Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 28 November 2025

C.G. ChazahGroup, Ed.
Organization ChazahGroup

27 May 2025

SW103K PROTOCOL
draft-rfcxml-rfc-swl-103k-04

Abstract

The SW103k protocol 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 IoT, satellite, 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

Implementations include Linux kernel modules, FPGA encoders, and userspace daemons. The protocol supports interoperability with industrial standards like PROFINET and EtherCAT through custom mappings.

Table of Contents

1. Introduction

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.

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. Implement communication logic to send and receive data packets based on the current protocol state. Implement functions for integrity verification, such as constructing a merkelsw103k Tree and comparing hashes. Utilize this protocol in your application by calling its functions based on your specific use case. 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

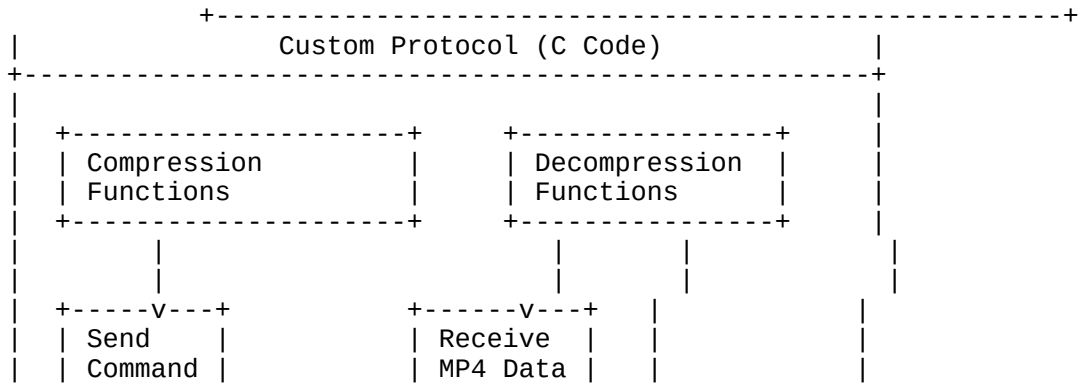
Second term: HaviPackets
Definition is the packets name on the transport layer

Compression Command C
Decompression Command D

Table 1

```
<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



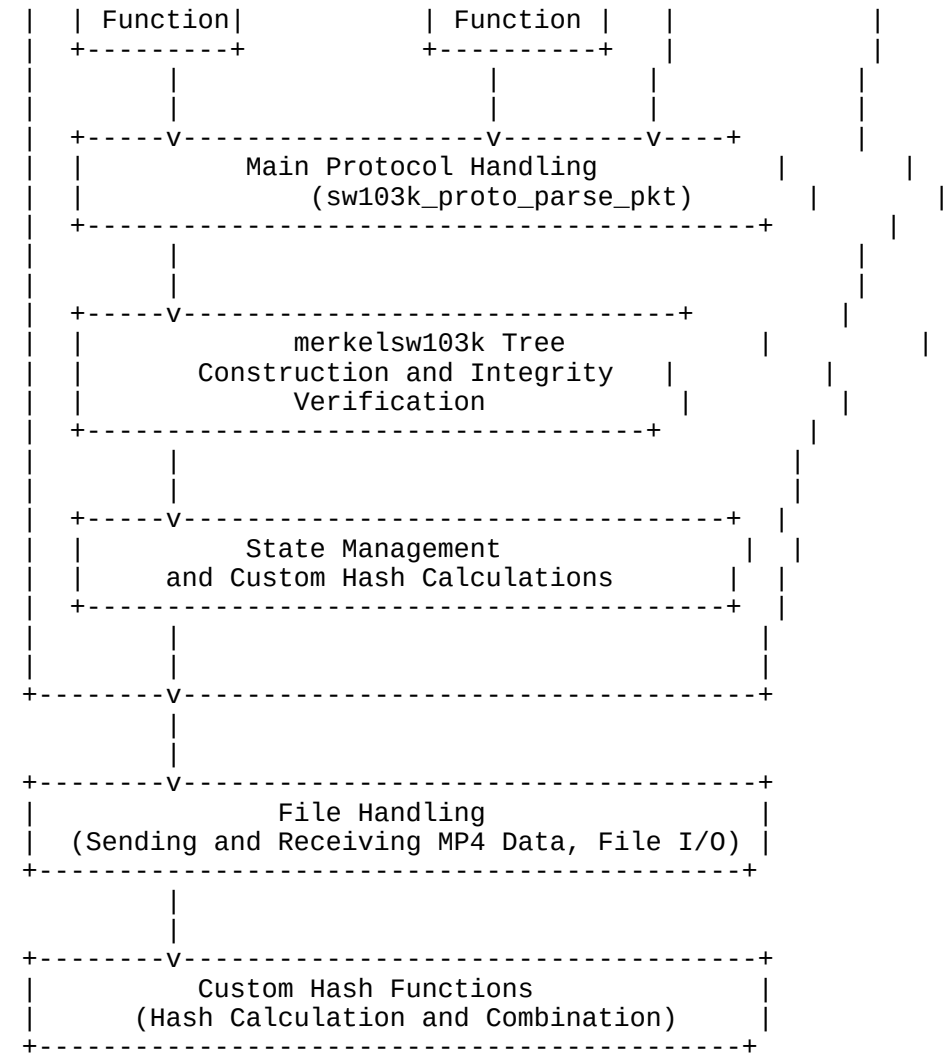


Figure 2: Diagram

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

This document should not affect the security of the Internet.

5. References

5.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

5.2. Informative References

[exampleRefMin] garith, Initials O.J., "Title", 2006.

[exampleRefOrg] IEEE 802 working group, "Title LLC", 1984, <http://www.example.com/>.

Appendix A. Appendix 1

Appendix

Acknowledgements

This work is supported by chazha 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

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