Skynet/Link32: A Tactical Communication Protocol and Reference Implementation

Namdak Tonpa

June 15, 2025

Abstract

Link32 is a tactical communication protocol designed for low-latency, secure, and scalable data exchange in contested environments, supporting swarm coordination, real-time position location information (PLI), command and control (C2), and tactical chat over UDP-based multicast networks. We present its design and reference implementation.

Contents

1 Introduction

Link32 is a tactical communication protocol inspired by standards such as VMF, LINK16, TSM, SRW, and MQTT, tailored for military applications requiring robust, low-latency, and secure data exchange in contested environments. It facilitates swarm coordination, real-time position location information (PLI), command and control (C2), and tactical chat using UDP-based multicast networks. Skynet is the reference server implementation of Link32, developed in C99 with minimal dependencies to ensure portability and performance on resource-constrained devices.

1.1 Properties

Link32 and Skynet are designed with the following properties:

- **Implementation**: Written in C99 for portability and performance.
- **Message Size**: Minimum 32-byte header (48 bytes with AES-256-GCM tag) to optimize bandwidth.
- **Security**: ECDH key exchange over secp384r1, AES-256-GCM encryption for all messages.
- **Latency**: Microsecond-precision timing using monotonic clocks and non-blocking I/O.
- **Concurrency**: Lock-free atomic operations (CMPXCHG) for thread-safe queue management.
- **Networking**: UDP multicast with topic-based subscriptions.
- **Footprint**: $\sim$64KB L1 cache usage, $\sim$2000 lines of code (LOC).
- **Dependencies**: Single dependency on OpenSSL for cryptography.
- **Threat Model**: Prioritizes confidentiality and integrity, no non-repudiation.

1.2 Principles

Link32 adheres to the following design principles:

- **Key Provisioning**: Manual public key distribution for controlled setup.
- **Mandatory Encryption**: All messages encrypted with AES-256-GCM.
- **Node Identification**: Node names hashed to 32-bit using FNV-1a.
- **Lock-Free Design**: Uses atomic compare-and-swap (CMPXCHG) for concurrency.
- **Topic Architecture**: Topics map to IP multicast groups for publish-subscribe.

- **Queue Management**: Global network queue with per-topic subscriber queues.
- **Key Storage**: Separate key stores per executable.

2 Link32 Protocol

2.1 S-Message Format

The Link32 message structure, `SkyNetMessage`, is compact for large-scale swarm communication:

```
typedef struct {
    uint8_t version : 4;    // Protocol version (current: 1)
    uint8_t type : 4;       // Message type (0-6)
    uint8_t qos : 4;        // Quality of Service (0-3)
    uint8_t hop_count : 4; // Hop count for routing (0-15)
    uint32_t npg_id;        // Topic identifier (1-103)
    uint32_t node_id;       // Sender node ID (FNV-1a hash)
    uint32_t seq_no;        // Sequence number for deduplication
    uint8_t iv[16];         // AES-256-GCM initialization vector
    uint16_t payload_len;  // Payload length (0-32767)
    uint8_t payload[MAX_BUFFER]; // Encrypted payload + 16-byte GCM tag
} SkyNetMessage;
```

- **Header Size**: 32 bytes.
- **Total Size**: 48 bytes minimum (32-byte header + 16-byte GCM tag).
- **Payload**: Up to 32720 bytes.

2.2 Message Types

The protocol defines seven message types, as shown in Table **??**:

Table 1: Link32 Message Types

ID	Type	Description
0	Key Exchange	Exchanges ECC public keys for ECDH session setup.
1	Slot Request	Requests a TDMA slot from the server.
2	Chat	Sends tactical chat messages.
3	Ack	Acknowledges slot assignments or control messages.
4	Waypoint	Specifies navigation waypoints for C2.
5	Status	Reports position, velocity, or sensor data (e.g., PLI).
6	Formation	Coordinates swarm formations.

2.3 Multicast Topics

Link32 uses multicast topics mapped to IP multicast groups, as listed in Table **??**:

Table 2: Multicast Topics

NPG	Name	Multicast	Purpose
1	npg_control	239.255.0.1	Network control (key exchange, slot requests).
6	npg_pli	239.255.0.6	Position information (status messages).
7	npg_surveillance	239.255.0.7	Sensor data forwarding.
29	npg_chat	239.255.0.29	Tactical chat and acknowledgments.
100	npg_c2	239.255.0.100	Command and control (waypoints, formations).
101	npg_alerts	239.255.0.101	Network alerts and self-healing.
102	npg_logistics	239.255.0.102	Logistical coordination (status, chat).
103	npg_coord	239.255.0.103	Inter-agent coordination (chat, waypoints).

2.4 Slot Management

Link32 employs a Time Division Multiple Access (TDMA)-like slot manager to minimize collisions:

- **Slot Array**: Fixed-size array (`slots[SLOT_COUNT=256]`) in `ServerState`.
- **Dynamic Topics**: Each slot creates a temporary multicast group (239.255.1.¡slot_id % 256¿).
- **Allocation**: First-come, first-serve with no timeouts.
- **Timing**: Slots cycle every `TIME_SLOT_INTERVAL_US=1000`μ`s`.

Clients send `SKYNET_MSG_SLOT_REQUEST` to NPG 1, and the server assigns slots via `SKYNET_MSG_ACK`.

2.5 Deduplication

A fixed-size circular buffer (`seq_cache`) prevents message loops:

- **Structure**: Stores {`node_id, seq_no, timestamp`}.
- **Memory**: ~16KB (1024 $\times$ 16 bytes).
- **Complexity**: O(1) lookup using FNV-1a hashing.
- **Threshold**: Discards duplicates within 1 second.

2.6 Security

Security mechanisms include:

- **Key Exchange**: ECDH over secp384r1 for 256-bit AES keys.
- **Encryption**: AES-256-GCM with 16-byte IV and 16-byte tag.
- **Key Storage**: Server: `~/.skynet/ecc/secp384r1/<node_hash>.`{`ec_priv,ec_pub`}; Client: `~/.skynet_client/ecc/secp384r1/<node_hash>.`{`ec_priv,ec_pub`}.

- **Key Derivation**: HKDF-SHA256 for AES keys.
- **Self-Sent Handling**: Skips messages where `msg->node_id == state->node_id`.

2.7 Subscriptions

Nodes subscribe to topics based on roles, as shown in Table **??**:

Table 3: Role-Based Subscriptions

Role	NPGs	Purpose
Infantry	1, 29	Network control and tactical chat.
Drone	1, 6, 7, 100, 101	Control, PLI, surveillance, C2, alerts.
Air	1, 6, 7, 100, 101, 103	Control, PLI, surveillance, C2, alerts, coordination.
Sea	1, 7, 29, 102, 103	Control, surveillance, chat, logistics, coordination.
Ground	1, 7, 29, 102	Control, surveillance, chat, logistics.
Relay	1, 6, 101	Control, PLI, alerts for relaying.
Controller	1, 6, 100, 101	Control, PLI, C2, alerts for command posts.

3 Skynet Implementation

3.1 Dependencies

- **OpenSSL**: For ECC, ECDH, and AES-256-GCM.
- **C99 Compiler**: GCC or equivalent.
- **POSIX Environment**: For threading, epoll, and timerfd.

3.2 Build

To build Skynet:

```
$ git clone git@github.com:BitEdits/skynet
$ cd skynet
$ gcc -o skynet_client skynet_client.c skynet_proto.c -lcrypto
$ gcc -o skynet skynet.c skynet_proto.c -pthread -lcrypto
```

3.3 Installation

The provisioning script `skynet.sh` generates ECC key pairs:

```
# ./skynet.sh
Generated keys for node npg_control (hash: 06c5bc52) in /secp/
Generated keys for node npg_pli (hash: c9aef284) in /secp/
Generated keys for node npg_surveillance (hash: 4d128cdc) in /secp/
Generated keys for node npg_chat (hash: 9c69a767) in /secp/
Generated keys for node npg_c2 (hash: 89f28794) in cc/secp/
Generated keys for node npg_alerts (hash: 9f456bca) in /secp/
Generated keys for node npg_logistics (hash: 542105cc) in /secp/
Generated keys for node npg_coord (hash: e46c0c22) in /secp/
Generated keys for node server (hash: 40ac3dd2) in /secp/
Generated keys for node client (hash: 8f929c1e) in /client/secp/
# cp /secp/*.ec_pub /client/secp/
```

3.4 Server Operation

The server binds to UDP port 6566, joins multicast groups, and processes messages using a global queue. Example output:

```
# skynet server
Node 40ac3dd2 bound to 0.0.0.0:6566.
Joined multicast group 239.255.0.1 (NPG 1: control).
Joined multicast group 239.255.0.6 (NPG 6: PLI).
Message received, from=8f929c1e, to=1, size=231.
Decryption successful, from=8f929c1e, to=1, size=215.
Saved public key for client 8f929c1e.
Assigned slot 0 to node 8f929c1e.
Message received, from=8f929c1e, to=6, size=40.
Decryption successful, from=8f929c1e, to=6, size=24.
Message sent from=8f929c1e, to=6, seq=3, multicast=239.255.1.0, latency=36643.
```

3.5 Client Operation

The client joins topic-specific multicast groups and sends key exchange, slot requests, and status messages. Example output:

```
# skynet_client client
Node 8f929c1e connecting to port 6566.
Joined multicast group 239.255.0.1 (NPG 1).
Joined multicast group 239.255.0.6 (NPG 6).
Sent key exchange message to server.
Sent slot request message to server.
Received slot assignment: slot=0.
Joined slot multicast group 239.255.1.0.
Sent status message: multicast=239.255.1.0,
                     pos=[0.1, 0.1, 0.1],
                     vel=[0.0, 0.0, 0.0],
                     seq=2.
```

3.6 Usage

Skynet includes five utilities:

Keys Provisioning

Generates ECC key pairs.

```
skynet_keygen <node> [--server|--client]
```

Message Encryption

Encrypts a test message to `<npg_id>.sky`.

```
skynet_encrypt <sender> <recipient> <file>
```

Message Decryption

Decrypts `<file.sky>`.

```
skynet_decrypt <sender> <recipient> <file.sky>
```

Skynet Server

Runs the server with FNV-1a hashed `<node>`.

```
skynet <node>
```

Skynet Client

Runs the client with FNV-1a hashed `<node>`.

```
skynet_client <node>
```

3.7 Limitations

- **Slot Scalability**: Fixed `SLOT_COUNT=256` limits nodes to 256.
- **No Retransmission**: Dropped messages are not retransmitted.
- **Key Management**: Manual public key copying required.
- **Deduplication**: `SEQ_CACHE_SIZE=1024` may cause collisions.

4 Conclusion

Link32 and Skynet provide a robust framework for tactical communication, combining low-latency, security, and scalability. Future improvements could address slot scalability, automated key distribution, and enhanced deduplication to support larger networks.

References

[1] U.S. Department of Defense, “MIL-STD-6016: Tactical Data Link (TDL) J Message Standard,” 2008.

[2] U.S. Department of Defense, “Link 16 Network Management and Operations,” ADA404334, 2003.