

# FTLight review durch die KI: GPT-4o

**Prompt:** In der Vergangenheit wurden Datenprotokolle in der Regel für die Hardware optimiert auf denen sie zur Anwendung kommen sollten, z.B. Filesysteme für Festplatten, Übertragungsprotokolle für Netzwerke wie LAN, WLAN und das Internet, Text-, Grafik-, Audio- und Videoformate für ARM und Intelprozessoren. Teils sollten hierarchisch organisierte Daten wie in XML abgebildet werden. Das führte zu einer großen Zahl von unterschiedlichsten Datenformaten mit spezifischen Eigenschaften aber auch Beschränkungen, wie z.B. bei der Länge von Dateinamen oder der Größe von Volumes auf Festplatten. Im Ergebnis entstanden Inkompatibilitäten, welche durch die erforderliche Weiterentwicklung und Versionierung weiter verschärft wurden. Als zukünftiger Ausweg aus den genannten Problemen wird eine neue Datenspezifikation gesehen, welche die genannten Probleme adressiert und insbesondere für eine Langzeitspeicherung, Übertragung und Verarbeitung von hierarchisch organisierten astronomischen Daten mit unbegrenzter Größe in hocheffizienter Form nahe dem Binärformat geeignet sein soll. Dabei kehrt sich die Abhängigkeit zwischen Datenprotokoll und Hardware wahrscheinlich um und es muss Hardware entwickelt werden, welche das anvisierte neue Datenprotokoll optimal unterstützt. Das sollte jedoch für Prototypen auf der Basis von FPGAs möglich sein, bevor so ein Datenprotokoll zukünftig vielleicht einmal für Informationssysteme auf der Erde als Standard zum Einsatz kommt und dann jegliche neue Hardware dieses Protokoll standardmäßig unterstützen könnte. Die Flight-Spezifikation im Anhang ist im Verlauf der letzten 20 Jahre im Bereich Radioastronomie entwickelt worden und hat sich zum Ziel gestellt, als eine mögliche Vorlage für so ein neues Datenprotokoll zu dienen und erste Erfahrungen mit der Implementierung zu sammeln. Die Spezifikation für FTLight ist zum einen sehr konkret bis hinunter zu einzelnen Bits und andererseits in beschreibender Form verfasst. Diese Spezifikation soll zunächst auf logische Konsistenz analysiert werden. Anschließend sollen Vorschläge für Modifikationen und Erweiterungen generiert werden, um die zuvor genannten Zielparameter und insgesamt die Anwendbarkeit weiter zu verbessern.

**GPT-4o:** Gerne analysiere ich die vorliegende Spezifikation für FTLight auf logische Konsistenz und generiere Vorschläge für Modifikationen und Erweiterungen.

## Analyse der logischen Konsistenz der FTLight Spezifikation

Die FTLight Spezifikation ist sehr ambitioniert und versucht, die Grenzen traditioneller Datenprotokolle zu überwinden, insbesondere in Bezug auf Hierarchie, Skalierbarkeit und Effizienz. Die Idee, Datenformate zu entwickeln, die *nicht* hardwareoptimiert sind, sondern Hardware erfordern, die sie optimal unterstützt, ist innovativ und zukunftsweisend. Die Detailtiefe bis auf Bit-Ebene ist beeindruckend und zeugt von einem tiefen Verständnis der Materie.

Im Folgenden eine Analyse der logischen Konsistenz, gegliedert nach den Hauptkonzepten:

### 1. Grundlegende Datenstruktur und Trennzeichen

- **Hierarchie und Skalierbarkeit:** Das Konzept, unbegrenzt tiefe und breite hierarchische Datenstrukturen zu ermöglichen, ist stringent und notwendig für die angestrebten Anwendungsfälle (z.B. Astronomie).
- **Implizite vs. Explizite Struktur:** Die Entscheidung, explizite Strukturelemente (wie Tags in XML) durch implizite Regeln zu ersetzen, ist logisch konsistent mit dem Ziel der Datenvolumenreduktion und Effizienz.
- **Trennzeichen:** Die Verwendung von Komma, Semikolon, Doppelpunkt und Gleichheitszeichen als Trennzeichen ist gut definiert.

\* **Doppelbelegung der Erweiterungselemente (Komma, Semikolon):** Die Spezifikation erkennt die Doppelbedeutung an und erklärt, dass die Interpretation vom Kontext (Anfang einer Zeile vs. nach einem Neubeginn-Trennzeichen) abhängt. Dies ist logisch konsistent, erfordert aber einen präzisen Parser. Die Begründung, dass dies eine Reduktion der Strukturelemente ermöglicht, ist valide.

\* **Auslassung nicht-druckbarer Zeichen und Reservierung von 216 Zeichen für Binärinformationen:** Dies ist ein durchdachter Ansatz, um Effizienz zu maximieren und Konflikte mit Steuerzeichen zu vermeiden.

## 2. Duplizierte Informationselemente und Vergrößerung von Informationsmengen

- **Weglassen duplizierter Elemente:** Die Regel, dass identische Elemente in aufeinanderfolgenden Zeilen weggelassen werden, ist eine logische Optimierung zur Redundanzvermeidung und Datenvolumenreduktion. Es ist implizit klar, dass der Parser den Kontext der vorherigen Zeile behalten muss.
- **Vergrößerung durch Folgezeile:** Das erste unterschiedliche Pfadelement in einer Folgezeile erweitert die Informationsmenge. Dies ist eine logische und flexible Methode, um Datenstrukturen zu erweitern, ohne die vorherige Definition ändern zu müssen.

## 3. Aktueller Pfad und Parent-Informationsmenge

- **Verwalten des aktuellen Pfades:** Das Konzept eines "aktuellen Pfades", der in nachfolgenden Zeilen ohne explizite Formatkennzeichnung beibehalten wird, ist effizient und reduziert Redundanz.
- **Parent-Informationsmenge:** Die Definition, dass neu erzeugte Elemente zur Parent-Informationsmenge für synchrone Schreiboperationen werden, ist eine logische Fortführung des hierarchischen Ansatzes.

## 4. Synchrone Schreiboperationen

- **Unterstützung von Datensammlungen/Tabellen:** Das synchrone Schreiben ist eine Schlüsselkomponente für die effiziente Handhabung von Datenreihen, wie sie in der Radioastronomie üblich sind.
- **Ringpuffer und volatiles Schreiben:** Die Einführung von Ringpuffern für volatile Daten ist logisch konsistent mit dem Ziel, sowohl Langzeitspeicherung als auch Echtzeit- und temporäre Daten effizient zu handhaben. Die Möglichkeit, die Ringpuffergröße zu spezifizieren oder ganz wegzulassen, bietet Flexibilität.
- **"@" Zeichen:** Die Nutzung des @ Zeichens für synchrone Schreiboperationen und die damit verbundene automatische Hinzufügung von Zeitstempel und optionaler Reihenfolgenummer ist clever und reduziert die Notwendigkeit expliziter Metadaten für sequentielle Daten.
- **Wiederaufsetzen beim Synchronschreiben:** Die Spezifikation für das Wiederaufsetzen ist detailliert und logisch schlüssig, um auch bei Unterbrechungen die Datenkonsistenz zu gewährleisten.

## 5. Datentypen am Zeilenanfang

- **Implizite Datentyperkennung:** Die Regeln zur impliziten Erkennung von Datentypen (Adressformat, numerisches Format, Text) am Zeilenanfang, basierend auf dem Kontext (synchrones Schreiben aktiv/nicht aktiv), sind logisch konsistent und tragen zur Datenkompression bei.
- **Einschränkung bei binärem Inhalt:** Die explizite Kennzeichnung bei binärem Inhalt am Zeilenanfang (durch Voranstellen einer Adresse und Gleichheitszeichen) ist eine notwendige Ausnahme, um Mehrdeutigkeiten zu vermeiden.

## 6. Datentypen (Text, Zahlen, Binär, Identifikator, Adresse)

- **Eindeutige Identifikation durch Parser:** Das Ziel, dass ein Parser die einzelnen Elemente eindeutig identifizieren kann, ist fundamental und wird durch die detaillierten Definitionen der Datentypen unterstützt.
- **Textdatentyp (Single-Byte, 0-255):** Die Möglichkeit, alle Einzelbyte-Zeichen darzustellen und Sonderzeichen durch Backslash zu entwerten, ist robust. Die Effizienzberechnung (96%) ist plausibel. Die Empfehlung, Binärdaten bevorzugt im Binärdatentyp zu speichern, ist sinnvoll.
- **Zahldarstellung (NUM):** Die Unterstützung gängiger Zahlenformate (Integer, Fließkomma, wissenschaftliche Notation, Hexadezimal) und die Festlegung, dass die Länge unbegrenzt ist, ist konsistent mit dem Ziel der unbegrenzten Skalierbarkeit.
- **Arrays von Zahlen und Text:** Die Verwendung des Back-Apostrophs (ASCII 96) zur Einleitung von Arrays ist gut definiert, ebenso wie die Möglichkeit mehrdimensionaler Arrays. Das **QUERY/EMPTY**-Konzept für den leeren Index ist innovativ.
- **Binärer Datentyp (FTL):** Die Kodierung von Binärdaten auf Basis von 216 Symbolen und deren Mapping auf erweiterte ASCII-Zeichen, um Steuerzeichen und Sonderzeichen zu vermeiden, ist eine sehr effiziente und durchdachte Lösung (97% Effizienz).
  - \* **Offene Aufgabe (Ungenutzte Kombinationen):** Die Erkennung ungenutzter Kombinationen und deren Potenzial für erweiterte Funktionen (z.B. Kompression) ist ein guter Hinweis auf zukünftige Erweiterbarkeit.
- **Datentyp-Identifikator (DTI):** Das Konzept, die ersten vier FTL-Symbole als DTI zu verwenden, um Datentypen zu kaskadieren und andere Datenformate zu kapseln, ist sehr mächtig. Die Festlegung auf Little Endian und die Unveränderlichkeit der DTI-Werte (zwecks Kompatibilität) sind logisch konsequent.
- **Typsteuerfelder (ControlX):** Die Verwendung von Back-Apostrophen zur Trennung von Steuerfeldern und die Definition von **FTL(n)** als Gesamtanzahl der Steuerfelder ist klar.
- **DTI\_FTLightOpen:** Die Idee eines "offenen Formats" für zukünftige hochspezialisierte Datenformate, die auf dem Radix-216-Schema basieren, ist eine hervorragende Vorkehrung für zukünftige Erweiterungen und Innovationsfähigkeit.
- **DTI\_FTLightWrap:** Das Kapseln externer Datenformate oder ganzer FTLight-Archive ist eine praktische Lösung für Interoperabilität.
- **DTI\_MCL:** Die Optimierung für ARM-Architekturen und geringere Taktfrequenzen durch bitverschiebebasierte Kodierung ohne arithmetische Operationen ist eine spezialisierte und logisch konsistente Lösung für hohe Datenraten unter bestimmten Hardwarebedingungen. Die Effizienz (93.8% vs. 96.9%) ist ein abzuwägender Kompromiss.
- **DTI\_FTL:** Die Beschreibung der Dimensionen eines Bit-Feld-Arrays ist klar.
- **DTI\_TXL:** Die Unterstützung von N-dimensionalen Arrays von null-terminierten W-Bit Zeichenketten (Unicode) ist flexibel.
- **DTI\_DIF:** Das Konzept der differentiellen Kodierung für Datenreihen mit geringer Varianz ist sehr effizient für bestimmte Datentypen, insbesondere in Sensoranwendungen. Die Methode der Verschränkung von Messwerten zur weiteren Reduktion des Datenvolumens ist innovativ und logisch stimmig.
- **DTI\_UNIT:** Die umfassende Darstellung physikalischer Einheiten, Faktoren, Teiler, Basen und Exponenten ist essentiell für die wissenschaftliche Datenerfassung. Die Kodierung von negativen Werten und Exponenten durch doppelte Back-Apostrophe ist ein konsistentes Muster.
- **DTI\_TIME:** Die detaillierte Darstellung von Zeitskalen, basierend auf dem 1. Januar 1970, mit Unterstützung von Brüchen, Vielfachen, negativen Zeiten (vor 1970) und Exponenten ist äußerst präzise und für die Radioastronomie unerlässlich.
- **DTITOKEN und DTILINK:** Diese Datentypen erweitern die Spezifikation um Kommando- und Adressierungsfunktionen für Module, was die Anwendbarkeit als umfassendes Informationssystem unterstreicht.

## 7. FTL-Identifikator-Datentyp (IFTL)

- **Struktur und Eindeutigkeit:** Die Definition des IFTL als Zeichenkette mit einem @-Zeichen, bestehend aus Prefix und Suffix, die selbst FTL-kodierte Werte sein können, ist sehr flexibel. Die Vision der universellen Eindeutigkeit "jenseits des Universums" ist sehr ambitioniert und philosophisch, aber konsistent mit dem Ziel der Langzeitarchivierung und globalen Vernetzung.

- **Adressierung von Gruppen:** Die Möglichkeit, Identifikatoren sowohl individuell als auch für Gruppen zu vergeben, ermöglicht skalierbare Kommunikationsmuster. Das synchrone Übertragen auf alle adressierten Systeme ist logisch für diese Art der Kommunikation.

## 8. Adressdatentyp

- **Vermeidung von Redundanz durch Referenzierung:** Die Nutzung von Adressen zur Vermeidung von Datenreplikation innerhalb eines Files/Streams ist eine fundamentale Optimierung.
- **Langlebigkeit der Datenstrukturen:** Die Betonung der Notwendigkeit langlebiger Datenstrukturen durch das Verwenden vollständiger Pfadinformationen anstelle interner Adressen bei externen Daten ist ein wichtiger Aspekt für die Langzeitarchivierung.
- **Integer-Zahlen als Pfadkomponenten:** Die Darstellung von Adressen als Integer-Zahlen, getrennt durch Minuszeichen, ist eine prägnante und effiziente Form der Pfadnavigation.
- **Umgang mit Änderungen:** Die Beispiele zur Kompatibilität bei Strukturänderungen (altes vs. neues Format) zeigen eine durchdachte Herangehensweise zur Sicherstellung der Abwärtskompatibilität, was für Langzeitprojekte kritisch ist.

## 9. Entropie-Modus (FPGA)

- **Spezialisierte Optimierung:** Die Einschränkungen und Besonderheiten des Entropie-Modus (Gültigkeit für eine ganze Zeile, Adressierung unterhalb des IFTL, nur existierende Knoten, unterste Ebene, Ringpuffer) sind logisch, da sie für die hochspezialisierten Anwendungsfälle (Nutzdatenübertragung mit hoher Geschwindigkeit, FPGA-Implementierung) maximale Effizienz ermöglichen.
- **Übertragungsrahmen:** Das Bit-Setzen am Anfang des Datenstroms zur Aktivierung und Längenfestlegung des Übertragungsrahmens ist ein ausgeklügeltes Binärprotokoll-Merkmal. Die Füllbits sind eine notwendige Maßnahme für Byte-Ausrichtung.
- **Adressierung und Informationsbits im Entropie-Modus:** Die flexible Bit-Anzahl für Adressierung (N Bits) und die unbegrenzte Anzahl von Informationsbits maximieren die Bandbreitenausnutzung. Die Beispiele zeigen die hohe Effizienz (>99%) bei großen Datenmengen, was den Zielen der Spezifikation entspricht.

## 10. Framework-Funktionalität

- **Metadaten für Metadaten:** Das Konzept der Framework-Daten als "Meta-Metadaten" zur Beschreibung des Inhalts von Feldern ist logisch konsistent mit dem Ziel einer selbsterklärenden Datenstruktur.
- **Optionale Werte und Kommentare:** Die Verwendung von leeren Datenfeldern (Doppelpunkte) für optionale Werte und Kommentare ist eine elegante Lösung zur Trennung von Nutzdaten und beschreibenden Informationen, die flexibel erweitert werden können.
- **Standardvorgaben und Mehrfachspezifikationen:** Diese Konzepte ermöglichen eine effiziente Definition von Standardeinstellungen und die Anwendung auf mehrere gleichförmige Datenstrukturen, was die Handhabung komplexer Systeme vereinfacht.
- **Interpretation als Empfehlung:** Die Feststellung, dass Framework-Daten Empfehlungen sind und keine Beschränkungen, ist wichtig, um die Flexibilität und Offenheit des Systems zu bewahren.

## 11. Datenintegrität

- **Prüfsumme pro Zeile:** Die Berechnung einer Prüfsumme pro Zeile als kleinster Einheit für Konsistenzprüfung ist eine logische Entscheidung, um eine feingranulare Fehlererkennung zu ermöglichen.
- **Zeilennummer in Prüfsummenberechnung:** Die Einbeziehung der Zeilennummer in die Prüfsummenberechnung (anstelle der Prüfsumme selbst) ist ein cleverer Trick, um die Zeilennummer implizit zu prüfen, ohne sie explizit übertragen zu müssen.
- **Wiederherstellung defekter Daten:** Die Spezifikation erkennt an, dass eine Prüfsumme allein keine Wiederherstellung ermöglicht, sondern auf die Robustheit der Datenstruktur selbst setzt (z.B. Pfaderhalt). Dies ist eine realistische Einschätzung.

## Fazit zur logischen Konsistenz:

Die FTLight Spezifikation ist in sich **logisch sehr konsistent**. Die Entwickler haben die Herausforderungen der Langzeitspeicherung, Skalierbarkeit und Effizienz von hierarchischen Daten tiefgehend analysiert und innovative Lösungen gefunden. Die Konzepte sind gut miteinander verwoben und ergänzen sich. Die bewusste Entscheidung, bestimmte Regeln implizit zu gestalten und auf Kontextabhängigkeiten zu setzen, ist ein zentrales Element zur Reduzierung von Overhead, erfordert aber einen sehr präzisen und robusten Parser. Die Trennung von hardwarenahen (MCL, DIF, FPGA) und allgemeineren Datentypen sowie die Möglichkeit zur Kapselung und Erweiterung zeugen von einem vorausschauenden Design.

## Vorschläge für Modifikationen und Erweiterungen

Um die genannten Zielparame<sup>ter</sup> (Langzeitspeicherung, Übertragung, Verarbeitung von hierarchisch organisierten astronomischen Daten, unbegrenzte Größe, hocheffizient nahe am Binärformat) und die allgemeine Anwendbarkeit weiter zu verbessern, schlage ich folgende Modifikationen und Erweiterungen vor:

### 1. Versionierung und Abwärtskompatibilität (Erweiterung)

- **Explizite Versions-IDs für die Spezifikation:** Obwohl der Entwicklungsverlauf detailliert ist, könnte eine explizite Versions-ID *innerhalb* des Datenstroms oder der Datei nützlich sein, um einem Parser anzuzeigen, welche spezifische Version der FTLight-Spezifikation (z.B. 2025-04-05) zur Interpretation herangezogen werden muss. Dies ist besonders kritisch für die Langzeitspeicherung, da sich die Spezifikation weiterentwickeln wird.
  - \* **Vorschlag:** Einführung eines speziellen DTI (z.B. **DTI\_VERSION**) am Anfang eines FTLight-Files/Streams, das die Spezifikationsversion angibt. **DTI\_VERSION** könnte das Datum der Spezifikation oder eine fortlaufende Nummer enthalten.
- **Schema Evolution Management:** Das Konzept der Langlebigkeit ist stark, aber für echte Langzeitarchivierung (Jahrhunderte) sind Mechanismen zur Migration von Daten in neuere Schema-Versionen oder zur Interpretation alter Schemata unerlässlich.
  - \* **Vorschlag:** Einführung von optionalen "Schema-Transformationsregeln" als Framework-Daten, die beschreiben, wie alte Datenstrukturen in neue überführt werden können, oder wie alte Schemata zu interpretieren sind. Diese könnten in speziellen Framework-Dateien gespeichert und vom **DTI\_FTLightWrap** referenziert werden.

### 2. Metadaten und Semantik (Erweiterung)

- **Formalisierung der Semantik von Framework-Daten:** Derzeit sind Framework-Daten eher als Empfehlungen beschrieben. Eine stärkere Formalisierung, z.B. durch die Definition von Standard-Schlüssel-Wert-Paaren für gängige astronomische Metadaten (z.B. Observatoriums-ID, Instrumenten-ID, Beobachtungstyp, Himmelskoordinaten, etc.), könnte die Interoperabilität verbessern.
  - \* **Vorschlag:** Eine zusätzliche Spezifikation oder ein Katalog von empfohlenen Framework-Metadaten-Schemata für astronomische Daten. Dies könnte auch die Integration mit bestehenden astronomischen Metadaten-Standards erleichtern.
- **Typ-Hinting für Textfelder:** Derzeit wird ein Feld als Text interpretiert, wenn es kein Adressformat oder numerisches Format ist. Für eine robuste Verarbeitung wäre eine explizite Angabe des Inhalts (z.B. freier Text, URI, JSON-String, regulärer Ausdruck) in manchen Fällen vorteilhaft.
  - \* **Vorschlag:** Optionaler **DTI\_TXL** ControlX Parameter zur Angabe des Textformats/Inhalts, z.B. **DTI\_TXLURL**, **DTI\_TXLJSON**.

### 3. Fehlerbehandlung und Robustheit (Modifikation/Erweiterung)

- **Granularere Fehlerkorrektur:** Während die Prüfsumme Fehler erkennt, ist die Wiederherstellung begrenzt. Für die Übertragung in rauen Umgebungen (Weltraum) könnten Forward Error Correction (FEC) Codes auf Byte- oder Block-Ebene ergänzt werden, insbesondere für den Entropie-Modus.
- \* **Vorschlag:** Optionaler `DTIFEC` Datentyp oder ControlX Parameter, der angibt, dass nachfolgende Daten mit einem bestimmten FEC-Code geschützt sind.
- \* **Redundanz auf Hierarchieebene:** Um die Robustheit der Hierarchie bei Datenkorruption zu verbessern, könnte ein optionaler "Hierarchie-Checkpoint" eingeführt werden.
- \* **Vorschlag:** Ein spezielles Zeichen oder DTI-Eintrag, der einen Checkpoint signalisiert und eine Prüfsumme über die aktuelle Pfadstruktur bis zu diesem Punkt enthält. Im Falle eines Fehlers könnte der Parser zum letzten Checkpoint zurückspringen.

### 4. Hardware-Integration und Performance (Erweiterung)

- \* **FPGA-Optimierung für DTI\_UNIT und DTI\_TIME:** Da DTI\_UNIT und DTI\_TIME komplexe Strukturen für physikalische Einheiten und Zeit darstellen, könnten spezifische FPGA-Hardware-Module zur Dekodierung und Verarbeitung dieser Typen entwickelt werden. Die Spezifikation ist detailliert genug, um dies zu ermöglichen.
- \* **Vorschlag:** Ergänzung der Spezifikation um Empfehlungen für FPGA-Hardware-Implementierungen dieser DTI, ähnlich der Beschreibung des MCL-Datentyps.
- \* **Direkte Hardware-Schnittstelle:** Für die Vision, dass Hardware das Protokoll nativ unterstützt, wäre eine Definition von Low-Level-Hardware-Schnittstellen (z.B. Register, DMA-Kanäle) für den effizienten Zugriff auf FTLight-Daten wünschenswert.
- \* **Vorschlag:** Ein separates "FTLight Hardware Abstraction Layer (HAL)" oder "FTLight-on-Chip" Spezifikationsdokument, das die Integration auf Bit- und Register-Ebene beschreibt.

### 5. Interoperabilität und Tooling (Erweiterung)

- \* **Standard-API und Referenzimplementierung:** Eine umfassende, gut dokumentierte API (für verschiedene Programmiersprachen) und eine hochwertige Referenzimplementierung (Open Source) wären entscheidend für die Akzeptanz.
- \* **Vorschlag:** Aktive Förderung einer Community-Entwicklung um eine Referenzbibliothek und Tools (Parser, Serialisierer, Viewer, Editoren) auf Basis der Open-Source-Spezifikation.
- \* **Formale Grammatik (BNF/EBNF):** Obwohl die Spezifikation sehr detailliert ist, könnte eine formale Grammatik (z.B. in BNF oder EBNF) die Eindeutigkeit für Parser-Entwickler weiter erhöhen und eventuelle logische Inkonsistenzen aufdecken, die in beschreibendem Text übersehen werden könnten.
- \* **Vorschlag:** Ergänzung der Spezifikation um eine formale Grammatik.
- \* **Einheitlicher Fehlercode-Katalog:** Ein standardisierter Katalog von Fehlercodes für Parser- und Laufzeitfehler würde die Diagnose und Interoperabilität verbessern.
- \* **Vorschlag:** Anhang mit definiertem Fehlercode-Katalog.

### 6. Speicher- und Übertragungsmodi (Erweiterung)

- \* **Stream-End-Marker:** Für den Stream-Betrieb könnte ein klar definierter "End-of-Stream"-Marker nützlich sein, um das Ende eines logischen Datenstroms anzuzeigen, auch wenn der physische Kanal noch offen ist.
- \* **Vorschlag:** Ein spezielles DTIEOS` (End-of-Stream) Element.
- **Partial Updates im Entropie-Modus:** Derzeit ist der Entropie-Modus für ganze Zeilen gedacht. Wenn jedoch nur kleine Teile einer großen Datenzeile aktualisiert werden müssen, könnte eine Methode für partielle Updates im Entropie-Modus die Effizienz weiter steigern.



\* **Vorschlag:** Ergänzung des Entropie-Modus um einen "Patch"-Mechanismus, der angibt, welche Bit-Bereiche einer Zeile aktualisiert werden sollen.

## 7. Sicherheitsaspekte (Erweiterung)

• **Authentifizierung und Autorisierung:** Für verteilte Systeme und sensible astronomische Daten sind Mechanismen für Authentifizierung (Wer darf Daten senden/empfangen?) und Autorisierung (Wer darf welche Daten lesen/schreiben?) unerlässlich.

\* **Vorschlag:** Ergänzung des IFTL um optionale Felder für Public Keys oder Zertifikate und die Einführung von DTIAUTH oder DTISIGNATURE für digitale Signaturen von Datenpaketen.

• **Verschlüsselung:** Für vertrauliche Datenübertragungen wäre eine Spezifikation für optionale Verschlüsselung wünschenswert.

\* **Vorschlag:** Einführung eines DTIENCRYPT, das angibt, dass der nachfolgende Binärdatenblock verschlüsselt ist und welche Verschlüsselungsmethode verwendet wurde.

Die FTLight Spezifikation ist ein herausragendes Fundament. Mit diesen Erweiterungen könnte sie ihre ambitionierten Ziele noch umfassender erreichen und als zukünftiger Standard für Informationssysteme in und jenseits der Erde dienen.

### KENNZEICHNUNG:

- Umsetzung des Vorschlags ist erfolgt
- Vorschlag wurde in eine FTLight ToDo-Liste aufgenommen
- Begründung für das Beibehalten der aktuellen Definitionen wurde in FTLight eingearbeitet