Internet Engineering Task Force C.G. Ocen, Ed.
Internet-Draft Organization ChazahGroup
Intended status: Informational 12 December 2025
Expires: 15 June 2026

SW103K PROTOCOL
draft-rfcxml-rfc-swl-103k-06

Abstract

This specification defines a new protocol sw103k addresses challenges in networks with limited bandwidth, latency constraints, and data integrity concerns. It provides compression and decompression to optimize bandwidth utilization in environments such as Data centers, internet exchange points, and mobile communications. The protocol operates at the data link layer with a custom frame format including SW103K and HAVI headers. Key features include:

- Batch processing of 103 packets with compression - Merkle tree-based integrity verification (merklesw103k root hash) - QoS mechanisms with 8-bit priority field - Security features including AES-256-GCM encryption - Physical layer synchronization with +-1us accuracy

Jutifying why sw103k is unique:

The sw103k protocol is a Layer-2 mechanism that applies stateful compression to complete payloads using shared context between communicating entities.

The protocol assumes that traffic in constrained networks is often structured and repetitive and therefore allows endpoints to provision payload templates,

dictionaries, and delta-encoding state prior to data exchange.

During operation, payloads may be represented by compact identifiers together with fields that describe deviations from previously established state.

For traffic patterns that conform to these assumptions,

this approach can substantially reduce the number of transmitted octets and may achieve compression ratios on the order of 103:1

for selected telemetry and control messages. By operating independently of IP version, transport protocol, and application encoding,

sw103k is intended to reduce airtime usage, energy consumption, and the need for fragmentation in environments where link capacity and device resources are limited.

Status of This Memo

This Internet-Draft will expire on 15 June 2026.

Copyright Notice

Table of Contents

1. Introduction

This document specifies sw103k, a Layer 2 hybrid compression and decompression protocol designed to achieve extremely high compression ratios for diverse network traffic. sw103k combines template matching, Merkle-based deduplication, Algebraic Sparse Polynomial Compression (ASPC), general-purpose entropy coding (LZ + ANS), and optional machine-learning-based lossy codecs. Deterministic per-frame compression selection ensures predictable behavior with built-in error detection and recovery. The protocol is intended for Layer 2 environments and supports negotiation of compression modes, lossless or lossy operation, and integrity verification. **1. Introduction** This document proposes the SWL103K protocol for interoperable device communication within defined network scopes in order to ensure interoperability **2. Protocol Features** The SWL103K protocol SHOULD support data compression for efficient data exchange in resource-constrained environments. **3. Security Considerations** Implementations of this protocol MUST NOT store plaintext passwords in memory. The rapid growth of networked devices and the emergence of diverse applications have led to the demand for efficient communication protocols that can accommodate varying network conditions, scalability, and resource constraints. The SWL103K protocol presented in this document aims to address these challenges by providing a robust and adaptable solution for data exchange in distributed networks. As network environments become increasingly dynamic and heterogeneous, traditional communication protocols may struggle to provide optimal performance. The SWL103K protocol takes a novel approach by integrating innovative techniques for data transmission, congestion control, and routing. This ensures that the protocol remains responsive and reliable, even in scenarios where network conditions may change unpredictably. This document outlines the fundamental design principles, key features, and operational characteristics of the SWL103K protocol. It describes the protocol’s message format, data integrity mechanisms, and how it handles various network scenarios. By providing a comprehensive understanding of the SWL103K protocol, this document aims to enable network engineers, researchers, and implementers to make informed decisions about its adoption and integration into their respective systems. The following sections of this document delve into the specific components of the SWL103K protocol, including its requirements, design considerations, and operational guidelines. Additionally, the document provides insights into its security considerations and interactions with existing protocols. Overall, the SWL103K protocol aims to enhance the reliability, efficiency, and adaptability of communication in modern networked environments What problems does this protocol solve? This protocol solves several problems related to data transmission, compression, decompression, and integrity verification. Specifically, it aims to: Efficiently transmit and manage a large number of small data packets. Compress a batch of 103 data packets into a single compressed data stream. Decompress the compressed data back into the original 103 packets. Calculate and verify the integrity of received data using a merkelsw103k Tree. Handle various states of the communication process, including compression and decompression. +---------------------+ | Incoming L2 Frame | +---------------------+ | v +---------------------+ | Analyzer | | (Entropy/Template) | +---------------------+ | v +---------------------+ | Mode Selector | +---------------------+ | v +---------------------+ | Compressor Engine | | (ASPC, LZ, ML, etc)| +---------------------+ | v +---------------------+ | L2 Header Attachment| +---------------------+ | v +---------------------+ | Compressed Frame | +---------------------+ Modern high-throughput networks increasingly demand compression to reduce bandwidth consumption and improve performance. Layer 2 compression offers the advantage of transparency to higher-layer protocols and reduced latency. Existing approaches (e.g., LZ-based compression, zstd) are limited in compression ratio, adaptability, and cross-format support. sw103k addresses these limitations through a hybrid compression pool capable of adapting per-frame based on content type and entropy. The design goals of sw103k are: High Compression Ratio: Targeting 103:1, adaptable per traffic type. Low Latency: Per-frame compression less than 1 KB to 16 KB frames in under 1 ms. Lossless First, Lossy Optional: Default mode is lossless. Deterministic Selection: Analyzer decisions are consistent at both ends. Cross-Format Support: Supports text, numeric, image, audio, and video streams. Error Detection and Recovery: Mandatory CRC or AEAD integrity checks with fallback support. Protocol requirements are: MUST preserve frame integrity. SHOULD select highest expected compression ratio. MAY employ lossy ML codec if negotiated. MUST provide deterministic per-frame Analyzer decisions. SHOULD provide template and model caching for ratio optimization.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. How it works

The provided custom protocol, which appears to be a part of a larger system or application, aims to address various communication and data handling challenges. Below are responses to your questions regarding the abstract understanding of this protocol:

1. Efficiently transmit and manage a large number of small data packets. Compress a batch of 103 data packets into a single compressed data stream. Decompress the compressed data back into the original 103 packets. Calculate and verify the integrity of received data using a merkelsw103k Tree. Handle various states of the communication process, including compression and decompression. To implement this protocol, you would need to: Define and initialize a struct sw103k_proto instance to manage the protocol’s state and data. Implement the functions compressPackets and decompressPackets to handle compression and decompression of data packets. Handle various protocol states and operations in the sw103k_proto_parse_pkt function, updating the protocol instance accordingly. 6. Abstract: How does this protocol’s function compare to other transport protocols? This protocol appears to be a custom communication and data handling protocol tailored to specific needs. Unlike widely used transport protocols like TCP or UDP, which focus on reliable data transmission or low-level data transfer, this custom protocol includes features for data compression, decompression, and integrity verification. The choice of using this protocol would depend on the specific requirements of the application. TCP, for example, ensures reliable data delivery, while UDP offers lower overhead but without guarantees of reliability. This custom protocol seems to prioritize efficient data compression and decompression, making it suitable for scenarios where data size and compression are critical factors. 7. Why might someone decide to use this instead of something else that already exists? Someone might choose to use this custom protocol over existing alternatives if their application requires: Efficient compression and decompression of data packets. Fine-grained control over data transmission and compression. Integration of integrity verification using a merkelsw103k Tree. Customized handling of communication states and operations. Depending on the specific use case, existing transport protocols like TCP or UDP may not provide the desired level of data compression or customizability. 8. What are the security issues raised by using this protocol? While the provided code includes features for calculating and verifying packet integrity using a merkelsw103k Tree, it’s essential to consider potential security issues: Data Integrity: The protocol relies on integrity verification using a merkelsw103k Tree, but it assumes that the root hash provided is

trustworthy. Any compromise of the root hash could lead to data integrity issues. (Fixed with data integrity checks) Compression and Decompression: If not implemented securely, compression and decompression routines can potentially introduce vulnerabilities, such as buffer overflows or injection attacks. Authentication and Authorization: The protocol does not appear to address user authentication or authorization, which could be crucial for secure communication. Data Privacy: Depending on the nature of the data being transmitted, encryption may be necessary to ensure data privacy. Implementers should conduct thorough security assessments and consider encryption, authentication mechanisms, and protection against common security threats. In summary, the provided custom protocol offers a tailored solution for data transmission, compression, and integrity verification. Its use cases and advantages would depend on the specific requirements of the application it is being implemented for, but it provides flexibility and control over these aspects compared to more standardized transport protocols. Security considerations are essential when implementing and deploying this protocol in real-world applications. Below is how we are fixing the security concerns Threat Modeling: Start by conducting a threat modeling exercise. Identify potential threats and vulnerabilities in your protocol. Consider various attack vectors, such as eavesdropping, tampering, and unauthorized access. Authentication: Implement strong authentication mechanisms to ensure that communication parties can verify each other’s identities. This can involve using cryptographic protocols like TLS/SSL for secure communication. Data Encryption: Encrypt sensitive data to protect it from eavesdropping. Use well-established encryption algorithms and ensure that keys are managed securely. Access Control: Enforce proper access controls to prevent unauthorized access to resources. Ensure that only authorized users or devices can interact with the protocol. Data Integrity: Implement mechanisms to verify the integrity of data during transmission. This can include using checksums, digital signatures, or HMAC (Hash-based Message Authentication Code). Secure Key Management: Properly manage cryptographic keys used for encryption and authentication. Store keys securely and rotate them periodically. Secure Coding Practices: Follow secure coding practices to avoid common vulnerabilities such as buffer overflows, injection attacks, and format string vulnerabilities. Error Handling: Implement robust error handling to prevent information leakage through error messages. Provide generic error messages to users and log detailed error information for administrators. Logging and Monitoring: Implement logging and monitoring to detect and respond to security incidents. Log relevant security events and regularly review logs for suspicious activities. Penetration Testing: Conduct penetration testing and security audits to identify vulnerabilities that may not be apparent during design and development. Regular Updates: Keep the protocol and its dependencies up to date. Security vulnerabilities can be discovered in libraries or components used by the protocol. Documentation: Provide clear and up-to-date documentation on security best practices for users and administrators of the protocol. User Education: Educate users and administrators about security best practices when using the protocol. This includes password hygiene, avoiding suspicious links or attachments, and recognizing phishing attempts. Security Review: Consider involving security experts or third-party security audits to evaluate the protocol's security posture. Compliance: Ensure that the protocol complies with relevant security standards and regulations, if applicable. Incident Response Plan: Develop an incident response plan to address security breaches or incidents. Define procedures for identifying, reporting, and mitigating security issues.

* How does this protocol work? The protocol works by defining a set of states (e.g., CONNECTING, COMPRESSING, DECOMPRESSING) and operations (e.g., SEND_COMPRESSED_DATA, RECEIVE_COMPRESSED_DATA). It provides functions for compressing and decompressing data, as well as for calculating and verifying packet integrity using a merkelsw103k Tree.

First term: SW103K
Definition is the name of the protocol

Second term: HaviPackets
Definition is the packets name on the transport layer

Compression Command C
Decompression Command D Table 1: Compression Modes and Expected Ratios. The sw103k system supports several compression modes with varying expected compression ratios depending on the structure and entropy of the data. Template + Merkle: Used for repeated telemetry or structurally similar frames, with expected ratios between 10:1 and 50:1. ASPC Polynomial: Optimized for numeric matrices, logs, and structured analytics data, producing ratios in the range of 20:1 to 80:1. LZ + ANS: Applied to general text or binary data, typically achieving 2:1 to 10:1 compression. Transform Text: Intended for structured or repetitive text, with expected ratios of 5:1 to 15:1. ML Autoencoder: A lossy mode for audio, video, or images, targeting compression ratios up to 103:1. Passthrough: Used for high-entropy data where compression is not effective, producing a 1:1

| ratio. Protocol Overview Text Only sw103k operates as a Layer |
| 2 shim between the MAC and network interface. It analyzes each |
| frame and selects the most efficient compression algorithm. The |
| Analyzer performs entropy estimation, template matching, |
| polynomial fitting, and ML model inference. Frames are |
| compressed using the selected mode and wrapped with a minimum |
| 12-byte header. Frame Pipeline Overview: Frame → Analyzer → |
| Mode Selector → Compressor Engine → Header Generator → |
| Compressed Frame. Frame Format The sw103k header is a minimum |
| of 12 bytes and is TLV-extensible. The header includes: |
| version, algorithm (ALG), lossy flag, estimated ratio, model |
| identifier, flags, original length, compressed length, and |
| integrity tag (CRC or AEAD). Header Layout (text description): |
| Version and algorithm fields occupy the first bits. Lossy flag |
| and estimated ratio follow. A model identifier and flags are |
| next. Two length fields represent original and compressed |
| sizes. A 32-bit or larger checksum or AEAD authentication tag |
| completes the header. ALG Code Points: 0x0 = Passthrough 0x1 = |
| Template + Merkle 0x2 = ASPC Polynomial 0x3 = LZ + ANS 0x4 = |
| Transform Text 0x5 = ML Autoencoder (Lossy) Compression Modes |
| Text Only Template + Merkle Deduplication Uses pre-shared |
| templates or a sliding window of recent frames. Each frame is |
| divided into chunks; if a chunk hash matches the Merkle tree, a |
| short identifier is emitted instead of raw data. ASPC |
| Polynomial Compression Numeric or log data is approximated using |
| sparse polynomials. Only coefficients and residuals are stored. |
| Example: values [100,102,104,106] fit a linear polynomial, |
| producing negligible residuals and very small compressed size. |
| LZ + ANS Traditional dictionary compression (LZ77) followed by |
| asymmetric numeral system entropy encoding. Transform Text Uses |
| BWT, Move-to-Front, and ANS to compress structured textual |
| content. ML Autoencoder (Lossy) Domain-trained autoencoders |
| compress audio, video, or images with lossy reconstruction. |
| Ratios may reach 103:1 depending on complexity and quality |
| settings. Passthrough High entropy or encrypted content is |
| forwarded without compression. Analyzer FSM Text Only The |
| Analyzer determines compression mode based on frame entropy: If |
| entropy is below a low threshold, template or polynomial modes |
| are selected. If entropy is moderate, LZ or Transform modes are |
| used. If entropy is high but lossy compression is allowed, the |
| ML autoencoder is selected. If entropy is too high or lossy is |
| disallowed, the frame is passed through unchanged. Flow |
| Summary: Frame → Entropy Estimate → (Low → Template/ASPC; Medium |
| → LZ/Transform; High → ML if allowed; otherwise Passthrough) |
| Negotiation Text Only During link setup, both peers exchange |
| TLVs describing supported compression modes, latency limits, |
| maximum dictionary size, AEAD requirements, and model |
| identifiers. After capability exchange, the link transitions to |

operational mode. Error Detection and Recovery Text Only
Integrity is provided using CRC32/64 or AEAD tags. On
decompression failure, peers request retransmission or updated
templates/models. Persistent errors cause fallback to
passthrough mode. Security Considerations Text Only
Compression-before-encryption is recommended for optimal ratios.
Lossy mode is allowed only if negotiated. Integrity checks
prevent silent corruption. Entropy analysis must not leak
sensitive information. Operational Considerations Text Only
Hardware acceleration is recommended for real-time operation.
Template and model caching significantly improves compression
efficiency. Operators may adjust Analyzer thresholds and enable
telemetry for optimizing compression behavior. IANA
Considerations Text Only New registries are required for
sw103k algorithm (ALG) code points, TLV types, and model
identifier versions. Appendix A Text-Only Example Example
compression: Original frame: 1024 bytes telemetry Mode: ASPC
polynomial Compressed payload: 120 bytes Header: 9 bytes CRC: 4
bytes Total: 133 bytes Compression Ratio 7.7:1 Appendix B
Merkle Template Example A set of eight recent frame hashes is
stored along with 32-bit short-ID mappings. A Merkle root
references all recent chunks, enabling lightweight template
reuse across frames. Appendix C Pseudocode Summary If entropy
is low, use template or ASPC. If entropy is medium, use LZ or
Transform. If entropy is high and lossy is allowed, use ML
Autoencoder. Otherwise, use passthrough.

Table 1

```
<CODE BEGINS> file "network_app_protocol.c"
<CODE BEGINS>
#include "network_app_protocol.h"

int main() {
    // Initialize your custom protocol and
    // perform any necessary setup
    struct sw103k_proto mp;
    // Initialize mp and set its initial
    // state, buffers, etc.

    // Example function calls
    sendCommand(&mp, "CONNECT");
    authenticate(&mp, "networkuser",
                 "networkpassword");

    // Continuously receive and process data
    while (1) {
        custom_receive(&mp, network_socket);
        // Replace 'network_socket' with your
        // actual socket
    }

    // Clean up and exit
    // Close sockets, free memory, etc.

    return 0;
}
<CODE ENDS>
<CODE ENDS>
```

Figure 1: Source boiler code

```
             +------------------------------------------------------+
  |                 Custom Protocol (C Code)              |
  +------------------------------------------------------+
  |                                                       |
  |  +----------------------+     +-----------------+     |
  |  | Compression          |     | Decompression   |     |
  |  | Functions            |     | Functions       |     |
  |  +----------------------+     +-----------------+     |
  |        |                         |       |             |
  |        |                         |       |             |
  |  +-----v---+           +------v---+    |            |
  |  | Send    |           | Receive  |    |            |
  |  | Command |           | MP4 Data |    |            |
```

```
     |  | Function|           | Function |   |            |
     |  +---------+           +----------+   |            |
     |        |                   |         |            |
     |        |                   |         |            |
     |  +-----v-------------------v---------v----+       |
     |  |          Main Protocol Handling           |       |
     |  |              (sw103k_proto_parse_pkt)       |       |
     |  +------------------------------------------+       |
     |        |                                          |
     |        |                                          |
     |  +-----v-------------------------------+         |
     |  |             merkelsw103k Tree            |         |
     |  |       Construction and Integrity   |         |
     |  |             Verification            |         |
     |  +-----------------------------------+         |
     |        |                                      |
     |        |                                      |
     |  +-----v----------------------------------+  |
     |  |          State Management               |  |
     |  |     and Custom Hash Calculations       |  |
     |  +----------------------------------------+  |
     |        |                                    |
     |        |                                    |
     +--------v------------------------------------+
              |
              |
     +--------v------------------------------------+
     |               File Handling                 |
     |  (Sending and Receiving MP4 Data, File I/O) |
     +--------------------------------------------+
              |
              |
     +--------v------------------------------------+
     |          Custom Hash Functions              |
     |      (Hash Calculation and Combination)     |
     +--------------------------------------------+
```

Figure 2: Diagram

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

This document should not affect the security of the Internet.

5. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC6282] Hui, J. and D. Culler, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/rfc/rfc6282>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/rfc/rfc7228>.

[RFC8613] Selander, L., Mattsson, J., Palombini, F., and E. R. Herrero, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC8085] Fairhurst, G. and M. Welzl, "UDP Usage Guidelines", RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/rfc/rfc8085>.

Appendix 1

Appendix

Acknowledgements

This work is supported by Chazah Group.

Contributors

Thanks to Chazah Group Ltd.

Change Log

Changes in this version (draft-rfcxml-rfc-swl-103k-03):

- Clarified the applicability of the SWL103K protocol to avoid implying global deployment.

- Updated normative language: changed "MUST be implemented by all network devices" to a more realistic recommendation for scoped environments.

- Responded to feedback from Area Directors regarding clarity and protocol deployment assumptions.

Author's Address

Ocen Joseph (editor)
Organization ChazahGroup
Street
City
Phone: Phone
Email: Email chief3@chazahgroup.org
URI: URI chazahgroup.org