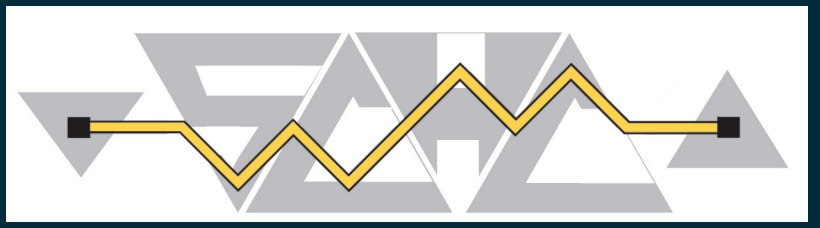
SCHClets: A Modular Approach to SCHC



draft-pelov-schclet-architecture

Alexander Pelov

Internet Engineering Task Force
© 2022 IETF Trust
Production by Meetecho

IETF®

What is a SCHClet?

- Objective: Enable use of the SCHC Framework everywhere (e.g. Diet-ESP), avoiding monolithic approach (e.g. all-or-nothing).
  - A technology which uses a SCHClet can be considered as built using the SCHC Framework. It is not necessary to have a Full SCHC Implementation.
- A SCHClet is a modular, atomic sub-function of SCHC
  - Inspired by chiplet design in integrated circuits
  - Encapsulates specific SCHC functionality (e.g., compression, fragmentation).
  - Self-contained and independently deployable.

- SCHClet characteristics
  - Implements a specific SCHC operation (e.g., NoAck Fragmentation)
  - Can operate with fixed parameters or be configurable

Formal definition

- Full SCHC Implementation: An implementation that covers all mandatory aspects of SCHC as defined in RFC8724, potentially extended by additional functionalities introduced in subsequent/related RFCs.

- Full Configuration: The set of SCHC Profiles/configurations/parameters supported by a Full SCHC Implementation.

A SCHClet MUST be able to interoperate with a Full SCHC Implementation with the appropriate configuration of the latter.

IETF

Formal definition

- Full SCHC Implementation: An implementation that covers all mandatory aspects of SCHC as defined in RFC8724, potentially extended by additional functionalities introduced in subsequent/related RFCs.
- Full Configuration: The set of SCHC Profiles/configurations/parameters supported by a Full SCHC Implementation.

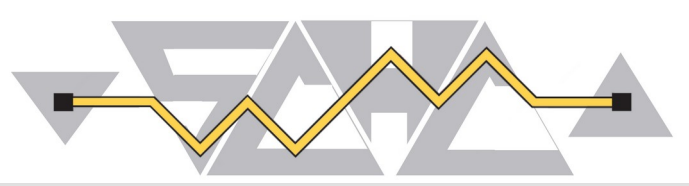
A SCHClet MUST be able to interoperate with a Full SCHC Implementation with the appropriate configuration of the latter.

SCHClet: A piece of software/hardware that can interoperate with a Full SCHC Implementation under with a specific configuration of the latter. The set of supported configurations by this software/hardware is called SCHClet Configuration.

➔ A SCHClet is therefore fully defined by its corresponding SCHClet Configuration.

IETF

Simplifications Enabled by SCHClets



- Simplifications Enabled by SCHClets
  - Simplification of the use of SCHC Framework
  - Targeted Functionality – Embed only necessary code
  - Can use fragmentation without compression
  - Can use compression without No-compression rule (otherwise required per RFC8724)
- SCHClet examples
  - Compression SCHClet: all compression configurations of RFC8724
  - NoAck Fragmentation SCHClet: all supported parameters
  - NoAck Fragmentation SCHClet, with one RuleID=8 bits, DTag size=0 bits, RCS =CRC32 polynomial 0xEDB88320, Inactivity Timer=12h
  - Ack-On-Err Fragmentation SCHClet: all supported parameters
  - Ack-On-Err Fragmentation SCHC, with the following parameters: RuleID size=3 bits, DTag size=0 bits, Window Index size=2 bits, FCN size=3 bits, MAX_ACK_REQUESTS=5, WINDOW_SIZE=7, Regular tile size=11 bytes, All-1 tile size=0 to 10 bytes, Retransmission Timer=12h, Inactivity Timer=12h, RCS size=3 bits

IETF®

Example: Minimal Fixed-Field Compression SCHClet

The goal of this example is to show how a SCHClet may be a single, constant-time function, of extreme simplicity - yet still interoperable with a Full SCHC Implementation.

This SCHClet targets the first four bytes of the IPv6 header, which correspond to:

- Version (4 bits): 6
- Traffic Class (8 bits): 0
- Flow Label (20 bits): 0

To simplify the example, we use a RuleID of length 8 bits, with two rules:

- RuleID 01100000 (0x60) - "pass-through", which elides the first byte (which is 0x60), but then adds the ruleID (which is 0x60), effectively making the compression operation a no-operation.
- RuleID 11111111 (0xFF) - "compression", which elides the first 4 bytes

SCHClet configuration – fully defines the SCHClet

```
{
  "schc": {
    "rule": [
      {
        "rule-id-value": 0x60,
        "rule-id-length": 8,
        "nature-compression": {
          "field": [
            {
              "field-id": "fid-ipv6-version",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 6
              },
              "matching-operator": "mo-equal",
              "comp-decomp-action": "cda-not-sent"
            },
            {
              "field-id": "fid-ipv6-trafficclass",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 0
              },
              "matching-operator": "mo-msb",
              "matching-operator-value":
              {
                  "index": 0,
                  "value": 4
              },
              "comp-decomp-action": "cda-lsb",
              "comp-decomp-action-value":
              {
                  "index": 0,
                  "value": 4
              }
            },
            {
              "field-id": "fid-ipv6-flowlabel",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 0
              },
```

```
"matching-operator": "mo-equal",
              "comp-decomp-action": "cda-value-sent"
            }
          ]
        }
      },
      {
        "rule-id-value": 0xFF,
        "rule-id-length": 8,
        "nature-compression": {
          "field": [
            {
              "field-id": "fid-ipv6-version",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 6
              },
              "matching-operator": "mo-equal",
              "comp-decomp-action": "cda-not-sent"
            },
            {
              "field-id": "fid-ipv6-trafficclass",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 0
              },
              "matching-operator": "mo-equal",
              "comp-decomp-action": "cda-not-sent"
            },
            {
              "field-id": "fid-ipv6-flowlabel",
              "field-position": 1,
              "direction": "bi",
              "target-value": {
                  "index": 0,
                  "value": 0
              },
              "matching-operator": "mo-equal",
              "comp-decomp-action": "cda-not-sent"
            }
          ]
        }
      }
    ]
  }
}
```

SCHClet C implementation – “The SCHClet itself”

```c
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#define RULE_ID_PASSTHROUGH 0x60 // 01100000
#define RULE_ID_COMPRESSED 0xFF // 11111111

// Compress packet in-place or pass through, based on content.
// Returns pointer to compressed packet and sets output length.
const uint8_t* compress_ipv6(uint8_t *packet, size_t in_len, size_t *out_len)
{
	// Match compressible pattern (Version, TC, FL = 0)
	if (in_len >= 4 && packet[0] == 0x60 && packet[1] == 0x00 &&
	packet[2] == 0x00 && packet[3] == 0x00) {
	// We "compress" by replacing the 4th byte with RuleID
	packet[3] = RULE_ID_COMPRESSED;
	*out_len = in_len - 3;
	return &packet[3];
	}

	// Pass-through: return original packet, unchanged
	*out_len = in_len;
	return packet;
}
```

```c
size_t decompress_ipv6(const uint8_t *compressed, size_t comp_len, uint8_t *out_buf,
size_t out_max) {
	if (comp_len == 0 || out_max < comp_len + 3) return 0;

	if (compressed[0] == RULE_ID_COMPRESSED) {
	// Insert fixed prefix before compressed payload
	out_buf[0] = 0x60;
	out_buf[1] = 0x00;
	out_buf[2] = 0x00;
	out_buf[3] = 0x00;
	memcpy(out_buf + 4, compressed + 1, comp_len - 1);
	return comp_len - 1 + 4;
	}

	// Rule ID is not compressed — assume passthrough
	memcpy(out_buf, compressed, comp_len);
	return comp_len;
}
```

IETF

SCHClet C implementation – running code

```
int main() {
	uint8_t packet[40] = {
	0x60, 0x00, 0x00, 0x00, // Compressible prefix
	0x3A, 0x40 // Next header, hop limit...
	};
	size_t in_len = 40;
	size_t comp_len;

	const uint8_t *compressed = compress_ipv6(packet, in_len, &comp_len);
	if (!compressed) {
	printf("Compression failed.\n");
	return 1;
	}

	printf("Compressed length: %zu\n", comp_len);
	printf("Compressed first byte: 0x%02X\n", compressed[0]);

	uint8_t restored[64];
	size_t restored_len = decompress_ipv6(compressed, comp_len, restored,
	sizeof(restored));

	printf("Decompressed first 4 bytes: %02X %02X %02X %02X\n",
	restored[0], restored[1], restored[2], restored[3]);

	return 0;
}
```

IETF

The way forward

- Enable the SCHC technology to be used in a simpler, more flexible and wider manner
- Standard-track document, which provides formal definition and way to verify interoperability
- Interest to collaborate / contribute ?
- WG adoption ?