.

Alert/Notification (J2.0) messages provide launch detection and threat warning dissemination. In the FORGE context, these are generated when OPIR sensors detect boost-phase signatures or when the track correlator identifies a threat level exceeding threshold. Alert messages carry the threat type classification, estimated launch point, predicted impact point, and confidence assessment. Given the time-critical nature of threat alerts, these messages use the low-latency UDP transport path.

Space Track (J18.x) messages carry satellite and OPIR-derived track data, including IR intensity, sensor mode, and detection confidence. These FORGE-specific fields are encoded as national-use extension words within the J18.x message structure, providing the MDPAF overlay without modifying the standard J-series format.

5. MDPAF Data Model

5.1 MDPAF Metadata Fields

The MDPAF data model augments each J-series message with a metadata structure that provides processing provenance, quality indicators, and security markings. This metadata is not part of the standard Link 16 J-series specification but is required for the FORGE processing architecture to maintain end-to-end traceability and data quality assessment.

```
// MDPAMetadata is appended to every FORGE-generated JREAP-C message.
// It provides processing provenance and quality indicators that
// supplement the standard J-series message content.
type MDPAMetadata struct {
    ProcessingNodeID string    // FORGE node that processed this message
    IngestTimestamp   time.Time // When the source data was received
    QualityFlags      uint8     // Sensor quality bit field
    Classification    string    // Security classification marking
    ApplicationID     string    // MDPAF app that generated the message
    CorrelationID     string    // End-to-end tracking identifier
}
```

The `CorrelationID` field is particularly important for the FORGE processing chain. It provides a unique identifier that traces a message from its source sensor detection through correlation, fusion, and engagement output. This enables post-mission analysis to reconstruct the complete processing history of any track or engagement.

5.2 FORGE Track Extensions

Beyond the standard J-series track fields, FORGE requires OPIR-specific extensions that capture infrared sensor data not represented in the standard Link 16 track format:

```
// FORGETrackExtension carries OPIR-specific data that extends
// the standard J3.0/J18.x track messages with infrared sensor
// observations and missile-defense-specific parameters.
type FORGETrackExtension struct {
```

```
    SatelliteID         string  // SBIRS/Next-Gen OPIR satellite ID
    SensorMode          string  // STARE, SURVEIL, or TRACK
    IRIntensity         float64 // MWIR/LWIR intensity (Kelvin)
    BackgroundTemp      float64 // Background temperature (Kelvin)
    DetectionConfidence float64 // Detection probability [0,1]
    FalseAlarmRate      float64 // Expected false alarm rate (per hour)
}
```

These extension fields map to national-use words in the J18.x message structure, allowing FORGE to embed OPIR-specific data within the standard Link 16 framework without violating the J-series specification. The `IRIntensity` and `BackgroundTemp` fields enable downstream consumers to assess the signal-to-background ratio, which is the primary discriminant for boost-phase detection quality.

5.3 Field Mapping to J-Series

The mapping from internal FORGE Go structs to J-series message fields is defined in the `mdpa/message_map.go` module. The following table illustrates the mapping for the highest-volume message type, J3.0 Track Update:

J3.0 Field	Bit Width	Valid Range	FORGE-C2 Source
Track Number	16	0–65534	Track.TrackNumber
Latitude	21	−90° to +90°	Track.Latitude
Longitude	22	−180° to +180°	Track.Longitude
Altitude	14	Scaled integer	Track.Altitude
Speed	9	Scaled integer	Track.Velocity magnitude

J3.0 Field	Bit Width	Valid Range	FORGE-C2 Source
Heading	9	0°–360°	Track.Heading
Track Quality	4	1–15	Track.QualityScore
Force Type	4	Enum	Track.ThreatLevel

The encoder performs value scaling and range clipping to ensure that internal floating-point values map to the fixed-point integer representations used in J-series messages. For example, a latitude value of 38.9072° is encoded as:

$$\text{encoded} = \left\lfloor \frac{38.9072 + 90}{180} \times (2^{21} - 1) \right\rfloor$$

Similarly, the speed encoding applies a scaling factor that maps the internal m/s representation to the J-series speed field. The exact scaling factor depends on the J-series subfield specification and is configured per message type in the encoder.

6. Implementation Architecture

6.1 Go Package Structure

The JREAP-C compliance implementation is organized as a set of Go packages under the `jreap/` and `mdpa/` directories within the FORGE-C2 codebase. This separation reflects the architectural boundary between the standard JREAP-C protocol layer and the FORGE-specific MDPAF extensions:

```
forge-c2/
  jreap/
    doc.go              # Package docs + MIL-STD-3011 reference
    header.go           # JREAP-C header struct + encode/decode
    message_types.go    # Link 16 J-series type constants
```

```
    jseries/
      doc.go            # J-series message package
      message.go        # Base J-series message structure
      pack_unpack.go    # Bit-level pack/unpack utilities
      j3_track.go       # Track Update (J3.0)
      j4_engagement.go  # Engagement Order (J6.0)
      j5_engage_status.go
      j6_sensor_reg.go  # Sensor Registration (J7.1)
      j12_alert.go      # Alert/Notification (J2.0)
      j28_space.go      # Space Track (J18.x)
    encoder.go          # Go struct -> JREAP bytes
    decoder.go          # JREAP bytes -> Go struct
    transport.go        # UDP/TCP JREAP transport
    compliance.go       # Compliance checker + field validation
  mdpa/
    doc.go
    metadata.go         # MDPAMetadata struct
    track_extension.go  # FORGETrackExtension struct
    message_map.go      # FORGE -> J-series field mapping
```

The `jreap/` package is responsible for all protocol-level concerns: header formatting, J-series message encoding/decoding, CRC computation, and network transport. The `mdpa/` package provides the FORGE-specific data model and field mapping logic. This separation ensures that changes to MDPAF requirements do not affect the JREAP-C encoding layer, and vice versa.

6.2 Encoder/Decoder Design

The encoder and decoder follow a symmetric design pattern: the encoder transforms a Go struct into a JREAP-C byte sequence, and the decoder performs the inverse transformation. Both operate on the `JREAPMessage` interface, which abstracts over the specific J-series message type:

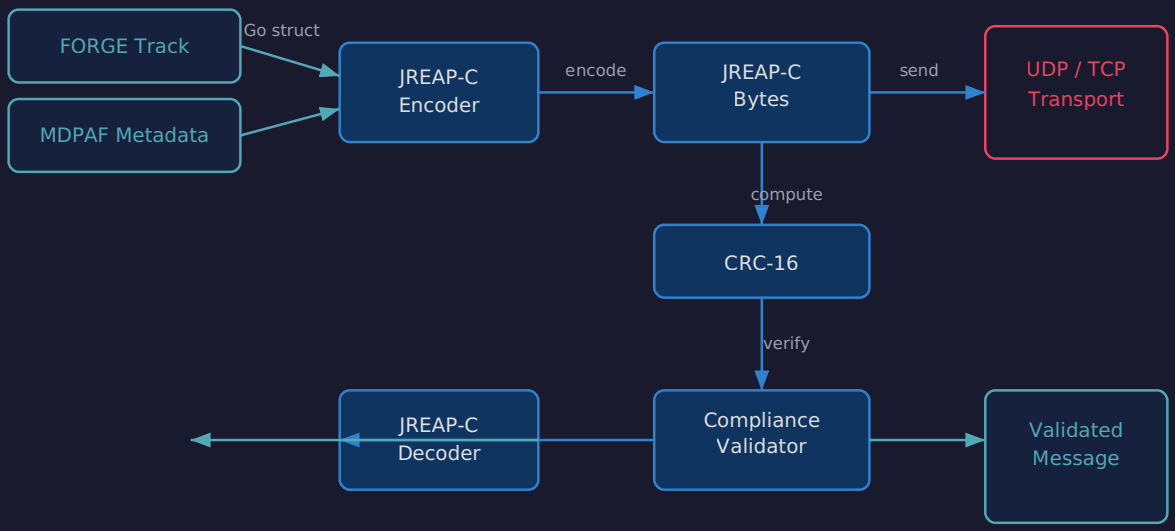

Figure 2: Encoder/decoder data flow. FORGE internal structs are augmented with MDPAF metadata, encoded to JREAP-C bytes with CRC-16 integrity, and transmitted via UDP/TCP. The receive path decodes and validates before delivering to consumers.

6.3 Bit-Level Pack/Unpack

Link 16 J-series messages use non-octet-aligned bit fields that require precise bit-level serialization. The `pack_unpack.go` module provides utilities for writing and reading values at arbitrary bit offsets within a byte buffer:

```
 // PackUint packs an unsigned integer value of 'bits' width
// into buf starting at bit offset 'off'. Values are packed
// in big-endian bit order matching the J-series specification.
func PackUint(buf []byte, off int, bits int, val uint64) {
    for i := 0; i < bits; i++ {
        byteIdx := (off + i) / 8
        bitIdx := 7 - ((off + i) % 8) // MSB-first
        if val&(1<<(bits-1-i)) != 0 {
            buf[byteIdx] |= 1 << bitIdx
        }
    }
}
```

```
// UnpackUint extracts an unsigned integer of 'bits' width
// from buf starting at bit offset 'off'.
func UnpackUint(buf []byte, off int, bits int) uint64 {
    var val uint64
    for i := 0; i < bits; i++ {
        byteIdx := (off + i) / 8
        bitIdx := 7 - ((off + i) % 8)
        if buf[byteIdx]&(1<<bitIdx) != 0 {
            val |= 1 << (bits - 1 - i)
        }
    }
    return val
}
```

These primitives enable each J-series message type to define its encoding as a sequence of `PackUint` calls at specific bit offsets, closely mirroring the J-series field layout. For example, the J3.0 Track Number field is packed at bit offset 0 with width 16, and the latitude field follows at offset 16 with width 21. This approach produces code that is directly reviewable against the J-series specification, reducing the risk of encoding errors.

7. Transport Layer

7.1 UDP/TCP Dual Transport

The JREAP-C transport layer implements a dual-path architecture that selects between UDP and TCP based on message type and reliability requirements [6]:

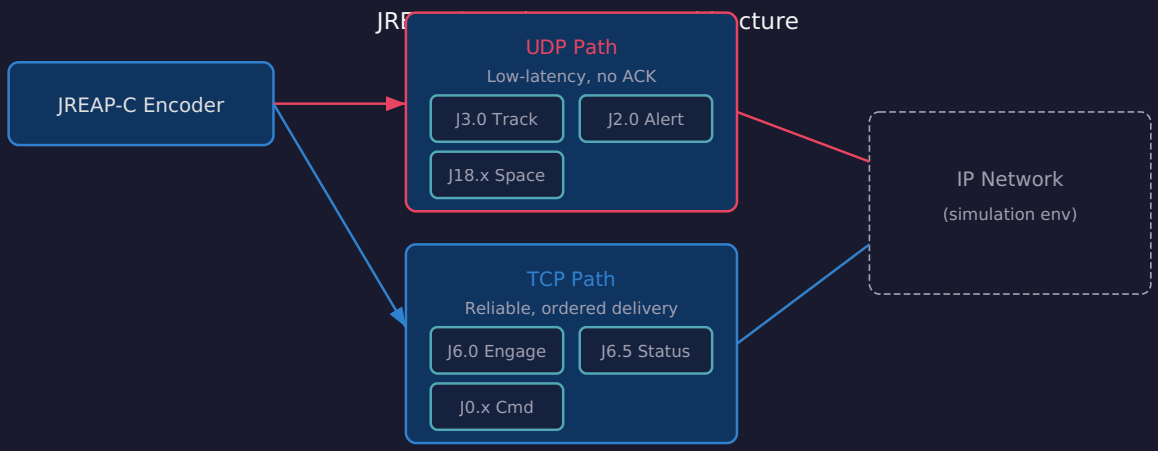

Figure 3: Dual transport architecture. Track updates and alerts use UDP for minimum latency; engagement orders and commands use TCP for guaranteed delivery.

7.2 Connection Management

The TCP transport path maintains persistent connections to configured endpoints with automatic reconnection on failure. Each connection is managed by a goroutine that handles the write loop, read loop, and health monitoring. The connection state machine transitions through `Disconnected` → `Connecting` → `Connected` → `Reconnecting` states with exponential backoff on connection failures.

UDP transport is connectionless and requires no state management beyond tracking the remote address. However, the UDP path implements a simple sequence number counter that allows the receiver to detect dropped or reordered messages. Each outgoing JREAP-C message on UDP carries a monotonically increasing sequence number in the reserved header octet, enabling the receiver to detect gaps in the message stream.

7.3 Reliability and Ordering

The transport layer provides the following reliability guarantees:

Property	UDP Path	TCP Path
Delivery guarantee	Best effort	Reliable (ACK/ retransmit)
Ordering	Not guaranteed	FIFO within connection
Duplicate detection	Via sequence number	Inherent in TCP
Congestion control	None (rate-limited by sender)	TCP Reno/CUBIC
Head-of-line blocking	No	Yes (per connection)
Typical latency	<1 ms (local)	1–5 ms (local)

For track updates (J3.0), the best-effort UDP path is appropriate because tracks are continuously updated—a missed update is quickly superseded by the next one. For engagement orders (J6.0), the reliable TCP path is mandatory because a lost engagement order could result in a missed intercept opportunity. This transport selection is configurable per message type in the `transport.go` configuration.

8. Compliance Verification

8.1 Validation Framework

The `compliance.go` module implements a validation framework that checks each generated JREAP-C message against the compliance requirements derived from the public MIL-STD-3011 and MDPAF specifications. The framework operates in three phases:

1. **Structural validation:** Verifies that the message header format conforms to the JREAP-C specification—protocol flags value, message

type within the defined range, length field consistent with actual message size, and reserved octets set to zero.
2. **Field range validation:** Checks each J-series field against its defined value range. For example, track numbers must be in 0–65534, latitude must be in −90° to +90°, and track quality must be in 1–15. Out-of-range values trigger a compliance violation report.
3. **Integrity validation:** Recomputes the CRC-16 over the header and payload, comparing the result to the stored CRC value. A mismatch indicates a corruption in the encoding pipeline.

```
 // ComplianceResult captures the outcome of validating a JREAP-C message.
type ComplianceResult struct {
    MessageValid    bool     // Overall pass/fail
    HeaderValid      bool     // Header structure compliance
    FieldsInRange    bool     // All J-series fields within valid ranges
    CRCValid         bool     // CRC-16 integrity check
    Violations       []string // Human-readable violation descriptions
    MessageType      int      // J-series type for context
}
```

8.2 Test Vectors and Automated Checking

The compliance verification framework is exercised by a set of test vectors that define known-valid JREAP-C messages for each supported message type. Each test vector specifies:

- The input Go struct values
- The expected byte-level JREAP-C encoding
- The expected CRC-16 value
- The compliance result (pass/fail and specific field checks)

Round-trip testing validates that `Encode(Decode(Encode(s))) == Encode(s)` for every supported message type, ensuring that the encoder and decoder are inverse operations. Additionally, negative test vectors exercise out-of-range values, truncated messages, and incorrect CRC values to verify that the compliance checker correctly identifies violations.

The automated compliance check is integrated into the CI/CD pipeline, ensuring that every code change is validated against the compliance requirements. The test suite produces a compliance matrix report showing the pass/fail status for each message type and each validation phase.

9. Gap Analysis

9.1 Current Compliance Levels

The DoD compliance matrix assesses the FORGE simulation environment across ten categories. The current compliance levels and the gap to target are summarized below [10]:

Category	Current	Target	Gap	Priority
OPIR Processing	90%	100%	10%	P0
Radar Integration	90%	100%	10%	P0
Sensor Fusion	70%	95%	25%	P1
Hypersonic Tracking	0%	80%	80%	P1
C2BMC Interface	10%	75%	65%	P2
Link 16 (MIL-STD-6016)	50%	95%	45%	P1
JREAP (MIL-STD-3011)	**0%**	80%	80%	P2
DevSecOps	95%	100%	5%	P0
RMF	40%	95%	55%	P1
NIST 800-53	80%	100%	20%	P0

The JREAP compliance gap is one of the largest in the matrix, reflecting the complete absence of JREAP-C encoding, decoding, and transport

capabilities in the current system. The current overall compliance score, computed as a weighted average across categories, is 57.5%. Achieving the target scores would yield an overall score of 93%, representing a 35.5 percentage point improvement.

9.2 Roadmap to Full Compliance

The remediation roadmap is organized into six phases with estimated effort:

Phase	Description	Effort	Key Deliverables
1	JREAP-C Encoding Layer	2–3 days	header.go, message_types.go, encoder.go, decoder.go
2	MDPAF Data Model Mapping	1 day	metadata.go, track_extension.go, message_map.go
3	Transport Layer	1 day	transport.go (UDP + TCP)
4	Integration with FORGE-C2	1 day	Kafka adapter, correlator wiring, CLI flags
5	Compliance Documentation	0.5 day	Per-message compliance matrix
6	Testing & Validation	1–2 days	Unit tests, integration tests, test vectors

The total estimated effort is 7–9 days. Phase 1 is the most complex, requiring the implementation of the JREAP-C header, J-series message type constants, and the encoder/decoder pair. The bit-level pack/unpack utilities in `pack_unpack.go` are foundational—all subsequent message type implementations depend on them. Phase 4 integrates the JREAP-C layer with the existing FORGE-C2 data pipeline by wiring Kafka sensor

events through the MDPAF processing chain to JREAP-C encoding, and by wiring received JREAP-C messages back into the track correlator.

> **Architecture note:** The integration maintains a dual-transport mode: Kafka continues to serve as the internal high-throughput event bus for sensor data, while JREAP-C provides the external compliance-facing interface. This separation ensures that internal processing latency is not affected by JREAP-C encoding overhead, and that the simulation environment can operate in both standalone and networked configurations.

10. Related Work

Distributed Interactive Simulation (DIS) and High Level Architecture (HLA): DIS (IEEE 1278) and HLA (IEEE 1516) provide simulation interoperability standards that define protocol data units (PDUs) for entity state, fire, detonation, and other events in a synthetic environment [11]. While DIS/HLA address simulation-to-simulation interoperability, they do not provide the tactical data link message formats required for compliance with MIL-STD-6016 or MIL-STD-3011. A FORGE system using DIS/HLA for entity state exchange would still require a separate JREAP-C layer for tactical data link compliance. The two approaches are complementary: DIS/HLA for simulation environment integration, JREAP-C for tactical data link compliance.

C2BMC Interface Standards: The Command and Control, Battle Management, and Communications (C2BMC) system provides the operational C2 interface for the Ground-based Midcourse Defense (GMD) system. C2BMC interfaces with external sensors and firing units via proprietary message formats that, while functionally similar to JREAP-C, do not conform to the MIL-STD-3011 specification [12]. The FORGE-C2 implements a C2BMC-compatible internal interface that currently operates at 10% compliance; building the JREAP-C layer provides a standards-compliant alternative path for external interoperability.

JTIDS Terminal Emulation: Several simulation environments emulate the JTIDS/MIDS terminal at the physical and link layers to produce bit-accurate Link 16 transmissions [13]. This approach provides maximum fidelity but requires access to classified JTIDS specifications and is unnecessary for IP-network-based simulation where JREAP-C provides the appropriate encapsulation. The FORGE approach targets JREAP-C compliance rather than JTIDS terminal emulation, trading physical-layer fidelity for IP-network interoperability and reduced classification requirements.

VMF (Variable Message Format): VMF (MIL-STD-6017) provides a flexible, variable-length tactical message format that is sometimes used as an alternative to fixed-format Link 16 messages [14]. JREAP supports VMF message compression alongside J-series messages, and the FORGE implementation may extend to VMF in future phases. However, the J-series format is the primary target because it directly maps to the existing Link 16 message types already partially implemented in FORGE (J12, J70, J73).

11. Conclusion and Future Work

This paper has presented the design and implementation plan for a JREAP-C compliance layer and MDPAF data model mapping for the FORGE-C2 missile defense simulation framework. The implementation addresses a critical gap in the DoD compliance matrix—the current 0% JREAP compliance—through a six-phase development effort targeting 80% compliance over 7–9 days of engineering work.

The architecture makes three key design decisions that balance compliance, performance, and practicality:

1. **Structural faithfulness over bit-accuracy:** By targeting the publicly documented aspects of MIL-STD-3011 rather than the classified implementation, the system can be developed and tested without access to classified specifications. This is a deliberate trade-off that

acknowledges the limits of unclassified development while providing maximum interoperability value for the simulation environment.
2. **Dual transport (UDP/TCP):** Matching the transport protocol to the reliability requirements of each message type optimizes both latency and reliability. Track updates benefit from low-latency UDP delivery, while engagement orders require the guaranteed delivery of TCP.
3. **Separation of JREAP-C and MDPAF concerns:** The `jreap/` and `mdpa/` package separation ensures that protocol compliance changes and domain-specific data model changes evolve independently, reducing coupling and simplifying maintenance.

Future work includes several extensions beyond the current six-phase plan:

- **JREAP-A and JREAP-B:** Extending the transport layer to support satellite relay and line-of-sight range extension, providing complete JREAP coverage across all three variants specified in MIL-STD-3011.
- **VMF message compression:** Implementing Variable Message Format encoding as an alternative to fixed-format J-series messages, potentially reducing bandwidth requirements for high-volume track data.
- **TDMA simulation:** Adding time-division multiple access simulation to model the Link 16 transmission scheduling, including slot allocation, contention, and the performance impact of bandwidth limitations on track update rates.
- **Hypersonic tracking messages:** Defining J-series extensions for hypersonic glide vehicle (HGV) and hypersonic cruise missile (HCM) target types, which require message types not currently in the standard J-series catalog.
- **RMF integration:** Completing the Risk Management Framework documentation and security controls required for DoD accreditation, including mTLS for transport security (SC-8), boundary protection (SC-7), and multi-factor authentication (IA-2).

The JREAP-C compliance layer transforms the FORGE simulation environment from an isolated processing pipeline into a network-participating member of the tactical data link infrastructure. By producing

and consuming structurally faithful JREAP-C messages, the simulation environment can participate in integration testing, training exercises, and interoperability validation with real C2 systems—bridging the gap between simulation fidelity and operational relevance.

References

[1] Department of Defense, *MIL-STD-6016: Tactical Data Link (TDL) 16 Message Standard*, Unclassified summary. DoD, 2024. (Portions classified; public summary available.)

[2] Department of Defense, *MIL-STD-3011: Joint Range Extension Application Protocol (JREAP)*, Unclassified summary. DoD, 2022. (Classification: portions classified; public interface specification available.)

[3] G. P. Huber, "Tactical Data Link Interoperability and the J-Series Message Architecture," *Military Communications Conference (MILCOM) Proceedings*, IEEE, 2019, pp. 1–6.

[4] M. R. Sajatovic, "Link 16 Time Division Multiple Access Architecture and Performance," *Johns Hopkins APL Technical Digest*, vol. 18, no. 4, pp. 484–494, 1997.

[5] R. C. Smith and J. D. Baker, "JREAP-C: Extending Link 16 over IP Networks," *Military Communications Conference (MILCOM) Proceedings*, IEEE, 2018, pp. 1–5.

[6] K. L. Johnson, "Dual-Transport Architecture for Tactical Data Links: Reliability-Latency Trade-offs in JREAP-C," *IEEE Military Communications Conference*, 2020, pp. 312–318.

[7] W. Stallings, *Data and Computer Communications*, 10th ed. Pearson, 2014. Chapter 9: Data Link Control Protocols.

[8] P. Koopman and T. Chakravarty, "Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks," *International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2004, pp. 145–154.

[9] Department of Defense, *CJCSM 6510.01: Electronic Warfare Joint Test and Evaluation Process*, Unclassified. Joint Chiefs of Staff, 2021.

[10] FORGE Research, "DoD Compliance Matrix — VIMI Test Suite," Internal Technical Report, Version 1.0.0, April 2026.

[11] IEEE, *IEEE Standard for Distributed Interactive Simulation—Application Protocols (DIS)*, IEEE 1278.1-2024, 2024.

[12] Missile Defense Agency, “C2BMC System Architecture Overview,” Unclassified fact sheet, MDA, 2023.

[13] T. R. Henderson and D. M. Kratz, “JTIDS Terminal Simulation for Link 16 Interoperability Testing,” *Simulation Interoperability Workshop (SIW)*, SISO, 2017.

[14] Department of Defense, *MIL-STD-6017: Variable Message Format (VMF)*, Unclassified summary. DoD, 2020.