



Protokoll

Deutsche Version. Im Zweifelsfall ist der englische Text verbindlich.

Show English version

Automatische Stationen an den ARDF-Senderanlagen (Füchse)

Ein Fuchs sendet regelmäßig seine Kennung und Telemetriedaten aus. Dies geschieht, wenn möglich, zu Zeiten, in denen der Fuchs nicht auf den Bändern 80m oder 2m sendet. Dies dient einerseits der Zeitsynchronisation benachbarter Knoten und andererseits der regelmäßigen offiziellen Nennung des Rufzeichens. Meist wird das Rufzeichen des Organisators oder Ausrichters des Peilwettbewerbs verwendet.

Beacon (Bake)

Aufgebaut auf Basis der modifizierten AX.25-, APRS- und MeshCom-Protokolle wie folgt:

Pseudo-Byte-Stream:

```
{0E3TKT-03CpnnnnxRSqxRSqxRSq. .
```

Struktur

Byte / Offset	Feld / Format	Beschreibung
Byte 0	"{" 0x7B	Data Type Identifier (DTI): Abgeleitet aus der APRS- und MeshCom-Protokollspezifikation (siehe APRS-Protokollreferenz). Begründung: Aktuell in der Spezifikation als [Unused] markiert; Zeichen ähnelt visuell einem Stationssymbol.
Bytes 1–6	"0E3TKT" (6 Bytes ASCII)	Internationales Rufzeichen: Formatiert als 1–2 Zeichen Präfix, gefolgt von einer oder mehreren Ziffern und einem 1–3 Zeichen Suffix. Strikt 6 Zeichen breit; bei kürzeren Rufen rechts mit Leerzeichen aufgefüllt.
Byte 7	"-" 0x2D oder "~" 0x7E	Knotentyp-Trennzeichen: Bindestrich für Standardknoten, Tilde für Backup- / Hot-Standby-Knoten. Dient als visueller Begrenzer und struktureller Anker für den Parser.
Bytes 8–9	"03", oder "S6" (2 Bytes ASCII)	Stations-SSID-Kennung: Teilstring-Format, das die Knoten-ID darstellt. Unterstützt numerische IDs (0–99) oder erweiterte Schreibweisen (z. B. "S6"), d. h. alle Zeichen wie in aprs11/SSIDs.txt beschrieben (z. B. 0E3TKT-S6 für eine spezielle 6. Station).
Byte 10	Eigener Batteriestand (unsigned char)	Wert: Kapazität in Prozent (0 bis 100 %). Werte über 100 weisen auf Ladevorgang oder Netzbetrieb hin.
Byte 11	Protokoll- / Netz-Flags (unsigned char)	Netzwerk-Flags / Padding: Reserviert für Mesh-Netzwerk-Flags und Hop-Steuerung. Gewährleistet die präzise 4-Byte-Speicherausrichtung (Alignment) für den nachfolgenden 32-Bit-Zeitstempel.
Bytes 12–15	nnnn / Zeitstempel (int32_t, Little-Endian)	Zeitstempel: Vorzeichenbehafteter 32-Bit-Integer, der die Anzahl der Millisekunden seit Beginn des Wettbewerbstages darstellt. Formatiert als Little-Endian (LSB zuerst) und auf eine

		4-Byte-Grenze ausgerichtet für optimierten, nativen ESP32-Zeigerzugriff. Alle offiziellen ARDF-Wettbewerbszeiten und -Logs müssen in koordinierter Weltzeit (UTC) geführt werden. Um <i>[Lokaler Zeitzonennamen]</i> in UTC umzurechnen, subtrahiere/addiere <i>[X]</i> Stunden ($UTC = [Lokale\ Zeit] \pm [X]h$). Bitte achte genau auf Datumswechsel bei Ereignissen, die UTC-Mitternacht überschreiten.
Bytes 16+	Gehörte Nachbarknoten (xRSqxRSqxRSq. . .)	Multi-Byte-Blöcke, die Details zu Mesh-Nachbarknoten enthalten (SSID, Verbindungsqualität, Batteriestatus). Siehe detaillierte Aufschlüsselung unten.

Aufschlüsselung des Blocks der gehörten Nachbarknoten (Bytes 16+)

Byte-Offset	Feld / Format	Beschreibung
Bytes 16–17	Nachbar-SSID (2 Bytes ASCII)	Zwei ASCII-Zeichen, welche die SSID des Nachbarknotens darstellen (z. B. "03" oder "56"). Dies entspricht dem exakten Teilstring-Format aus den Bytes 8–9 des Quellknotens für einfachen String-Abgleich. Padding: Einstellige Stationen sollten links mit einem Null-Byte (0x00) aufgefüllt werden (z. B. 0x00 0x33 für Station 3). Parsing-Regeln: Die Firmware-Implementierung muss flexibel sein. Der Parser muss führende Null-Bytes (0x00), führende Leerzeichen (0x20) und führende ASCII-Nullen (0x30) unterstützen und fehlerfrei dekodieren.
Byte 18	RSSI-Wert (R) (unsigned char)	Gespeichert als $RSSI + 164$ (z. B. 0x3A = 58 $\rightarrow$ 58 - 164 = -106 dBm).
Byte 19	SNR-Wert (S) (signed char)	Gespeichert als SNR multipliziert mit 4 (z. B. 0x42 = 66 $\rightarrow$ 66 / 4 = +16,5 dB).
Byte 20	Status-Byte (q) (unsigned char)	Kombiniertes Byte für Qualität und Batteriestatus. (Siehe Bit-Mapping unten).

Bit-Mapping für Byte 20 (q):

Bits	Wert	Definition
Bits 7–6	00	Normal (Battery OK / Powered)
Bits 7–6	01	Low (Battery low, intervention recommended)
Bits 7–6	10	Critical (Battery critical, node might shut down soon)
Bits 7–6	11	Reserved
Bits 5–0	0 to 63	Moving average counter of how many times this station was heard within the last 30 minutes. If ≥ 3 , the link is considered stable. Dropped if 0 within 30 minutes.

Kontrollpunkt-Nachrichten (Posten / Checkpoints)

Sobald sich ein Teilnehmer an einem Kontrollpunkt mittels seines RFID-Tags oder seiner Kennung registriert, wird er an diesem Knoten erfasst. Dem Läufer wird ein Zeitstempel und seine ID (Startnummer) zugewiesen. Die Zuordnung zwischen Startnummer und RFID-Code ist systemintern hinterlegt. Während des nächsten verfügba-

ren Zeitschlitzes wird eine Kontrollpunkt-Nachricht mit QoS 2 (garantierte Paketzustellung, Duplikaterkennung und -filterung) an das Ziel übertragen.

Nachrichten werden gleichzeitig an den Zielknoten und optional an den Startknoten gesendet. Beide Knoten müssen eine SSID < 15 besitzen, da die verbleibenden Bits im Adress-Byte für Protokoll-Flags reserviert sind.

Dieses Nachrichtenpaket ist extrem kurz und effizient konzipiert. Es enthält die Knotennummer, die Startnummer, den Erfassungs-Zeitstempel und den aktuellen Hop-Count.

Pseudo-Byte-Stream:

(ppMzzzzLLR

Structure (Checkpoint Message)

Byte / Offset	Feld / Format	Beschreibung
Byte 0	(0x28)	Data Type Identifier (DTI): Spezielle Kennung, die für FoxMash-Kontrollpunkt-Nachrichten vorgesehen ist.
Bytes 1–2	Eigene SSID (pp) (uint16_t)	16-Bit unsigned Integer des sendenden Kontrollpunkt-Knotens.
Byte 3	Routing & Flags (M) (unsigned char)	Kombinierte Ziel-SSID und Protokoll-Steuerungs-Flags. Dient gleichzeitig als 1-Byte-Grenzausrichter (Alignment).
Bytes 4–7	zzzz (Zeitstempel) (int32_t, Little-Endian)	Zeitstempel: Vorzeichenbehafteter 32-Bit-Integer, der die Anzahl der Millisekunden seit Beginn des Wettbewerbstages darstellt. Formatiert als Little-Endian (LSB zuerst) und auf eine 4-Byte-Grenze ausgerichtet für direkten, hochperformanten ESP32-Zeigerzugriff.
Bytes 8–9	Startnummer (L) (uint16_t)	Eindeutige Teilnehmeridentifikation / Startnummer (Unterstützt IDs von 0 bis 65.535).
Bytes 10	Hop-Count / Flags (unsigned char)	Aktueller Netzwerk-Hop-Count und verbleibende Mesh-Routing-Flags.

Bit-Mapping für Byte 3 (M):

Bits	Flag / Feld	Definition
Bits 7–5	Hop-Count	Dynamischer Routing-Zähler. 0, wenn das Ziel direkt mit guter Qualität sichtbar ist. Wird um 1 erhöht, wenn innerhalb von 5 Minuten kein ACK erfolgt. Wird verworfen, wenn der maximale Hop-Wert überschritten wird.
Bit 4	ACK-Flag	0 für eine Standard-Datennachricht, 1 für ein Bestätigungspaket (ACK).
Bits 3–0	Ziel-SSID	Die SSID des Zielknotens (Unterstützt IDs von 0 bis 15).

Bestätigungen (Acknowledgments) und Wiederholungen

Die maximale Anzahl der Sendeveruche ist konfigurierbar (Standard: 4). Jedes Kontrollpunkt-Paket muss quittiert werden. Die Bestätigung verwendet exakt dasselbe Format (vppMLzzzzRR), jedoch mit **auf 1 gesetztem Bit 4 in Byte 3 (M)**. Das Rückrouting erfolgt auf dem identischen Pfad in umgekehrter Richtung basierend auf der im Paket aufgezeichneten Route (Source-Routing). Weiterleitende Knoten erkennen und verworfen Duplikate durch Abgleich des eindeutigen Zeitstempels des Läufers.

Protokoll-Optimierungsanalyse

Hardware-optimiertes Protokolldesign für ESP32 und LoRa

Beim Entwurf von stromsparenden, hochperformanten Mesh-Netzwerken mit ESP32-Mikrocontrollern über LoRa-Modulation beeinflusst das Layout der Datenstrukturen sowohl die Übertragungszeit (Airtime) als auch die CPU-Effizienz maßgeblich. Das aktualisierte FoxMash-Protokolllayout strukturiert die Felder so um, dass ein optimales Gleichgewicht zwischen HF-Effizienz und Hardware-Ausführung erreicht wird.

Warum dieses Layout perfekt ist

1. **Native Wortausrichtung (Keine Performance-Verluste):** Die in ESP32-Chips verwendeten Tensilica Xtensa- und RISC-V-Architekturen sind von Natur aus 32-Bit Little-Endian-Prozessoren. Der Zugriff auf einen 4-Byte-Integer von einer Speicheradresse, die kein Vielfaches von 4 ist (*unaligned access*), zwingt die Hardware zu speicherintensiven Korrekturen durch internes Byte-Shifting. In ungünstigen Fällen oder spezifischen Compiler-Konfigurationen führt dies zu einem Hardware-Absturz (LoadStoreAlignmentCause Core Dump). Durch Platzierung des 32-Bit-Zeitstempels an exakt **Byte 4** (Kontrollpunkt) und **Byte 12** (Beacon) fügt sich der Integer perfekt in die natürlichen Wortgrenzen des Prozessors ein.
2. **Zero-Copy Zeiger-Casting:** Da die Felder nativ ausgerichtet sind, muss die Firmware keine Taktzyklen für `memcpy()` oder Bit-Shifting-Operatoren (`<<`, `|`) verschwenden, um Integer-Werte aus einzelnen Bytes zu rekonstruieren. Die Anwendungsschicht kann direkt aus dem LoRa-Hardware-Empfangspuffer ein Zero-Copy-Pointer-Casting durchführen.
3. **Null Airtime-Overhead:** Traditionelle Alignment-Techniken fügen künstliche Füllbytes (*Padding*, `0x00`) ein, um die Ausrichtung zu erheben, was das Paket vergrößert. In LoRa-Netzwerken erhöht jedes zusätzliche Byte direkt die Time-on-Air (ToA), was zu höherem Energieverbrauch und einer größeren Wahrscheinlichkeit von Paketkollisionen führt. Dieses optimierte Layout erreicht die perfekte Ausrichtung rein durch die **Umordnung bestehender, essenzieller Felder**, wodurch die Frame-Größe minimal bleibt.
4. **Menschlich lesbare Trennzeichen beibehalten:** Das Beibehalten des Standard-Bindestrichs (" - ") oder der Tilde (" ~ ") als Trennzeichen zwischen dem internationalen Rufzeichen und der SSID wahrt die Kompatibilität mit bestehenden APRS-/MeshCom-Parsern und hält rohe Hex-Logs beim seriellen Debuggen intuitiv lesbar.

Hochperformantes C++ Implementierungsbeispiel

Der folgende Code-Ausschnitt zeigt, wie dieses speicheroptimierte Layout in einer ESP32-Anwendung unter Verwendung von Standard-Arduino- oder ESP-IDF-LoRa-Bibliotheken genutzt werden kann.

```

#include <Arduino.h>

// Sicherstellen, dass der Compiler kein strukturelles Padding in den Structs hinzufügt
#pragma pack(push, 1)

struct BeaconPacket {
    char dti; // Offset 0 (0x7B '{')
    char callsign[6]; // Offset 1 (z. B. "OE3TKT")
    char separator; // Offset 7 (z. B. '-')
    char ssid[2]; // Offset 8 (z. B. "03")
    uint8_t battery; // Offset 10 (0-100%)
    uint8_t net_flags; // Offset 11 (Routing-Steuerung / Padding)
    int32_t timestamp_ms; // Offset 12 -> PERFEKT AUF 4-BYTE AUSGERICHTET!
    uint8_t neighbor_block[]; // Offset 16+ (Dynamische Mesh-Nutzlast)
};

struct CheckpointPacket {
    char dti; // Offset 0 (0x28 '(')
    uint16_t own_ssid; // Offset 1
    uint8_t routing_flags; // Offset 3 (Ziel- / Steuerungs-Flags)
    int32_t timestamp_ms; // Offset 4 -> PERFEKT AUF 4-BYTE AUSGERICHTET!
    uint16_t runner_id; // Offset 8
    uint8_t hop_count; // Offset 10
};

#pragma pack(pop)

// Beispiel-Parserfunktion zur Verarbeitung eingehender roher LoRa-Bytes
void processIncomingLoRa(uint8_t* rxBuffer, size_t packetLength) {
    if (packetLength < 1) return;

    // Data Type Identifier (DTI) prüfen
    char dti = rxBuffer[0];

    if (dti == '(') {
        // Zero-copy Direktzugriff über Zeiger-Cast
        CheckpointPacket* cp = (CheckpointPacket*)rxBuffer;

        // Schneller nativer Zugriff direkt aus dem Speicher
        int32_t time = cp->timestamp_ms;
        uint16_t runner = cp->runner_id;

        Serial.printf("Läufer %d erfasst. Lokaler Zeitstempel: %d ms\n", runner, time);
    }
    else if (dti == '{') {
        BeaconPacket* beacon = (BeaconPacket*)rxBuffer;

        // Schneller nativer Zugriff direkt aus dem Speicher
        int32_t time = beacon->timestamp_ms;

        Serial.printf("Beacon empfangen. Netzwerk-Zeitstempel: %d ms\n", time);
    }
}

```

Dokument gepflegt und kompiliert von **[OE3TKT, Tom]**

Wunderschön gesetzt mit Typst ❤