Groupe de travail Réseau
Request for Comments : 4997
Catégorie : sur la voie de la normalisation
Traduction Claude Brière de L'Isle

R. Finking, Siemens/Roke Manor Research
G. Pelletier, Ericsson
juillet 2007

Notation formelle pour la compression d'en-tête robuste (ROHC-FN)

Statut du présent mémoire
Le présent document spécifie un protocole de l'Internet sur la voie de la normalisation pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. Prière de se référer à l'édition en cours des "Protocoles officiels de l'Internet" (STD 1) pour voir l'état de normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Notice de Copyright
Copyright (C) The IETF Trust (2007).

Résumé
Le présent document définit la notation formelle de compression d'en-tête robuste (ROHC-FN, *Robust Header Compression - Formal Notation*) une notation formelle de spécification des codages de champs pour les formats compressés quand on définit de nouveaux profils au sein du cadre de ROHC. ROHC-FN offre une bibliothèque de méthodes de codages qui sont souvent utilisés dans les profils ROHC qui peut ainsi aider à simplifier de futurs travaux de développement de profils.

Table des Matières

1. Introduction

La notation formelle de compression d'en-tête robuste (ROHC-FN, *Robust Header Compression - Formal Notation*) est une notation formelle destinée à aider à la définition de profils de compression d'en-tête ROHC [RFC4995]. Les profils précédents de compression d'en-tête ont été jusqu'à présent spécifiés en utilisant une combinaison de texte anglais avec une notation de boîte ASCII. Malheureusement, c'est parfois peu clair et ambigu, révélant les limitations de la définition de structures et codages complexes pour les formats compressés de cette façon. Le principal objectif de la notation formelle est de fournir un moyen plus rigoureux de définir les formats d'en-tête -- compressé et non compressé -- ainsi que les relations entre eux. Aucune autre notation formelle n'existe qui satifasse ces exigences, de sorte que ROHC-FN vise à les satisfaire.

De plus, ROHC-FN offre une bibliothèque de méthodes de codage qui sont souvent utilisées dans les profils ROHC, de sorte que la spécification de nouveaux profils utilisant la notation formelle peut être réalisée sans avoir à redéfinir cette bibliothèque à partir de rien. De façon informelle, une méthode de codage définit une transposition bijective entre les données non compressées et les données compressées.

2. Terminologie

Bibliothèque de méthodes de codage : elle contient un certain nombre de méthodes de codage couramment utilisées pour compresser des champs d'en-tête.

Champ : l'en-tête de protocole est divisé en un ensemble de schémas de bits contigus appelés des champs. Chaque champ est défini par une collection d'attributs qui indiquent sa valeur et sa longueur en bits pour les en-têtes compressés et non compressés. La façon dont l'en-tête est divisé en champs est spécifique de la définition d'un profil, et il n'est pas nécessaire que les divisions de champs soient identiques à celles données par la ou les spécifications pour l'en-tête de protocole qui est compressé.

Champ de contrôle : les champs de contrôle sont transmis d'un compresseur ROHC à un décompresseur ROHC, mais ne font pas partie de l'en-tête non compressé lui-même.

Contexte : ce sont des informations sur l'état actuel de (dé)compression du flux. Précisément, un contexte pour un champ spécifique peut être non initialisé, ou il peut inclure un ensemble d'une ou plusieurs valeurs pour les attributs du champ définies par l'algorithme de compression, où une valeur peut venir des attributs du champ correspondant à un paquet précédent. Voir aussi une définition plus générale au paragraphe 2.2 de la [RFC4995].

Format compressé : un format compressé consiste en une liste de champs qui fournissent des liens entre les codages et les champs qu'ils compressent. Un ou plusieurs formats compressés peuvent être combinés pour représenter un format entier d'en-tête compressé.

Format non compressé : il consiste en une liste des champs qui donnent l'ordre des champs à compresser pour un ensemble contigu de bits dont la disposition correspond à l'en-tête de protocole à compresser.

Méthodes de codage, codages : les méthodes de codage sont des relations bidirectionnelles qui peuvent être appliquées pour compresser et décompresser les champs d'un en-tête de protocole.

Profil : un profil ROHC [RFC4995] est une description de la façon de compresser une certaine pile de protocoles. Chaque profil consiste en un ensemble de formats (par exemple, formats compressés et non compressés) avec un ensemble de règles qui contrôlent le comportement du compresseur et du décompresseur.

Spécification ROHC-FN : spécification de l'ensemble des formats d'un profil ROHC utilisant ROHC-FN.

3. Vue d'ensemble de ROHC-FN

Cette Section donne une vue d'ensemble de ROHC-FN. Elle explique aussi comment ROHC-FN peut être utilisé pour spécifier la compression des champs d'en-tête au titre d'un profil ROHC.

3.1 Portée de la notation formelle

Ce paragraphe explique comment la notation formelle se rapporte au cadre ROHC et aux spécifications de profils ROHC.

Le cadre ROHC [RFC4995] donne les principes généraux pour effectuer une compression d'en-tête robuste. Il définit le concept de profil, qui fait de ROHC une plate-forme générale pour différents schémas de compression. Il établit les exigences de la couche de liaison, et en particulier les exigences de négociation, pour tous les profils ROHC. Il définit un ensemble de fonctions communes comme les identifiants de contexte (CID, *Context Identifier*) le bourrage, et la segmentation. Il définit aussi les formats communs (IR, IR-DYN, Feedback, Add-CID, etc.) et finalement, il définit un mécanisme de rétroaction générique, indépendant du profil.

Un profil ROHC est une description de la façon de compresser une certaine pile de protocole. Par exemple, les profils ROHC sont disponibles pour RTP/UDP/IP et de nombreuses autres piles de protocole.

D'une façon très générale, chaque profil ROHC consiste en un ensemble de formats (définissant les bits à transmettre) ainsi qu'un ensemble de règles qui contrôlent le comportement du compresseur et du décompresseur. L'objet des formats est de définir comment compresser et décompresser les en-têtes. Les formats définissent une ou plusieurs versions compressées de chaque en-tête non compressé, et simultanément, définissent l'inverse : comment remettre un en-tête compressé dans sa forme originale d'en-tête non compressé.

L'ensemble des formats va normalement définir la compression des en-têtes par rapport à un contexte de valeurs de champs à partir des en-têtes précédents dans un flux, améliorant la compression globale en tenant compte des redondances entre les en-têtes des paquets successifs. Donc, en plus de définir les formats, un profil doit :
- o spécifier comment gérer le contexte pour le compresseur et le décompresseur,
- o définir quand et quoi envoyer dans les messages de rétroaction, si il en est, du décompresseur au compresseur,
- o souligner les principes de compression pour rendre le profil robuste aux erreurs binaires et aux paquets abandonnés.

Tout cela est nécessaire pour assurer que les contextes du compresseur et du décompresseur restent cohérents l'un avec l'autre, tout en facilitant quand même les meilleures performances de compression possibles.

ROHC-FN est conçu pour aider à la spécification de formats compressés qui, lorsque mis ensemble sur la base de la définition de profil, constituent les formats utilisés dans un profil ROHC. Il offre une bibliothèque de méthodes de codage pour compresser les champs, et un mécanisme pour combiner ces méthodes de codage pour créer des formats compressés sur mesure pour une pile de protocoles spécifique.

La portée de ROHC-FN est limitée à la spécification des relations entre les formats compressés et non compressés. Pour former une spécification de profil complète, la logique de contrôle pour le comportement du profil doit être définie par d'autres moyens.

3.2 Fondamentaux de la notation formelle

Il y a deux éléments fondamentaux pour la notation formelle :
1. Les champs et leurs codages, qui définissent la transposition entre la forme non compressée et compressée d'un en-tête.
2. Les méthodes de codage, qui définissent la façon dont les en-têtes sont coupés dans les champs. Les méthodes de codage définissent les listes des champs non compressés et les listes de champs compressés en lesquels elles se transposent.

Ces deux éléments fondamentaux sont au cœur de la notation et sont précisés ci-dessous.

3.2.1 Champs et codages

Les en-têtes sont constitués de champs. Par exemple, numéro de version, longueur d'en-tête, et numéro de séquence sont tous des champs utilisés dans les protocoles réels.

Les champs ont des attributs. Les attributs décrivent diverses choses sur le champ. Par exemple :

field.ULENGTH

Ceci indique la longueur non compressés du champ. Un champ est dit avoir un attribut de valeur, c'est-à-dire, une valeur compressée ou une valeur non compressée, si l'attribut Longueur correspondant est supérieur à zéro. Voir au paragraphe 4.4 les détails des attributs des champs.

La relation entre les attributs compressés et non compressés d'un champ est spécifiée par les méthodes de codage, en utilisant la notation suivante :

field =:= encoding_method; *(champ =:= méthode de codage)*

Dans la définition de champ ci-dessus, le symbole "=:=" signifie "est codé par". Cette définition de champ ne représente pas une opération d'allocation du côté droit au côté gauche. Elle est plutôt une transposition bijective entre les attributs compressés et non compressés du champ. Elle représente à la fois l'opération de compression et de décompression dans une seule définition de champ, par un processus de correspondance bidirectionnelle.

La correspondance bidirectionnelle est une opération binaire qui tente de rendre les opérandes (c'est-à-dire, les attributs compressés et non compressés) correspondants. Ceci est similaire en logique au processus d'unification. Les opérandes représentent un objet de données non spécifié et un objet spécifié. Les valeurs peuvent être confrontées à partir de l'un ou l'autre opérande.

Durant la compression, les attributs non compressés du champ sont déjà définis. Le codage donné leur confronte les attributs compressés. Durant la décompression, les attributs compressés du champ sont déjà définis, de sorte que les attributs non compressés sont confrontés aux attributs compressés en utilisant la méthode de codage donnée. Donc, la compression et la décompression sont toutes deux définies par une seule définition de champ.

Donc, une méthode de codage (incluant tous les paramètres spécifiés) crée un lien réversible entre les attributs d'un champ. Au compresseur, un format peut être utilisé si un ensemble de liens qui est réussi pour tous les attributs dans tous ses champs peut être trouvé. Au décompresseur, l'opération est inversée en utilisant les mêmes liens et les attributs dans chaque champ sont remplis en accord avec les liens spécifiés ; le décodage échoue si le lien pour un attribut échoue.

Par exemple, la méthode de codage "static" crée un lien entre l'attribut correspondant à la valeur non compressée du champ et la valeur correspondante du champ dans le contexte.

- Pour le compresseur, le lien "static" est réussi quand la valeur de contexte et la valeur non compressée sont les mêmes. Si les deux valeurs diffèrent, alors le lien échoue.

- Pour le décompresseur, le lien "static" ne réussit que si une entrée de contexte valide contenant la valeur du champ non compressé existe. Autrement, le lien va échouer.

Les deux formes compressée et non compressée de chaque champ sont représentées comme une chaîne de bits ; le bit de poids fort en premier, de la longueur spécifiée par l'attribut Longueur. La chaîne de bits est la représentation binaire de l'attribut Valeur du champ , modulo "2^longueur", où "longueur" est l'attribut Longueur du champ. Cependant, c'est seulement la représentation des bits échangés entre le compresseur et le décompresseur, conçue pour permettre une efficacité maximum de compression. La FN utilise elle-même la gamme complète des entiers. Voir au paragraphe 4.4.2 d'autres détails.

3.2.2 Formats et méthodes de codage

ROHC-FN fournit une bibliothèque de méthodes de codage couramment utilisées. Les méthodes de codage peuvent être définies en utilisant l'anglais, ou en utilisant une définition formelle consistant, par exemple, en une collection d'expressions (paragraphe 4.7) et des déclarations "ENFORCE" (paragraphe 4.9).

ROHC-FN fournit aussi des mécanismes pour combiner les champs et leurs méthodes de codage en méthodes de codage de

niveau supérieur suivant une structure bien définie. Ceci est similaire à la définition des fonctions et procédures dans un langage de programmation ordinaire. Cela permet de traiter la complexité en la divisant en parties gérables. De nouvelles méthodes de codage sont définies au niveau supérieur d'un profil. Cela peut alors être utilisé dans la définition d'autres méthodes de codage de niveau supérieur, et ainsi de suite.

```
nouvelle_méthode_de_codage              //Ce bloc est une méthode de codage
{
 NON COMPRESSÉ {                        // Ce bloc est un format non compressé
   field_1   [ 16 ];
   field_2   [ 32 ];
   field_3   [ 48 ];
 }

 CONTROLE {                             // Ce bloc définit les champs de contrôle
     ctrl_field_1;
     ctrl_field_2;
   }

 DEFAUT {                               // Ce bloc définit les codages par défaut pour les champs spécifiés
   ctrl_field_2 =:= méthode_de_codage_2;
   field_1      =:= méthode_de_codage_1;
 }

 COMPRESSED format_0 {                  // Ce bloc est un format compressé
   field_1;
   field_2 =:= méthode_de_codage_2;
   field_3 =:= méthode_de_codage_3;
   ctrl_field_1 =:= méthode_de_codage_4;
   ctrl_field_2;
 }

 COMPRESSED format_1 {                  // Ce bloc est un format compressé
   field_1;
   field_2 =:= méthode_de_codage_3;
   field_3 =:= méthode_de_codage_4;
   ctrl_field_2 =:= méthode_de_codage_5;
   ctrl_field_3 =:= méthode_de_codage_6;          // C'est un champ de contrôle sans valeur non compressée
 }
}
```

Dans l'exemple ci-dessus, la méthode de codage définie est appelée "nouvelle_méthode_de_codage". La section intitulée "UNCOMPRESSED" indique l'ordre des champs dans l'en-tête non compressé, c'est-à-dire, le format de l'en-tête non compressé. Le nombre de bits dans chacun des champs est indiqué entre crochets. Après cela se trouve une autre section, "CONTROL", qui définit deux champs de contrôle. Ensuite est la section "DEFAULT" qui définit les méthodes de codage par défaut pour deux des champs (voir ci-dessous). Finalement, deux formats compressés alternatifs suivent, chacun défini dans les sections intitulées "COMPRESSED". Les champs qui se présentent dans les formats compressés sont soit :

- o les champs qui se présentent dans le format non compressé ; soit
- o les champs de contrôle qui ont une valeur non compressée et qui se présentent dans la section "CONTROL" ; soit
- o les champs de contrôle qui n'ont pas une valeur non compressée et donc sont définis au titre du format compressé.

Au centre de chacun de ces formats est une "liste des champs", qui définit les champs contenus dans le format et aussi l'ordre dans lequel ces champs apparaissent dans ce format. Pour les section "DEFAULT" et "CONTROL", l'ordre des champs n'est pas significatif.

En plus de spécifier l'ordre des champs, la liste des champs peut aussi spécifier des liens pour un ou tous les champs qu'elle contient. Les champs qui n'ont pas de liens définis pour eux en sont réduits à utiliser les liens par défaut spécifiés dans la section "DEFAULT" (voir le paragraphe 4.12.1.5).

Les champs venant du format compressé ont le même nom que celui qu'ils avaient dans le format non compressé. Si il y a des champs qui sont présents exclusivement dans le format compressé, mais ont bien une valeur non compressée, ils doivent être déclarés dans la section "CONTROL" de la définition de la méthode de codage (voir au paragraphe 4.12.1.3 les

détails de la définition des champs de contrôle).

Les champs qui n'ont pas de valeur non compressée n'apparaissent pas dans une liste de champs "UNCOMPRESSED" et n'ont pas à apparaître non plus dans la liste de champs "CONTROL". Il sont seulement déclarés dans les listes de champs compressés où ils sont utilisés.

Dans l'exemple ci-dessus, tous les champs qui apparaissent dans le format compressé se trouvent aussi dans le format non compressé, ou la liste des champs de contrôle, sauf ctrl_field_3 ; ceci est possible parce que ctrl_field_3 n'a pas de valeur "non compressée" du tout. Les champs comme une somme de contrôle dans les informations compressées entrent dans cette catégorie.

3.3 Exemple avec IPv4

Cette section donne une vue d'ensemble de la façon d'utiliser la notation au moyen d'un exemple. L'exemple va développer la notation formelle pour une méthode de codage capable de compresser un seul en-tête bien connu : l'en-tête IPv4 [RFC0791].

La première étape est de spécifier la structure globale de l'en-tête IPv4. Pour ce faire, on utilise une méthode de codage qu'on va appeler "ipv4_en-tête". Des détails sur les définitions des méthodes de codage se trouvent au paragraphe 4.12. La notation est la suivante :

```
ipv4_en-tête
{
```

Le fragment de notation ci-dessus déclare la méthode de codage "ipv4_en-tête", la définition suit l'accolade d'ouverture (voir au paragraphe 4.12).

Les définitions dans la paire d'accolades sont locales pour "ipv4_en-tête". Ce mécanisme de portée aide à clarifier quels champs appartiennent à quels formats ; il est aussi utile lors de la compression de piles de protocoles complexes avec plusieurs en-têtes, souvent avec les mêmes noms de champ survenant dans plusieurs en-têtes (voir au paragraphe 4.2).

L'étape suivante est de spécifier les champs contenus dans l'en-tête IPv4 non compressé pour représenter le format non compressé pour lequel la méthode de codage va définir un ou plusieurs formats compressés. Ceci est accompli en utilisant ROHC-FN comme suit :

```
UNCOMPRESSED {
  version          [  4 ];
  header_length    [  4 ];
  dscp             [  6 ];
  ecn              [  2 ];
  length           [ 16 ];
  id               [ 16 ];
  reserved         [  1 ];
  dont_frag        [  1 ];
  more_fragments   [  1 ];
  offset           [ 13 ];
  ttl              [  8 ];
  protocol         [  8 ];
  checksum         [ 16 ];
  src_addr         [ 32 ];
  dest_addr        [ 32 ];
}
```

La largeur de chaque champ est indiquée entre crochets. Cette partie de la notation est utilisée dans l'exemple pour illustration afin d'aider le lecteur à comprendre. Cependant, indiquer les longueurs de champ de cette façon est facultatif car la longueur de chaque champ peut aussi normalement être déduite du codage utilisé pour le compresser/décompresser pour un format spécifique. Cette partie de la notation est formellement définie au paragraphe 4.10.

L'étape suivante est de spécifier le format compressé. Cela inclut les codages pour chaque champ qui se transpose entre les formes compressées et non compressées du champ. Dans l'exemple, ces méthodes de codage sont principalement tirées de

la bibliothèque ROHC-FN (voir au paragraphe 4.11). Comme l'intention est ici d'illustrer l'utilisation de la notation, plutôt que de décrire la méthode optimale de compression des en-têtes IPv4, cet exemple utilise seulement trois méthodes de codage.

La méthode de codage "uncompressed_value" (définie au paragraphe 4.11.1) peut compresser tout champ dont la longueur et la valeur non compressées sont fixées, ou peuvent être calculées en utilisant une expression. Aucun bit compressé n'a besoin d'être envoyé parce que le champ non compressé peut être reconstruit en utilisant sa taille et valeur connues. La méthode de codage "uncompressed_value" est utilisée pour compresser cinq champs dans l'en-tête IPv4, comme décrit ci-dessous :

```
COMPRESSED {
  header_length      =:= uncompressed_value(4, 5);
  version            =:= uncompressed_value(4, 4);
  reserved           =:= uncompressed_value(1, 0);
  offset             =:= uncompressed_value(13, 0);
  more_fragments     =:= uncompressed_value(1, 0);
```

Le premier paramètre indique la longueur du champ non compressé en bits, et le second paramètre donne sa valeur entière.

Noter que l'ordre des champs dans le format compressé est indépendant de l'ordre des champs dans le format non compressé.

La méthode de codage "irregular" (définie au paragraphe 4.11.3) peut être utilisée pour coder tout champ pour lequel les deux attributs non compressés (ULENGTH et UVALUE) sont définis, et dont l'attribut ULENGTH est fixé ou peut être calculé en utilisant une expression. C'est une méthode de codage sûre qui peut être utilisée pour de tels champs dans le cas où aucune autre méthode de codage ne s'applique. Tous les bits en forme non compressée du champ sont aussi présents dans la forme compressée ; donc, ce codage ne réalise aucune compression.

```
  src_addr  =:= irregular(32);
  dest_addr =:= irregular(32);
  length    =:= irregular(16);
  id        =:= irregular(16);
  ttl       =:= irregular(8);
  protocol  =:= irregular(8);
  dscp      =:= irregular(6);
  ecn       =:= irregular(2);
  dont_frag =:= irregular(1);
```

Finalement, la troisième méthode de codage est spécifique pour le seul format non compressé défini ci-dessus pour l'en-tête IPv4, "inferred_ip_v4_header_checksum":

```
    checksum      =:= inferred_ip_v4_header_checksum [ 0 ];
  }
}
```

La méthode de codage "inferred_ip_v4_header_checksum" est différente des deux autres méthodes de codage en ce qu'elle n'est pas définie dans la bibliothèque ROHC-FN des méthodes de codage. Sa définition pourrait être donnée soit en utilisant la notation formelle au titre de la définition de profil elle-même (voir au paragraphe 4.12) soit en utilisant du texte anglais en clair (voir au paragraphe 4.13).

Dans notre exemple, la "inferred_ip_v4_header_checksum" est une méthode de codage spécifique qui calcule la somme de contrôle IP à partir du reste des valeurs de l'en-tête. Comme dans la méthode de codage "uncompressed_value", aucun bit compressé n'a besoin d'être envoyé, car la valeur du champ peut être reconstruite au décompresseur. Ceci est noté explicitement en spécifiant, entre crochets, une longueur de 0 pour le champ Somme de contrôle dans le format compressé. Là encore, cette notation est facultative car la méthode de codage elle-même va être définie comme envoyant des bits zéro compressés, cependant il est utile pour le lecteur d'inclure cette notation (voir au paragraphe 4.10 les détails sur cette partie de la notation).

Finalement, la définition du format est terminée par une accolade de clôture. À ce point, l'exemple ci-dessus a défini un format compressé qui peut être utilisé pour représenter l'en-tête IPv4 compressé entier, et fournit assez d'informations pour permettre à une mise en œuvre de construire le format compressé à partir d'un format non compressé (compression) et vice

versa (décompression).

4. Définition normative de ROHC-FN

Cette Section donne la définition normative de ROHC-FN. ROHC-FN est un langage déclaratif qui est référentiellement transparent, sans effets secondaires. Cela signifie que chaque fois qu'une expression est évaluée, il n'y a pas d'autres effets de l'obtention de la valeur de l'expression ; il est donc garanti que la même expression va avoir la même valeur chaque fois qu'elle apparaît dans la notation, et elle peut toujours être échangée avec sa valeur dans tous les formats où elle apparaît (sous réserve des règles de portée des identifiants au paragraphe 4.2).

La notation formelle décrit la structure des formats et les relations entre leur forme non compressée et compressée, plutôt que de décrire comment la compression et la décompression sont effectuées.

Dans divers endroits de cette Section, du texte entre crochets a été utilisé comme un fourre-tout descriptif. L'utilisation de crochets de cette façon est seulement pour l'agrément du lecteur de ce document. Ni les crochets, ni leur contenu ne font partie de la notation.

4.1 Structure d'une spécification

La spécification des formats compressés d'un profil ROHC en utilisant ROHC-FN est appelé une spécification ROHC-FN. Les spécifications ROHC-FN sont sensibles à la casse et sont écrites dans le jeu de caractères de 7 bit ASCII (comme défini dans la [RFC2822]) et consiste en une séquence de zéro, une ou plusieurs définitions de constantes (paragraphe 4.3) une liste facultative globale de champs de contrôle (paragraphe 4.12.1.3) et une ou plusieurs définitions de méthode de codage (paragraphe 4.12).

Les méthodes de codage peuvent être définies en utilisant la notation formelle ou être des méthodes de codage prédéfinies.

Les méthodes de codage sont définies en utilisant la notation formelle en donnant un ou plusieurs formats non compressés pour représenter l'en-tête non compressé et un ou plusieurs formats compressés. Ces formats sont en relation les uns avec les autres par "les champs", dont chacun décrit une certaine partie d'un en-tête non compressé et/ou compressé. En plus des formats, chaque méthode de codage peut contenir des champs de contrôle, des valeurs initiales, et sections de codage de champs par défaut. Les attributs d'un champ sont liés en utilisant pour eux une méthode de codage et/ou en utilisant des déclarations "ENFORCE" (paragraphe 4.9) dans les formats. Chacun d'eux est terminé par un caractère point-virgule.

Les méthodes de codage prédéfinies ne sont pas définies dans la notation formelle. Elles sont plutôt définies en donnant une courte référence textuelle qui explique où la méthode de codage est définie. Il n'est pas nécessaire de définir la bibliothèque des méthodes de codage contenue dans le présent document de cette façon, leur définition est implicite à l'usage de la notation formelle.

4.2 Identifiants

Dans ROHC-FN, les identifiants sont utilisés pour une des choses suivantes :
- o les méthodes de codage
- o les formats
- o les champs
- o les paramètres
- o les constantes

Tous les identifiants peuvent être d'une longueur quelconque et peuvent contenir toute combinaison de caractères alphanumériques et soulignés, avec les restrictions définies dans ce paragraphe.

Tous les identifiants doivent commencer par un caractère alphabétique.

Il est illégal d'avoir deux identifiants ou plus qui diffèrent d'un de l'autre par la seule casse, dans la même portée.

Toutes les lettres dans les identifiants de constantes doivent être en majuscules.

Il est illégal d'utiliser ce qui suit comme identifiant (y compris dans une casse différente) :

- "false", "true"
- "ENFORCE", "THIS", "VARIABLE"
- "ULENGTH", "UVALUE"
- "CLENGTH", "CVALUE"
- "UNCOMPRESSED", "COMPRESSED", "CONTROL", "INITIAL", ou "DEFAULT"

Les noms de format ne peuvent pas être référencés dans la notation, bien qu'ils soient considérés comme des identifiants. (Voir au paragraphe 4.12.3.1 plus de détails sur les noms de format.)

Tous les identifiants utilisés dans ROHC-FN ont une "portée". La portée d'un identifiant définit les parties de la spécification où cet identifiant s'applique et d'où il peut être référencé. Si un identifiant a une portée "globale", il s'applique alors à toute la spécification qui le contient et peut être référencé à partir de n'importe où en son sein. Si un identifiant a une portée "locale", il s'applique alors seulement à la méthode de codage dans laquelle il est défini, il ne peut pas être référencé de l'extérieur de la portée locale de cette méthode de codage. Si un identifiant a une portée locale, cet identifiant peut donc être utilisé dans plusieurs portées locales différentes pour se référer à des éléments différents.

Toutes les instances d'un identifiant au sein de sa portée se réfèrent au même élément. Il n'est pas possible d'avoir différents éléments qui se réferent à un seul identifiant au sein d'une portée donnée. Pour cette raison, si il y a un identifiant qui a une portée globale, il ne peut pas être utilisé séparément dans une portée locale, car un identifiant de portée globale est déjà applicable dans toutes les portées locales.

Les identifiants pour chaque méthode de codage et chaque constante ont tous une portée globale. Chaque format et champ a aussi un identifiant. La portée des identifiants de format et de champ est locale, à l'exception des champs de contrôle globaux, qui ont une portée globale. Donc il est illégal pour un format ou champ d'avoir le même identifiant qu'un autre format ou champ dans la même portée, ou qu'une méthode de codage ou constante (car elles ont une portée globale).

Noter que bien que les noms de format (voir au paragraphe 4.12.3.1) soient considérés comme des identifiants, ils ne sont pas référencés dans la notation, mais sont principalement pour l'agrément du lecteur.

4.3 Définitions des constantes

Les valeurs de constantes peuvent être définies en utilisant l'opérateur "=". Les identifiants de constantes doivent être tout en majuscules. Par exemple :

```
SOME_CONSTANT = 3;
```

Les constantes sont définiee par une expression (voir au paragraphe 4.7) sur le côté droit de l'opérateur "=". L'expression doit donner une valeur constante. C'est-à-dire, l'expression doit être une dont les termes sont tous des constantes ou des littéraux et ne doit pas varier selon que l'en-tête est compressé ou non.

Les constantes ont une portée globale. Les constantes doivent être définies au niveau supérieur, en dehors de toute définition de méthode de codage. Les constantes sont entièrement équivalentes à la valeur à laquelle elles se réfèrent, et sont complètement interchangeables avec cette valeur. À la différence des attributs de champs, qui peuvent changer d'un paquet à l'autre, les constantes ont la même valeur pour tous les paquets.

4.4 Champs

Les champs sont les blocs de construction de base de la spécification ROHC-FN. Les champs sont les unités dans lesquelles sont divisés les en-têtes. Chaque champ peut avoir deux formes : une forme compressée et une forme non compressée. Les deux formes sont représentées comme des bits échangés entre le compresseur et le décompresseur de la même façon qu'une chaîne de bits non signés ; le bit de poids fort en premier.

Les propriétés de la forme compressée d'un champ sont définies par une méthode de codage et/ou des déclarations "ENFORCE". Cela caractérise entièrement les relations entre les formes non compressées et compressées de ce champ. Ceci est réalisé en spécifiant les relations entre les attributs du champ.

La notation définit quatre attributs de champ, deux pour la forme non compressée et deux correspondants pour la forme compressée. Les attributs disponibles pour chaque champ sont :
attributs non compressés d'un champ : "UVALUE" et "ULENGTH",

attributs compressés d'un champ : "CVALUE" et "CLENGTH".

Les deux attributs de valeur contiennent les valeurs numériques respectives du champ, c'est-à-dire, "UVALUE" donne la valeur numérique de la forme non compressée du champ, et l'attribut "CVALUE" donne la valeur numérique de la forme compressée du champ. Les valeurs numériques sont déduites en interprétant les représentations en chaîne binaire du champ comme des chaînes de bits; le bit de poids fort en premier.

Les deux attributs de longueur indiquent la longueur en bits de la chaîne de bits associée : "ULENGTH" pour la forme non compressée, et "CLENGTH" pour la forme compressée.

Les attributs sont indéfinis sauf si ils sont liés à une valeur, et dans ce cas deviennent définis. Si deux liens en conflit sont donnés pour un attribut de champ, alors les liens échouent avec les formats (leur combinaison) dans lesquels ces liens ont été définis.

Les attributs non compressés ne refètent pas toujours un aspect de l'en-tête non compressé. Certains des champs ne sont pas originaires de l'en-tête non compressé, mais sont des champs de contrôle.

4.4.1 Références d'attribut

On fait formellement référence aux attributs d'un champ particulier en utilisant le nom du champ suivi par un "." et l'identifiant de l'attribut.

Par exemple :

```
rtp_seq_number.UVALUE
```

Cela donne la valeur non compressée du champ rtp_seq_number *(numéro de séquence RTP)*. La principale raison pour faire référence aux attributs est pour les utiliser dans les expressions, ce qui est expliqué au paragraphe 4.7.

4.4.2 Représentation des valeurs de champ

Les champs sont représentés comme des chaînes de bits. La chaîne de bits est calculée en utilisant l'attribut de valeur ("val") et l'attribut de longueur ("len"). La chaîne de bits est la représentation binaire de "val % (2 ^ len)".

Par exemple, si l'attribut "CLENGTH" d'un champ est 8, et son attribut "CVALUE" est -1, la représentation compressée du champ va être "-1 % (2 ^ 8)", qui est égale à "-1 % 256", soit 255, 11111111 en binaire.

ROHC-FN prend en charge la gamme complète des entiers à utiliser dans les expressions (voir au paragraphe 4.7) mais la représentation des formats (c'est-à-dire, des bits échangés entre compresseur et décompresseur) est dans cette forme.

4.5 Groupement de champs

Comme l'ordre des champs dans une liste de champs "COMPRESSED" (paragraphe 4.12.1.2) n'a pas à être le même que celui des champs dans une liste de champs "UNCOMPRESSED" (paragraphe 4.12.1.1) il est possible de grouper tout nombre de champs qui sont contigus dans un format "COMPRESSED", pour leur permettre d'être tous codés en utilisant une seule méthode de codage. Le groupe des champs est spécifié immédiatement à gauche de "=:=" au lieu d'un seul nom de champ.

Le groupe est noté en donnant une liste séparée par des points-virgules des champs à grouper. Par exemple, il peut y avoir deux champs non contigus dans un en-tête non compressé qui sont deux moitiés de ce qui est effectivement un seul numéro de séquence :

```
grouping_exemple
{
 UNCOMPRESSED {
  minor_seq_num;  // 12 bits
  other_field;    //  8 bits
  major_seq_num;  //  4 bits
 }
```

```
  COMPRESSED {
    other_field =:= irregular(8);
    major_seq_num
    : minor_seq_num =:= lsb(3, 0);
  }
}
```

Le groupe des champs est présenté à la méthode de codage comme un groupe de bits contigus, assemblés par l'enchaînement des champs dans l'ordre dans lequel ils sont donnés dans le groupe. Le bit de poids fort du champ combiné est le bit de poids fort du premier champ de la liste, et le bit de moindre poids du champ combiné est le bit de moindre poids du dernier champ de la liste.

Finalement, les attributs de longueur du champ combiné sont égaux à la sommme des attributs de longueur correspondants pour tous les champs du groupe.

4.6 "THIS"

Dans la définition d'une méthode de codage, il est possible de se référer au champ (c'est-à-dire, le groupe de bits contigus) que code la méthode, en utilisant le mot-clé "THIS".

"This" est utile pour obtenir l'accès aux attributs du champ à coder. Par exemple il est souvent utile pour connaître la longueur totale non compressée du format non compressé qui est à coder :

```
THIS.ULENGTH
```

4.7 Expressions

ROHC-FN inclut le style des expressions infix usuel, avec des parenthèses "(" et ")" utilisées pour grouper. Les expressions peuvent être constituées de tout composant décrit dans les paragraphes qui suivent.

La sémantique des expressions est généralement similaire à celle des expressions dans le langage de programmation ANSI-C [C90]. La liste définitive des expressions dans ROHC-FN est donnée dans les paragraphes qui suivent ; la liste ci-dessous donne des exemples de la différence entre les expressions en ANSI-C et dans ROHC-FN :

- il n'y a pas de limite à la gamme des entiers,
- "x ^ y" s'évalue à x à la puissance y. Cela a la préséance sur *, / et %, mais pas sur unaire - et est associatif de droite à gauche,
- il n'y a pas d'opérateur virgule,
- il n'y a pas d'opérateurs "modify" (pas d'opérateurs d'allocation et pas d'incrément ou décrément),
- il n'y a pas d'opérateur au bit près.

Les expressions peuvent se référer à tout attribut d'un champ (comme décrit au paragraphe 4.4) à toute constante définie (voir au paragraphe 4.3) et aussi aux paramètres de méthode de codage, si il en est dans la portée (voir au paragraphe 4.12).

Si un attribut, constante, ou paramètre utilisé dans l'expression est indéfini, la valeur de l'expression est indéfinie. Les expressions indéfinies causent l'échec de l'environnement (par exemple, le format compressé) dans lequel elles sont utilisées si une valeur définie est exigée. Les valeurs définies sont exigées pour tous les attributs compressés des champs qui apparaissent dans le format compressé. Les valeurs définies ne sont pas exigées pour tous les attributs non compressés des champs qui apparaissent dans le format non compressé. Il appartient au créateur du profil de définir ce qui arrive aux attributs de champ non liés dans ce cas. On devrait noter que dans ce cas, la transparence du processus de compression va être perdue ; c'est-à-dire, il ne sera pas possible au décompresseur de reproduire l'en-tête original.

Les expressions ne peuvent pas être utilisées directement comme méthodes de codage parce que elles ne caractérisent pas complètement un champ. Les expressions spécifient seulement une valeur tandis que un champ est constitué de plusieurs valeurs : ses attributs. Par exemple, ce qui suit est illégal :

```
tcp_list_length =:= (data_offset + 20) / 4;
```

Il y a ici juste assez d'information pour définir un seul attribut de "tcp_list_length". Bien que cela n'ait formellement pas de

sens, cela pourrait intuitivement être lu comme définissant l'attribut "UVALUE". Cependant, cela laisserait quand même la longueur du champ non compressé indéfinie au décompresseur. Un tel usage est donc interdit.

4.7.1 Littéraux entiers

Les entiers peuvent être exprimés comme des valeurs décimales, des valeurs binaires (préfixées par "0b") ou hexadécimales (préfixées par "0x"). Les entiers négatifs sont préfixés par un signe "-". Par exemple "10", "0b1010", et "-0x0a" sont tous des littéraux entiers valides, ayant respectivement les valeurs 10, 10, et -10.

4.7.2 Opérateurs d'entiers

Les opérateurs "d'entier" suivants sont disponibles, qui prennent des arguments d'entier et retournent un résultat entier :

- ^, pour l'exponentiation. "x ^ y" retourne la valeur de "x" à la puissance "y".
- *, / pour la multiplication et la division. "x * y" retourne le produit de "x" par "y". "x / y" retourne le quotient, arrondi à l'entier inférieur (le suivant vers l'infini négatif).
- +, - pour l'addition et la soustraction. "x + y" retourne la somme de "x" et de "y". "x - y" retourne la différence.
- % pour modulo. "x % y" retourne "x" modulo "y" ; x - y * (x / y).

4.7.3 Littéraux booléens

Les littéraux booléens sont "false" *(faux)*, et "true" *(vrai)*.

4.7.4 Opérateurs booléens

Les opérateurs "booléens" sont disponibles, qui prennent des arguments booléens et retournent un résultat booléen :
- &&, pour le "et" logique. Retourne vrai si les deux arguments sont vrais. Retourne faux autrement.
- ||, pour le "ou" logique. Retourne vrai si au moins un argument est vrai. Retourne faux autrement.
- !, pour le "non" logique. Retourne vrai si son argument est faux. Retourne faux autrement.

4.7.5 Opérateurs de comparaison

Les opérateurs de "comparaison" suivants sont disponibles, qui prennent des arguments d'entier et retournent un résultat booléen :

- ==, !=, pour l'égalité et sa négation. "x == y" retourne vrai si x est égal à y. Retourne faux autrement. "x != y" retourne vrai si x est n'est pas égal à y. Retourne faux autrement.
- <, >, pour inférieur à et supérieur à. "x < y" retourne vrai si x est inférieur à y. Retourne faux autrement. "x > y" retourne vrai si x est supérieur à y. Retourne faux autrement.
- >=, <=, pour supérieur ou égal à et inférieur ou égal à, les fonctions inverses de <, >. "x >= y" retourne faux si x est inférieur à y. Retourne vrai autrement. "x <= y" retourne faux si x est supérieur à y. Retourne vrai autrement.

4.8 Commentaires

Du texte libre en anglais peut être inséré dans une spécification ROHC-FN pour expliquer pourquoi quelque chose a été fait d'une certaine façon, pour préciser la signification prévue de la notation, ou pour développer un point.
La FN utilise une fin de ligne de style commentaire, qui utilise le marqueur de commentaire "//". Tout texte entre le marqueur "//" et la fin de ligne n'a pas de signification formelle. Par exemple :

```
//---------------------------------------------------------------
//   IR-REPLICATE formats d'en-tête
//---------------------------------------------------------------
```

```
// Les champs suivants sont inclus dans tous les formats d'en-tête IR-REPLICATE :
//
UNCOMPRESSED {
 discriminator;   //  8 bits
 tcp_seq_number;  // 32 bits
 tcp_flags_ecn;   //  2 bits
```

Le commentaire n'affecte pas la signification formelle de ce qui est noté, mais peut être utilisé pour améliorer la lisibilité. Son utilisation est facultative.

Les commentaires peuvent aider à fournir des éclaircissements au lecteur, et servir à différents objets pour les mises en œuvre. Les commentaires ne devraient donc pas être considérés de moindre importance quand on les insère dans une spécification ROHC-FN ; ils devraient être cohérents avec la partie normative de la spécification.

4.9 Déclarations "ENFORCE"

La déclaration "ENFORCE" donne un moyen d'ajouter des prédicats à un format, dont tous doivent être respectés pour que le format réussisse. Une déclaration "ENFORCE" a quelques similarités avec une méthode de codage. Précisément, tandis qu'une méthode de codage lie plusieurs attributs de champ en une fois, une déclaration "ENFORCE" lie normalement juste un d'eux. En fait, tous les liens que créent les méthodes de codage peuvent être exprimés en termes de collection de déclarations "ENFORCE". Voici un exemple de déclaration "ENFORCE" qui lie l'attribut "UVALUE" d'un champ à 5.

```
ENFORCE(field.UVALUE == 5);
```

Une déclaration "ENFORCE" doit seulement être utilisée dans une liste de champs (voir au paragraphe 4.12). Elle tente de forcer l'expression donnée à être vraie pour le format à laquelle elle appartient.

Une forme abrégée d'une déclaration "ENFORCE" est disponible pour lier les attributs de longueur en utilisant "[" et "]", voir au paragraphe 4.10.

Comme une méthode de codage, une déclaration "ENFORCE" peut seulement réussir quand elle est utilisée dans un format si le lien qu'elle décrit est réalisable. Un format contenant l'exemple de déclaration "ENFORCE" ci-dessus ne serait pas utilisable si le champ avait aussi été lié dans ce même format avec un codage "uncompressed_value", qui lui donne une "UVALUE" autre que 5.

Une déclaration "ENFORCE" prend une expression booléenne comme paramètre. Il peut être utilisé pour affirmer que l'expression est vraie, afin de choisir un format particulier dans une liste de formats possibles spécifiés dans une méthode de codage (voir au paragraphe 4.12) ou juste pour lier une expression comme dans l'exemple ci-dessus. La forme générale d'une déclaration "ENFORCE" est donc :

```
ENFORCE(<expression booléenne>);
```

Il y a trois conditions possibles dans lesquelles peut être l'expression :

1. L'expression booléenne s'évalue à faux, et dans ce cas la portée locale du format qui contient la déclaration "ENFORCE" ne peut pas être utilisée.

2. L'expression booléenne s'évalue à vrai, et dans ce cas le lien est créé et réussi.

3. La valeur de l'expression booléenne est indéfinie. Dans ce cas, le lien est aussi créé et réussi.

Dans les trois cas, tout terme indéfini devient lié par l'expression. D'une façon générale, une déclaration "ENFORCE" est soit utilisée comme une allocation (condition 3 ci-dessus) soit utilisée pour vérifier si un format particulier est utilisable, comme c'est le cas avec les conditions 1 et 2.

4.10 Spécification formelle des longueurs de champs

Dans de nombreux exemples, chaque champ est suivi d'un commentaire indiquant la longueur du champ. Indiquer la

longueur d'un champ comme cela est facultatif, mais peut être très utile pour le lecteur. Cependant, bien qu'utiles pour le lecteur, les commentaires n'ont pas de signification formelle.

Une des plus courantes utilisations des déclarations "ENFORCE" (voir au paragraphe 4.9) est de définir explicitement la longueur d'un champ dans un en-tête. Utiliser des déclarations "ENFORCE" à cette fin a une signification formelle mais n'est pas très facile à lire. Donc, une forme abrégée est fournie pour cette utilisation de "ENFORCE", qui est à la fois facile à lire et a une signification formelle.

Une expression qui définit la longueur d'un champ peut être spécifiée entre crochets après l'apparition de ce champ dans un format. Si le champ peut prendre plusieurs longueurs possibles, alors les expressions qui définissent ces longueurs peuvent être énumérées dans une liste séparées de virgules entre les crochets. Par exemple :

```
field_1              [ 4 ];
field_2              [ a+b, 2 ];
field_3 =:= lsb(16, 16)  [ 26 ];
```

L'attribut de longueur réel, qui est lié par cette notation, dépend de si il apparaît dans une liste de champs "COMPRESSED", "UNCOMPRESSED", ou "CONTROL" (voir au paragraphe 4.12.1 et ses sous paragraphes). Dans une liste de champs "COMPRESSED", l'attribut "CLENGTH" du champ est lié. Dans des listes de champs "UNCOMPRESSED" et "CONTROL", l'attribut "ULENGTH" du champ est lié. Les déclarations "ENFORCE" abrégées ne sont pas permises dans les sections "DEFAULT" (voir au paragraphe 4.12.1.5). Donc, la notation ci-dessus ne serait pas permise dans une section "DEFAULT". Cependant, si elle apparaît dans une section "UNCOMPRESSED" ou "CONTROL", elle serait équivalente à :

```
field_1 ;  ENFORCE(field_1.ULENGTH == 4);
field_2 ;  ENFORCE((field_2.ULENGTH == 2) || (field_2.ULENGTH == a+b));
field_3 =:= lsb(16, 16); ENFORCE(field_3.ULENGTH == 26);
```

Un cas particulier existe pour les champs qui ont une longueur variable que le notateur ne souhaite pas, ou n'est pas capable de définir en utilisant une expression. Le mot-clé "VARIABLE" peut être utilisé dans le cas suivant :

```
variable_length_field  [ VARIABLE ];
```

Formellement, cela ne donne pas de restriction sur le champ longueur, mais se transpose en tout entier positif ou en une valeur de zéro. Il va donc être nécessaire de définir ailleurs la longueur du champ (voir les alinéas de la fin des paragraphes 4.12.1.1 et 4.12.1.2). Cela peut être soit dans la notation soit dans le texte anglais du profil dans lequel la FN est contenue. Dans les crochets, le mot-clé "VARIABLE" peut être utilisé comme un terme dans une expression, juste comme tout autre terme qui apparaît normalement dans une expression. Par exemple :

```
field  [ 8 * (5 + VARIABLE) ];
```

Cela définit un champ dont la longueur est un nombre entier d'octets et d'au moins 40 bits (5 octets).

4.11 Bibliothèque des méthodes de codage

Un certain nombre de techniques courantes pour compresser des champs d'en-tête sont définies au titre de la bibliothèque ROHC-FN afin qu'elles puissent être réutilisées lors de la création de nouvelles spécifications ROHC-FN. Leur notation est décrite ci-dessous.

Comme solution de remplacement, ou en complément à cette bibliothèque des méthodes de codage, une spécification ROHC-FN peut définir son propre ensemble de méthodes de codage, en utilisant la notation formelle (voir au paragraphe 4.12) ou en utilisant une définition textuelle (voir au paragraphe 4.13).

4.11.1 uncompressed_value

La méthode de codage "uncompressed_value" est utilisée pour coder des champs d'en-tête pour lesquels la valeur non compressée peut être définie en utilisant une expression mathématique (incluant des valeurs constantes). Cette méthode de codage est définie comme suit :

```
uncompressed_value(len, val) {
```

```
UNCOMPRESSED {
  field;
  ENFORCE(field.ULENGTH == len);
  ENFORCE(field.UVALUE == val);
}
COMPRESSED {
  field;
  ENFORCE(field.CLENGTH == 0);
 }
}
```

Pour donner un exemple de l'usage du codage "uncompressed_value", le numéro de version d'en-tête IPv6 est un champ de 4 bits qui a toujours la valeur 6 :

```
version  =:= uncompressed_value(4, 6);
```

Voici un autre exemple de codage de valeur, en utilisant une expression pour calculer la longueur :

```
bourrage =:= uncompressed_value(nbits - 8, 0);
```

L'expression ci-dessus utilise un paramètre de méthode de codage, "nbits", qui dans cet exemple spécifie combien de bits significatifs sont dans les données pour calculer combien de bits de bourrage utiliser. Voir au paragraphe 4.12.2 plus d'informations sur les paramètre de méthode de codage.

4.11.2 compressed_value

La méthode de codage "compressed_value" est utilisée pour définir les champs en formats compressés pour lesquels il n'y a pas de contrepartie dans le format non compressé (c'est-à-dire, des champs de contrôle). Elle peut être utilisée pour spécifier les champs compressé dont la valeur peut être définie en utilisant une expression mathématique (incluant des valeurs constantes). Cette méthode de codage est définie comme suit :

```
compressed_value(len, val) {
  UNCOMPRESSED {
    field;
    ENFORCE(field.ULENGTH == 0);
  }
  COMPRESSED {
    field;
    ENFORCE(field.CLENGTH == len);
    ENFORCE(field.CVALUE == val);
  }
}
```

Une utilisation possible de cette méthode de codage est de définir du bourrage en format compressé :

```
pad_to_octet_boundary =:= compressed_value(3, 0);
```

Une utilisation plus courante est de définir un champ discriminant pour permettre de différencier entre différents formats compressés dans une méthode de codage (voir au paragraphe 4.12). Pour simplifier, la notation donne la syntaxe pour spécifier le codage "compressed_value" sous la forme d'une chaîne binaire. La chaîne binaire à coder est simplement données entre de simples apostrophes ; l'attribut "CLENGTH" du champ lie au nombre de bits dans la chaîne, tandis que son attribut "CVALUE" lie à la valeur donnée par la chaîne. Par exemple :

```
discriminator =:= '01101';
```

Cela a exactement la même signification que :

```
discriminator =:= compressed_value(5, 13);
```

4.11.3 irregular

La méthode de codage "irregular" est utilisée pour coder un champ en format compressé avec un schéma de bits identique à celui du champ non compressé. Cette méthode de codage est définie comme suit :

```
irregular(len) {
  UNCOMPRESSED {
    field;
    ENFORCE(field.ULENGTH == len);
  }
  COMPRESSED {
    field;
    ENFORCE(field.CLENGTH == len);
    ENFORCE(field.CVALUE == field.UVALUE);
  }
}
```

Par exemple, le champ Somme de contrôle dans l'en-tête TCP est un champ de 16 bits qui ne suit aucun schéma prévisible d'un en-tête à l'autre (et donc il ne peut pas être compressé):

```
tcp_checksum  =:= irregular(16);
```

Noter que la longueur n'a pas à être constante, par exemple, une expression peut être utilisée pour déduire la longueur du champ de la valeur d'un autre champ.

4.11.4 static

La méthode de codage "static" compresse un champ dont la longueur et la valeur sont les mêmes que pour un en-tête précédent dans le flux, c'est-à-dire, où le champ correspond complètement à une entrée existante dans le contexte :

```
field =:= static;
```

Les attributs "UVALUE" et "ULENGTH" du champ se lient à leurs valeurs respectives dans le contexte et l'attribut "CLENGTH" est lié à zéro.

Comme la valeur du champ est la même que celle d'un champ précédent, le champ entier peut être reconstruit à partir du contexte, de sorte qu'il est compressé à des bits zéro et n'apparaît pas dans le format compressé.

Par exemple, l'accès de source de l'en-tête TCP est un champ dont la valeur ne change pas d'un paquet au suivant pour un flux donné :

```
src_port =:= static;
```

4.11.5. lsb

La méthode de codage "bits de moindre poids, "lsb", compresse un champ dont la valeur diffère d'une petite quantité de la valeur mémorisée dans le contexte. Les bits de moindre poids de la valeur du champ sont transmis à la place de la valeur originelle du champ.

```
field =:= lsb(<num_lsbs_param>, <offset_param>);
```

Ici, "num_lsbs_param" est le nombre de bits de moindre poids à utiliser, et "offset_param" est le décalage d'intervalle d'interprétation comme défini ci-dessous.

Le paramètre "num_lsbs_param" lie à l'attribut "CLENGTH", l'attribut "UVALUE" lie à la valeur dans l'intervalle dont les bits de moindre poids correspondent à l'attribut "CVALUE". La valeur de "ULENGTH" peut être déduite des informations mémorisées dans le contexte.

Par exemple, le numéro de séquence TCP :

```
tcp_sequence_number =:= lsb(14, 8192);
```

Cela prend 14 bits, et peut communiquer toute valeur qui est entre 8192 inférieure à la valeur du champ mémorisé dans le contexte et 8191 au dessus d'elle.

L'intervalle d'interprétation peut être décrit comme une fonction d'une valeur mémorisée dans le contexte, ref_value, et de num_lsbs_param :

```
f(context_value, num_lsbs_param) = [ref_value - offset_param,
        ref_value + (2^num_lsbs_param - 1) - offset_param]
```

où offset_param est un entier.

```
      <-- intervalle d'interprétation (la taille est 2^num_lsbs_param) -->
      |----------------------------------+-----------------------------------------|
limite inférieure                ref_value                              limite supérieure
```

où :
limite inférieure = ref_value - offset_param
limite supérieure = ref_value + (2^num_lsbs_param-1) - offset_param

La méthode de codage "lsb" peut donc compresser un champ dont la valeur se tient entre les limites inférieure et supérieure, incluses, de l'intervalle d'interprétation. En particulier, si offset_param = 0, alors la valeur du champ peut seulement rester la même ou augmenter par rapport à la valeur de référence ref_value. Si offset_param = -1, alors elle peut seulement augmenter, tandis que si offset_param = 2^num_lsbs_param, elle peut seulement diminuer.

Le champ compressé prend jusqu'au nombre spécifié de bits dans le format compressé (c'est-à-dire, num_lsbs_param).

Le compresseur peut n'être pas capable de déterminer la valeur de référence exacte mémorisée dans le contexte du décompresseur et qui va être utilisée par le décompresseur, car certains paquets qui auraient mis à jour le contexte peuvent avoir été perdus ou endommagés. Cependant, à partir des rétroactions reçues ou en faisant des hypothèses, le compresseur peut limiter l'ensemble de valeurs candidates. Le compresseur peut alors choisir un format qui utilise le codage "lsb", défini avec des valeurs convenables pour ses paramètres num_lsbs_param et offset_param, de façon que quelle que soit la valeur de contexte dans l'ensemble candidat que le décompresseur utilise, la décompression résultante est correcte. Si ce n'est pas possible, la méthode de codage "lsb" échoue (ce qui résulte normalement en le choix d'un format compressé moins efficace par le compresseur). Comment le compresseur détermine quelles valeurs de référence il mémorise et maintient dans son ensemble de références candidates sort du domaine d'application de la notation.

4.11.6 crc

La méthode de codage "crc" fournit un contrôle de redondance cyclique (CRC) calculé sur un bloc de données. L'algorithme utilisé pour calculer le CRC est celui qui est spécifié dans la [RFC4995]. La méthode "crc" prend certains paramètres :

- o le nombre de bits pour le CRC (crc_bits),
- o le schéma de bits pour le polynome (bit_pattern),
- o la valeur initiale pour le registre de CRC (initial_value),
- o la valeur du bloc de données, représentée en utilisant l'attribut "UVALUE" ou "CVALUE" d'un champ (block_data_value) ; et
- o la taille en octets du bloc de données (block_data_length).

C'est à dire :

```
field =:= crc(<num_bits>, <bit_pattern>, <initial_value>, <block_data_value>, <block_data_length>);
```

Quand on spécifie le schéma de bits pour le polynome, chaque bit représente le coefficient pour le terme correspondant dans le polynome. Noter que le terme d'ordre le plus élevé est toujours présent (par définition) et donc n'a pas besoin d'être spécifié dans le schéma de bits. Donc, un polynome de CRC de n termes est représenté par un schéma de bits de n-1 bits.

Le CRC est calculé dans l'ordre du bit de moindre poids (LSB, *least significant bit*).

Par exemple :

```
// 3 bit CRC, C(x) = x^0 + x^1 + x^3
  crc_field =:= crc(3, 0x6, 0xF, THIS.CVALUE, THIS.CLENGTH);
```

L'usage du mot-clé "THIS" (voir au paragraphe 4.6) comme ci-dessus, est normal en utilisant le codage "crc". Par exemple, quand il est utilisé dans la méthode de codage pour un en-tête entier, il cause le calcul du CRC sur tous les champs de l'en-tête.

4.12 Définition des méthodes de codage

De nouvelles méthodes de codage peuvent être définies dans une spécification formelle. Elles composent des groupes de champs individuels dans un bloc contigu.

Les méthodes de codage ont des noms et peuvent avoir des paramètres ; elles peuvent aussi être utilisées de la même manière que toute autre méthode de codage à partir de la bibliothèque des méthodes de codage. Comme elles peuvent contenir des références à d'autres méthodes de codage, les formats compliqués peuvent être réduits en portions gérables de façon hiérarchique.

Cette section décrit les diverses caractéristiques utilisées pour définir de nouvelles méthodes de codage.

4.12.1 Structure

Cette plus simple forme de définition d'une méthode de codage est de spécifier un seul codage. Par exemple :

```
compound_encoding_method
{
 UNCOMPRESSED {
  field_1;  //  4 bits
  field_2;  // 12 bits
 }

 COMPRESSED {
  field_2 =:= uncompressed_value(12, 9); //  0 bits
  field_1 =:= irregular(4);              //  4 bits
 }
}
```

On commence par l'identifiant de la nouvelle méthode, "compound_encoding_method". La définition de la méthode suit entre les accolades, "{" et "}". Le premier élément de la définition est la liste de champs "UNCOMPRESSED", qui donne l'ordre des champs dans le format non compressé. C'est suivi par la liste des champs en format compressé ("COMPRESSED"). Cette liste donne l'ordre des champs en format compressé et donne aussi la méthode de codage pour chaque champ.

Dans l'exemple, les deux formats mentionnent chaque champ exactement une fois. Cependant, il est parfois nécessaire de spécifier plus d'un lien pour un certain champ, ce qui signifie qu'il apparaît plus d'une fois dans la liste des champs. Dans ce cas, c'est la première occurrence du champ dans la liste qui indique sa position dans l'ordre des champs. Les occurrences suivantes du champ spécifient seulement les informations de lien, pas les informations d'ordre des champs.

Les différents composants de cet exemple sont décrits plus en détail ci après ainsi que les autres composants qui peuvent être utilisés dans la définition des méthodes de codage.

4.12.1.1 Format non compressé - "UNCOMPRESSED"

La liste des champs non compressés est définie par "UNCOMPRESSED", qui spécifie les champs de format non compressé dans l'ordre où ils apparaissent dans l'en-tête non compressé. La somme des longueurs de chaque champ non compressé individuel dans la liste doit être égale à la longueur du champ à coder. Finalement, la représentation du format non compressé décrit en utilisant la liste des champs dans la section "UNCOMPRESSED", pour laquelle les formats compressés sont définis, consiste toujours en un seul bloc de bits contigus.

Dans l'exemple du paragraphe 4.12.1, la liste des champs non compressés est "field_1", suivi par "field_2". Cela signifie qu'un champ à coder par cette méthode est divisé en deux sous champs, "field_1" et "field_2". La longueur totale non compressée de ces deux champs est donc égale à la longueur du champ à coder :

field_1.ULENGTH + field_2.ULENGTH == THIS.ULENGTH

Dans l'exemple, il y a seulement deux champs, mais tout nombre de champs peut être utilisé. Cette relation s'applique quel que soit le nombre des champs réellement utilisés. Tout arrangement des champs qui décrit efficacement le contenu de l'en-tête non compressé peut être choisi -- il n'est pas besoin que ce soit le même que celui décrit dans les spécifications pour l'en-tête de protocole à compresser.

Par exemple, il peut y avoir un protocole dont l'en-tête contient un numéro de séquence de 16 bits, mais dont les sessions tendent à être d'une courte durée de vie. Cela signifierait que les bits de poids fort du numéro de séquence sont presque toujours constants. Le format "UNCOMPRESSED" pourrait refléter cela en partageant le champ non compressé original en deux champs, un champ pour représenter la partie presque toujours à zéro du numéro de séquence, et un second champ pour représenter la partie caractéristique.

Une liste de champs "UNCOMPRESSED" peut spécifier des méthodes de codage de la même façon qu'une liste de champs "COMPRESSED" dans l'exemple. Les méthodes de codage spécifiées ici sont utilisées chaque fois qu'un paquet avec ce format non compressé est à coder. Le codage d'un paquet avec un certain format non compressé ne peut réussir que si toutes ses méthodes de codage et déclarations "ENFORCE" réussissent (voir au paragraphe 4.9).

La longueur totale de chaque format non compressé doit toujours être définie. La longueur de chacun des champs dans un format non compressé doit aussi être définie. Cela signifie que les liens dans les listes de champs "UNCOMPRESSED", "COMPRESSED" (voir au paragraphe 4.12.1.2) "CONTROL" (voir au paragraphe 4.12.1.3) "INITIAL" (voir au paragraphe 4.12.1.4) et "DEFAULT" (voir au paragraphe 4.12.1.5) doivent, entre eux, définir l'attribut "ULENGTH" de chaque champ dans un format non compressé afin qu'il y ait une transposition non ambiguë des bits dans le format non compressé en les champs mentionnés dans la liste des champs "UNCOMPRESSED".

4.12.1.2 Format compressé - "COMPRESSED"

Comme pour la liste des champs non compressés, les champs dans l'en-tête compressé vont apparaître dans l'ordre spécifié par la liste des champs compressés donnée pour un format compressé. Chaque champ individuel est codé de la façon donnée pour ce champ. La longueur totale des données compressée va être la somme des longueurs compressées de tous les champs individuels. Dans l'exemple du paragraphe 4.12.1, les méthodes de codage utilisées pour ces champs indiquent que ce sont des bits longs de zéro et 4 bits, faisant un total de 4 bits.

L'ordre des champs spécifié dans une liste de champs "COMPRESSED" n'a pas à correspondre à l'ordre dans lequel ils apparaissent dans la liste des champs "UNCOMPRESSED". Il peut être souhaitable de réordonner les champs dans le format compressé pour aligner l'en-tête compressé sur une limite d'octet, ou pour d'autres raisons. Dans l'exemple ci-dessus, l'ordre est en fait l'opposé de celui du format non compressé.

La liste des champs compressés spécifie que le codage pour "field_1" est "irregular", et prend jusqu'à 4 bits dans les deux formats compressé et non compressé. Le codage pour "field_2" est "uncompressed_value", ce qui signifie que le champ a une valeur fixe, de sorte qu'il peut être compressé à zéro bit. La valeur qu'il prend est 9, et il est long de 12 bits dans le format non compressé.

Les champs comme "field_2", qui se compressent à zéro bit en longueur, peuvent apparaître n'importe où dans la liste des champs sans changer le format compressé parce que leur position dans la liste n'est pas significative. En fait, si la méthode de codage pour ce champ était définie ailleurs (par exemple, dans la section "UNCOMPRESSED") ce champ pourrait être omis aussi de la section "COMPRESSED" :

```
compound_encoding_method
{
 UNCOMPRESSED {
  field_1;                                  //  4 bits
  field_2 =:= uncompressed_value(12, 9);  // 12 bits
 }

 COMPRESSED {
```

```
    field_1 =:= irregular(4);          //  4 bits
  }
}
```

La longueur totale de chaque format compressé doit toujours être définie. La longueur de chacun des champs en format compressé doit aussi être définie. Cela signifie que les liens dans les listes de champs "UNCOMPRESSED", "COMPRESSED", "CONTROL" (voir au paragraphe 4.12.1.3) "INITIAL" (voir au paragraphe 4.12.1.4) et "DEFAULT" (voir au paragraphe 4.12.1.5) doivent définir entre eux l'attribut "CLENGTH" de chaque champ en format compressé afin qu'il y ait une transposition non ambiguë des bits en format compressé aux champs mentionnés dans la liste des champs "COMPRESSED".

4.12.1.3 Champs de contrôle - "CONTROL"

Les champs de contrôle sont définis en utilisant la liste de champs "CONTROL". La liste des champs de contrôle spécifie tous les champs qui n'apparaissent pas dans le format non compressé, mais qui ont une valeur non compressée (spécifiquement ceux qui ont une "ULENGTH" supérieure à zéro). Ces champs peuvent être utilisés pour aider à compresser plus efficacement les champs à partir du format non compressé. Un champ de contrôle pourrait être utilisé pour améliorer l'efficacité en représentant des facteurs communs entre un certain nombre de champs non compressés, ou en représentant des informations sur le flux qui ne sont pas explicitement contenues dans les en-têtes du protocole.

Par exemple dans IPv4, le comportement du champ IP-ID dans un flux varie selon la façon dont les points d'extrémité traitent les identifiants IP. Parfois le comportement est effectivement aléatoire et parfois l'IP-ID suit une séquence prévisible. Le type de comportement IP-ID est de l'information qui n'est jamais communiquée explicitement dans l'en-tête non compressé.

Cependant, un profil peut toujours être conçu pour identifier le comportement et ajuster la stratégie de compression en accord avec le comportement identifié, améliorant ainsi les performances de compression. Pour ce faire, la spécification ROHC-FN peut introduire un champ explicite pour communiquer le comportement IP-ID en format compressé -- ceci est fait en introduisant un champ de contrôle :

```
ipv4
{
 UNCOMPRESSED {
   version;          // 4 bits
   hdr_length;       // 4 bits
   protocol;         // 8 bits
   dscp;             // 6 bits
   ip_ecn_flags;     // 2 bits
   ttl_hopl;         // 8 bits
   df;               // 1 bit
   mf;               // 1 bit
   rf;               // 1 bit
   frag_offset;      // 13 bits
   ip_id;            // 16 bits
   src_addr;         // 32 bits
   dst_addr;         // 32 bits
   checksum;         // 16 bits
   length;           // 16 bits
 }

 CONTROL {
   ip_id_behavior; // 1 bit
     :
     :
```

La liste de champs "CONTROL" est équivalente à la liste de champs "UNCOMPRESSED" pour les champs qui n'apparaissent pas dans le format non compressé. Elle définit un champ qui a les mêmes propriétés (les mêmes attributs définis, etc.) que les champs apparaissant dans le format non compressé.

Les champs de contrôle sont initialisés en utilisant les méthodes de codage appropriées et/ou en utilisant des déclarations "ENFORCE". Ceci peut donc être fait dans la liste de champs "CONTROL".

Par exemple :

```
exemple_encoding_method_definition
{
 UNCOMPRESSED {
  field_1 =:= some_encoding;
 }

 CONTROL {
  scaled_field;
  ENFORCE(scaled_field.UVALUE == field_1.UVALUE / 8);
  ENFORCE(scaled_field.ULENGTH == field_1.ULENGTH - 3);
 }

 COMPRESSED {
  scaled_field =:= lsb(4, 0);
 }
}
```

Ce champ de contrôle est utilisé pour réduire un champ en format non compressé d'un facteur de 8 avant le codage avec la méthode de codage "lsb". Le réduire rend le codage "lsb" plus efficace.

Les champs de contrôle peuvent aussi être utilisés avec une portée globale. Dans ce cas, leur déclaration doit être en dehors de toute définition de méthode de codage. Ils sont alors visibles dans toute méthode de codage, permettant donc aux informations d'être partagées directement entre les méthodes de codage.

4.12.1.4 Valeurs initiales - "INITIAL"

Afin de permettre aux champs dans le tout premier usage d'un format spécifique d'être compressés avec les méthodes de codage "static", "lsb", ou autres qui dépendent du contexte, il est possible de spécifier des liens initiaux pour de tels champs. Ceci est fait en utilisant "INITIAL", par exemple :

```
INITIAL {
  field =:= uncompressed_value(4, 6);
}
```

Cela initialise la "UVALUE" de "field" à 6 et initialise sa "ULENGTH" à 4. À la différence des autres liens spécifiés dans la notation formelle, ces liens sont appliqués au contexte du champ, si le contexte du champ est indéfini. Ceci est particulièrement utile quand on utilise des méthodes de codage qui s'appuient sur la présence du contexte, comme "static" ou "lsb", avec le premier paquet dans un flux.

Parce que la liste de champs "INITIAL" est utilisée pour lier le contexte seul, il n'y a pas de sens à spécifier des liens initiaux qui eux-mêmes s'appuient sur le contexte, par exemple, "lsb". Un tel usage n'est pas permis.

4.12.1.5 Liens de champ par défaut - "DEFAULT"

Des liens par défaut peuvent être spécifiés pour chaque champ ou attribut. Les méthodes de codage par défaut spécifient la méthode de codage à utiliser pour un champ si aucun lien n'est donné ailleurs pour la valeur de ce champ. Ceci est utile pour garder concise la définition des formats, car la même méthode de codage n'a pas besoin d'être répétée pour chaque format quand on définit plusieurs formats (voir au paragraphe 4.12.3).

Les liens par défaut sont facultatifs et peuvent être donnés pour toute combinaison des champs et attributs qui sont dans la portée.

La syntaxe pour spécifier les liens par défaut est similaire à celle utilisée pour spécifier un format compressé ou non compressé. Cependant, l'ordre des champs dans la liste des champs n'affecte pas l'ordre des champs dans le format compressé ou non compressé. C'est parce que l'ordre des champs est spécifié individuellement pour chaque format "COMPRESSED" et "UNCOMPRESSED".

Voici un exemple :

```
DEFAULT {
  field_1 =:= uncompressed_value(4, 1);
  field_2 =:= uncompressed_value(4, 2);
  field_3 =:= lsb(3, -1);
  ENFORCE(field_4.ULENGTH == 4);
}
```

Ici les liens par défaut sont spécifiés pour les champs 1 à 3. Un lien par défaut pour l'attribut "ULENGTH" de field_4 est aussi spécifié.

Les champs pour lesquels il y a une méthode de codage par défaut n'ont pas besoin que leurs liens soient spécifiés dans la liste des champs de tout format qui utilise la méthode de codage par défaut pour ce champ. Tout format qui n'utilise pas la méthode de codage par défaut doit explicitement spécifier un lien pour la valeur des attributs de ce champ.

Si un lien n'est pas spécifié ailleurs pour les attributs d'un champ, la méthode de codage par défaut est utilisée. Si la méthode de codage par défaut compresse toujours le champ à zéro bit, le champ peut être omis de la liste des champs du format compressé. Comme tout autre champ de zéro bit, sa position dans la liste des champ n'est pas significative.

La liste de champs "DEFAULT" peut contenir des liens par défaut pour des attributs individuels en utilisant des déclarations "ENFORCE". Un lien par défaut pour un attribut individuel va seulement être utilisé si il n'y a pas ailleurs de lien donné pour cet attribut ou le champ auquel il appartient. Si il y a ailleurs une déclaration "ENFORCE" qui lie cet attribut, ou une méthode de codage liant le champ auquel il appartient, le lien par défaut pour l'attribut ne va pas être utilisé. Ceci s'applique même si la méthode de codage spécifiée ne lie pas l'attribut particulier donné dans la section "DEFAULT". Cependant, une déclaration "ENFORCE" ailleurs qui lie seulement la longueur du champ permet quand même l'utilisation des liens par défaut, sauf pour les déclarations "ENFORCE" par défaut qui ne lient que la longueur du champ.

Pour préciser, on supposant les liens par défaut donnés dans l'exemple ci-dessus, les trois premiers des quatre formats compressés suivants n'utiliseraient pas le lien par défaut pour "field_4.ULENGTH" :

```
COMPRESSED format1 {
  ENFORCE(field_4.ULENGTH == 3); // règle ULENGTH à 3
  ENFORCE(field_4.UVALUE == 7);   // règle UVALUE à 7
}

COMPRESSED format2 {
  field_4 =:= irregular(3);      // règle ULENGTH à 3
}

COMPRESSED format3 {
  field_4 =:= '1010';            // règle ULENGTH à zero
}

COMPRESSED format4 {
  ENFORCE(field_4.UVALUE == 12); // utilise ULENGTH par défaut
}
```

Le quatrième format est le seul qui utilise le lien par défaut pour "field_4.ULENGTH".

En résumé, les liens par défaut d'une méthode de codage sont seulement utilisés pour les formats qui ne spécifient pas déjà un lien pour la valeur de tous leurs champs. Pour les formats qui utilisent les liens par défaut, seuls les champs et attributs dont les liens ne sont pas spécifiés sont recherchés dans la liste des champs "DEFAULT".

4.12.2 Arguments

Les méthodes de codage peuvent prendre des arguments qui contrôlent la transposition entre les champs compressés et non compressés. Ils sont spécifiés immédiatement après le nom de la méthode, entre parenthèses, comme une liste séparée par des virgules.

Par exemple :

```
poor_mans_lsb(variable_length)
{
 UNCOMPRESSED {
  constant_bits;
  variable_bits;
 }

 COMPRESSED {
  variable_bits =:= irregular(variable_length);
  constant_bits =:= static;
 }
}
```

Comme avec toute méthode de codage, tous les arguments prennent des valeurs individuelles, comme un littéral entier ou un attribut de champ, plutôt que des champs entiers. Bien que des champs entiers ne puissent pas être passés comme des arguments, il est possible de passer plutôt chacun de leurs attributs, ce qui est équivalent.

On rappelle que tous les liens sont bidirectionnels, de sorte que plutôt que les arguments agissent comme "entrées" à la méthode de codage, le résultat d'une méthode de codage peut être de lier les paramètres qui lui sont passés.

Par exemple :

```
set_to_double(arg1, arg2)
{
 CONTROL {
  ENFORCE(arg1 == 2 * arg2);
 }
}
```

Cette méthode de codage va tenter de lier le premier argument à deux fois la valeur du second. En fait cette méthode de "codage" est pathologique. Comme elle ne définit pas de champs, elle ne fait en réalité aucun codage. Les sections "CONTROL" sont d'un usage plus approprié que "UNCOMPRESSED" pour ce faire.

4.12.3 Formats multiples

Les méthodes de codage peuvent aussi définir plusieurs formats pour un certain en-tête. Cela permet d'utiliser différentes méhodes de compression selon ce qui est la façon la plus efficace de compresser un en-tête particulier.

Par exemple, un champ peut avoir une valeur fixe la plupart du temps, mais la valeur peut occasionnellement changer. En utilisant un seul format pour le codage, ce champ devrait être codé en utilisant "irregular" (voir au paragraphe 4.11.3) même si la valeur ne change que rarement. Cependant, en définissant plusieurs formats, on peut donner deux codages alternatifs : un pour quand la valeur reste fixe et un autre pour quand la valeur change.
C'est le sujet des paragraphes qui suivent

4.12.3.1 Convention de désignation

Quand des formats compressés sont définis, ils doivent l'être en utilisant le mot clé réservé "COMPRESSED". De même, les formats non compressés doivent être définis en utilisant le mot réservé "UNCOMPRESSED". Après chacun de ces mots-clés, un nom peut être donné au format. Si aucun nom n'est donné au format, le nom du format est vide.

Les noms de format, sauf dans le cas où le nom est vide, suivent les règles syntaxique des identifiants, comme décrit au paragraphe 4.2.

Les noms de format doivent être uniques dans la portée de la méthode de codage à laquelle ils appartiennent, sauf le nom vide, qui peut être utilisé pour un format "COMPRESSED" et un format "UNCOMPRESSED".

4.12.3.2 Discrimination de format

Chacun des formats compressés a sa propre liste de champs. Un compresseur peut prendre un de ces formats proposés pour compresser un en-tête, pour autant que les liens de champ qu'il emploie puissent être utilisés avec le format non compressé. Par exemple, le compresseur pourrait choisir de ne pas utiliser un format compressé qui a un codage "static" pour un champ dont l'attribut "UVALUE" diffère de sa valeur correspondante dans le contexte.

Plus formellement, le compresseur peut choisir toute combinaison d'un format non compressé et d'un format compressé pour lequel aucun lien pour aucun des attributs du champ "n'échoue", c'est-à-dire que les méthodes de codage et déclarations "ENFORCE" (voir au paragraphe 4.9) qui lient leurs attributs compressés réussissent. Si il y a plusieurs combinaisons réussies, le compresseur peut choisir l'une d'elles. Autrement si il n'y a pas de combinaison réussie, la méthode de codage "échoue". Un format ne va jamais échouer parce qu'il ne définit pas l'attribut "UVALUE" d'un champ. Un format n'échoue que si il manque à définir un des attributs compressés d'un des champs dans le format compressé, ou laisse indéfinie la longueur du format non compressé.

Parce que le compresseur a un choix, il doit être possible au décompresseur de discriminer entre les différents formats compressés que le compresseur pourrait avoir choisi. Une approche simple de ce problème est que chaque format compressé inclue un "discriminant" qui identifie de façon univoque ce format "COMPRESSED" particulier. Un discriminant est un champ de contrôle; il n'est pas déduit d'une valeur de champ non compressé (voir au paragraphe 4.11.2).

4.12.3.3 Exemple de formats multiples

En mettant tout cela ensemble, voici un exemple complet de la définition d'une méthode de codage avec plusieurs formats compressés :

```
exemple_multiple_formats
{
 UNCOMPRESSED {
  field_1;  //  4 bits
  field_2;  //  4 bits
  field_3;  // 24 bits
 }

 DEFAULT {
  field_1 =:= static;
  field_2 =:= uncompressed_value(4, 2);
  field_3 =:= lsb(4, 0);
 }

 COMPRESSED format0 {
  discriminator =:= '0'; // 1 bit
  field_3;               // 4 bits
 }

 COMPRESSED format1 {
  discriminator =:= '1';          //  1 bit
  field_1       =:= irregular(4); //  4 bits
  field_3       =:= irregular(24); // 24 bits
 }
}
```

Noter que :

- "field_1" et "field_3" ont tous deux spécifiées pour eux des méthodes de codage par défaut, qui sont utilisées dans "format0", mais sont outrepassées dans "format1" ; la méthode de codage par défaut de "field_2" n'est cependant pas outrepassée.
- "field_1" et "field_2" ont des méthodes de codage par défaut qui compressent à zéro bit. Quand cela est utilisé dans "format0", les noms de champs n'apparaissent pas dans la liste des champs.
- "field_3" a une méthode de codage qui ne compresse pas à zéro bit, de sorte que tandis que "field_3" n'a pas de codage spécifié pour lui dans la liste des champs de "format0", il a quand même besoin d'apparaître dans la liste des champs pour spécifier où il va dans le format compressé.
- Dans l'exemple, tous les champs dans le format non compressé ont des méthodes de codage par défaut spécifiées pour

eux, mais ce n'est pas une exigence. Des codages par défaut peuvent être spécifiés pour seulement quelques ou même aucun des champs du format non compressé.

- Dans l'exemple, toutes les méthodes de codage par défaut sont sur les champs du format non compressé, mais ce n'est pas une exigence. Des méthodes de codage par défaut peuvent être spécifiées pour des champs de contrôle.

4.13 Méthodes de codage spécifiques du profil

La bibliothèque des méthodes de codage définie par ROHC-FN au paragraphe 4.11 fournit un ensemble de base et générique de méthodes de codage de champs. Quand on utilise une spécification ROHC-FN dans un profil ROHC, des codages supplémentaires spécifiques de l'en-tête de protocole particulier à compresser peuvent cependant être nécessaires, comme des méthodes qui déduisent la valeur d'un champ d'autres valeurs.

Ces méthodes sont spécifiques des propriétés du protocole compressé et vont donc devoir être définies dans la spécification de profil elle-même. De telles méthodes de codage spécifiques du codage, définies dans la syntaxe de ROHC-FN ou rigoureusement en texte, peuvent être référencées dans la spécification ROHC-FN des formats de profil de la même façon que toute méthode de la bibliothèque ROHC-FN.

Les méthodes de codage qui ne sont pas définies dans la notation formelle sont spécifiées en donnant leur nom, suivi par une brève description de où elles sont définies, entre guillemets, et un point-virgule.

Par exemple :

```
inferred_ip_v4_header_checksum "défini dans la RFCxxxx paragraphe 6.4.1";
```

5. Considérations sur la sécurité

Le présent document décrit une notation formelle similaire à l'ABNF [RFC4234], et donc n'est pas estimé soulever de problèmes de sécurité (noter que l'ABNF a un objet complètement différent de celui de la notation formelle ROHC).

6. Contributeurs

Richard Price a fait beaucoup pour le travail de fondation de la notation formelle. Il est l'auteur du document initial qui décrit une notation formelle sur laquelle se fonde le présent document.

Kristofer Sandlund a contribué au présent travail en appliquant de nouvelles idées au profil ROHC-TCP, en fournissant des réactions, et en aidant à résoudre différents problèmes durant tout le développement de la notation.

Carsten Bormann a fourni la traduction de la syntaxe de la notation formelle en utilisant l'ABNF dans l'Appendice A, et a aussi contribué avec des réactions et des relectures à valider la complétude et la correction de la notation.

7. Remerciements

Un certain nombre de concepts et idées importants ont été empruntés à ROHC [RFC3095].

Merci à Mark West, Eilert Brinkmann, Alan Ford, et Lars-Erik Jonsson pour leurs contributions, relectures, et réactions qui ont conduit à des améliorations significatives de la lisibilité, de la complétude, et de la qualité globale de la notation.

Merci à Stewart Sadler, Caroline Daniels, Alan Finney, et David Findlay de leur relecture et commentaires. Merci à Rob Hancock et Stephen McCann de leur travail antérieur sur la notation formelle. Les auteurs tiennent aussi à remercier Christian Schmidt, Qian Zhang, Hongbin Liao, et Max Riegel de leurs commentaires et précieux apports.

Remerciements supplémentaires : ce document a été revu durant le dernier appel du groupe de travail par les réviseurs mandatés Mark West, Carsten Bormann, et Joe Touch, ainsi que par Sally Floyd qui a fourni une relecture à la demande des directeurs de la zone Transport. Merci aussi à Magnus Westerlund de ses retours dans la préparation de la revue par l'IESG.

8. Références

8.1 Références normatives

[C90] ISO/CEI 9899:1990, "Technologies de l'information -- Langage C de programmation ", avril 1990.

[RFC2822] P. Resnick, "Format de message Internet", avril 2001. *(Remplace la* RFC0822, STD 11, *Remplacée par* RFC5322)

[RFC4234] D. Crocker et P. Overell, "BNF augmenté pour les spécifications de syntaxe : ABNF", octobre 2005. *(Remplace* RFC2234, *remplacée par* RFC5234)

[RFC4995] L-E. Jonsson et autres, "Cadre de la compression d'en-tête robuste (ROHC)", juillet 2007. *(Remplacée par* RFC5795)

8.2 Références pour information

[RFC0791] J. Postel, éd., "Protocole Internet - Spécification du protocole du programme Internet", STD 5, septembre 1981.

[RFC3095] C. Bormann et autres, "Compression d'en-tête robuste (ROHC) : cadre et quatre profils", juillet 2001. (*MàJ par* RFC3759, RFC4815) *(P.S.)*

Appendice A. Syntaxe formelle de ROHC-FN

Cette Section donne une définition de la syntaxe de ROHC-FN en ABNF [RFC4234], en utilisant "fnspec" comme règle de début.

```
; structure globale
fnspec = S *(constdef S) [globctl S] 1*(methdef S)
constdef = constname S "=" S expn S ";"
globctl = CONTROL S formbody
methdef = id S [parmlist S] "{" S 1*(formatdef S) "}" / id S [parmlist S] STRQ *STRCHAR STRQ S ";"
parmlist = "(" S id S *( "," S id S ) ")"
formatdef = formhead S formbody
formhead = UNCOMPRESSED [ 1*WS id ] / COMPRESSED [ 1*WS id ] / CONTROL / INITIAL / DEFAULT
formbody = "{" S *((fielddef/enforcer) S) "}"
fielddef = fieldgroup S [":=" S encspec S] [lenspec S] ";"
fieldgroup = fieldname *( S ":" S fieldname )
fieldname = id
encspec = """ *("0"/"1") """ / id [ S "(" S expn S *( "," S expn S ) ")"]
lenspec = "[" S expn S *("," S expn S) "]"
enforcer = ENFORCE S "(" S expn S ")" S ";"

; expressions
expn = *(expnb S "||" S) expnb
expnb = *(expna S "&&" S) expna
expna = *(expn7 S ("=="/"!=") S) expn7
expn7 = *(expn6 S ("<"/"<="/">"/">=") S) expn6
expn6 = *(expn4 S ("+"/"-") S) expn4
expn4 = *(expn3 S ("*"/"/"/"%") S) expn3
expn3 = expn2 [S "^" S expn3]
expn2 = ["!" S] expn1
expn1 = expn0 / attref / constname / litval / id
expn0 = "(" S expn S ")" / VARIABLE
attref = fieldnameref "." attname
fieldnameref = fieldname / THIS
```

```
attname = ( U / C ) ( LENGTH / VALUE )
litval = ["-"] "0b" 1*("0"/"1") / ["-"] "0x" 1*(DIGIT/"a"/"b"/"c"/"d"/"e"/"f") / ["-"] 1*DIGIT / false / true

; catégories lexicales
constname = UPCASE *(UPCASE / DIGIT / "_")
id = ALPHA *(ALPHA / DIGIT / "_")
ALPHA = %x41-5A / %x61-7A
UPCASE = %x41-5A
DIGIT = %x30-39
COMMENT = "//" *(SP / HTAB / VCHAR) CRLF
SP = %x20
HTAB = %x09
VCHAR = %x21-7E
CRLF = %x0A / %x0D.0A
NL = COMMENT / CRLF
WS = SP / HTAB / NL
S = *WS
STRCHAR = SP / HTAB / %x21 / %x23-7E
STRQ = %x22

; littéraux sensibles à la casse
C = %d67
COMPRESSED = %d67.79.77.80.82.69.83.83.69.68
CONTROL = %d67.79.78.84.82.79.76
DEFAULT = %d68.69.70.65.85.76.84
ENFORCE = %d69.78.70.79.82.67.69
INITIAL = %d73.78.73.84.73.65.76
LENGTH = %d76.69.78.71.84.72
THIS = %d84.72.73.83
U = %d85
UNCOMPRESSED = %d85.78.67.79.77.80.82.69.83.83.69.68
VALUE = %d86.65.76.85.69
VARIABLE = %d86.65.82.73.65.66.76.69
false = %d102.97.108.115.101
true = %d116.114.117.101
```

Appendice B. Exemple au niveau du bit

Cette Section donne un exemple élaboré au niveau du bit, montrant comment une simple spécification ROHC-FN décrit la compression de données réelles d'un en-tête de protocole imaginaire. L'exemple utilisé est très simple, tout en visant quand même à illustrer certaines des complications qui surviennent dans l'utilisation de la notation. En particulier, les champs ont été gardés courts pour rendre possible la lecture de la représentation binaire des en-têtes sans trop de difficulté.

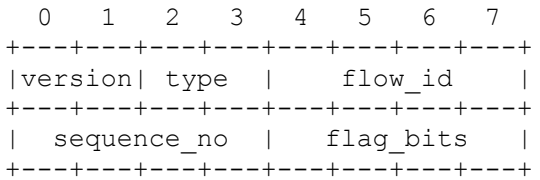
B.1 Exemple de format de paquet

Notre en-tête imaginaire fait juste 16 bits, et consiste en les champs suivants :

1. numéro de version -- 2 bits
2. type -- 2 bits
3. identifiant de flux -- 4 bits
4. numéro de séquence -- 4 bits
5. bits fanions -- 4 bits

Ainsi par exemple 0101000100010000 indique un en-tête avec un numéro de version de un, un type de un, un identifiant de flux de un, un numéro de séquence de un, et tous les bits fanions réglés à zéro.

Voici un diagramme de notation de boîte ASCII de l'en-tête imaginaire :



B.2 Codage initial

Voici une définition initiale fondée seulement sur les informations ci-dessus :

```
eg_header
{
 UNCOMPRESSED {
  version_no   [ 2 ];
  type         [ 2 ];
  flow_id      [ 4 ];
  sequence_no  [ 4 ];
  flag_bits    [ 4 ];
 }

 COMPRESSED initial_definition {
  version_no  =:= irregular(2);
  type        =:= irregular(2);
  flow_id     =:= irregular(4);
  sequence_no =:= irregular(4);
  flag_bits   =:= irregular(4);
 }
}
```

Cela définit bien le format, mais n'offre en fait aucune compression. Si on l'utilise pour coder l'en-tête ci-dessus, on obtient :

```
En-tête non compressé : 0101000100010000
En-tête compressé     : 0101000100010000
```

C'est parce que nous avons déclaré que tous les champs sont "irregular" -- c'est-à-dire, nous n'avons rien spécifié sur leur comportement.

Noter que comme on a seulement un format compressé et un format non compressé, il n'y a pas de différence si les méthodes de codage pour chaque champ sont spécifiées dans le format compressé ou non compressé. Il n'y aurait pas de différence du tout si on écrivait à la place :

```
eg_header
{
 UNCOMPRESSED {
  version_no =:= irregular(2);
  type =:= irregular(2);
  flow_id =:= irregular(4);
  sequence_no =:= irregular(4);
  flag_bits =:= irregular(4);
 }

 COMPRESSED initial_definition {
  version_no   [ 2 ];
  type         [ 2 ];
  flow_id      [ 4 ];
  sequence_no  [ 4 ];
  flag_bits    [ 4 ];
 }
}
```

B.3 Compression de base

Afin de réaliser une compression on a besoin de noter plus d'informations sur l'en-tête et son comportement dans un flux. Par exemple, on peut connaître les faits suivants sur l'en-tête :

1. Le numéro de version -- il indique quelle version du protocole c'est : toujours un pour cette version du protocole.
2. Le type -- peut prendre n'importe quellle valeur.
3. L'identifiant de flux -- peut prendre n'importe quellle valeur.
4. Le numéro de séquence -- prend n'importe quelle valeur.
5. Les bits fanions -- contiennnent trois fanions, a, b, et c, dont chacun peut être établi ou à zéro, et un bit de fanion réservé, qui est toujours à zéro.

On pourrait noter ces informations comme suit :

```
eg_header
{
 UNCOMPRESSED {
   version_no     [ 2 ];
   type           [ 2 ];
   flow_id        [ 4 ];
   sequence_no    [ 4 ];
   abc_flag_bits  [ 3 ];
   reserved_flag  [ 1 ];
 }

 COMPRESSED basic {
   version_no =:= uncompressed_value(2, 1)    [ 0 ];
   type =:= irregular(2)                      [ 2 ];
   flow_id =:= irregular(4)                   [ 4 ];
   sequence_no =:= irregular(4)               [ 4 ];
   abc_flag_bits =:= irregular(3)             [ 3 ];
   reserved_flag =:= uncompressed_value(1, 0) [ 0 ];
 }
}
```

En utilisant ce simple schéma, on a réussi à coder le fait qu'un des champs a une valeur fixe permanente de un, et donc ne contient pas d'informations utiles. On a aussi codé le fait que le bit fanion final est toujours zéro, qui là encore ne contient pas d'informations utiles. Ces deux faits ont été notés en utilisant la méthode de codage "uncompressed_value" (voir au paragraphe 4.11.1).

En utilisant ce nouveau codage sur l'en-tête ci-dessus, on obtient :

```
En-tête non compressé : 0101000100010000
En-tête compressé :   0100010001000
```

Cela réduit la quantité de données dont on a besoin pour transmettre environ 20 %. Cependant, ce codage ne tire pas parti des relations entre les valeurs d'un champ dans un paquet et sa valeur dans les paquets suivants. Par exemple, chaque en-tête dans la séquence suivante est compressé par la même quantité en dépit des similarités entre eux :

```
En-tête non compressé : 0101000100010000
En-tête compressé       : 0100010001000

En-tête non compressé : 0101000101000000
En-tête compressé       : 0100010100000

En-tête non compressé : 0110000101110000
En-tête compressé       : 1000010111000
```

B.4 Compression inter-paquets

Le profil qu'on a défini jusqu'à présent n'a pas compressé du tout le numéro de séquence ni l'identifiant de flux des champs, car ils peuvent prendre n'importe quelle valeur. Cependant la valeur de chacun de ces champs dans un en-tête a une relation très simple avec sa valeur dans les en-têtes précédents :

- o le numéro de séquence est inhabituel -- il augmente de trois à chaque fois,
- o l'identifiant de flux reste le même -- il a toujours la même valeur que dans l'en-tête précédent dans le flux,
- o les bits de fanion abc restent les mêmes la plupart du temps -- ils ont généralement la même valeur que dans l'en-tête précédent dans le flux.

Un façon évidente de noter cela est :

```
// Ce codage évident ne va pas fonctionner (le codage correct est donné en dessous)
    eg_header
    {
     UNCOMPRESSED {
       version_no     [ 2 ];
       type           [ 2 ];
       flow_id        [ 4 ];
       sequence_no    [ 4 ];
       abc_flag_bits  [ 3 ];
       reserved_flag  [ 1 ];
     }

     COMPRESSED obvious {
       version_no =:= uncompressed_value(2, 1);
       type =:= irregular(2);
       flow_id =:= static;
       sequence_no =:= lsb(0, -3);
       abc_flag_bits =:= irregular(3);
       reserved_flag =:= uncompressed_value(1, 0);
     }
    }
```

La dépendance aux paquets précédents est notée en utilisant les méthodes de codage "static" et "lsb" (voir respectivement les paragraphes 4.11.4 et 4.11.5). Cependant la notation ci-dessus pose quelques problèmes.

D'abord, et le plus important, le champ "flow_id" est noté comme "static", ce qui signifie qu'il ne change pas d'un paquet à l'autre. Cependant, la notation n'indique pas comment communiquer initialement la valeur du champ. Il ne sert à rien de dire "c'est la même valeur que la dernière fois" si il n'y a pas eu une première fois où on définit ce qu'est cette valeur, afin qu'on puisse s'y référer. La notation ci-dessus ne fournit pas de moyen de communiquer cela. De même avec le numéro de séquence -- il faut qu'il y ait un moyen de communiquer sa valeur initiale. En fait, sauf pour la notation explicite qui indique leurs longueurs, les longueurs de ces deux champs vont rester indéfinies. Ce problème va être résolu plus loin, dans l'Appendice B.5.

Ensuite, le champ Numéro de séquence est communiqué très efficacement en bits zéro, mais ce n'est pas du tout robuste en cas de perte de paquet. Si un paquet est perdu, il n'y a alors aucun moyen de traiter le numéro de séquence manquant. Quand on communique les numéros de séquence, ou tout autre champ codé avec le codage "lsb", une considération très importante pour le notateur est quelle est la robustesse à la perte de paquet que devrait avoir le protocole compressé. Cela va varier beaucoup d'une pile de protocole à l'autre. Pour l'exemple de protocole on va supposer des flux courts, à faibles frais généraux et on dit qu'il doit être robuste à la perte de juste un paquet, ce qu'on peut réaliser avec deux bits de codage "lsb" (un bit n'est pas assez car le numéro de séquence augmente de trois à chaque fois -- voir au paragraphe 4.11.5). Cela va être traité à l'Appendice B.5.

Finalement, bien que les bits fanions soient généralement les mêmes que dans l'en-tête précédent dans le flux, le profil n'utilise pas ce fait ; comme ils sont parfois différents de ceux de l'en-tête précédent, il n'est pas sûr de dire qu'ils sont toujours les mêmes, de sorte que le codage "static" ne peut pas être utilisé exclusivement. Ce problème sera résolu plus tard par l'utilisation de formats multiples dans l'Appendice B.6.

B.5 Spécification des valeurs initiales

Pour communiquer es valeurs initiales pour les champs compressés avec un codage dépendant du contexte comme "static" ou "lsb" on utilise une liste de champs "INITIAL". Cela peut aider avec les champs dont la valeur de départ est fixe et connue. Par exemple, si on sait qu'au début du flux "flow_id" va toujours être 1 et "sequence_no" va toujours être 0, on pourrait noter cela par :

```
// Ce codage ne va pas fonctionner non plus (le codage correct est en-dessous)
    eg_header
    {
     UNCOMPRESSED {
      version_no      [ 2 ];
      type            [ 2 ];
      flow_id         [ 4 ];
      sequence_no     [ 4 ];
      abc_flag_bits   [ 3 ];
      reserved_flag   [ 1 ];
     }

     INITIAL {    //on règle les valeurs initiales des champs avant le début du flux
      flow_id     =:= uncompressed_value(4, 1);
      sequence_no =:= uncompressed_value(4, 0);
     }

     COMPRESSED obvious {
      version_no =:= uncompressed_value(2, 1);
      type =:= irregular(2);
      flow_id =:= static;
      sequence_no =:= lsb(2, -3);
      abc_flag_bits =:= irregular(3);
      reserved_flag =:= uncompressed_value(1, 0);
     }
    }
```

Cependant, cette utilisation de "INITIAL" n'est pas bonne car les valeurs initiales de "flow_id" et de "sequence_no" varient d'un flux à l'autre. "INITIAL" n'est applicable que lorsque la valeur initiale d'un champ est fixe, comme c'est souvent le cas avec les champs de contrôle.

B.6 Formats de paquets multiples

Pour communiquer correctement les valeurs initiales pour les champs Numéro de séquence et Identifiant de flux, et pour tirer parti du fait que les bits fanions sont généralement les mêmes que dans l'en-tête précédent, on doit abandonner le codage d'un seul format qu'on a utilisé jusqu'à présent pour l'utilisation de formats multiples. Ici, on a exprimé les codages pour deux des champs dans le format non compressé, car ils vont toujours être vrais pour les en-têtes non compressés de ce format. Les champs restants, dont la méthode de codage peut dépendre de la façon exacte dont l'en-tête est compressé, ont leurs codages spécifiés dans les formats compressés.

```
    eg_header
    {
     UNCOMPRESSED {
      version_no =:= uncompressed_value(2, 1)    [ 2 ];
      type                                       [ 2 ];
      flow_id                                    [ 4 ];
      sequence_no                                [ 4 ];
      abc_flag_bits                              [ 3 ];
      reserved_flag =:= uncompressed_value(1, 0) [ 1 ];
     }

     COMPRESSED irregular_format {
      discriminator =:= '0'          [ 1 ];
      version_no                     [ 0 ];
```

```
  type =:= irregular(2)             [ 2 ];
  flow_id =:= irregular(4)          [ 4 ];
  sequence_no =:= irregular(4)  [ 4 ];
  abc_flag_bits =:= irregular(3) [ 3 ];
  reserved_flag                     [ 0 ];
 }

 COMPRESSED compressed_format {
  discriminator =:= '1'          [ 1 ];
  version_no                     [ 0 ];
  type =:= irregular(2)          [ 2 ];
  flow_id =:= static             [ 0 ];
  sequence_no =:= lsb(2, -3)  [ 2 ];
  abc_flag_bits =:= static       [ 0 ];
  reserved_flag                  [ 0 ];
 }
}
```

Noter qu'on a ajouté un champ "discriminator", afin que le décompresseur puisse dire quel format a été utilisé par le compresseur. Le format avec un identifiant de flux "static" et un numéro de séquence codé comme "lsb" est maintenant long de 5 bits. Noter qu'en dépit d'avoir à ajouter le champ "discriminator", ce format a encore la même taille que l'original incorrect "obvious" parce qu'il tire parti du fait que les bits fanions abc changent rarement.

Cependant, le format original "basic" a aussi augmenté d'un bit à cause de l'ajout du discriminant ("irregular_format"). Une importante considération en créant des formats multiples est si chaque format apparaît assez fréquemment pour tirer parti de ce que la longueur d'en-tête compressé moyenne est plus courte par suite de son usage. Par exemple, si en fait les bits fanions changent toujours entre les paquets, le codage "compressed_format" ne pourrait jamais être utilisé ; tout ce qu'on aurait fait serait d'allonger d'un bit le format "basic".

En utilisant la notation ci-dessus, on obtient maintenant :

En-tête non compressé : 0101000100010000
En-tête compressé : 00100010001000

En-tête non compressé : 0101000101000000
En-tête compressé : 10100 ; 00100010100000

En-tête non compressé : 0110000101110000
En-tête compressé : 11011 ; 01000010111000

Le premier en-tête dans le flux est compressé de la même façon qu'avant, sauf qu'il a maintenant le bit supplémentaire du discriminant au début (0). Quand un second en-tête arrive avec le même identifiant de flux que le premier et son numéro de séquence supérieur de trois, il peut être compressé de deux façons possibles : soit en utilisant "compressed_format", soit de la même façon que précédemment en utilisant "irregular_format".

Noter qu'on montre tous les codages théoriquement possibles d'un en-tête comme défini par la spécification ROHC-FN, séparés par des points-virgules. L'un ou l'autre des codages ci-dessus pour chaque en-tête pourrait être produit par une mise en œuvre valide, bien qu'une bonne mise en œuvre viserait toujours à prendre le codage qui conduit à la meilleure compression. Une bonne mise en œuvre tiendrait aussi compte de la robustesse et donc ne supposerait probablement pas sur le second paquet que le décompresseur ait disponible le contexte nécessaire pour décompresser la forme la plus courte de "compressed_format".

Finalement, on note que les champs dont les méthodes de codage sont spécifiées dans le format non compressé ont une longueur zéro lorsque ils sont compressés. Cela signifie que leur position dans le format compressé n'est pas significative. Dans ce cas, il n'est pas nécessaire de les noter lors de la définition des formats compressés. Dans la partie suivante de l'exemple on va voir qu'ils ont été aussi supprimés des formats compressés.

B.7 Discriminants de longueur variable

Supposons qu'on fasse une analyse sur les flux de notre exemple de protocole et qu'on découvre que alors qu'il est usuel

pour les paquets successifs d'avoir les mêmes fanions, lorsque ce n'est pas le cas, le paquet est presque toujours un paquet avec des "fanions établis" dans lequel les trois fanions abc sont établis. Pour coder le flux plus efficacement, un format doit être écrit pour refléter cela.

Cela donne maintenant un total de trois formats, ce qui signifie qu'on a besoin de trois discriminants pour les différencier. La solution évidente ici est d'augmenter le nombre de bits dans le discriminant de un à deux et d'utiliser les discriminants 00, 01, et 10 par exemple. Cependant on peut faire légèrement mieux que cela.

Un discriminant identifiable de façon univoque va suffire, de sorte qu'on puisse utiliser 00, 01, et 1. Si le discriminant commence avec 1, c'est la chose entière. Si il commence par 0, le décompresseur sait qu'il doit vérifier un bit de plus pour déterminer le format.

Noter qu'il faut faire attention en utilisant des discriminants de longueur variable. Par exemple, il serait erroné d'utiliser 0, 01, et 10 comme discriminants car après avoir lu un 0 initial, le décompresseur n'aurait pas de moyen de savoir si le prochain bit était un second bit de discriminant, ou le premier bit du prochain champ dans le format. Cependant, 0, 10, et 11 serait correct, car le premier bit indique là encore si il y a ou non d'autres bits discriminants à suivre.

Cela nous donne ce qui suit :

```
eg_header
{
 UNCOMPRESSED {
  version_no =:= uncompressed_value(2, 1)    [ 2 ];
  type                                       [ 2 ];
  flow_id                                    [ 4 ];
  sequence_no                                [ 4 ];
  abc_flag_bits                              [ 3 ];
  reserved_flag =:= uncompressed_value(1, 0) [ 1 ];
 }

 COMPRESSED irregular_format {
  discriminator =:= '00'            [ 2 ];
  type =:= irregular(2)             [ 2 ];
  flow_id =:= irregular(4)          [ 4 ];
  sequence_no =:= irregular(4)      [ 4 ];
  abc_flag_bits =:= irregular(3)    [ 3 ];
 }

 COMPRESSED flags_set {
  discriminator =:= '01'                     [ 2 ];
  type =:= irregular(2)                      [ 2 ];
  flow_id =:= static                         [ 0 ];
  sequence_no =:= lsb(2, -3)                 [ 2 ];
  abc_flag_bits =:= uncompressed_value(3, 7) [ 0 ];
 }

 COMPRESSED flags_static {
  discriminator =:= '1'             [ 1 ];
  type =:= irregular(2)             [ 2 ];
  flow_id =:= static                [ 0 ];
  sequence_no =:= lsb(2, -3)        [ 2 ];
  abc_flag_bits =:= static          [ 0 ];
 }
}
```

Voici un exemple de résultat :

En-tête non compressé : 0101000100010000
En-tête compressé : 000100010001000

En-tête non compressé : 0101000101000000

En-tête compressé : 10100 ; 000100010100000

En-tête non compressé : 0110000101110000
En-tête compressé : 11011 ; 001000010111000

En-tête non compressé : 0111000110101110
En-tête compressé : 011110 ; 001100011010111

On a ici une séquence très similaire à celle de la dernière fois, sauf qu'il y a maintenant un message supplémentaire sur le bout qui a les bits fanions établis. Le codage pour le premier message dans le flux est maintenant plus grand d'un bit, le codage pour les deux messages suivants est le même que précédemment, car ce format n'a pas augmenté, grâce à l'utilisation de discriminants de longueur variable. Finalement, le paquet qui arrive avec tous les bits fanions établis peut être codé juste en six bits, seulement un bit de plus que le format le plus courant. Sans ce format supplémentaire, ce dernier paquet aurait dû être codé en utilisant le plus long format et aurait pris jusqu'à 14 bits.

B.8 Codage par défaut

Certaines des méthodes de codage courantes utilisées jusqu'à présent ont été "mises en facteurs" dans la définition du format non compressé, ce qui signifie qu'elles n'ont pas besoin d'être définies pour chaque format compressé. Cependant, il y a encore un peu de redondance dans la notation. Pour un certain nombre de champs, la même méthode de codage est utilisée plusieurs fois dans différent formats (bien que pas nécessairement dans tous) mais le codage du champ est redéfini explicitement à chaque fois. Si le codage d'un de ces champs change à l'avenir, alors chaque format qui utilise ce codage va devoir être modifié pour refléter ce changement.

Ce problème peut être évité en spécifiant des méthodes de codage par défaut pour ces champs. Le faire peut aussi conduire à une notation plus concise du profil :

```
eg_header
{
 UNCOMPRESSED {
  version_no =:= uncompressed_value(2, 1)    [ 2 ];
  type                                       [ 2 ];
  flow_id                                    [ 4 ];
  sequence_no                                [ 4 ];
  abc_flag_bits                              [ 3 ];
  reserved_flag =:= uncompressed_value(1, 0) [ 1 ];
 }

 DEFAULT {
  type =:= irregular(2);
  flow_id =:= static;
  sequence_no =:= lsb(2, -3);
 }

 COMPRESSED irregular_format {
  discriminator =:= '00'           [ 2 ];
  type                             [ 2 ];   // Utilise la valeur par défaut
  flow_id =:= irregular(4)         [ 4 ];    // Outrepasse la valeur par défaut
  sequence_no =:= irregular(4)     [ 4 ];    // Outrepasse la valeur par défaut
  abc_flag_bits =:= irregular(3)   [ 3 ];
 }

 COMPRESSED flags_set {
  discriminator =:= '01' [ 2 ];
  type                   [ 2 ];            // Utilise la valeur par défaut
  sequence_no            [ 2 ];            // Utilise la valeur par défaut
  abc_flag_bits =:= uncompressed_value(3, 7);
 }

 COMPRESSED flags_static {
```

```
  discriminator =:= '1' [ 1 ];
  type                 [ 2 ]; // Utilise la valeur par défaut
  sequence_no          [ 2 ]; // Utilise la valeur par défaut
  abc_flag_bits =:= static;
 }
}
```

Le profil ci-dessus se comporte exactement de la même façon que celui noté précédemment , car il a la même signification. Noter que l'idée derrière les différents formats devient plus claire avec la mise en facteur des méthodes de codage par défaut : tout ce qui reste sont les codages qui sont spécifiques de chaque format. Noter aussi que les méthodes de codage par défaut qui compressent à zéro bit sont devenues complètement implicites. Par exemple, les formats compressés en utilisant le codage par défaut pour "flow_id" ne le mentionnent pas (le codage par défaut est "static" qui compresse à zéro bit).

B.9 Champs de contrôle

Une inefficacité dans le schéma de compression produit donc jusque ici est qu'il utilise deux bits pour fournir le numéro de séquence codé en "lsb" avec robustesse à la perte de juste un paquet. En théorie, un seul bit devrait être nécessaire. La racine du problème est le numéro de séquence inhabituel que le protocol utilise -- il compte en incréments de trois. Afin de le coder avec une efficacité maximale, on a besoin de traduire cela en un champ qui incrémente de un à chaque fois. On le fait en utilisant un champ de contrôle.

Un champ de contrôle est fait de données supplémentaires qui sont communiquées dans le format compressé, mais qui ne sont pas un codage direct d'une partie de l'en-tête non compressé. Les champs de contrôle peuvent être utilisés pour communiquer les informations supplémentaires dans le format compressé, ce qui permet que les autres champs soient compressés plus efficacement.

Le champ de contrôle qu'on introduit diminue le numéro de séquence d'un facteur trois. Au lieu de coder le numéro de séquence original dans le paquet compressé, on code le numéro de séquence adapté, ce qui nous permet d'avoir la robustesse à la perte d'un paquet en utilisant juste un bit de codage "lsb" :

```
eg_header
{
 UNCOMPRESSED {
  version_no =:= uncompressed_value(2, 1)    [ 2 ];
  type                                       [ 2 ];
  flow_id                                    [ 4 ];
  sequence_no                                [ 4 ];
  abc_flag_bits                              [ 3 ];
  reserved_flag =:= uncompressed_value(1, 0) [ 1 ];
 }

 CONTROL {
  // on a besoin de l'arithmétique modulo pour calculer correctement l'adaptation, à cause du retour à zéro sur 4 bits
  scaled_seq_no   [ 4 ];
  ENFORCE(sequence_no.UVALUE == (scaled_seq_no.UVALUE * 3) % 16);
 }

 DEFAULT {
  type =:= irregular(2);
  flow_id =:= static;
  scaled_seq_no =:= lsb(1, -1);
 }

 COMPRESSED irregular_format {
  discriminator =:= '00'             [ 2 ];
  type                               [ 2 ];
  flow_id =:= irregular(4)           [ 4 ];
  scaled_seq_no =:= irregular(4) [ 4 ]; // Outrepasse la valeur par défaut
  abc_flag_bits =:= irregular(3)   [ 3 ];
 }
```

```
  COMPRESSED flags_set {
    discriminator =:= '01' [ 2 ];
    type                  [ 2 ];
    scaled_seq_no         [ 1 ];        // Utilise la valeur par défaut
    abc_flag_bits =:= uncompressed_value(3, 7);
  }

  COMPRESSED flags_static {
    discriminator =:= '1' [ 1 ];
    type                  [ 2 ];
    scaled_seq_no         [ 1 ];        // Utilise la valeur par défaut
    abc_flag_bits =:= static;
  }
}
```

Normalement, la ou les méthodes de codage utilisées pour coder un champ spécifient la longueur du champ. Dans la notation ci-dessus, comme il n'y a pas de méthode de codage qui utilise directement "sequence_no", sa longueur doit être définie explicitement en utilisant une déclaration "ENFORCE". Ceci est fait en utilisant la syntaxe abrégée, à la fois pour la cohérence et aussi pour faciliter la lisibilité. Noter que ceci est inhabituel : tandis que la majorité des indications de longueur de champ sont redondantes (et donc facultatives) celles-ci ne le sont pas. Si elles étaient retirées de la notation ci-dessus, la longueur du champ "sequence_no" serait indéfinie.

Voici un exemple de résultat :

```
En-tête non compressé : 0101000100010000
En-tête compressé :   000100011011000

En-tête non compressé : 0101000101000000
En-tête compressé :   1010 ; 000100011100000

En-tête non compressé : 0110000101110000
En-tête compressé :   1101 ; 001000011101000

En-tête non compressé : 0111000110101110
En-tête compressé :   01110 ; 001100011110111
```

Sous cette forme, on voit que cela nous donne une économie d'un bit de plus dans la plupart des paquets. En supposant que le gros d'un flux soit constitué d'en-têtes "flags_static", la taille moyenne des en-têtes dans un flux compressé est maintenant juste un peu plus du quart de leur taille dans un flux non compressé.

B.10 Utilisation des déclarations "ENFORCE" comme conditionnelles

On a créé précédemment un nouveau format "flags_set" pour traiter les paquets avec les trois bits fanions établis. Comme cela arrive, ces trois fanions sont toujours tous établis pour les paquets de "type 3", et ne sont jamais tous établis pour les autres types de paquet (un paquet de "type 3" est celui où le champ "type" est réglé à trois).

Cela permet une efficacité supplémentaire dans le codage de tels paquets. On sait que le type est trois, donc on n'a pas besoin de coder le champ "type" dans l'en-tête compressé. Le champ "type" a été précédemment codé comme "irregular(2)", qui est long de deux bits. Les supprimer réduit la taille du format "flags_set" de cinq bits à trois, le rendant le plus petit format dans la définition de méthode de codage.

Afin de noter que le format "flags_set" devrait seulement être utilisé pour les en-têtes de "type 3", et que le format "flags_static" seulement quand le type n'est pas trois, il est nécessaire de déclarer ces conditions à l'intérieur de chaque format. Cela peut être fait avec une déclaration "ENFORCE" :

```
eg_header
{
 UNCOMPRESSED {
   version_no =:= uncompressed_value(2, 1) [ 2 ];
```

```
  type                                              [ 2 ];
  flow_id                                           [ 4 ];
  sequence_no                                       [ 4 ];
  abc_flag_bits                                     [ 3 ];
  reserved_flag =:= uncompressed_value(1, 0) [ 1 ];
 }

 CONTROL {
  // on a besoin de l'arithmétique modulo pour calculer correctement l'adaptation, à cause du retour à zéro sur 4 bits
  scaled_seq_no   [ 4 ];
  ENFORCE(sequence_no.UVALUE == (scaled_seq_no.UVALUE * 3) % 16);
 }

 DEFAULT {
  type =:= irregular(2);
  scaled_seq_no =:= lsb(1, -1);
  flow_id =:= static;
 }

 COMPRESSED irregular_format {
  discriminator =:= '00'              [ 2 ];
  type                                [ 2 ];
  flow_id       =:= irregular(4)      [ 4 ];
  scaled_seq_no =:= irregular(4) [ 4 ];
  abc_flag_bits =:= irregular(3)   [ 3 ];
 }

 COMPRESSED flags_set {
  ENFORCE(type.UVALUE == 3); // condition redondante
  discriminator =:= '01'                          [ 2 ];
  type =:= uncompressed_value(2, 3)               [ 0 ];
  scaled_seq_no                                   [ 1 ];
  abc_flag_bits =:= uncompressed_value(3, 7)  [ 0 ];
 }

 COMPRESSED flags_static {
  ENFORCE(type.UVALUE != 3);
  discriminator =:= '1'      [ 1 ];
  type                       [ 2 ];
  scaled_seq_no              [ 1 ];
  abc_flag_bits =:= static [ 0 ];
 }
}
```

Les deux déclarations "ENFORCE" dans les deux derniers formats agissent comme "gardes". Les gardes empêchent les formats d'être utilisés dans de mauvaises circonstances. En fait, la déclaration "ENFORCE" dans "flags_set" est redondante. La condition contre laquelle elle garde est déjà appliquée par la nouvelle méthode de codage utilisée pour le champ "type". La méthode de codage "uncompressed_value(2,3)" lie l'attribut "UVALUE" à trois. C'est exactement ce que fait la déclaration "ENFORCE", de sorte q'elle peut être supprimée sans changer le sens. La méthode de codage "uncompressed_value" n'est par ailleurs pas redondante. Elle spécifie d'autres liens sur le champ "type" en plus de celui que spécifie la déclaration "ENFORCE". Donc il ne serait pas possible de supprimer la méthode de codage et laisser juste la déclaration "ENFORCE".

Noter qu'un garde est seulement préventif. Un garde ne peut jamais forcer qu'un format soit choisi par le compresseur. Un format peut seulement être garanti d'être choisi dans une certaine situation si il n'y a pas d'autres formats qui puissent être utilisés à la place. Ceci est démontré dans l'exemple de résultat ci-dessous. Le compresseur peut quand même choisir le format "irregular" si il le souhaite :

En-tête non compressé : 0101000100010000
En-tête compressé : 000100011011000

En-tête non compressé : 0101000101000000
En-tête compressé : 1010 ; 000100011100000

En-tête non compressé : 0110000101110000
En-tête compressé : 1101 ; 001000011101000

En-tête non compressé : 0111000110101110
En-tête compressé : 010 ; 001100011110111

Cela économise juste deux bits supplémentaires (une économie de 7 %) dans l'exemple de flux.

Adresse des auteurs

Robert Finking
Siemens/Roke Manor Research
Old Salisbury Lane
Romsey, Hampshire SO51 0ZN
UK
téléphone : +44 (0)1794 833189
mél : robert.finking@roke.co.uk
URI : http://www.roke.co.uk

Ghyslain Pelletier
Ericsson
Box 920
Lulea SE-971 28
Sweden
téléphone : +46 (0) 8 404 29 43
mél : ghyslain.pelletier@ericsson.com

Déclaration complète de droits de reproduction

Copyright (C) The IETF Trust (2007)

Le présent document est soumis aux droits, licences et restrictions contenus dans le BCP 78, et sauf pour ce qui est mentionné ci-après, les auteurs conservent tous leurs droits.

Le présent document et les informations contenues sont fournis sur une base "EN L'ÉTAT" et le contributeur, l'organisation qu'il ou elle représente ou qui le/la finance (s'il en est), la INTERNET SOCIETY, le IETF TRUST et la INTERNET ENGINEERING TASK FORCE déclinent toutes garanties, exprimées ou implicites, y compris mais non limitées à toute garantie que l'utilisation des informations encloses ne viole aucun droit ou aucune garantie implicite de commercialisation ou d'aptitude à un objet particulier.

Propriété intellectuelle
L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourraient être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur le répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr.

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf-ipr@ietf.org.

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.