SCHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: 31 January 2026

Q. Lampin
M. Dumay
P. Surbayrole
Orange

30 July 2025

SCHC Minimal Architecture
draft-lampin-schc-minimal-architecture-00

Abstract

This document investigates the requirements of a minimal architecture for the Static Context Header Compression (and fragmentation) protocol (SCHC). The intent is to identify the essential components their relationships and interfaces. To do so, the document considers scenarios of increasing complexity involving the use of SCHC, from a simple point-to-point communication to a more complex deployment involving multiple SCHC Instances communicating with each other. In this process, the authors hope to identify the essential components of a SCHC architecture and their relationships.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 31 January 2026.

Table of Contents

1. Introduction

The SCHC Working Group has developed the [RFC8724] SCHC protocol for Low-Power Wide-Area (LPWA) networks, providing efficient header compression and fragmentation mechanisms. As SCHC adoption expands beyond its original scope, there is a need to define a minimal architecture that identifies only the essential elements required for proper SCHC operation. This document does not aim to replace the SCHC architecture defined in [DRAFT-ARCH], but rather to investigate the minimal set of components and their relationships that are

necessary for SCHC to function effectively in various deployment scenarios.

This document provides:

* A definition of the minimal components required for SCHC operation

* The essential interactions between these components

* Problems and challenges addressed by each component

2. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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.

3. Terminology

This section defines terminology and abbreviations used in this document. It borrows from the terminology of [RFC8724] and [DRAFT-ARCH].

In the following, terms used in the terminology are assumed to be defined in the context of the SCHC protocol unless specified otherwise, _.e.g_ Endpoint refers to a SCHC Endpoint, Instance refers to a SCHC Instance, and so on.

Endpoint: A network host capable of compressing and decompressing headers and optionally fragmenting and reassembling packets.

Instance: A logical component of an Endpoint that implements the SCHC protocol, including header compression, fragmentation, and context management.

Context: A set of rules and parameters that define how SCHC operations are performed by Instances that implement this Context. It includes the Set of Rules (SoR) and the parser ID.

Endpoint Manager: A logical component that manages the lifecycle and configuration of Instances within an Endpoint. It is responsible for creating, updating, and deleting Instances as needed, synchronizing Contexts and Profiles, and managing the Dispatcher.

Session: A communication session between two Instances that share a common context for compression and fragmentation operations. Whenever the Context is updated, a new or updated Session is established.

Domain: A logical grouping of Instances that share a common set of Contexts for compression and fragmentation operations.

Dispatcher: A logical component that routes packets to the appropriate Instances based on defined admission rules. It can be integrated into the network stack or implemented as a separate component.

Profile: A set of configurations that define how SCHC operations are performed within a specific Instance. It includes parameters for the different SCHC components.

Domain Manager: A logical component that manages the Domain, including context synchronization and profile distribution.

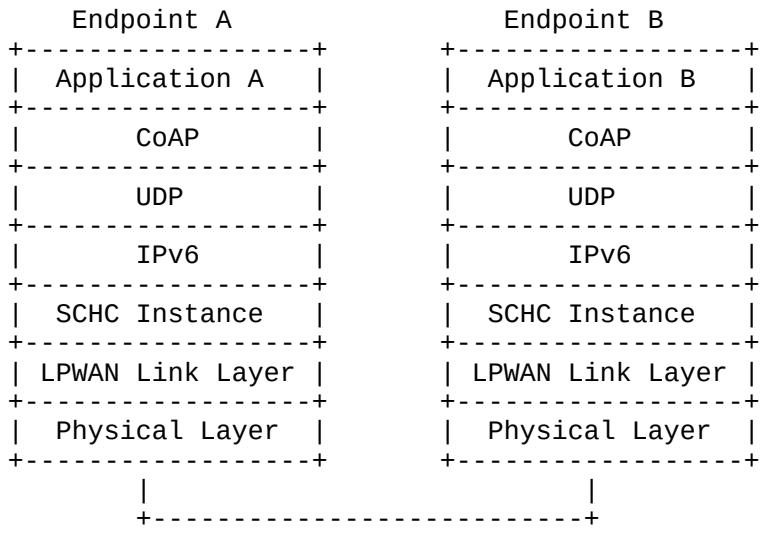
Context Repository: A logical component that stores and manages the Contexts used by its Domain.

4. Minimal Architecture Components, a case study & discussions

In this section, we investigate the minimal components required for SCHC operation in the context of typical deployment scenarios.

4.1. The simplest deployment scenario

This section considers a simple point-to-point deployment scenario where two Endpoints communicate with each other using SCHC. The devices are assumed to have a very simple network stack, as shown in the figure below:



In this scenario,

* Both Endpoint A and Endpoint B implement the SCHC protocol for header compression and decompression.

* Both Endpoints feature a single application and all traffic is sent and received using the CoAP protocol over UDP over IPv6.

* The SCHC protocol is used to compress the CoAP, UDP, and IPv6 headers before sending the packets over the LPWAN link layer.

* The SCHC protocol is implemented as a single Instance on each Endpoint.

* The Instance is hardwired into the protocol stack of each Endpoint, meaning that it is not dynamically loaded or unloaded.

* All of the traffic is compressed and decompressed using those Instances.

In this simplistic scenario, which is representative of some LPWAN deployments, the requirements for the minimal architecture are as follows:

* The Set Of Rules (SoR) of Endpoint A MUST be compatible with the SoR of Endpoint B. Such compatitibility requires that rules IDs and Rule Descriptors are consistent between the two Instances. Parsers of both Instances MUST be compatible, meaning that they MUST delineate the same header fields in the same order and with the same semantics.

* Whenever Endpoint A compresses a packet, it MUST use the same Context as Endpoint B. This means that the Context MUST be synchronized between the two Instances. This communication session is referred to as a Session.

Why Instance? Here we use the term Instance to refer to the SCHC protocol routine that is running on each endpoint. This is different from the SCHC Instance defined in [DRAFT-ARCH], which refers to a pair of SCHC endpoints that communicate through SCHC.

The rationale for this terminology is that the term Instance is often used to refer to a specific realization of a class in object-oriented programming, and in this case, the SCHC Instance is a specific realization of the SCHC protocol that is running on each endpoint.

Session vs Instance: In this document, we use the term Session to refer to a communication session between two (or more) Instances that are communicating with each other using SCHC, using the same Context.

The rationale for this is that the term Session is often used to refer to a specific communication session between two endpoints and this definition extends this concept to all Instances that maintain a common context.

Why not use the terminology of [DRAFT-ARCH]? We believe that version -04 introduced changes in the terminology that are prone to confusion. In particular, the term Instance is defined as:

SCHC Instance. The **session** between SCHC end points in two or more peer nodes operating SCHC to communicate using a common SoR and a matching SoV. There are 2 SCHC Instances or more involved per SCHC stratum, one for the SCHC Stratum Header and one or more for the SCHC payload, i.e., the SCHC-compressed data.

while the end point is defined as:

SCHC end point. An entity (e.g., Device, Application and Network Gateway) involved in the SCHC process. Each SCHC end point will have its Set of Rules (SoR), based on the profile, the protocols, the device, the behaviour and a Set of Variables (SoV).

Those definitions are confusing as they do not clearly identify the components which they are referring to. For example, the term Instance can be used to refer to session between two applications that are using SCHC, or to refer to two hardware devices that are using SCHC, etc.

The proposed terminology in this document aims to clarify the definitions of the architecture components by calling a session a Session and therefore reuses the term Instance to refer to the SCHC protocol routine that is running on an endpoint.

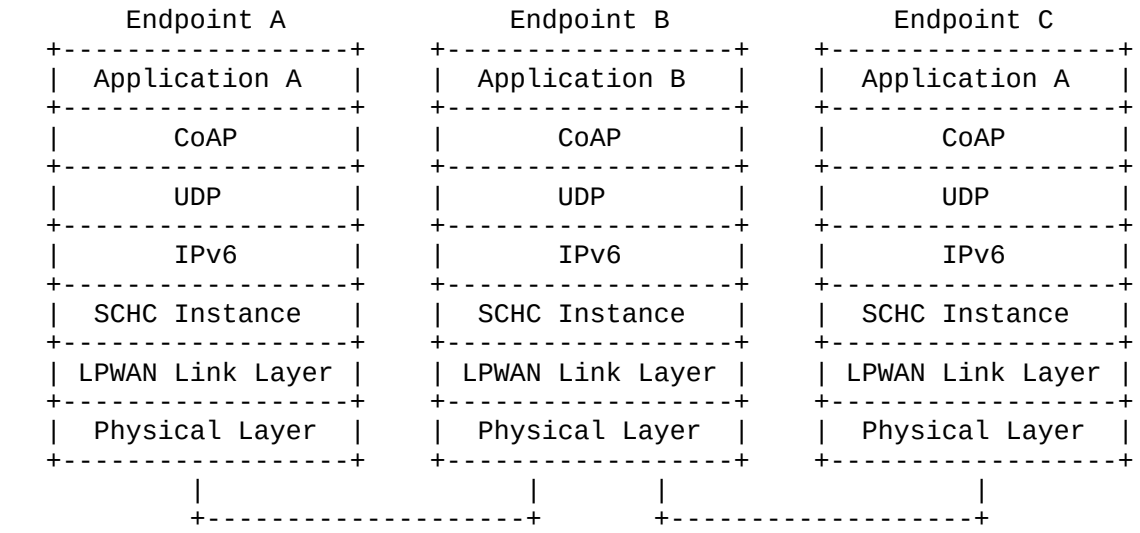
This terminology is consistent with prior versions of the architecture document, such as version -03, which had the following definitions:

* SCHC Instance. The different stages of SCHC in a host. Each instance will have its Set of Rules (SoR), based on the profile, the protocols, the device, the behaviour and a Set of Variables (SoV). There are 2 SCHC Instances involved per SCHC stratum, one for the SCHC header and one for the SCHC payload, i.e., the SCHC-compressed data.

* SCHC Session. The association of SCHC Instances in two or more peer nodes operating SCHC to communicate using a common SoR and a matching SoV.

4.2. The three endpoints deployment scenario

In this section, we consider a more complex deployment scenario where two or more endpoints communicate with the same SCHC Instance/ Endpoint. This scenario is common in IoT deployments where multiple sensors or devices communicate with a central gateway or server using SCHC.



In this scenario, we have three Endpoints, Endpoint A, Endpoint B, and Endpoint C, that communicate with each other using SCHC. Here, Endpoint A and Endpoint C are typically sensors or devices that send data to Endpoint B, which is a gateway or server that collects and processes the data.

We further assume that Endpoints A and C have very similar traffic patterns, meaning that they send similar packets to Endpoint B. This allows the SCHC Instances on Host A, B and C to share the same Context, which reduces the complexity of administration and management of this deployment. In the following, we refer to Instances that share the same Contexts as a Domain.

In this typical IoT deployment scenario, the requirements for the minimal architecture are as follows:

* The Context of all three Endpoints MUST be compatible. This means that the SoR, parsers, and rule IDs are consistent between the three Instances.

* The Context *MUST* be synchronized between the three Instances. This means that an updated Session between all three Instances is established whenever the Context is updated or modified.

* The Domain *MUST* be able to manage the Contexts of all Instances that belong to it. This includes the Endpoints enrollment, provisioning of Contexts and synchronization. This role is assumed by a logical component of the Domain, referred to as the Domain Manager.

Why synchronize the Contexts of A and C? Synchronizing the Contexts of Endpoint A and Endpoint C is desirable. This reduces the complexity of managing multiple Contexts at Endpoint B and eventually reduces the size the Rules IDs, impacting the SCHC packet size.

Domain Manager, why introduce a new entity? In a first approach, the Domain Manager could be implemented as a component of the Instance that is running on an Endpoint, here B. However, this would require distinguishing between different types of Instances: those capable of managing other instances from those that are not, those that are in charge of managing others from those that are not. By separating the Domain Manager from the Instance, we allow for a more flexible and modular architecture that can be adapted to different deployment scenarios.

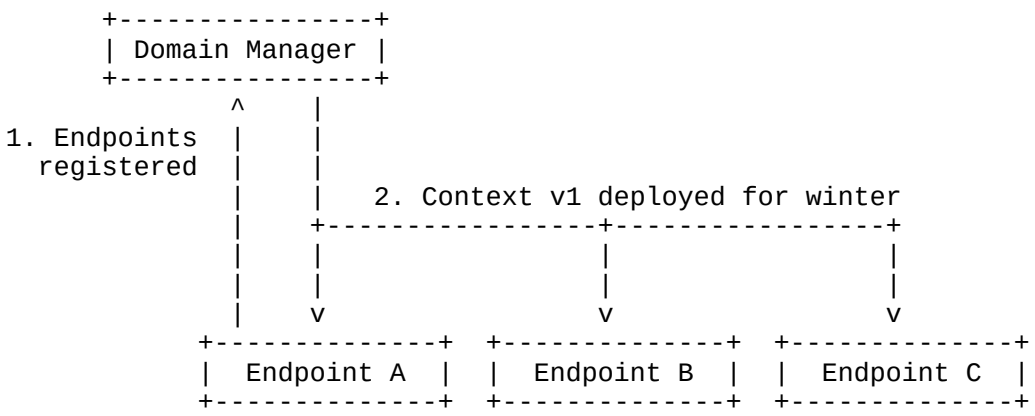
Context compatibility or equality? While not strictly required, it is desirable that the Contexts of all Instances that belong to the same Domain are equal, meaning that they share the same SoR, parsers, and rule IDs. This simplifies the management of the Contexts and reduces the risk of misconfiguration or incompatibility between Instances. However, this is not strictly required, as long as the Contexts are compatible, meaning that they can be used interchangeably without causing issues in compression or fragmentation operations.

4.3. The dynamic traffic scenario

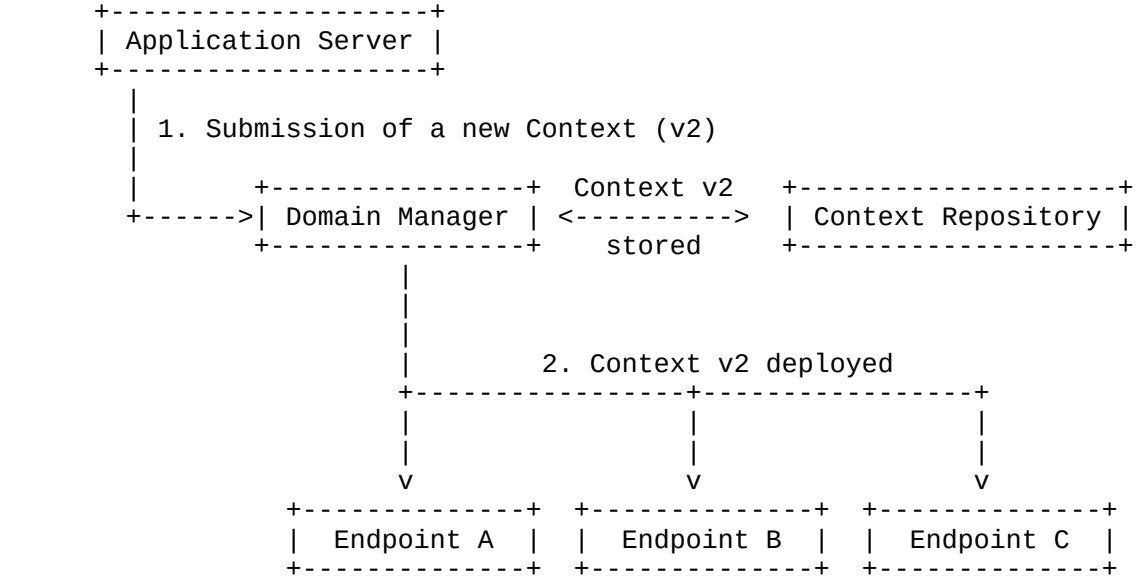
In this section, we consider a deployment scenario where multiple Endpoints communicate with each other using SCHC, but the traffic patterns of these Endpoints are dynamic and change over time. This scenario typically occurs in Smart Buildings applications, where different configurations are deployed based on the current season, occupancy, etc. For example, thermostats setpoints are set differently in winter and summer.

In this scenario, we have three Endpoints A, B and C. A, B feature a temperature and thermostat functionality, while C is a server that collects and processes the data from A and B.

We further assume that the Endpoints A, B and C are first registered with the Domain Manager of the domain which they are part of, and that they are provisioned with an initial Context. This Context is used to compress the temperature and thermostat data sent by A and B to C, and is tailored to the specifics of the temperatures recorded during winter, and the thermostat setpoints used during this season, as shown in the figure below:



Then comes the spring, and the temperature and thermostat setpoints change. The Domain Manager is responsible for updating the Context of all Instances that belong to the domain, i.e. A, B and C. This update is done dynamically, meaning that the Context is updated without interrupting the communication between the Endpoints.



This scenario highlights the need for a dynamic context update mechanism that allows the Domain Manager to update the Context of all Instances belonging to the Domain.

This raises a number of questions, such as:

* How to ensure that all Instances are updated with the new Context?

* How to process packets sent before the Context update but received after?

Answering those specific questions is critical for the proper operation of SCHC in this scenario as unsynchronized Contexts can lead to packet loss or misinterpretation at the receiving end.

It is worth noting that the same questions arise in the context of configuration management and are possibly addressed by existing IETF protocols.

Nevertheless, we can already identify the need for the following:

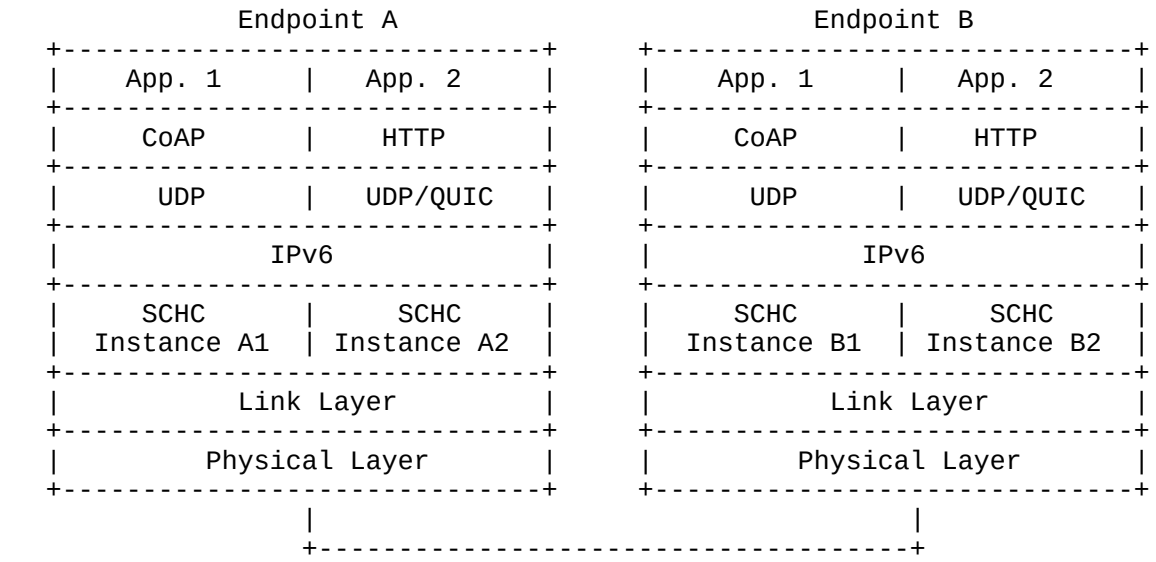
* A Context Repository that is responsible for storing the Contexts of the domain. In case of disagreement between Instances, the Context Repository is used to resolve the disagreement. Having one identified source of truth for the Contexts helps to maintain consistency across the domain. This is also useful when (new) nodes join the domain later, as the Context Repository can provide the necessary Context information to new or existing Instances.

* A mechanism for versioning Contexts, allowing the Domain Manager to manage multiple versions of a Context and facilitate rollbacks if needed.

* A mechanism for notifying Endpoints of Context updates, ensuring that all Endpoints are aware of the latest Context version and can adapt their behavior accordingly.

4.4. Multiple SCHC Instances in the same Endpoint

In this scenario, a single Endpoint that hosts multiple Instances is considered. This scenario involves each Instance being configured with different Contexts. This can be useful for supporting multiple applications or services with distinct traffic patterns. One such use-case arises when a single Endpoint needs to handle different types of traffic, potentially sent and received on different network interfaces, each requiring its own Instance with tailored compression and fragmentation settings.



In the above example, two Endpoints, A and B, each host two Instances. Endpoint A hosts Instance A1 and Instance A2, while Endpoint B hosts Instance B1 and Instance B2. Instance A1 and Instance B1 are configured to handle CoAP traffic and share a common Context C1 while Instance A2 and Instance B2 are configured to handle HTTP traffic and share a common Context C2.

This new scenario introduces the following challenges:

* *Datagram Dispatch*: The Endpoint must be able to dispatch packets to the appropriate Instance based on the protocol or application in use. This requires an additional functional entity, the Dispatcher, that can identify and dispatch packets to the correct Instance for compression and fragmentation. Conversely, the Dispatcher must also be able to route inbound compressed and fragmented packets to the correct Instance for decompression and reassembly.

  In the above example, uncompressed CoAP/UDP packets must be dispatched to Instances A1 and B1, while uncompressed HTTP/QUIC packets must be dispatched to Instances A2 and B2. Additionally, compressed CoAP/UDP packets must be dispatched to Instances A1 and B1, while compressed HTTP/QUIC packets must be dispatched to Instances A2 and B2 for decompression. The criteria for dispatching compressed packets to the correct Instance is called the Discriminator in [DRAFT-ARCH].

  Those challenges are discussed further in section Section 4.7.5.

* *Instance/Context Identification*: In addition to the need for a Dispatch mechanism, which addresses the routing of packets to the correct Instance, each Instance or Context must be uniquely identifiable to allow the Domain Manager to update the Context of a specific Instance.

Dispatcher and Discriminator In [DRAFT-ARCH], the authors introduce the concept of a Discriminator that is used to identify the Instance or Context that must be used to decompress a packet. As explained in the document, the Discriminator is a criterion that is used to select the appropriate Instance or Context for decompression.

The Dispatcher is different from the Discriminator in that it is the functional unit responsible for routing packets to the correct Instance based on:

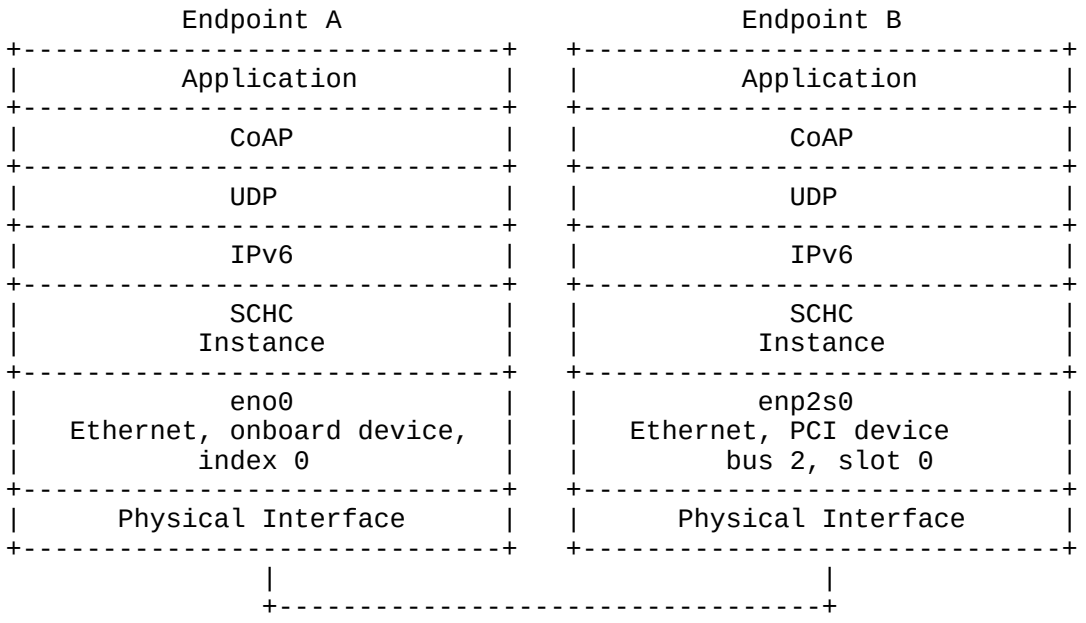
* the Discriminator value for decompression and reassembly.

* the Matching Operators and Target Values of the Contextes for compression and fragmentation.

4.5. Heterogeneous Endpoints

This additional scenario introduces heterogeneous Endpoints that feature different hardware and software configurations. These differences may include the Operating System (OS) or the hardware on which the Instance is run. Those differences are the source of a challenge in the deployment of Instances in configurations.

To illustrate this challenge, we consider the following example, illustrated in the figure below, where two Endpoints A and B are both running the same OS, here a Linux-based distribution using udev for device management, but on two different hardware platforms.



Endpoint A features an Ethernet interface which is located on the mainboard and is referred to as eno0: _en_ standing for "Ethernet", _o_ for "onboard", and _0_ for "index 0" in the udev naming convention.

Endpoint B features an Ethernet interface which is located on a PCI bus and is referred to as enp2s0: _en_ standing for "Ethernet", _p_ for "PCI", _2_ for "bus 2", and _s0_ for "slot 0" in the same naming convention.

In this scenario, the Context which contains the Set of Rules (SoR) and parser ID, i.e. the configuration which is shared by all Instances of a Domain, provides no information on how to instruct the Dispatcher to route packets from the appropriate Network Interface to the appropriate Instance.

Without further configuration and unless the Dispatcher intercepts all packets from all network interfaces, we cannot guarantee the correct dispatching of packets to the appropriate Instances. Obviously, intercepting traffic from all network interfaces is not a viable solution, as it would require the inspection of all packets, regardless of their destination, which requires significant processing power and may introduce unacceptable latency on high-speed links.

Additionally, different OSes may use different filtering/dispatch frameworks, e.g. Netfilter on Linux, BPF on FreeBSD, PF on OpenBSD and macOS etc. This means that the foundation on which the Dispatcher implementation relies may vary significantly between different platforms and could require different configurations to dispatch packets to Instances based on the same Context but running on different Endpoints.

This assessment advocates for the introduction of a device-specific configuration that is independent of the Context, which we name Profile.

Such Profiles are Endpoint specific so their actual content is out of scope of this document. However, the Profile *SHALL* include the following:

* The configuration of the Dispatcher, i.e. the admission rules for compression and decompression.

* The rule matching policy, i.e. the policy used to select the appropriate compression or decompression rule based on the SCHC packet and the Context.

As the Profile is Endpoint specific, it is not shared between Instances of different Endpoints. However, the Profile required for a given Instance may need to change when the Context is updated, e.g. the filtering rules may need to be adjusted to account for the new traffic patterns and C/D rules.

This means that the Profile may need to be updated whenever the Context is updated, and that the Domain Manager *SHALL* therefore be responsible for managing the Profile delivery to Instances and their synchronization with the Context.

4.6. The cold boot scenario

In this scenario, we consider the case where an Endpoint A is powered on and needs to establish a SCHC Session with another Endpoint B. Endpoint A does not have any Context or Profile stored in memory, and it needs to retrieve or negotiate this information before establishing the session.

Two hypothetical scenarios are considered here:

* _S1_. Endpoint A is provisioned on the appropriate Domain and is configured with the address/URI of the Domain Manager and Context Repository.

* _S2_. Endpoint A is provisioned on the appropriate Domain but is not configured with the address/URI of the Domain Manager and Context Repository. In this scenario, the Domain Manager is in charge of advertising its presence and push the context to Endpoint A.

In scenario _S1_, Endpoint A initiates the configuration phase, effectively pulling the Context and Profile from the Domain Manager and the Context Repository. This scenario implies that the Domain Manager exposes a management interface that allows Endpoints to retrieve the necessary configuration information. Additionally, a logical component referred to as Endpoint Manager is required to manage the Instances of the Endpoint.

In scenario _S2_, the Domain Manager is in charge of advertising its presence and Endpoint A is pulling the Contexts and Profiles from it. The advertisement can be done using a discovery mechanism, such as DNS-SD or a predefined multicast address. This scenario implies that Endpoints are capable of discovering the Domain Manager and retrieving the necessary configuration information. This further requires that Endpoints feature a service discovery mechanism and a Management Protocol that enables them to interact with the Domain Manager and request the required Context and Profile.

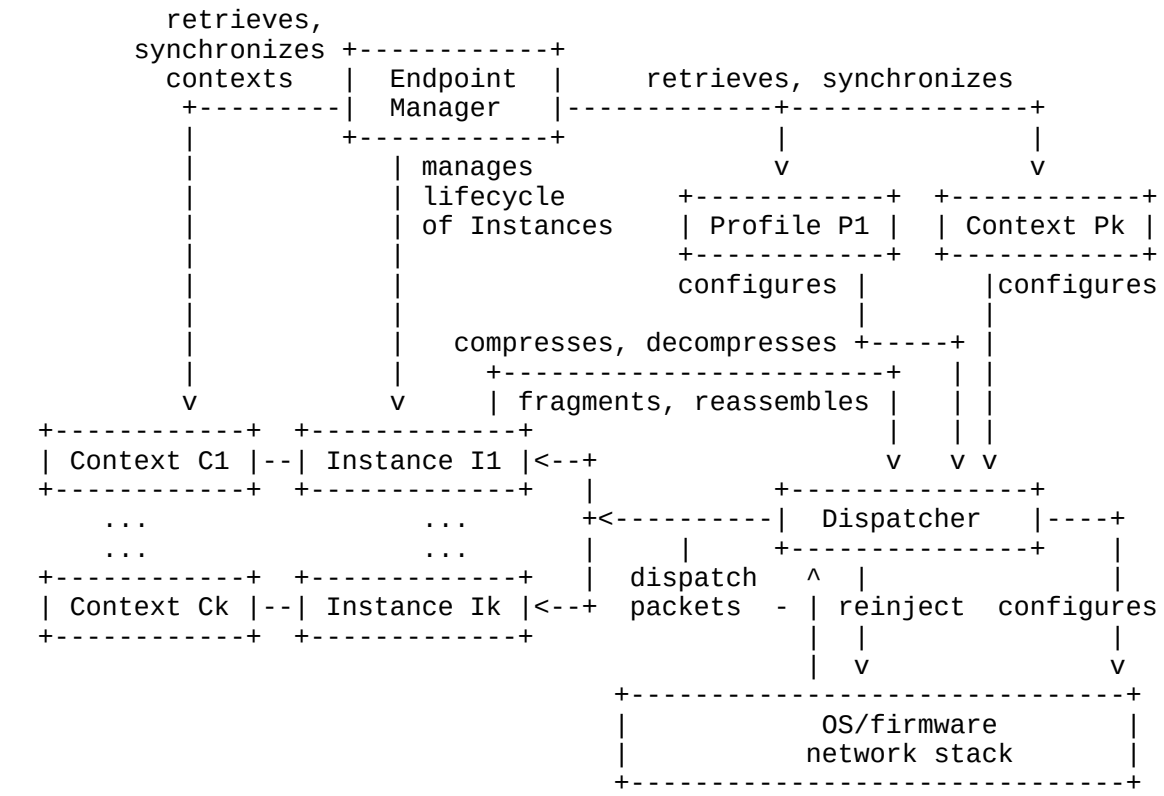
In both scenarios, a management protocol is required to enable the retrieval of the Context and Profile from the Domain Manager. [DRAFT-CORECONF] provides initial ideas on how to implement such a management protocol for SCHC in a Constrained Environment, e.g. IoT devices.

4.7. Core Components Illustrated

This section provides an overview of the SCHC Core components their interactions and key functionalities and interfaces.

4.7.1. Endpoint

An Endpoint is a network host capable of compressing and decompressing headers and optionally fragmenting and reassembling packets. It implements the SCHC protocol as defined in [RFC8724]. An Endpoint can host multiple Instances, each with its own Context and Profile.



4.7.2. Instance

An Instance is the fundamental component that implements the SCHC protocol as defined in [RFC8724]. An Endpoint MAY execute several Instances in its protocol stack. Each Instance operates independently, with its own context and profile.

An Instance MUST implement the following components:

* Header Compression and Decompression (C/D) engine

* Context Manager

* Profile Manager

Its configuration MUST include:

* a SCHC Context, which defines the set of C/D and F/R rules - or Set of Rules - and the parser to be used to delineate the header field.

* a SCHC Profile, which defines the configuration of the Dispatch Engine, the rule matching policy, and the device-specific configuration.

A SCHC Instance MAY implement:

* Fragmentation and Reassembly (F/R) functionality.

* Dynamic context update mechanisms.

* Performance monitoring and reporting.

4.7.2.1. Header Compression and Decompression (C/D) engine

This component is responsible for compressing and decompressing headers using the SCHC protocol, as described in [RFC8724]. It applies the rules defined in the SCHC Context.

The C/D engine MUST expose the following interface:

* compress(buffer, context, profile): Compresses the provided buffer using the SCHC Context and the profile.

* decompress(buffer, context, profile): Decompresses the provided buffer using the SCHC Context and the profile.

Internally, on compression, the C/D engine:

* delineates the fields using the parser identified in the SCHC Context.

* chooses the appropriate compression rule based on the SCHC Context and the matching policy defined in the profile.

* applies the compression rule to the fields of the header.

* generates the compressed SCHC packet.

On decompression, the C/D engine:

* identifies the C/D rule based on the SCHC compressed packet.

* applies the decompression rules to reconstruct the original header.

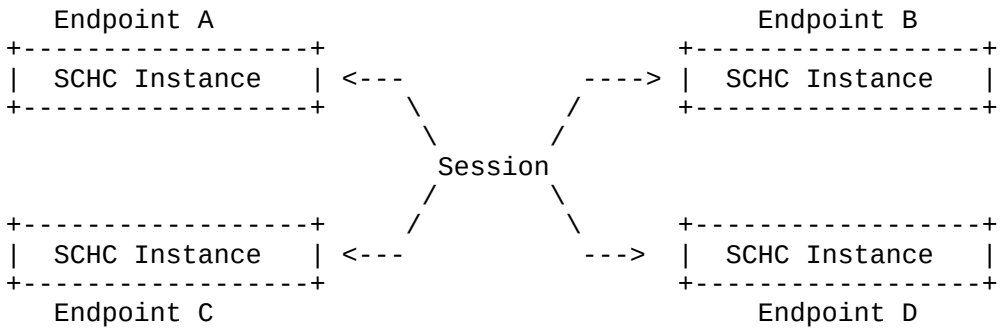
* reconstructs the original packet from the decompressed header and payload.

4.7.2.2. Fragmentation and Reassembly (F/R)

This component is responsible for fragmenting larger packets into smaller fragments and reassembling them at the receiving end. It is optional in the minimal architecture but recommended for scenarios where packet sizes exceed the maximum transmission unit (MTU) of the underlying network.

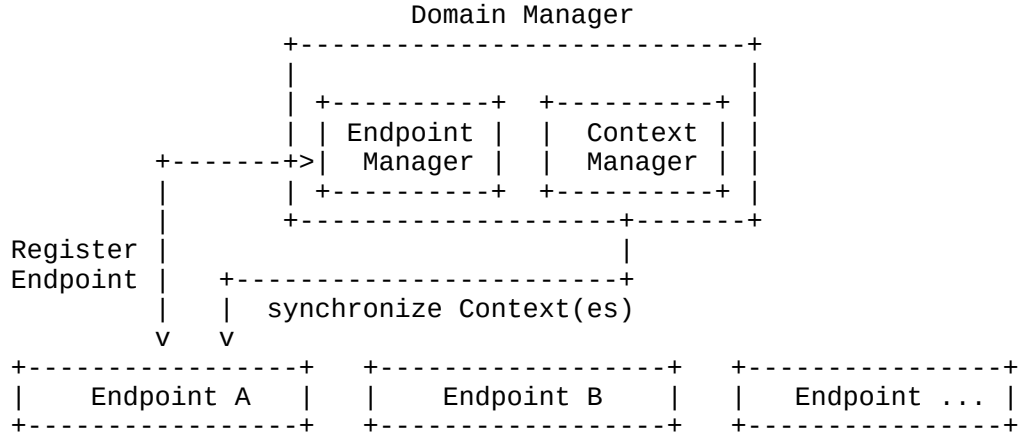
4.7.3. SCHC Session

As illustrated in the figure below, the Session is a communication session between two or more Instances that share a common Context, i.e. they are part of the same Domain. It is established whenever the Context is updated or modified.

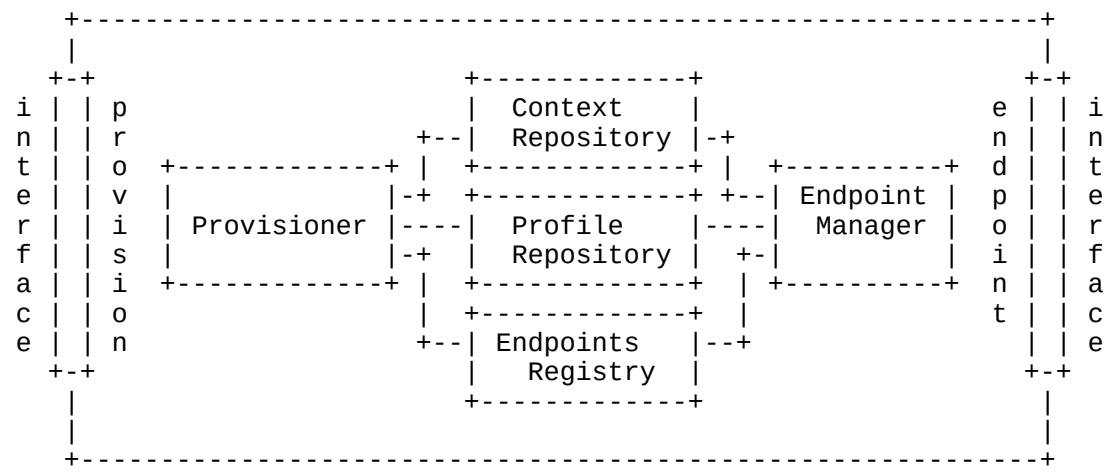


4.7.4. SCHC Domain & Domain Manager

The SCHC Domain is an administrative unit, whose role is to manage the SCHC Contexts of all Instances that belong to it. The Domain Manager is the component responsible for this management. It handles Endpoints Enrollment, and Context synchronization.



SCHC Domain, different illutration.

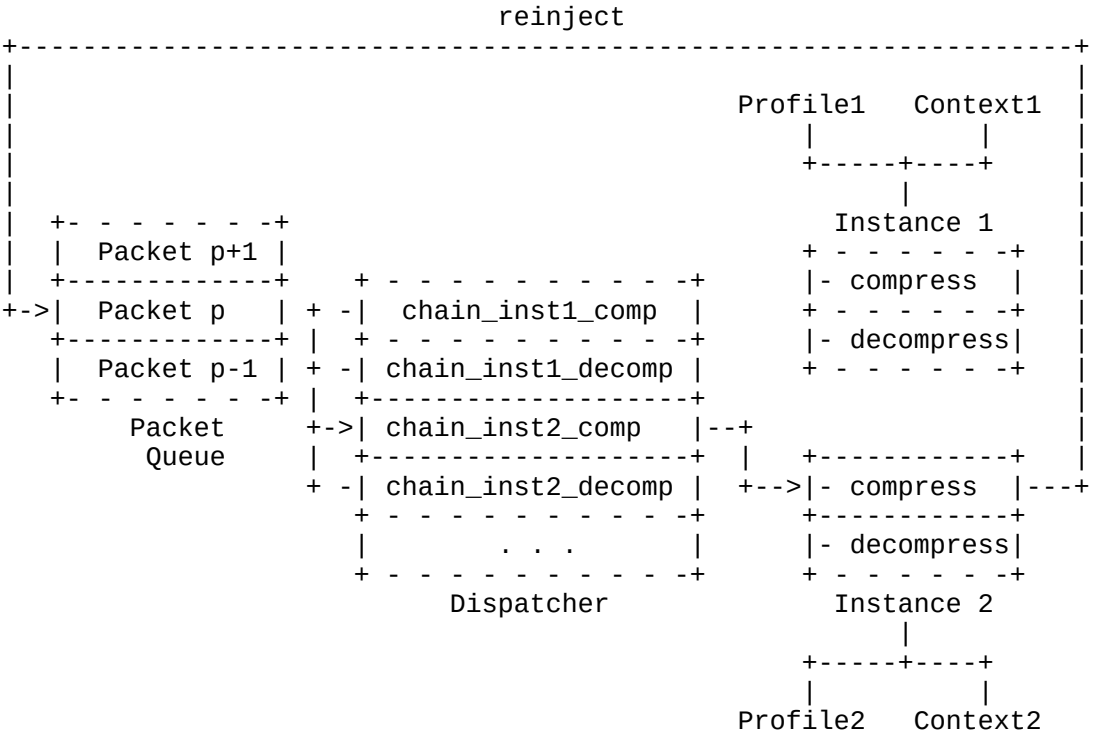


4.7.5. Dispatcher

The Dispatcher is responsible for delivering compressed packets to the correct SCHC Instance. It ensures that the compressed packets are sent to the appropriate destination and that the decompressed packets are delivered to the correct application or protocol routine.

The Dispatcher is a key component that enables the coexistence of multiple SCHC Instances on the same network host, allowing different protocols or applications to use SCHC compression and decompression mechanisms. It also allows regular traffic to coexist with SCHC-compressed traffic.

Dispatcher is illustrated in the figure below, where two SCHC Instances are running on the same Endpoint. The Dispatcher is responsible for routing packets to the appropriate Instance based on admission criteria defined in the Profiles and the Contexts.



There are two types of admission criteria that are used by the Dispatcher:

1. Discriminator values for decompression and reassembly. Those criteria are used to identify packets that should be decompressed and reassembled by SCHC. The Discriminator is a value that is either included in the packet, such as a field in the packet headers (MPLS label, UDP port) or is derived from the metadata associated with the packet, such as the Network Interface ID, as described in [DRAFT-ARCH].

2. Matching Operators and Target Values of the Contexts for compression and fragmentation. Those criteria are used to identify packets that should be compressed and fragmented by SCHC. The Matching Operators and Target Values are defined in the SCHC Contexts and are used to match specific header fields or values in the packet headers. The Dispatcher uses these criteria to determine whether a packet should be compressed or fragmented by a given Instance.

TODO: LINK between discriminator, MO/TV and filter chains. *What follows is WIP, needs to be completed.*

4.7.6. Context Management

Context management is responsible for maintaining the shared state between SCHC entities. This includes:

* Context synchronization between entities

* Rule lifecycle management

* Profile distribution and updates

4.7.7. Context Repository

A Context Repository provides centralized storage and management of SCHC contexts and profiles. While not mandatory for minimal deployments, it becomes essential for larger deployments requiring centralized management.

4.7.8. Management Interface

A Management Interface provides operational control and monitoring capabilities for SCHC deployments. This may include:

* Configuration management

* Performance monitoring

* Troubleshooting tools

5. Security Considerations

Security considerations for SCHC minimal architecture include:

* Context integrity and authenticity

* Profile distribution security

* Protection against context manipulation attacks

6. IANA Considerations

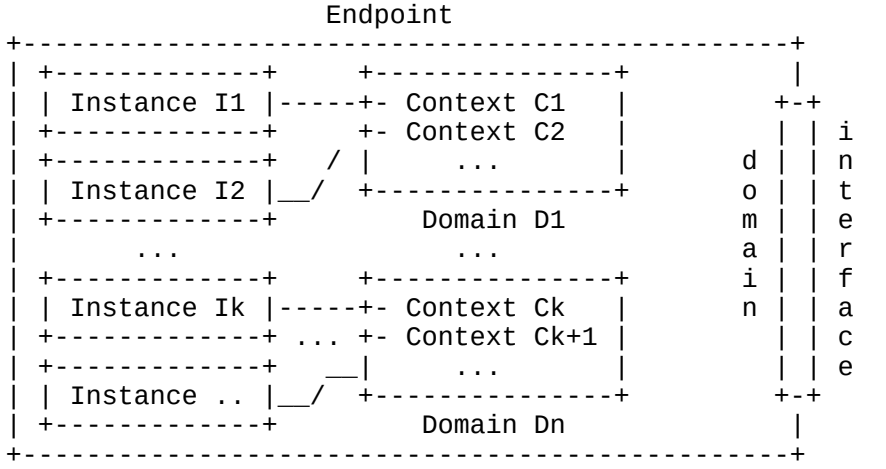
This document has no IANA actions.

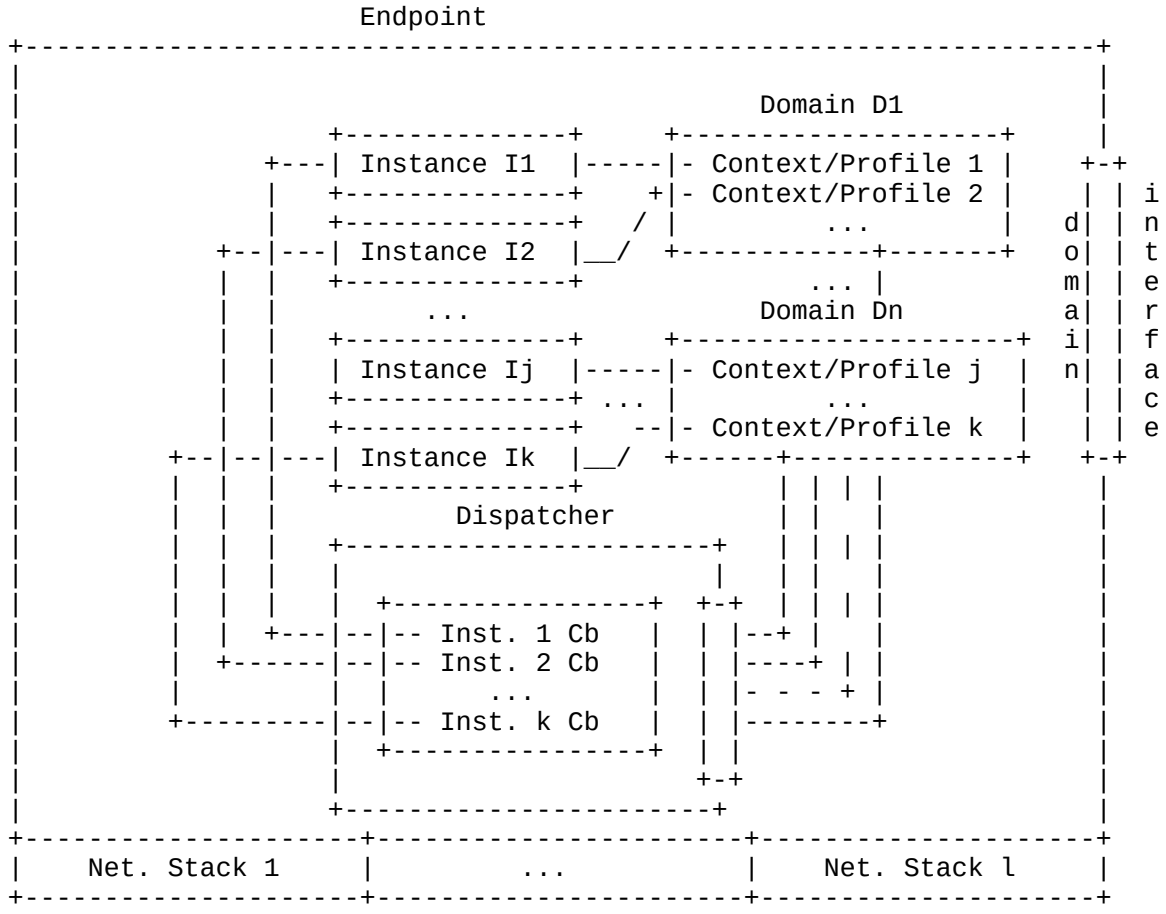
7. Acknowledgments

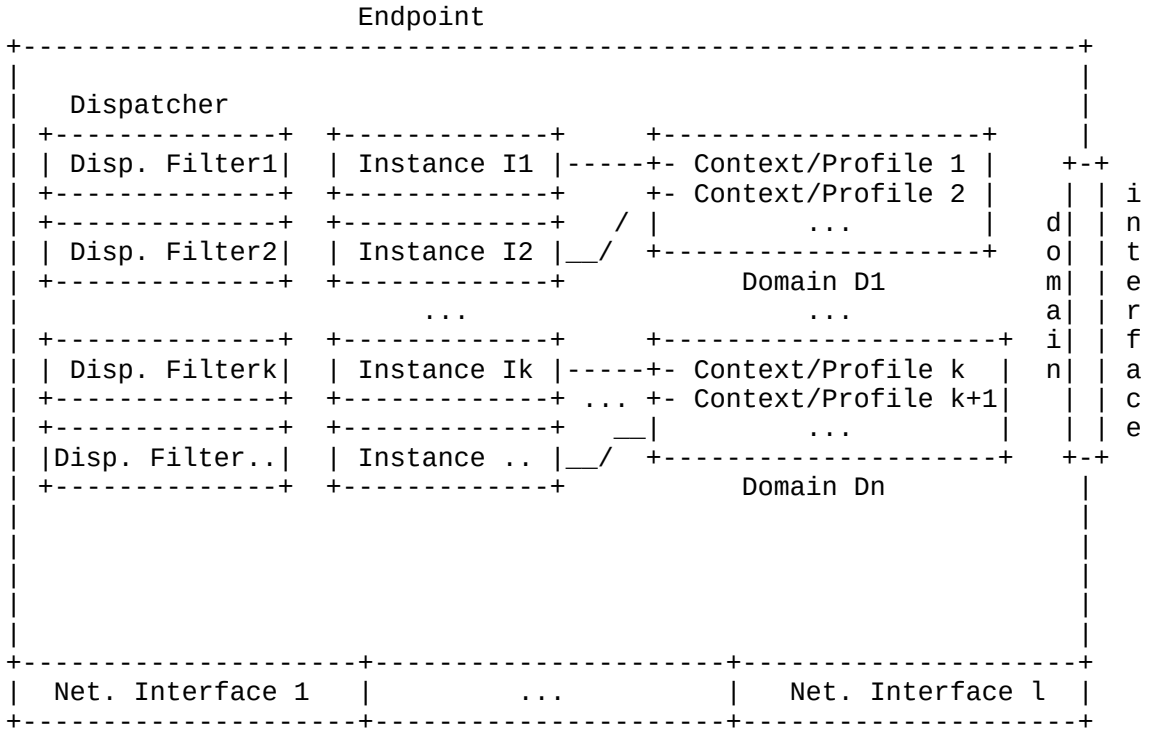
The authors would like to thank the SCHC Working Group for their contributions and feedback on this document.

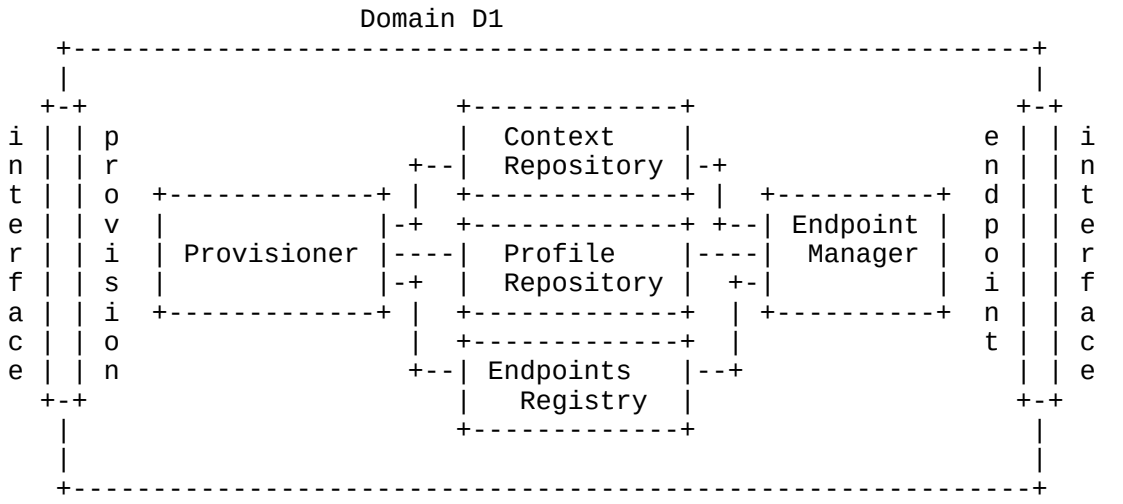
8. Bits and pieces

THIS SECTION IS NOT PART OF THE DOCUMENT, IT IS A COLLECTION OF IDEAS, CONCEPTS AND ILLUSTRATIONS THAT ARE NOT YET FULLY DEVELOPED OR INTEGRATED INTO THE DOCUMENT. IT IS PROVIDED FOR REFERENCE AND DISCUSSION PURPOSES ONLY.



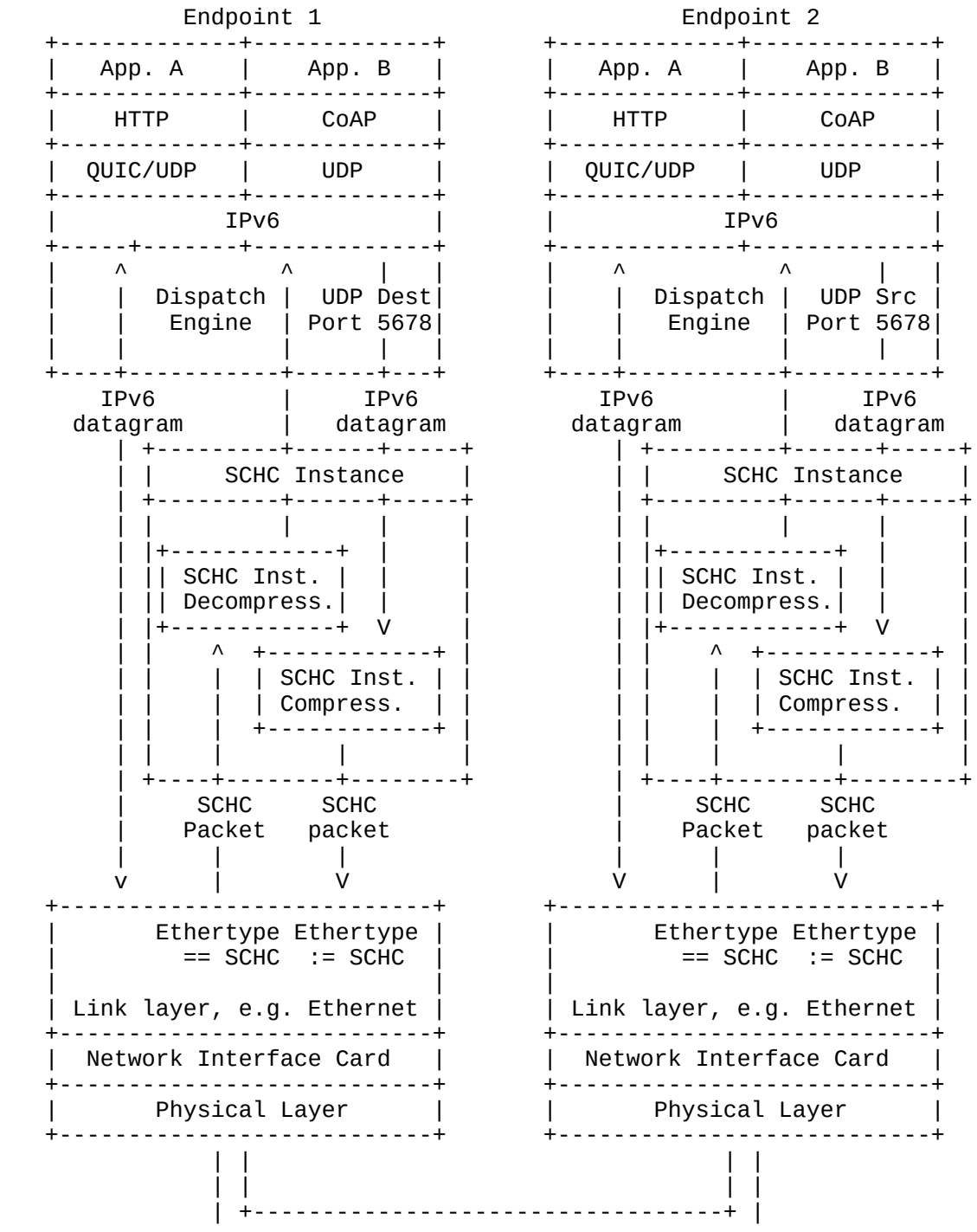

Endpoint
Domain D1
Instance I1
Context/Profile 1
Context/Profile 2
Instance I2
Domain Dn
Instance Ij
Context/Profile j
Instance Ik
Context/Profile k
Dispatcher
Inst. 1 Cb
Inst. 2 Cb
Inst. k Cb
Net. Stack 1
Net. Stack l
Interface





Dispatch scenarios:

Case 1: The Dispatch Engine is integrated into the network stack and a single SCHC Instance is used.


Endpoint 1
App. A
App. B
HTTP
CoAP
QUIC/UDP
UDP
IPv6
Dispatch
Engine
UDP Dest
Port 5678
IPv6
datagram
IPv6
datagram
SCHC Instance
SCHC Inst.
Decompress.
SCHC Inst.
Compress.
SCHC
Packet
SCHC
packet
Ethertype Ethertype
== SCHC := SCHC
Link layer, e.g. Ethernet
Network Interface Card
Physical Layer
Endpoint 2
App. A
App. B
HTTP
CoAP
QUIC/UDP
UDP
IPv6
Dispatch
Engine
UDP Src
Port 5678
IPv6
datagram
IPv6
datagram
SCHC Instance
SCHC Inst.
Decompress.
SCHC Inst.
Compress.
SCHC
Packet
SCHC
packet
Ethertype Ethertype
== SCHC := SCHC
Link layer, e.g. Ethernet
Network Interface Card
Physical Layer

In this simple scenario, the Dispatch Engine is integrated into the network stack and there is a unique predefined SCHC Instance for a specific protocol stack, such as CoAP over UDP over IPv6. This is the classic case for SCHC over LPWAN networks, as described in [RFC8724], [RFC8824], [RFC9363].

The dispatching is done based on a identified header field, such as the an ethertype, the IPv6 Next Header field, a specific UDP port, etc.

This implementation scenario therefore assumes that the endpoint Operating System (OS) implements the SCHC protocol as part of its network stack and that SCHC is allocated the appropriate ethertype, IPv6 Next Header value or UDP port from IANA.

In the example above,

* On Endpoint 1, the Dispatch "intercepts" outbound packets whose UDP destination port is 5678, which is used by CoAP. It then routes these packets to the SCHC Instance for CoAP over UDP over IPv6. The SCHC instance then compresses the CoAP, UDP and IPv6 headers, and calls the Link Layer interface to send the compressed packet over the network, setting the appropriate SCHC ethertype in the link layer header.

* On Endpoint 2, incoming packets whose SCHC ethertype is set to the SCHC value are routed to the SCHC Instance for CoAP over UDP over IPv6. The SCHC Instance decompresses the SCHC packets and delivers them to the IPv6 layer.

Note that in this example, regular HTTP over QUIC traffic is also present on the same Endpoint. The Dispatch Engine is able to discriminate those packets from packets that are compressed by SCHC, as the HTTP over QUIC packets do not not match the admission rules defined in the SCHC profile, here UDP Destination Port == 5678.

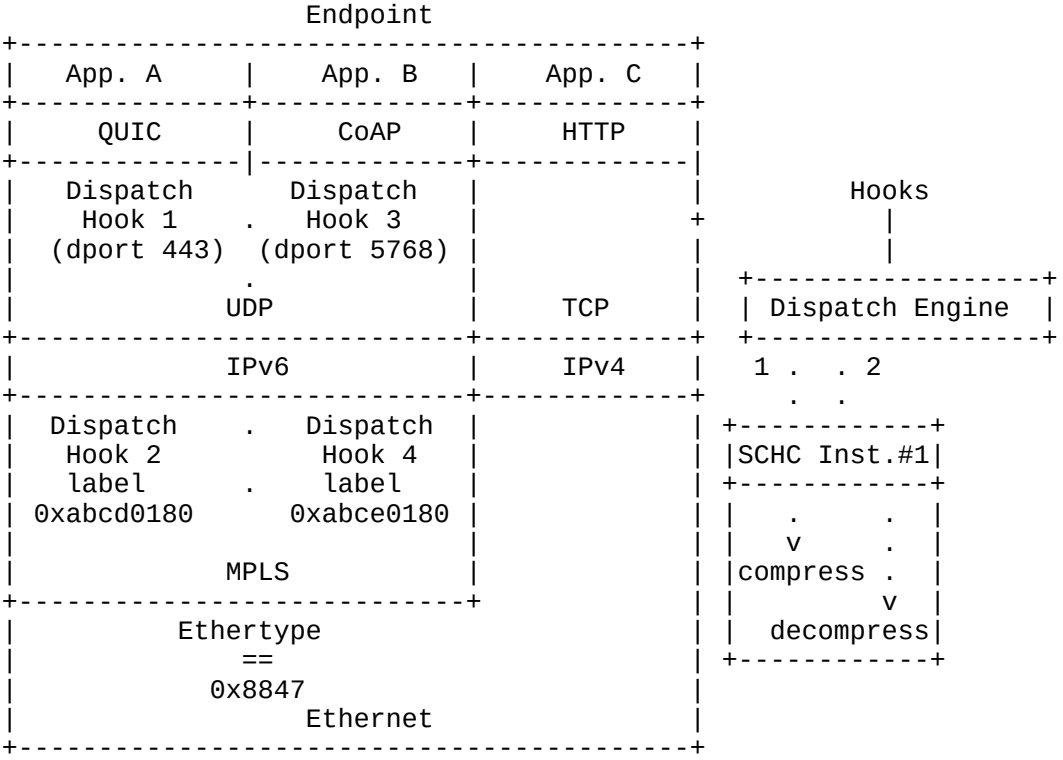
Case 2: The Dispatch engine lives outside of the network stack.

In this case, the Dispatch Engine is a separate component that interacts with multiple SCHC Instances. It is responsible for routing packets to the appropriate SCHC Instance based on the packet type and supplied admission rules.

* On Linux, this can be implemented using netfilter hooks or similar mechanisms to intercept packets and route them to and from the appropriate SCHC Instance.

* On macOS, the Dispatch Engine can be implemented as a kernel extension or user-space application that make use of PF, the native packet filter.

The exact implementation details of the Dispatch Engine will depend on the Operating System, which therefore is not specified in this document. However, a description of packets criteria and admission rules is provided in the SCHC profile, which is used by the Dispatch Engine to determine how to route packets.



In the example above, the Dispatch Engine is implemented as a filter that intercepts packets based on their UDP destination port. In this instance, it routes packets with a destination port of 5768 to the SCHC Instance for CoAP over UDP over IPv6. The Dispatch Engine then compresses the CoAP, UDP, and IPv6 headers, adds a MPLS header with appropriate tag and sends the compressed packet over the network.

When receiving packets, the Dispatch Engine checks the SCHC ethertype and MPLS label and routes matching packets (MPLS label == 0xabcd0180 && UDP destination port == 5768) them to the appropriate SCHC Instance based on the defined admission rules in the profile.

9. References

9.1. Normative References

[DRAFT-ARCH] Pelov, A., Thubert, P., and A. Minaburo, "SCHC Architecture", Work in Progress, Internet-Draft, draft-ietf-schc-architecture-04, February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-schc-architecture-04>.

[DRAFT-CORECONF] Minaburo, A., Toutain, L., Banier, C., and M. Dumay, "CORECONF Rule management for SCHC", Work in Progress, Internet-Draft, draft-toutain-schc-coreconf-management-00, May 2025, <https://datatracker.ietf.org/doc/html/draft-toutain-schc-coreconf-management-00>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

9.2. Informative References

[RFC9363] Minaburo, A. and L. Toutain, "A YANG Data Model for Static Context Header Compression (SCHC)", RFC 9363, DOI 10.17487/RFC9363, March 2023, <https://www.rfc-editor.org/info/rfc9363>.

Appendix A. Change Log

A.1. Changes from -00 to -01

* Initial version

Authors' Addresses

Quentin Lampin
Orange
Orange 3 Massifs - 22 Chemin du Vieux Chene
38240 Meylan
France
Email: quentin.lampin@orange.com

Marion Dumay
Orange
Orange 3 Massifs - 22 Chemin du Vieux Chene
38240 Meylan
France
Email: marion.dumay@orange.com

Philippe Surbayrole
Orange
Orange 3 Massifs - 22 Chemin du Vieux Chene
38240 Meylan
France
Email: philippe.surbayrole@orange.com