Groupe de travail Réseau
Request for Comments : 4741
Catégorie : Sur la voie de la normalisation

R. Enns, éditeur, Juniper Networks
décembre 2006
Traduction Claude Brière de L'Isle

Protocole de configuration NETCONF

Résumé
Le protocole de configuration de réseau (NETCONF, *Network Configuration Protocol*) défini dans ce document fournit des mécanismes pour installer, manipuler, et supprimer la configuration des appareils du réseau. Il utilise un codage de données fondé sur le langage de balisage extensible (XML, *Extensible Markup Language*) pour les données de configuration ainsi que les messages de protocole. Les opérations du protocole NETCONF sont réalisées par dessus une simple couche d'appel de procédure à distance (RPC, *Remote Procedure Call*).

Table des matières

1. Introduction....2
1.1 Vue d'ensemble du protocole....3
1.2 Capacités....3
1.3 Séparation des données de configuration et d'état....4
2. Exigences du protocole de transport....4
2.1 Fonctionnement en mode connexion....4
2.2 Authentification, intégrité, et confidentialité....5
2.3 Authentification....5
2.4 Protocole de transport obligatoire....5
3. Considérations XML....5
3.1 Espace de noms....5
3.2. Pas de déclarations de type de document....5
4. Modèle RPC....5
4.1 Élément <rpc>....6
4.2 Élément <rpc-reply>....6
4.3 Élément <rpc-error>....7
4.4 Élément <ok>....9
4.5 Travail en parallèle....9
5. Modèle de configuration....9
5.1 Mémorisation des données de configuration....9
5.2 Modélisation des données....10
6. Filtrage de sous arborescence....10
6.1 Vue d'ensemble....10
6.2 Composants de filtre de sous arborescence....10
6.3 Traitement de filtre de sous arborescence....12
6.4 Exemples de filtrage de sous arborescence....13
7. Opérations du protocole....19
7.1 <get-config>....19
7.2 <edit-config>....21
7.3 <copy-config>....24
7.4 <delete-config>....25
7.5 <lock>....25
7.6 <unlock>....27
7.7 <get>....27

7.8. <close-session>....28
7.9 <kill-session>....29
8. Capacités....29
8.1 Échange de capacités....30
8.2 Capacité :writable-courant....30
8.3 Capacité de configuration candidate....31
8.4 Capacité :confirmed-commit....33
8.5 Capacité de retour en arrière sur erreur....34
8.6 Validation de capacité....35
8.7 Capacités de démarrage distinctes....36
8.8. Capacité URL....37
8.9 Capacité XPath....38
9. Considérations sur la sécurité....39
10. Considérations relatives à l'IANA....40
10.1 Espace de noms XML NETCONF....40
10.2 Schéma XML NETCONF....40
10.3 URN de capacité NETCONF....40
11. Auteurs et remerciements....40
12. Références....41
12.1 Références normatives....41
12.2 Références pour information....42
Appendice A. Liste d'erreurs NETCONF....42
Appendice B. Schéma XML pour NETCONF RPC et les opérations de protocole....45
Appendice C. Gabarit de capacité....52
Appendice D. Configuration de plusieurs appareils avec NETCONF....53
D.1 Opérations sur les appareils individuels....53
D.2 Opérations sur plusieurs appareils....56
Appendice E. Caractéristiques différées....57
Adresse de l'éditeur....57
Déclaration complète de droits de reproduction....57

1. Introduction

Le protocole NETCONF définit un mécanisme simple par lequel un appareil du réseau peut être géré, des informations de données de configuration peuvent être restituées, et de nouvelles données de configuration peuvent être téléchargées et manipulées. Le protocole permet à l'appareil d'exposer une pleine interface formelle de programmation d'application (API, *application programming interface*). Les applications peuvent utiliser directement cette API pour envoyer et recevoir des ensembles de données de configuration complets et partiels.

Le protocole NETCONF utilise un paradigme d'appel de procédure à distance (RPC, *remote procedure call*). Un client code un RPC en XML [XML] et l'envoie à un serveur en utilisant une session sécurisée, en mode connexion. La réponse du serveur est codée en XML. Le contenu de la demande et de la réponse sont complètements décrits en DTD XML ou en schémas XML, ou les deux, ce qui permet aux deux parties de reconnaître les contraintes de syntaxe imposées à l'échange.

Un aspect clé de NETCONF est qu'il permet à la fonction de protocole de gestion de refléter étroitement la fonction native de l'appareil. Cela réduit les coûts de mise en œuvre et permet un accès en temps utile à de nouvelles caractéristiques. De plus, les applications peuvent accéder au contenu syntaxique et sémantique de l'interface d'utilisateur native de l'appareil.

NETCONF permet à un client de découvrir l'ensemble des extensions de protocole prises en charge par un serveur. Ces "capacités" permettent au client d'ajuster son comportement pour tirer parti des caractéristiques exposées par l'appareil. Les définitions de capacités peuvent être facilement étendues d'une façon non centralisée. Des capacités standard et non standard peuvent être définies avec une rigueur sémantique et syntaxique. Les capacités sont discutées à la Section 8.

Le protocole NETCONF est un bloc de construction dans un système de configuration automatisée. XML est la langue commune de l'échange, fournissant un mécanisme de codage souple mais pleinement spécifié pour un contenu hiérarchisé. NETCONF peut être utilisé de concert avec des technologies de transformation fondées sur XML, comme XSLT [XSLT], pour fournir un système de génération automatique de configurations complètes et partielles. Le système peut interroger une ou plusieurs bases de données sur des données de topologies de réseautage, de liaisons, de politiques, de consommateurs, et de services. Ces données peuvent être transformées en utilisant un ou plusieurs descriptifs XSLT à

partir d'un schéma de données en mode tâche, indépendant du fabricant sous une forme spécifique du fabricant, du produit, du système d'exploitation, et de la livraison du logiciel. Les données résultantes peuvent être passées à l'appareil en utilisant le protocole NETCONF.

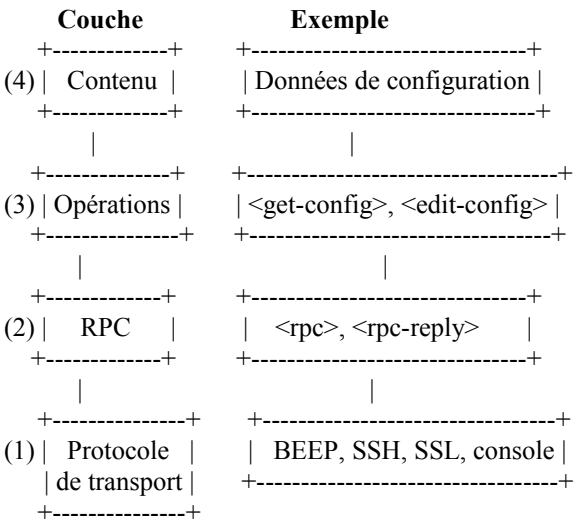
Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

1.1 Vue d'ensemble du protocole

NETCONF utilise un mécanisme simple fondé sur RPC pour faciliter la communication entre un client et un serveur. Le client peut être un descriptif ou une application fonctionnant normalement au titre d'un gestionnaire de réseau. Le serveur est normalement un appareil du réseau. Les termes "appareil" et "serveur" sont utilisés de façon interchangeable dans ce document, comme le sont "client" et "application".

Une session NETCONF est la connexion logique entre un administrateur ou application de configuration de réseau et un appareil du réseau. Un appareil DOIT prendre en charge au moins une session NETCONF et DEVRAIT prendre en charge plusieurs sessions. Des attributs globaux de configuration peuvent être changés durant toute session autorisée, et les effets sont visibles dans toutes les sessions. Les attributs spécifiques d'une session affectent seulement la session dans laquelle ils sont changés.

NETCONF peut être conceptuellement partagé en quatre couches :



1. La couche Protocole de transport fournit un chemin de communication entre le client et le serveur. NETCONF peut être mis en couche sur tout protocole de transport qui fournit un ensemble d'exigences de base. La Section 2 expose ces exigences.

2. La couche RPC fournit un mécanisme de tramage simple, indépendant du transport pour coder les RPC. La Section 4 documente ce protocole.

3. La couche Opérations définit un ensemble d'opérations de base invoquées dans les méthodes de RPC avec des paramètres codés en XML. La Section 7 détaille la liste des opérations de base.

4. La couche Contenu sort du domaine d'application de ce document. Étant donnée la nature propriétaire des données de configuration manipulées, la spécification de ce contenu dépend de la mise en œuvre de NETCONF. On suppose que sera entrepris un effort particulier pour spécifier un langage standard de définition de données et un contenu standard.

1.2 Capacités

Une capacité NETCONF est un ensemble de fonctionnalités qui complètent la spécification NETCONF de base. La capacité est identifiée par un identifiant de ressource universel (URI, *uniform resource identifier*). Ces URI devraient suivre

les lignes directrices de la Section 8.

Les capacités augmentent les opérations de base de l'appareil, décrivant à la fois des opérations supplémentaires et le contenu permis à l'intérieur des opérations. Le client peut découvrir les capacités du serveur et utiliser toute opération, paramètre et contenu supplémentaire défini par ces capacités.

La définition de capacité peut désigner une ou plusieurs capacités dépendantes. Pour prendre en charge une capacité, le serveur DOIT prendre en charge toutes les capacités dont elle dépend.

La Section 8 définit les échanges de capacités qui permettent au client de découvrir les capacités du serveur. La Section 8 fait aussi la liste des capacités définies dans ce document.

Des capacités supplémentaires peuvent être définies à tout moment dans d'autres documents, permettant à l'ensemble des capacités de s'étendre au fil du temps. Les organismes de normalisation peuvent définir des capacités normalisées, et les mises en œuvre peuvent en définir qui sont propriétaires. Un URI de capacité DOIT suffisamment distinguer l'autorité de désignation pour éviter les collisions de dénomination.

1.3 Séparation des données de configuration et d'état

Les informations qui peuvent être restituées d'un système en fonctionnement sont séparées en deux classes, les données de configuration et les données d'état. Les données de configuration sont l'ensemble des données écrivables qui est exigé pour transformer un système de son état initial par défaut à son état en cours. Les données d'état sont les données additionnelles sur un système qui ne sont pas des données de configuration comme des informations d'état en lecture seule et des statistiques collectées. Quand un appareil effectue des opérations de configuration, un certain nombre de problèmes vont se poser si des données d'état ont été incluses :

- Les comparaisons des ensembles de données de configuration seraient dominées par des entrées non pertinentes comme des statistiques différentes.
- Les données entrantes pourraient contenir des demandes absurdes, comme des tentatives d'écriture sur des données en lecture seule.
- Les ensembles de données pourraient être grands.
- Les données archivées pourraient contenir des valeurs pour des éléments de données en lecture seule, compliquant les processus requis pour restaurer les données archivées.

Pour tenir compte de ces problèmes, le protocole NETCONF reconnaît la différence entre données de configuration et données d'état et fournit des opérations pour chacune. L'opération <get-config> restitue seulement les données de configuration, tandis que l'opération <get> restitue les données de configuration et d'état.

Noter que le protocole NETCONF est centré sur les informations requises pour mettre l'appareil dans l'état de fonctionnement désiré. L'inclusion des autres données persistantes importantes est spécifique de la mise en œuvre. Par exemple, les fichiers et bases de données d'utilisateur ne sont pas traitées comme des données de configuration par le protocole NETCONF.

Si une base de données locale de données d'authentification d'utilisateur est mémorisée sur l'appareil, son inclusion dans des données de configuration est une décision qui dépend de la mise en œuvre.

2. Exigences du protocole de transport

NETCONF utilise un paradigme de communication fondé sur RPC. Un client envoie une série d'une ou plusieurs opérations de demande RPC, qui cause la réponse du serveur avec une série correspondante de réponses RPC.

Le protocole NETCONF peut être mis en couches sur tout protocole de transport qui fournit l'ensemble requis de fonctionnalités. Il n'est pas lié à un protocole de transport particulier, mais permet une transposition pour définir comment il peut être mis en œuvre sur tout protocole spécifique.

Le protocole de transport DOIT fournit un mécanisme pour indiquer le type de session (client ou serveur) à la couche de protocole NETCONF.

Cette section détaille les caractéristiques que NETCONF exige du protocole de transport sous-jacent.

2.1 Fonctionnement en mode connexion

NETCONF est en mode connexion, exigeant une connexion persistante entre les homologues. Cette connexion doit fournir une livraison des données fiable, en séquence.

Les connexions NETCONF sont de longue durée, persistant entre les opérations de protocole. cela permet au client de faire des changements à l'état de la connexion qui vont perdurer pour la durée de vie de la connexion. Par exemple, les informations d'authentification spécifiées pour une connexion restent en vigueur jusqu'à la fermeture de la connexion.

De plus, les ressources demandées au serveur pour une connexion particulière DOIVENT être automatiquement libérées quand la connexion ferme, rendant la récupération sur défaillance plus simple et plus robuste. Par exemple, quand un verrouillage est acquis par un client, le verrouillage persiste jusqu'à ce qu'il soit explicitement libéré ou que le serveur détermine que la connexion s'est terminée. Si une connexion est terminée alors que le client détient un verrouillage, le serveur peut effectuer toute récupération appropriée. L'opération de verrouillage est décrite plus en détails au paragraphe 7.5.

2.2 Authentification, intégrité, et confidentialité

Les connexions NETCONF doivent assurer l'authentification, l'intégrité des données, et la confidentialité. NETCONF dépend du protocole de transport pour cette capacité. Un homologue NETCONF suppose que les niveaux de sécurité et de confidentialité appropriés sont fournis indépendamment du présent document. Par exemple, les connexions peuvent être chiffrées en TLS [RFC4346] ou SSH [RFC4251], selon le protocole sous-jacent.

2.3 Authentification

Les connexions NETCONF doivent être authentifiées. Le protocole de transport est responsable de l'authentification. L'homologue suppose que les informations d'authentification de la connexion ont été validées par le protocole sous-jacent en utilisant des mécanismes suffisamment dignes de confiance et que l'identité de l'homologue a été suffisamment prouvée.

Un but de NETCONF est de fournir une interface programmatique à l'appareil qui suive étroitement la fonctionnalité de l'interface native de l'appareil. Donc, on s'attend à ce que le protocole sous-jacent utilise les mécanismes d'authentification existants définis par l'appareil. Par exemple, un appareil qui prend en charge RADIUS [RFC2865] devrait permettre l'utilisation de RADIUS pour authentifier les sessions NETCONF.

Le processus d'authentification devrait résulter en une identité dont les permissions sont connues de l'appareil. Ces permissions DOIVENT être mises en application durant le reste de la session NETCONF.

2.4 Protocole de transport obligatoire

Une mise en œuvre de NETCONF DOIT prendre en charge la transposition SSH de protocole de transport [RFC4742].

3. Considérations XML

XML sert de format de codage pour NETCONF, permettant d'exprimer des données hiérarchiques complexes dans un format de texte qui peut être lu, sauvegardé, et manipulé avec les outils de texte traditionnels et des outils spécifiques de XML.

Cette Section expose un petit nombre de considérations relatives à XML relevant de NETCONF.

3.1 Espace de noms

Tous les éléments de protocole NETCONF sont définis dans l'espace de noms suivant :

urn:ietf:params:xml:ns:netconf:base:1.0

Les noms de capacités NETCONF DOIVENT être des URI [RFC3986]. Les capacités NETCONF sont discutées à la Section 8.

3.2. Pas de déclarations de type de document

Des déclarations de type de document NE DOIVENT PAS apparaître dans le contenu NETCONF.

4. Modèle RPC

Le protocole NETCONF utilise un modèle de communication fondé sur RPC. Les homologues NETCONF utilisent les éléments <rpc> et <rpc-reply> pour fournir un tramage indépendant du protocole de transport des demandes et réponses NETCONF.

4.1 Élément <rpc>

L'élément <rpc> est utilisé pour enclore une demande NETCONF envoyée du client au serveur.
L'élément <rpc> a un attribut obligatoire "message-id", qui est une chaîne arbitraire choisie par l'envoyeur du RPC qui va généralement coder un entier d'accroissement monotone. Le receveur du RPC ne décode ni n'interprète cette chaîne mais la sauvegarde simplement pour l'utiliser dans un attribut "message-id" dans tout message <rpc-reply> résultant. Par exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <méthode quelconque>
    <!-- paramètres de la méthode... -->
  </méthode quelconque>
</rpc>
```

Si des attributs supplémentaires sont présents dans un élément <rpc>, un homologue NETCONF DOIT les retourner non modifiés dans l'élément <rpc-reply>.
Le nom et les paramètres d'un RPC sont codés comme contenu de l'élément <rpc>. Le nom du RPC est un élément directement à l'intérieur de l'élément <rpc>, et tous les paramètres sont codés à l'intérieur de cet élément.

L'exemple suivant invoque une méthode appelée <ma-propre-méthode>, qui a deux paramètres, <mon-premier-paramètre>, avec une valeur de "14", et <un-autre-paramètre>, avec une valeur de "fred" :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  < ma-propre-méthode xmlns="http://exemple.net/moi/ma-propre/1.0">
    <mon-premier-paramètre>14</mon-premier-paramètre>
    <un-autre-paramètre>fred</un-autre-paramètre>
  </ma-propre-méthode>
</rpc>
```

L'exemple suivant invoque une méthode <rock-the-house> avec un paramètre <zip-code> de "27606-0100" :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="http://exemple.net/rock/1.0">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>
```

L'exemple suivant invoque la méthode NETCONF <get> sans paramètre :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>
```

4.2 Élément <rpc-reply>

Le message <rpc-reply> est envoyé en réponse à une opération <rpc>.

L'élément <rpc-reply> a un attribut obligatoire "message-id", qui est égal à l'attribut "message-id" du <rpc> dont il est une réponse.

Un homologue NETCONF DOIT aussi retourner tous les attributs supplémentaires inclus dans l'élément <rpc> non modifiés dans l'élément <rpc-reply>.

Le nom de réponse et les données de réponse sont codés comme le contenu de l'élément <rpc-reply>. Le nom de la réponse est un élément directement à l'intérieur de l'élément <rpc-reply>, et toutes les données sont codées à l'intérieur de cet élément.

Par exemple :

L'élément <rpc> suivant invoque la méthode NETCONF <get> et inclut un attribut supplémentaire appelé "user-id". Noter que l'attribut "user-id" n'est pas dans l'espace de noms NETCONF. L'élément <rpc-reply> retourné donne l'attribut "user-id", ainsi que le contenu demandé.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:ex="http://exemple.net/content/1.0"
    ex:user-id="fred">
  <get/>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:ex="http://exemple.net/content/1.0"
    ex:user-id="fred">
  <data>
    <!-- le contenu vient ici ... -->
  </data>
</rpc-reply>
```

4.3 Élément <rpc-error>

L'élément <rpc-error> est envoyé dans les messages <rpc-reply> si une erreur survient durant le traitement d'une demande <rpc>.

Si un serveur rencontre plusieurs erreurs durant le traitement d'une demande <rpc>, la <rpc-reply> PEUT contenir plusieurs éléments <rpc-error>. Cependant, un serveur n'est pas obligé de détecter ou rapporter plus d'un élément <rpc-error>, si une demande contient plusieurs erreurs. Un serveur n'est pas obligé de vérifier des conditions d'erreur particulières dans une séquence spécifique. Un serveur DOIT retourner un élément <rpc-error> si des conditions d'erreur surviennent durant le traitement et DEVRAIT retourner un élément <rpc-error> si des conditions d'avertissement surviennent durant le traitement.

Un serveur NE DOIT PAS retourner d'informations d'erreur spécifiques de niveau application ou de modèle de données dans un élément <rpc-error> pour lequel le client n'a pas des droits d'accès suffisants.

L'élément <rpc-error> inclut les informations suivantes :

error-type : définit la couche conceptuelle où l'erreur s'est produite.
Énumération. Une de :
* transport
* rpc
* protocol

* application

error-tag : contient une chaîne identifiant la condition d'erreur. Voir à l'Appendice A les valeurs permises.

error-severity : contient une chaîne identifiant la sévérité de l'erreur, comme déterminée par l'appareil. Une de :
* erreur
* avertissement

error-app-tag : contient une chaîne identifiant la condition d'erreur spécifique du modèle de données ou de la mise en œuvre, si il en existe une. Cet élément ne sera pas présent si aucune étiquette d'erreur d'application appropriée ne peut être associée à une condition d'erreur particulière.

error-path : contient l'expression absolue XPath [XPath] qui identifie le chemin de l'élément vers le nœud qui est associé à l'erreur rapportée dans un élément rpc-error particulier. Cet élément ne sera pas présent si aucun élément de charge utile approprié ne peut être associé à une condition d'erreur particulière, ou si la QString 'mauvais élément' retournée dans le conteneur 'error-info' est suffisante pour identifier le nœud associé à l'erreur. Quand l'expression XPath est interprétée, l'ensemble des déclarations d'espace de noms est celui de la portée de l'élément rpc-error, incluant l'espace de noms par défaut.

error-message : contient une chaîne, convenable pour l'affichage à l'homme, qui décrit la condition d'erreur. Cet élément ne va pas être présent si aucun message approprié n'est fourni pour une condition d'erreur particulière. Cet élément DEVRAIT inclure un attribut xml:lang comme défini dans [XML] et discuté dans la [RFC3470].

error-info : Contient un contenu d'erreur spécifique du protocole ou du modèle de données. Cet élément ne va pas être présent si aucun contenu d'erreur n'est fourni pour une condition d'erreur particulière. La liste de l'Appendice A définit tous les contenus error-info obligatoires pour chaque erreur. Après tout contenu exigé par le protocole, une définition de modèle de données peut exiger l'inclusion de certaines informations d'erreur de couche d'application dans le conteneur error-info. Une mise en œuvre peut inclure des éléments supplémentaires pour fournir des informations de débogage étendues et/ou spécifiques de la mise en œuvre.

L'Appendice A énumère les erreurs NETCONF standard.

Exemple :
Une erreur est retournée si un élément <rpc> est reçu sans attribut message-id. Noter que c'est seulement dans ce cas qu'il est acceptable que l'homologue NETCONF omette l'attribut message-id dans l'élément <rpc-reply>.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <courant/>
    </source>
  </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>rpc</error-type>
    <error-tag>missing-attribute</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <bad-attribute>message-id</bad-attribute>
      <bad-element>rpc</bad-element>
    </error-info>
  </rpc-error>
</rpc-reply>
```

La <rpc-reply> suivante illustre le cas de retour de plusieurs éléments <rpc-error>.

Noter que les modèles de données utilisés dans les exemples de cette section utilisent l'élément <name> pour distinguer entre les instances de l'élément <interface>.

```
<rpc-reply message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
 xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
  <error-tag>invalid-value</error-tag>
  <error-severity>error</error-severity>
  <error-message xml:lang="fr">
   La valeur de MTU 25000 n'est pas dans la gamme 256 à 9192
  </error-message>
  <error-info>
   <top xmlns="http://exemple.com/schema/1.2/config">
    <interface>
     <name>Ethernet0/0</name>
     <mtu>25000</mtu>
    </interface>
   </top>
  </error-info>
 </rpc-error>

 <rpc-error>
  <error-type>application</error-type>
  <error-tag>invalid-value</error-tag>
  <error-severity>error</error-severity>
  <error-message xml:lang="fr">
   Adresse IP invalide pour une interface Ethernet1/0
  </error-message>
  <error-info>
   <top xmlns="http://exemple.com/schema/1.2/config">
    <interface xc:operation="replace">
     <name>Ethernet1/0</name>
     <address>
      <name>1.4</name>
      <prefix-length>24</prefix-length>
     </address>
    </interface>
   </top>
  </error-info>
 </rpc-error>
</rpc-reply>
```

4.4 Élément <ok>

L'élément <ok> est envoyé dans les messages <rpc-reply> si aucune erreur ni avertissement ne s'est produit durant le traitement d'une demande <rpc>. Par exemple

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ok/>
</rpc-reply>
```

4.5 Travail en parallèle

Les demandes NETCONF <rpc> DOIVENT être traitées à la suite par l'appareil géré. Des demandes <rpc> supplémentaires PEUVENT être envoyées avant que les précédentes aient été achevées. L'appareil géré DOIT envoyer les réponses dans l'ordre de réception des demandes.

5. Modèle de configuration

NETCONF fournit un ensemble initial d'opérations et un certain nombre de capacités qui peuvent être utilisées pour étendre la base. Les homologues NETCONF échangent les capacités d'appareil quand la session est initiée, comme décrit au paragraphe 8.1.

5.1 Mémorisation des données de configuration

NETCONF définit l'existence d'un ou plusieurs magasins de données de configuration et permet des opérations de configuration sur eux. Un magasin de données de configuration est défini comme l'ensemble complet de données de configuration qui est exigé pour amener un appareil de son état initial par défaut dans l'état de fonctionnement désiré. Le magasin de données de configuration ne comporte pas de données d'état ni de commandes exécutives.

Seul le magasin de données de configuration <courant> est présent dans le modèle de base. Des magasins de données de configuration supplémentaires peuvent être définis par des capacités. De tels magasins de données de configuration ne sont disponibles que sur des appareils qui annoncent les capacités.

o courant : la configuration complète actuellement active sur l'appareil du réseau. Un seul magasin de données de configuration de ce type existe sur l'appareil, et il est toujours présent. Les opérations du protocole NETCONF se réfèrent à ce magasin de données en utilisant l'élément <courant>.

Les capacités des paragraphes 8.3 et 8.7 définissent les magasins de données de configuration <candidate> et <startup>, respectivement.

5.2 Modélisation des données

Les questions de modélisation de données et de contenu sortent du domaine d'application du protocole NETCONF. On fait l'hypothèse que le modèle de données de l'appareil est bien connu de l'application et que les deux parties sont conscientes de questions comme la présentation, le conditionnement, le chiffrement, la recherche, le remplacement, et la gestion des données, ainsi que des autres contraintes imposées par le modèle de données.

NETCONF porte les données de configuration dans l'élément <config> qui est spécifique du modèle de données de l'appareil. Le protocole traite le contenu de cet élément comme des données opaques. L'appareil utilise les capacités pour annoncer l'ensemble de modèles de données que met en œuvre l'appareil. La définition de capacité détaille les opérations et contraintes imposées par le modèle de données.

Les appareils et gestionnaires peuvent prendre en charge plusieurs modèles de données, incluant des modèles de données aussi bien standard que propriétaires.

6. Filtrage de sous arborescence

6.1 Vue d'ensemble

Le filtrage de sous arborescence XML est un mécanisme qui permet à une application de choisir des sous arborescences XML particulières à inclure dans la <rpc-reply> pour une opération <get> ou <get-config>. Un petit ensemble de filtres d'inclusion, de correspondance exacte de simple contenu, et de sélection est fourni, qui permet des mécanismes de sélection très utiles, mais aussi très limités. L'agent n'a pas besoin d'utiliser de sémantique spécifique du modèle de données durant le traitement, ce qui permet des stratégies de mise en œuvre simples et centralisées.

Conceptuellement, un filtre de sous arborescence comporte zéro, une ou plusieurs sous arborescences d'éléments, qui représentent le critère de sélection du filtre. À chaque niveau de contenance d'une sous arborescence, l'ensemble de nœuds parents est traité logiquement par le serveur pour déterminer si sa sous arborescence et le chemin des éléments jusqu'à la racine sont inclus dans le résultat du filtre.

Tous les éléments présents dans une sous arborescence particulière au sein d'un filtre doivent correspondre aux nœuds associés présents dans le modèle conceptuel de données du serveur. Des espaces de noms XML peuvent être spécifiés (via des déclarations 'xmlns') au sein du modèle de données du filtre. Si il en est, l'espace de noms déclaré doit d'abord

correspondre exactement à un espace de noms accepté par le serveur. Noter que les valeurs de préfixes pour les espaces de noms qualifiés ne sont pas pertinents pour la comparaison des éléments de filtre avec des éléments dans le modèle de données sous-jacent. Seules les données associées à un espace de noms spécifié vont être incluses dans le résultat du filtre.

Chaque nœud spécifié dans un filtre de sous arborescence représente un filtre inclusif. Seuls les nœuds associés dans le ou les modèles de données sous-jacents au sein du magasin de données de configuration spécifié sur le serveur sont choisis par le filtre. Un nœud doit correspondre exactement à l'espace de noms et à la hiérarchie des éléments donnés dans les données de filtre, sauf que le nom de chemin absolu du filtre est ajusté pour commencer à la couche en dessous de <filtre>.

Les messages de réponse contiennent seulement les sous arborescences choisies par le filtre. Tout critère de choix présent dans la demande, au sein d'une sous arborescence choisie particulière, est aussi inclus dans la réponse. Noter que certains éléments exprimés dans le filtre comme des nœuds d'extrémités vont être expansés (c'est-à-dire que des sous arborescences vont être incluses) dans le résultat du filtre. Les instances de données spécifiques ne sont pas dupliquées dans la réponse au cas où la demande contiendrait plusieurs expressions de sous arborescence de filtre qui sélectionneraient les mêmes données.

6.2 Composants de filtre de sous arborescence

Un filtre de sous arborescence se compose d'éléments XML et de leurs attributs XML. Il y a cinq types de composants qui peuvent être présents dans un filtre de sous arborescence :

- Choix d'espace de noms
- Expressions de correspondance d'attributs
- Nœuds de contenance
- Nœuds de sélection
- Nœuds de correspondance de contenu

6.2.1 Choix d'espace de noms

Si des espaces de noms sont utilisés, alors le résultat du filtre va seulement inclure des éléments provenant de l'espace de noms spécifié. Un espace de noms est considéré comme correspondant (pour les besoins du filtre) si le contenu des attributs 'xmlns' est le même dans le filtre et dans le modèle de données sous-jacent. Noter que le choix d'espace de noms ne peut pas être utilisé par lui-même. Au moins un élément doit être spécifié dans le filtre parmi les éléments qui doivent être inclus dans le résultat du filtre.

Exemple :

```
<type de filtre ="sous-aborescence">
  <sommet xmlns="http://exemple.com/schema/1.2/config"/>
</filtre>
```

Dans cet exemple, l'élément <sommet> est un nœud de sélection, et seulement ce nœud et ses nœuds fils (provenant du modèle de données sous-jacent) dans l'espace de noms 'http://exemple.com/schema/1.2/config' vont être inclus dans le résultat du filtre.

6.2.2 Expressions de correspondance d'attributs

Un attribut qui apparaît dans un filtre de sous arborescence fait partie d'une "expression de correspondance d'attribut". Tout nombre d'attributs XML (qualifiés ou non qualifiés) peut être présent dans tout type de nœud filtre. En plus des critères de sélection normalement applicables à ce nœud, les données choisies doivent avoir des valeurs qui correspondent pour chaque attribut spécifié dans le nœud. Si un élément n'est pas défini comme incluant un attribut spécifié, il n'est alors pas choisi dans le résultat du filtre.

Exemple :

```
<type de filtre="sous arborescence">
  <t:top xmlns:t="http://exemple.com/schema/1.2/config">
    <t:interfaces>
      <t:interface t:ifName="eth0"/>
    </t:interfaces>
```

```
  </t:top>
</filtre>
```

Dans cet exemple, les éléments <top>, <interfaces>, et <interface> sont des nœuds contenants, et 'ifName' est une expression de correspondance d'attribut. Seuls les nœuds 'interface' dans l'espace de noms 'http://exemple.com/schema/1.2/config' qui ont un attribut 'ifName' de valeur 'eth0' et surviennent dans les nœuds 'interfaces' au sein des nœuds 'top' vont être inclus dans le résultat du filtre.

6.2.3 Nœuds contenants

Les nœuds qui contiennent des éléments fils dans un filtre de sous arborescence sont appelés des "nœuds contenants". Chaque élément fils peut être de tout type de nœud, incluant un autre nœud contenant. Pour chaque nœud contenant spécifié dans un filtre de sous arborescence, toutes les instances de modèle de données qui correspondent exactement à l'espace de noms spécifié, à la hiérarchie d'éléments, et toutes les expressions de correspondance d'attribut, sont incluses dans le résultat du filtre.

Exemple :

```
<type de filtre="sous arborescence">
  <top xmlns="http://exemple.com/schema/1.2/config">
    <users/>
  </top>
</filter>
```

Dans cet exemple, l'élément <top> est un nœud contenant.

6.2.4 Nœuds de sélection

Un nœud d'extrémité vide au sein d'un filtre est appelé un "nœud de sélection", et il représente un filtre de "sélection explicite" sur le modèle de données sous-jacent. La présence de tous nœuds de sélection au sein d'un ensemble de nœuds apparentés va causer la sélection par le filtre des sous arborescences spécifiées et la suppression du choix automatique de l'ensemble entier de nœuds apparentés dans le modèle de données sous-jacent. Pour les besoins du filtrage, un nœud d'extrémité vide peut être déclaré avec une étiquette vide (par exemple, <foo/>) ou avec des étiquettes explicites de début et de fin (par exemple, <foo> </foo>). Tous les caractères d'espace sont ignorés dans cette forme.

Exemple :

```
<filter type="subtree">
  <top xmlns="http://exemple.com/schema/1.2/config">
    <users/>
  </top>
</filter>
```

Dans cet exemple, l'élément <top> est un nœud contenant, et l'élément <users> est un nœud de sélection. Seuls les nœuds 'users' dans l'espace de noms 'http://exemple.com/schema/1.2/config' qui se produisent au sein d'un élément 'top' qui est la racine du magasin de données de configuration vont être inclus dans le résultat du filtre.

6.2.5 Nœuds de correspondance de contenu

Un nœud d'extrémité qui contient un contenu simple est appelé un "nœud de correspondance de contenu". Il est utilisé pour choisir certains ou tous ses nœuds apparentés comme résultat du filtre, et il représente un filtre de correspondance exacte sur le contenu de l'élément nœud d'extrémité. Les contraintes suivantes s'appliquent aux nœuds de correspondance de contenu :

- Un nœud de correspondance de contenu ne doit pas contenir des éléments incorporés (c'est-à-dire, il doit se résoudre en un simpleType dans la définition de schéma XML (XSD, *XML Schema Definition*)).
- Plusieurs nœuds de correspondance de contenu (c'est-à-dire, des nœuds apparentés) sont logiquement combinés dans une expression "ET".
- Le filtrage de contenu mixte n'est pas pris en charge.
- Le filtrage d'un contenu de liste n'est pas pris en charge.

- Le filtrage de contenu constitué seulement d'espaces n'est pas pris en charge.
- Un nœud de correspondance de contenu doit contenir des caractères non d'espace. Un élément vide (par exemple, <foo></foo>) va être interprété comme un nœud de sélection (par exemple, <foo/>).
- Les caractères d'espace en tête et en queue sont ignorés, mais tous les caractères espace au sein d'un bloc de caractères de texte ne sont ni ignorés ni modifiés.

Si tous les nœuds de correspondance de contenu apparentés spécifiés dans une expression de filtre de sous arborescence sont 'vrais', alors les nœuds du résultat du filtre sont sélectionnés de la manière suivante :
- Chaque nœud dont le contenu correspond dans l'ensemble apparenté est inclus dans le résultat du filtre.
- Si des nœuds contenants sont présents dans l'ensemble apparenté, alors ils sont traités et inclus si des critères de filtre incorporé sont aussi satisfaits.
- Si des nœuds de sélection sont présents dans l'ensemble apparenté, alors tous sont inclus dans le résultat du filtre.
- Autrement (c'est-à-dire, si il n'y a pas de nœud de sélection ou contenant dans l'ensemble apparenté filtré) tous les nœuds définis à ce niveau dans le modèle de données sous-jacent (et leurs sous arborescences si il en est) sont retournés dans le résultat du filtre.

Si un ou des essais de correspondance de contenu de nœud apparenté sont 'faux', alors aucun autre traitement de filtre n'est effectué sur cet ensemble apparenté, et aucune des sous arborescences apparentées n'est sélectionnée par le filtre, ni le ou les nœuds de correspondance de contenu.

Exemple :

```
<type de filtre="sous arborescence">
  <top xmlns="http://exemple.com/schema/1.2/config">
    <users>
      <user>
        <nom>fred</nom>
      </user>
    </users>
  </top>
</filter>
```

Dans cet exemple, les nœuds <users> et <user> sont tous deux des nœuds contenants, et <nom> est un nœud de correspondance de contenu. Comme aucun nœud apparenté de <nom> n'est spécifié (et donc aucun nœud contenant ou de sélection) tous les nœuds apparentés de <nom> sont retournés dans le résultat du filtre. Seuls les nœuds 'user' dans l'espace de noms 'http://exemple.com/schema/1.2/config' qui correspondent à la hiérarchie d'éléments et pour lesquels l'élément <nom> est égal à 'fred' vont être inclus dans le résultat du filtre.

6.3. Traitement de filtre de sous arborescence

Le résultat du filtre (l'ensemble des nœuds choisis) est initialement vide.

Chaque filtre de sous arborescence peut contenir un ou plusieurs fragments de modèle de données, qui représentent des portions du modèle de données qui devrait être choisi (avec tous les nœuds fils) dans le résultat du filtre.

Chaque fragment de données de sous-arborescence est comparé par le serveur aux modèles internes de données supportés par le serveur. Si le filtre entier de fragment de données de sous-arborescence (en commençant par la racine jusqu'à l'élément le plus interne spécifié dans le filtre) correspond exactement à une portion correspondante du modèle de données supporté, alors ce nœud et tous ses enfants sont inclus dans les données de résultat.

Le serveur traite ensemble tous les nœuds avec les mêmes nœuds parents (ensemble apparenté) en commençant de la racine vers les nœuds d'extrémité. Les éléments racines dans le filtre sont considérés dans le même ensemble apparenté (en supposant qu'ils sont dans le même espace de noms) même si il n'ont pas de parent commun.

Pour chaque ensemble apparenté, le serveur détermine quels nœuds sont inclus (ou potentiellement inclus) dans le résultat du filtre, et quelles sous arborescences apparentées sont exclues (élaguées) du résultat du filtre. Le serveur détermine d'abord quels types de nœuds sont présents dans l'ensemble apparenté et traite les nœuds en accord avec les règles pour leur type. Si aucun nœud de l'ensemble apparenté n'est choisi, le processus est alors appliqué de façon récurrente aux ensembles apparentés de chaque nœud choisi. L'algorithme continue jusqu'à ce que tous les ensembles apparentés dans toutes les sous arborescences spécifiées dans le filtre aient été traitées.

6.4 Exemples de filtrage de sous arborescence

6.4.1 Pas de filtre

Laisse le filtre sur l'opération "get" retourner le modèle de données entier.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <!-- ... ensemble entier de données retourné ... -->
  </data>
</rpc-reply>
```

6.4.2 Filtre vide

Un filtre vide ne va rien sélectionner parce que aucun nœud de correspondance de contenu ou de sélection n'est présent. Ce n'est pas une erreur. L'attribut type de filtre utilisé dans ces exemples est discuté au paragraphe 7.1.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
    </filter>
  </get>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
  </data>
</rpc-reply>
```

6.4.3 Choix de la sous arborescence <users> entière

Le filtre dans cet exemple contient un nœud de sélection (<users>) de sorte que juste cette sous arborescence est choisie par le filtre. Cet exemple représente le modèle de données <users> pleinement rempli dans la plupart des exemples de filtre qui suivent. Dans un modèle de données réel, le champ <company-info> ne serait probablement pas retourné avec la liste des utilisateurs d'un hôte ou réseau particulier.

Note : Les exemples de filtrage et de configuration utilisés dans ce document apparaissent dans l'espace de noms "http://exemple.com/schema/1.2/config". L'élément racine de cet espace de noms est <top>. L'élément <top> et ses descendants représente seulement un exemple de modèle de données de configuration.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <courant/>
    </source>
    <filter type="subtree">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <users/>
      </top>
```

```
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://exemple.com/schema/1.2/config">
      <users>
        <user>
          <nom>racine</nom>
          <type>superuser</type>
          <full-name>Charlie Root</full-name>
          <company-info>
            <dept>1</dept>
            <id>1</id>
          </company-info>
        </user>
        <user>
          <nom>fred</nom>
          <type>admin</type>
          <full-name>Fred Flintstone</full-name>
          <company-info>
            <dept>2</dept>
            <id>2</id>
          </company-info>
        </user>
        <user>
          <nom>barney</nom>
          <type>admin</type>
          <full-name>Barney Rubble</full-name>
          <company-info>
            <dept>2</dept>
            <id>3</id>
          </company-info>
        </user>
      </users>
    </top>
  </data>
</rpc-reply>
```

La demande de filtre suivante aurait produit le même résultat, mais seulement parce que le conteneur <users> définit un élément fils (<user>).

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <courant/>
    </source>
    <filter type="subtree">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <users>
          <user/>
        </users>
      </top>
    </filter>
  </get-config>
</rpc>
```

6.4.4 Choix de tous les éléments <nom> dans la sous arborescence <users>

Ce filtre contient deux nœuds contenants (<users>, <user>) et un nœud sélecteur (<nom>). Toutes les instances de l'élément <nom> dans le même ensemble apparenté sont choisies dans le résultat du filtre. Le gestionnaire peut avoir besoin de savoir que <nom> est utilisé comme identifiant d'instance dans cette structure de données particulière, mais le serveur n'a pas besoin de savoir ces méta-données pour traiter la demande.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get-config>
  <source>
   <courant/>
  </source>
  <filter type="subtree">
   <top xmlns="http://exemple.com/schema/1.2/config">
    <users>
     <user>
      <nom/>
     </user>
    </users>
   </top>
  </filter>
 </get-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <top xmlns="http://exemple.com/schema/1.2/config">
   <users>
    <user>
     <nom>racine</nom>
    </user>
    <user>
     <nom>fred</nom>
    </user>
    <user>
     <nom>barney</nom>
    </user>
   </users>
  </top>
 </data>
</rpc-reply>
```

6.4.5 Une entrée <user> spécifique

Ce filtre contient deux nœuds contenants (<users>, <user>) et un nœud de correspondance de contenu (<nom>). Toutes les instances de l'ensemble apparenté contenant <nom> pour lequel la valeur de <nom> égale "fred" sont choisies dans le résultat du filtre.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get-config>
  <source> <courant/> </source>
  <type de filtre="sous arborescence">
   <top xmlns="http://exemple.com/schema/1.2/config">
    <users>
     <user>
      <nom>fred</nom>
     </user>
    </users>
```

```
    </top>
   </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
   <top xmlns="http://exemple.com/schema/1.2/config">
    <users>
     <user>
      <nom>fred</nom>
      <type>admin</type>
      <full-name>Fred Flintstone</full-name>
      <company-info>
       <dept>2</dept>
       <id>2</id>
      </company-info>
     </user>
    </users>
   </top>
 </data>
</rpc-reply>
```

6.4.6 Éléments spécifiques 'une entrée <user> spécifique

Ce filtre contient deux nœuds contenants (<users>, <user>) un nœud de correspondance de contenu (<nom>) et deux nœuds sélecteurs (<type>, <full-name>). Toutes les instances des éléments <type> et <full-name> dans le même ensemble apparenté contenant <nom> pour lequel la valeur de <nom> égale "fred" sont choisies dans le résultat du filtre. L'élément <company-info> n'est pas inclus parce que l'ensemble apparenté contient des nœuds de sélection.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get-config>
   <source> <courant/> </source>
   <type de filtre="sous arborescence">
    <top xmlns="http://exemple.com/schema/1.2/config">
     <users>
      <user>
       <nom>fred</nom>
       <type/>
       <full-name/>
      </user>
     </users>
    </top>
   </filter>
 </get-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
   <top xmlns="http://exemple.com/schema/1.2/config">
    <users>
     <user>
      <nom>fred</nom>
      <type>admin</type>
      <full-name>Fred Flintstone</full-name>
     </user>
    </users>
```

```
    </top>
  </data>
</rpc-reply>
```

6.4.7 Sous arborescences multiples

Ce filtre contient trois sous arborescences (nom=racine, fred, barney).

Le filtre de sous arborescence "racine" contient deux nœuds contenants (<users>, <user>) un nœud de correspondance de contenu (<nom>), et un nœud sélecteur (<company-info>). Les critères de choix d'arborescence sont satisfaits, et seule la sous arborescence company-info pour "racine" est choisie dans le résultat du filtre.

Le filtre de sous arborescence "fred" contient trois nœuds contenants (<users>, <user>, <company-info>) un nœud de correspondance de contenu (<nom>), et un nœud sélecteur (<id>). Les critères de choix d'arborescence sont satisfaits, et seul l'élément <id> dans la sous arborescence company-info pour "fred" est choisi dans le résultat du filtre.

Le filtre de sous arborescence "barney" contient trois nœuds contenants (<users>, <user>, <company-info>) deux nœuds de correspondance de contenu (<nom>, <type>) et un nœud sélecteur (<dept>). Les critères de choix d'arborescence ne sont pas satisfaits parce que l'usager "barney" n'est pas un "super usager", et la sous arborescence entière pour "barney" (y compris son entrée <user> parente) est exclue du résultat du filtre.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source> <courant/> </source>
    <filter type="subtree">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <users>
          <user>
            <nom>racine</nom>
            <company-info/>
          </user>
          <user>
            <nom>fred</nom>
            <company-info>
              <id/>
            </company-info>
          </user>
          <user>
            <nom>barney</nom>
            <type>superuser</type>
            <company-info>
              <dept/>
            </company-info>
          </user>
        </users>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://exemple.com/schema/1.2/config">
      <users>
        <user>
          <nom>racine</nom>
          <company-info>
            <dept>1</dept>
```

```
          <id>1</id>
         </company-info>
        </user>
        <user>
         <nom>fred</nom>
         <company-info>
          <id>2</id>
         </company-info>
        </user>
       </users>
      </top>
     </data>
    </rpc-reply>
```

6.4.8 Éléments avec désignation d'attribut

Dans cet exemple, le filtre contient un nœud contenant (<interfaces>) une expression de correspondance d'attribut (ifName) et un nœud sélecteur (<interface>). Toutes les instances de l'arborescence <interface> qui ont un attribut ifName égal à "eth0" sont choisies dans le résultat du filtre. Les éléments et attributs de données du filtre doivent être qualifiés parce que l'attribut ifName ne va pas être considéré comme faisant partie de l'espace de noms 'schema/1.2' si il n'est pas qualifié.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <type de filtre="sous arborescence">
   <t:top xmlns:t="http://exemple.com/schema/1.2/stats">
    <t:interfaces>
     <t:interface t:ifName="eth0"/>
    </t:interfaces>
   </t:top>
  </filter>
 </get>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <t:top xmlns:t="http://exemple.com/schema/1.2/stats">
   <t:interfaces>
    <t:interface t:ifName="eth0">
     <t:ifInOctets>45621</t:ifInOctets>
     <t:ifOutOctets>774344</t:ifOutOctets>
    </t:interface>
   </t:interfaces>
  </t:top>
 </data>
</rpc-reply>
```

Si ifName était un nœud fils au lieu d'un attribut, la demande suivante aurait alors produit un résultat similaire.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <type de filtre="sous arborescence">
   <top xmlns="http://exemple.com/schema/1.2/stats">
    <interfaces>
     <interface>
      <ifName>eth0</ifName>
     </interface>
    </interfaces>
```

```
      </top>
    </filter>
  </get>
</rpc>
```

7. Opérations du protocole

Le protocole NETCONF fournit un petit ensemble d'opérations de bas niveau pour gérer les configurations d'appareils et restituer les informations d'état d'appareil. Le protocole de base fournit des opérations pour restituer, configurer, copier, et supprimer les magasins de données de configuration. Des opérations supplémentaires sont fournies, sur la base des capacités annoncées par l'appareil.

Le protocole de base inclut les opérations de protocole suivantes :
- o get *(obtenir)*
- o get-config *(obtenir la configuration)*
- o edit-config *(éditer la configuration)*
- o copy-config *(copîer la configuration)*
- o delete-config *(supprimer la configuration)*
- o lock *(verrouiller)*
- o unlock *(déverrouiller)*
- o close-session *(terminer la session)*
- o kill-session *(tuer la session)*

Une opération de protocole peut échouer pour diverses raisons, incluant "opération non prise en charge". Un initiateur ne devrait pas supposer que toutes les opérations vont toujours réussir. Les valeurs de retour dans toute réponse RPC devraient être vérifiées à la recherche de réponses d'erreur.

La syntaxe et le codage XML des opérations de protocole sont formellement définis dans le schéma XML de l'Appendice B. Les paragraphes suivants décrivent la sémantique de chaque opération du protocole.

7.1 <get-config>

Description : Restitue tout ou partie d'une configuration spécifiée.

Paramètres :
source : nom du magasin de données de configuration qui est interrogé, comme <courant/>.

filtre : l'élément filtre identifie les portions de la configuration d'appareil à restituer. Si cet élément n'est pas spécifié, la configuration entière est retournée. L'élément filtre peut facultativement contenir un attribut "type". Cet attribut indique le type de la syntaxe de filtrage utilisée dans l'élément filtre. Le mécanisme de filtrage par défaut dans NETCONF est appelé le filtrage de sous arborescence et est décrit à la Section 6. La valeur "sous arborescence" identifie explicitement ce type de filtrage. Si l'homologue NETCONF prend en charge la capacité :xpath (paragraphe 8.9) la valeur "xpath" peut être utilisée pour indiquer que l'attribut choisi sur l'élément filtre contient une expression XPath.

Réponse positive : Si l'appareil peut satisfaire la demande, le serveur envoie un élément <rpc-reply> contenant un élément <data> avec le résultat de l'interrogation.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être menée à bien pour une raison quelconque.

Exemple : pour restituer l'arborescence <users> entière :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <courant/>
    </source>
```

```
    <filter type="subtree">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <users/>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://exemple.com/schema/1.2/config">
      <users>
        <user>
          <nom>racine</nom>
          <type>superuser</type>
          <full-name>Charlie Root</full-name>
          <company-info>
            <dept>1</dept>
            <id>1</id>
          </company-info>
        </user>
        <!-- les éléments <user> supplémentaires apparaissent ici... -->
      </users>
    </top>
  </data>
</rpc-reply>
```

Si la configuration est disponible en plusieurs formats, comme XML et text, un espace de noms XML peut être utilisé pour spécifier quel format est désiré. Dans l'exemple suivant, le client utilise un élément spécifique (<config-text>) dans un espace de noms spécifique pour indiquer au serveur le désir de recevoir la configuration dans un autre format. Le serveur peut prendre en charge un nombre quelconque de formats ou vues distincts dans les données de configuration, avec le client qui utilise le paramètre <filtre> pour choisir entre eux.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <courant/>
    </source>
    <type de filtre="sous arborescence">
      <!-- demande une version texte de la configuration -->
      <config-text xmlns="http://exemple.com/text/1.2/config"/>
    </filtre>
  </get-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <config-text xmlns="http://exemple.com/text/1.2/config">
      <!-- texte de la configuration ... -->
    </config-text>
  </data>
</rpc-reply>
```

La Section 6 contient des exemples supplémentaires de filtrage de sous arborescence.

7.2 <edit-config>

Description : L'opération <edit-config> charge tout ou partie d'une configuration spécifiée à la configuration cible spécifiée. Cette opération permet d'exprimer la nouvelle configuration de plusieurs façons, comme d'utiliser un fichier local, un fichier distant, ou en ligne. Si la configuration cible n'existe pas, elle va être créée. Si un homologue NETCONF prend en charge la capacité :url (paragraphe 8.8), l'élément <url> peut apparaître à la place du paramètre <config> et devrait identifier un fichier de configuration local.

L'appareil analyse les configurations de source et de cible et effectue les changements demandés. La configuration cible n'est pas nécessairement remplacée, comme avec le message <copy-config>. La configuration cible est plutôt changée en accord avec les données et les opérations demandées de la source.

Attributs :

operation : des éléments dans la sous arborescence <config> peuvent contenir un attribut "operation". L'attribut identifie le point dans la configuration où effectuer l'opération et PEUT apparaître sur plusieurs éléments dans la sous arborescence <config>. Si l'attribut operation n'est pas spécifié, la configuration est fusionnée dans le magasin de données de configuration. L'attribut operation a une des valeurs suivantes :

merge : les données de configuration identifiées par l'élément contenant cet attribut sont fusionnées avec la configuration au niveau correspondant dans le magasin de données de configuration identifié par le paramètre cible. C'est le comportement par défaut.

replace : les données de configuration identifiées par l'élément contenant cet attribut remplacent toute configuration en rapport dans le magasin de données de configuration identifié par le paramètre cible. À la différence de l'opération <copy-config>, qui remplace la configuration cible entière, seule la configuration réellement présente dans le paramètre config est affectée.

create : les données de configuration identifiées par l'élément qui contient cet attribut sont ajoutées à la configuration si et seulement si les données de configuration n'existent pas déjà sur l'appareil. Si les données de configuration existent, un élément <rpc-error> est retourné avec une valeur <error-tag> de "data-exists".

delete : les données de configuration identifiées par l'élément qui contient cet attribut sont supprimées dans le magasin de données de configuration identifié par le paramètre cible.

Paramètres :

cible : nom du magasin de données de configuration qui est édité, comme <courant/> ou <candidate/>.

default-operation : choisit l'opération par défaut (comme décrit dans l'attribut "operation") pour cette demande <edit-config>. La valeur par défaut pour le paramètre default-operation est "merge". Le paramètre default-operation est facultatif, mais si il est fourni, il doit avoir une des valeurs suivantes :

- merge : les données de configuration dans le paramètre <config> sont fusionnées avec la configuration au niveau correspondant dans le magasin de données cible. C'est le comportement par défaut.
- replace : les données de configuration dans le paramètre <config> remplacent complètement la configuration dans le magasin de données cible. C'est utile pour changer des données de configuration précédemment sauvegardées.
- aucune : le magasin de données cible n'est pas affecté par la configuration dans le paramètre <config>, sauf si et tant que les données de la configuration entrante utilisent l'attribut "operation" pour demander une opération différente. Si la configuration dans le paramètre <config> contient des données pour lesquelles il n'y a pas de niveau correspondant dans le magasin de données cible, une <rpc-error> est retournée avec une valeur <error-tag> de data-missing *(données manquantes)*. Utiliser "aucune" permet d'éviter que des opérations comme "delete" suppriment involontairement la hiérarchie parente de l'élément.

test-option : l'élément test-option ne peut être spécifié que si l'appareil annonce la capacité :validate (paragraphe 8.6). L'élément test-option a une des valeurs suivantes :

- test-then-set: effectue un essai de validation avant de tenter d'établir. Si des erreurs de validation surviennent, ne pas effectuer l'opération <edit-config>. C'est le comportement par défaut pour test-option.
- set : effectue l'établissement sans essai préalable de validation.

error-option : l'élément error-option a une des valeurs suivantes :

- stop-on-error : interrompt l'opération edit-config à la première erreur. C'est l'error-option par défaut.
- continue-on-error : continue de traiter les données de configuration sur erreur ; l'erreur est enregistrée, et une

réponse négative est générée si des erreurs se produisent.

rollback-on-error : si une condition d'erreur survient, telle qu'un élément de sévérité d'erreur <rpc-error> est généré, le serveur va arrêter le traitement de l'opération edit-config et restaurer la configuration spécifiée à son état complet au début de cette opération edit-config. Cette option exige que le serveur prenne en charge la capacité "rollback-on-error" décrite au paragraphe 8.5.

config : hiérarchie de données de configuration définie par un des modèles de données de l'appareil. Le contenu DOIT être placé dans un espace de noms approprié, pour permettre à l'appareil de détecter le modèle de données approprié, et le contenu DOIT respecter les contraintes de ce modèle de données, comme défini par sa définition de capacités. Les capacités sont discutées à la Section 8.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui contient un élément <ok>.

Réponse négative : une réponse <rpc-error> est envoyée si la demande ne peut pas être satisfaite pour une raison quelconque.

Exemple :
Les exemples <edit-config> de ce paragraphe utilisent un modèle de données simple, dans lequel plusieurs instances de l'élément 'interface' peuvent être présentes, et une instance est distinguée par l'élément 'nom' dans chaque élément 'interface'.

On règle la MTU à 1500 sur une interface appelée "Ethernet0/0" dans la configuration courante :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <cible><courant/></cible>
    <config>
      <top xmlns="http://exemple.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Ajoute une interface nommée "Ethernet0/0" à la configuration courante, remplaçant toute interface précédente par ce nom :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>

    <cible><courant/></cible>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <interface xc:operation="replace">
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
          <address>
            <name>192.0.2.4</name>
            <prefix-length>24</prefix-length>
          </address>
        </interface>
```

```
      </top>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Supprime la configuration pour une interface nommée "Ethernet0/0" de la configuration courante :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <cible><courant/></cible>
    <default-operation>aucune</default-operation>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <interface xc:operation="delete">
          <name>Ethernet0/0</name>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Supprime l'interface 192.0.2.4 d'une zone OSPF (les autres interfaces configurées dans la même zone n'étant pas affectées) :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <cible><courant/></cible>
    <default-operation>aucune</default-operation>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <top xmlns="http://exemple.com/schema/1.2/config">
        <protocols>
          <ospf>
            <area>
              <name>0.0.0.0</name>
              <interfaces>
                <interface xc:operation="delete">
                  <name>192.0.2.4</name>
                </interface>
              </interfaces>
            </area>
          </ospf>
        </protocols>
      </top>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
  <ok/>
</rpc-reply>
```

7.3 <copy-config>

Description : crée ou remplace un magasin de données de configuration entier par le contenu d'un autre magasin de données de configuration complet. Si le magasin de données cible existe, il est écrasé. Autrement, un nouveau est créé, si c'est permis.

Si un homologue NETCONF prend en charge la capacité :url (paragraphe 8.8), l'élément <url> peut apparaître comme paramètre <source> ou <cible>.

Même si il annonce la capacité :écrivable-courant, un appareil peut choisir de ne pas prendre en charge le magasin de données de configuration <courant/> comme paramètre <cible> d'une opération <copy-config>. Un appareil peut choisir de ne pas prendre en charge des opérations de copie de distant à distant, où les deux paramètres <source> et <cible> utilisent l'élément <url>.

Si les paramètres source et cible identifient les mêmes URL ou magasins de données de configuration, une erreur DOIT être retournée avec une étiquette d'erreur contenant "valeur invalide".

Paramètres :
cible : nom du magasin de données de configuration à utiliser comme destination de l'opération de copie.

source : nom du magasin de données de configuration à utiliser comme source de l'opération de copie ou élément <config> contenant la sous arborescence de configuration à copier.

Réponse positive : si l'appareil a été capable de satisfaire la demande, un <rpc-reply> est envoyé qui inclut un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans le <rpc-reply> si la demande ne peut pas être satisfaite pour toute raison.

Exemple :

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <cible><courant/></cible>
    <source>
      <url>https://user@exemple.com:passphrase/cfg/new.txt</url>
    </source>
  </copy-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

7.4 <delete-config>

Description : supprime un magasin de données de configuration. Le magasin de données de configuration <courant> ne peut pas être supprimé. Si un homologue NETCONF prend en charge la capacité :url (paragraphe 8.8), l'élément <url> peut apparaître comme le paramètre <cible>.

Paramètres :

cible : nom du magasin de données de configuration à supprimer.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui inclut un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans le <rpc-reply> si la demande ne peut pas être satisfaite pour toute raison.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <delete-config>
  <cible><startup/></cible>
 </delete-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ok/>
</rpc-reply>
```

7.5 <lock>

Description : l'opération lock permet au client de verrouiller le système de configuration d'un appareil. De tels verrouillages sont destinés à être de courte durée et permettent à un client de faire des changements sans crainte d'interaction avec d'autres clients NETCONF, des clients non NETCONF (par exemple, SNMP et des scripts d'interface de ligne de commande (CLI, *command line interface*)) et des utilisateurs humains. Une tentative de verrouillage de la configuration DOIT échouer si une session existante ou une autre entité détient un verrouillage sur toute portion de la cible du verrouillage.

Quand le verrouillage est acquis, le serveur DOIT empêcher tout changement à la ressource verrouillée autre que demandé par cette session. Les demandes SNMP et CLI de modification de la ressource DOIVENT échouer avec une erreur appropriée.

La durée du verrouillage est définie comme commençant quand le verrouillage est acquis et dure jusqu'à ce que le verrouillage soit libéré ou que la session NETCONF ferme. La clôture de session peut être explicitement effectuée par le client, ou implicitement effectuée par le serveur sur la base de critères comme la défaillance du transport sous-jacent, ou une simple fin de temporisation d'inactivité. Ces critères dépendent de la mise en œuvre et du transport sous-jacent.

L'opération lock prend un paramètre obligatoire, cible. Le paramètre cible désigne la configuration qui va être verrouillée. Quand un verrouillage est actif, il va être interdit d'utiliser l'opération <edit-config> sur la configuration verrouillée et d'utiliser la configuration verrouillée comme cible de l'opération <copy-config> par toute autre session NETCONF. De plus, le système va s'assurer que ces ressources de configuration verrouillées ne vont pas être modifiées par d'autres opérations de gestion non NETCONF comme SNMP et CLI. Le message <kill-session> (à la couche RPC) peut être utilisé pour forcer la libération d'un verrou possédé par une autre session NETCONF. Il sort du domaine d'application du présent document de définir comment rompre les verrous détenus par d'autres entités.

Un verrou NE DOIT PAS être accordé si l'une des conditions suivantes est vraie :
* un verrou est déjà détenu par une session NETCONF ou une autre entité,
* la configuration cible est <candidate>, elle a déjà été modifiée, et ces changements n'ont pas été engagés ou inversés.

Le serveur DOIT répondre par un élément <ok> ou une <rpc-error>.

Un verrou sera libéré par le système si la session qui détient le verrouillage est terminée pour toute raison.

Paramètres :

cible : nom du magasin de données de configuration à verrouiller.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui contient un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour toute raison.

Si le verrouillage est déjà détenu, l'élément <error-tag> va être 'verrouillage refuse' et l'élément <error-info> va inclure le <session-id> du propriétaire du verrouillage. Si le verrouillage est détenu par une entité non NETCONF, un <session-id> de 0 (zéro) est inclus. Noter que toute autre entité effectuant un verrouillage sur un élément même partiel d'une cible va empêcher un verrouillage NETCONF (qui est global) d'être obtenu sur cette cible.

Exemple :
L'exemple suivant montre une acquisition réussie de verrou.

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <lock>
     <cible><courant/></cible>
   </lock>
</rpc>

<rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <ok/> <!-- verrouillage réussi -->
</rpc-reply>
```

Exemple :
L'exemple suivant montre un échec de tentative d'acquérir un verrou quand le verrouillage est déjà utilisé.

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <lock>
     <cible>
       <courant/>
     </cible>
   </lock>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error> <!-- échec du verrouillage -->
   <error-type>protocol</error-type>
   <error-tag>lock-denied</error-tag>
   <error-severity>error</error-severity>
   <error-message>
     échec du verrouillage, le verrouillage est déjà détenu
   </error-message>
   <error-info>
     <session-id>454</session-id>
     <!-- le verrouillage est détenu par la session NETCONF 454 -->
   </error-info>
  </rpc-error>
</rpc-reply>
```

7.6 <unlock>

Description : l'opération unlock est utilisée pour libérer un verrouillage de configuration, précédemment obtenu avec l'opération <lock>.
Une opération unlock ne va pas réussir si une des conditions suivante est vraie :
* le verrou spécifié n'est pas actuellement actif,
* la session qui produit l'opération <unlock> n'est pas la même session qui a obtenu le verrouillage.

Le serveur DOIT répondre avec un élément <ok> ou une <erreur rpc>.

Paramètres :

cible : Nom du magasin de données de configuration à déverrouiller. Un client NETCONF n'a pas la permission de déverrouiller un magasin de données de configuration qu'il n'a pas verrouillé.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui contient un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour une raison quelconque.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <cible><courant/></cible>
  </unlock>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

7.7 <get>

Description : restitue la configuration courante et les informations d'état de l'appareil.

Paramètres :

filtre : ce paramètre spécifie la portion de la configuration du système et les données d'état à restituer. Si ce paramètre est vide, toutes les informations de configuration et d'état de l'appareil sont retournées.
L'élément filtre peut facultativement contenir un attribut 'type'. Cet attribut indique le type de syntaxe de filtrage utilisée dans l'élément filtre. Le mécanisme de filtrage par défaut dans NETCONF est appelé un filtrage de sous arborescence et est décrit à la Section 6. La valeur 'subtree' identifie explicitement ce type de filtrage.
Si l'homologue NETCONF prend en charge la capacité :xpath (paragraphe 8.9) la valeur "xpath" peut être utilisée pour indiquer que l'attribut choisi de l'élément filtre contient une expression XPath.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée. La section <data> contient le sous ensemble approprié.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour toute raison.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <top xmlns="http://exemple.com/schema/1.2/stats">
        <interfaces>
          <interface>
            <ifName>eth0</ifName>
          </interface>
        </interfaces>
      </top>
```

```
   </filter>
  </get>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
   <top xmlns="http://exemple.com/schema/1.2/stats">
    <interfaces>
     <interface>
      <ifName>eth0</ifName>
      <ifInOctets>45621</ifInOctets>
      <ifOutOctets>774344</ifOutOctets>
     </interface>
    </interfaces>
   </top>
  </data>
</rpc-reply>
```

7.8. <close-session>

Description : demande la terminaison en douceur d'une session NETCONF.
Quand un serveur NETCONF reçoit une demande <close-session>, il va clore la session en douceur. Le serveur va libérer tous les verrous et ressources associés à la session et clore en douceur toutes les connexions associées. Toutes les demandes NETCONF reçues après une demande <close-session> vont être ignorées.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui inclut un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour une raison quelconque.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

7.9 <kill-session>

Description : Force la terminaison d'une session NETCONF.
Quand une entité NETCONF reçoit une demande <kill-session> pour une session ouverte, elle va interrompre toutes les opérations actuellement en cours, libérer tous les verrous et ressources associés à la session, et clore toutes les connexions associées.
Si un serveur NETCONF reçoit une demande <kill-session> lorsque il traite un engagement confirmé (paragraphe 8.4) il doit restaurer la configuration à son état d'avant la production de l'engagement confirmé.
Autrement, l'opération <kill-session> ne revient pas à la configuration ou autres modifications d'état d'appareil faites par l'entité qui détient le verrouillage.

Paramètres :

session-id : identifiant de session de la session NETCONF à terminer. Si cette valeur est égale à l'identifiant de session actuel, une erreur 'valeur-invalide' est retournée.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui inclut un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour une raison quelconque.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>4</session-id>
  </kill-session>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

8. Capacités

Cette section définit un ensemble de capacités qu'un client ou un serveur PEUT mettre en œuvre. Chaque homologue annonce ses capacités en les envoyant durant un échange initial de capacités. Chaque homologue a besoin de comprendre seulement les capacités qu'il pourrait utiliser et DOIT ignorer toutes les capacités reçues de l'autre homologue dont il n'a pas besoin ou qu'il ne comprend pas.

Des capacités supplémentaires peuvent être définies en utilisant le schéma de l'Appendice C. De futures définitions de capacités pourront être publiées comme des normes par les organismes de normalisation ou publiées comme des extensions propriétaires.

Une capacité NETCONF est identifiée avec un URI. Les capacités de base sont définies en utilisant des URN suivant la méthode décrite dans la [RFC3553]. Les capacités définies dans le présent document ont le format suivant :

urn:ietf:params:netconf:capability:{nom}:1.0

où {nom} est le nom de la capacité. Les capacités sont souvent référencées dans les discussions et les messages en utilisant l'abrégé :{nom}. Par exemple, la capacité foo aurait le nom formel "urn:ietf:params:netconf:capability:foo:1.0" et être appelée ":foo". la forme abrégée NE DOIT PAS être utilisée dans le protocole.

8.1 Échange de capacités

Les capacités sont annoncées dans les messages envoyés par chaque homologue durant l'établissement de session. Quand la session NETCONF est ouverte, chaque homologue (client et serveur) DOIT envoyer un élément <hello> contenant une liste des capacités de cet homologue. Chaque homologue DOIT envoyer au moins la capacité NETCONF de base, "urn:ietf:params:netconf:base:1.0".

Un serveur qui envoie l'élément <hello> DOIT inclure un élément <session-id> contenant l'identifiant de session de cette session NETCONF. Un client qui envoie l'élément <hello> NE DOIT PAS inclure d'élément <session-id>.

Un serveur qui reçoit un élément <session-id> NE DOIT PAS continuer la session NETCONF. De même, un client qui ne reçoit pas un élément <session-id> dans le message <hello> du serveur NE DOIT PAS continuer la session NETCONF. Dans les deux cas, le transport sous-jacent devrait être fermé.

Dans l'exemple qui suit, un serveur annonce la capacité NETCONF de base, une capacité NETCONF définie dans le document de base NETCONF, et une capacité spécifique de la mise en œuvre.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:netconf:base:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:startup:1.0
    </capability>
    <capability>
      http://exemple.net/router/2.3/myfeature
    </capability>
  </capabilities>
  <session-id>4</session-id>
</hello>
```

Chaque homologue envoie un élément <hello> simultanément aussitôt que la connexion est ouverte. Un homologue NE DOIT PAS attendre de recevoir l'ensemble de capacités de l'autre côté pour envoyer le sien.

8.2 Capacité :writable-courant

8.2.1 Description

La capacité :writable-courant indique que l'appareil prend en charge l'écriture directe au magasin de données de configuration <courant>. En d'autres termes, l'appareil prend en charge les opérations edit-config et copy-config où la configuration <courante> est la cible.

8.2.2 Dépendances

Aucune.

8.2.3 Identifiant de capacités

La capacité :writable-courant est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:writable-courant:1.0

8.2.4 Nouvelles opérations

Aucune.

8.2.5 Modifications à des opérations existantes

8.2.5.1 <edit-config>

La capacité :writable-courant modifie l'opération <edit-config> pour accepter l'élément <courant> comme <cible>.

8.2.5.2 <copy-config>

La capacité :writable-courant modifie l'opération <copy-config> pour accepter l'élément <courant> comme <cible>.

8.3 Capacité de configuration candidate

8.3.1 Description

La capacité de configuration candidate, :candidate, indique que l'appareil prend en charge un magasin de données de configuration candidat, qui est utilisé pour détenir des données de configuration qui peuvent être manipulées sans impacter la configuration actuelle de l'appareil. La configuration candidate est un plein ensemble de données de configuration qui

sert de lieu de travail pour créer et manipuler les données de configuration. Des ajouts, suppressions, et changements peuvent être faits à ces données pour construire les données de configuration désirées. Une opération <commit> peut être effectuée à tout moment qui cause l'établissement de la configuration courante de l'appareil à la valeur de la configuration candidate.

L'opération <commit> règle effectivement la configuration courante au contenu courant de la configuration candidate. Bien qu'elle pourrait être modélisée comme une simple copie, elle est faite comme une opération distincte pour un certain nombre de raisons. En conservant des concepts de haut niveau comme opérations de première classe, on permet aux développeurs de voir plus clairement ce que demande le client et ce que le serveur doit effectuer. Cela rend les intentions plus évidentes, les cas particuliers moins complexes, et les interactions entre opérations plus directes. Par exemple, la capacité :confirmed-commit (paragraphe 8.4) n'aurait pas de sens comme opération "copy confirmed".

La configuration candidate peut être partagée entre plusieurs sessions. Sauf si un client a des informations spécifiques disant que la configuration candidate n'est pas partagée, il doit supposer que d'autres sessions peuvent être capables de modifier la configuration candidate au même moment. Il est donc prudent pour un client de verrouiller la configuration candidate avant de la modifier.

Le client peut éliminer tout changement non engagé à la configuration candidate en exécutant l'opération <discard-changes>. Cette opération fait revenir le contenu de la configuration candidate au contenu de la configuration courante.

8.3.2 Dépendances

Aucune.

8.3.3 Identifiant de capacité

La capacité :candidate est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:candidate:1.0

8.3.4 Nouvelles opérations

8.3.4.1 <commit>

Description : Quand le contenu d'une configuration candidate est complet, les données de configuration peuvent être engagées, publiant l'ensemble de données au reste de l'appareil et demandant à l'appareil de se conformer au comportement décrit dans la nouvelle configuration. Pour engager la configuration candidate comme nouvelle configuration courante de l'appareil, on utilise l'opération <commit>. L'opération <commit> donne pour instruction à l'appareil de mettre en œuvre les données de configuration contenues dans la configuration candidate. Si l'appareil est incapable d'engager tous les changements dans le magasin de données de configuration candidate, la configuration courante DOIT alors rester inchangée. Si l'appareil réussit l'engagement, la configuration courante DOIT être mise à jour avec le contenu de la configuration candidate. Si le système n'a pas la capacité :candidate, l'opération <commit> n'est pas disponible.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui contient un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour une raison quelconque.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
  <ok/>
</rpc-reply>
```

8.3.4.2 <discard-changes>

Si le client décide que la configuration candidate ne devrait pas être engagée, l'opération <discard-changes> peut être utilisée pour revenir à la configuration courante.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <discard-changes/>
</rpc>
```

Cette opération élimine tous les changements non engagés en rétablissant la configuration candidate avec le contenu de la configuration courante.

8.3.5 Modifications d'opérations existantes

8.3.5.1 <get-config>, <edit-config>, <copy-config>, et <validate>

La configuration candidate peut être utilisée comme source ou cible de toute opération <get-config>, <edit-config>, <copy-config>, ou <validate> comme paramètre <source> ou <cible>. L'élément <candidate> est utilisé pour indiquer la configuration candidate :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config> <!-- toute opération NETCONF -->
    <source>
      <candidate/>
    </source>
  </get-config>
</rpc>
```

8.3.5.2 <lock> et <unlock>

La configuration candidate peut être verrouillée en utilisant l'opération <lock> avec l'élément <candidate> comme paramètre <cible> :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <cible>
      <candidate/>
    </cible>
  </lock>
</rpc>
```

De même, la configuration candidate est déverrouillée en utilisant l'élément <candidate> comme paramètre <cible> :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <cible>
      <candidate/>
    </cible>
  </unlock>
</rpc>
```

Quand un client échoue avec des changements en cours à la configuration candidate, la récupération peut être difficile.

Pour faciliter la récupération, tous les changements en cours sont éliminés quand le verrouillage est libéré, soit explicitement avec l'opération <unlock>, soit implicitement à partir de l'échec de la session.

8.4 Capacité :confirmed-commit

8.4.1 Description

La capacité :confirmed-commit indique que le serveur va prendre en charge les paramètres <confirmed> et <confirm-timeout> pour l'opération de protocole <commit>. Voir au paragraphe 8.3 plus de détails sur l'opération <commit>.

Une opération d'engagement confirmé DOIT être annulée si un engagement à la suite (appelé un "engagement de confirmation") n'est pas produit dans les 600 secondes (10 minutes). La période de temporisation peut être ajustée avec l'élément <confirm-timeout>. L'engagement de confirmation peut lui-même inclure un paramètre <confirmed>.

Si la session qui produit l'engagement confirmé est terminée pour une raison quelconque avant l'expiration du temporisateur de confirmation, le serveur DOIT restaurer la configuration à son état antérieur à la production de l'engagement confirmé.

Si l'appareil réamorce pour une raison quelconque avant l'expiration du temporisateur de confirmation, le serveur DOIT restaurer la configuration à son état antérieur à la production de l'engagement confirmé.

Si un engagement de confirmation n'est pas produit, l'appareil va revenir à sa configuration de l'état antérieur à la production de l'engagement confirmé. Noter que toute opération d'engagement, incluant un engagement qui introduit des changements supplémentaires à la configuration, va servir de confirmation d'engagement. Donc pour annuler un engagement confirmé et revenir sur les changements sans attendre l'expiration du temporisateur de confirmation, le gestionnaire peut explicitement restaurer la configuration à son état d'avant la production de l'engagement confirmé.

Pour les configurations partagées, cette caractéristique peut causer que des changements d'autres configurations (par exemple, via d'autres sessions NETCONF) soit altérées ou supprimées par inadvertance, sauf si la caractéristique de verrouillage de configuration est utilisée (en d'autres termes, le verrouillage est obtenu avant que l'opération edit-config soit commencée). Donc, il est fortement suggéré que, afin d'utiliser cette caractéristique avec des bases de données de configuration partagées, le verrouillage de configuration soit aussi être utilisé.

8.4.2 Dépendances

La capacité :confirmed-commit n'est pertinente que si la capacité :candidate est aussi prise en charge.

8.4.3 Identifiant de capacité

La capacité :confirmed-commit est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:confirmed-commit:1.0

8.4.4 Nouvelles opérations

Aucune.

8.4.5 Modifications à des opérations existantes

8.4.5.1 <commit>

La capacité :confirmed-commit permet deux paramètres supplémentaires à l'opération <commit>.

Paramètres :

confirmed : effectue une opération d'engagement confirmé.

confirm-timeout : période de temporisation pour l'engagement confirmé, en secondes. Si il n'est pas spécifié, la

temporisation de confirmation est de 600 secondes par défaut.

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <confirmed/>
    <confirm-timeout>120</confirm-timeout>
  </commit>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

8.5 Capacité de retour en arrière sur erreur

8.5.1 Description

Cette capacité indique que le serveur va prendre en charge la valeur de 'retour en arrière sur erreur' dans le paramètre <error-option> à l'opération <edit-config>.

Pour les configurations partagées, cette caractéristique peut être cause que d'autres changements de configuration (par exemple, via d'autres sessions NETCONF) soient altérés ou supprimés par inadvertance, sauf si la caractéristique de verrouillage de configuration est utilisée (en d'autres termes, le verrouillage est obtenu avant que commence l'opération edit-config). Donc, il est fortement suggéré qu'afin d'utiliser cette caractéristique avec des bases de données de configuration partagées, le verrouillage de configuration soit aussi utilisé.

8.5.2 Dépendances

Aucune

8.5.3 Identifiant de capacité

La capacité :rollback-on-error est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:rollback-on-error:1.0

8.5.4 Nouvelles opérations

Aucune.

8.5.5 Modifications à des opérations existantes

8.5.5.1 <edit-config>

La capacité :rollback-on-error permet la valeur 'rollback-on-error' du paramètre <error-option> sur l'opération <edit-config>.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <cible><courant/></cible>
    <error-option>rollback-on-error</error-option>
    <config>
      <top xmlns="http://exemple.com/schema/1.2/config">
```

```
      <interface>
        <name>Ethernet0/0</name>
        <mtu>100000</mtu>
      </interface>
    </top>
  </config>
 </edit-config>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ok/>
</rpc-reply>
```

8.6 Validation de capacité

8.6.1 Description

La validation consiste en une vérification des erreurs de syntaxe et de sémantique d'une configuration candidate avant d'appliquer la configuration à l'appareil.

Si cette capacité est annoncée, l'appareil prend en charge l'opération de protocole <validate> et vérifie au moins les erreurs de syntaxe. De plus, cette capacité prend en charge le paramètre test-option sur l'opération <edit-config> et, quand elle est fournie, vérifie au moins les erreurs de syntaxe.

8.6.2 Dépendances

Aucune.

8.6.3 Identifiant de capacité

La capacité :validate est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:validate:1.0

8.6.4 Nouvelles opérations

8.6.4.1 <validate>

Description : cette opération de protocole valide le contenu de la configuration spécifiée.

Paramètres :

source : nom du magasin de données de configuration à valider, comme un élément <candidate> ou <config> contenant la sous arborescence de configuration à valider.

Réponse positive : si l'appareil a été capable de satisfaire la demande, une <rpc-reply> est envoyée qui contient un élément <ok>.

Réponse négative : un élément <rpc-error> est inclus dans la <rpc-reply> si la demande ne peut pas être satisfaite pour une raison quelconque.

Une opération validate peut échouer pour une des raisons suivantes :
- erreur de syntaxe
- paramètres manquants
- références à des données de configuration indéfinies

Exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
    <source><candidate/></source>
  </validate>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

8.7 Capacités de démarrage distinctes

8.7.1 Description

L'appareil prend en charge des magasins de données de configuration courante et de démarrage séparés. Les opérations qui affectent la configuration courante ne vont pas être automatiquement copiées dans la configuration de démarrage. Une opération <copy-config> explicite de <courant> à <startup> doit être invoquée pour mettre à jour la configuration de démarrage au contenu actuel de la configuration courante. Les opérations du protocole NETCONF se réfèrent au magasin de donnés de démarrage en utilisant l'élément <startup>.

8.7.2 Dépendances

Aucune.

8.7.3 Identifiant de capacité

La capacité :startup est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:startup:1.0

8.7.4 Nouvelles opérations

Aucune.

8.7.5 Modifications aux opérations existantes

8.7.5.1 Généralités

La capacité :startup ajoute le magasin de données de configuration <startup/> aux arguments de plusieurs opérations NETCONF. Le serveur DOIT prendre en charge les valeurs supplémentaires suivantes :

Opération	**Paramètres**	**Notes**
<get-config>	<source>	
<copy-config>	<source> <cible>	
<lock>	<cible>	
<unlock>	<cible>	
<validate>	<source>	Si :validate est annoncé

Pour sauvegarder la configuration de démarrage, on utilise l'opération copy-config pour copier le magasin de données de configuration <courant> au magasin de données de configuration <startup>.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <source><courant/></source>
```

```
    <cible><startup/></cible>
  </copy-config>
</rpc>
```

8.8. Capacité URL

8.8.1 Description

L'homologue NETCONF a la capacité d'accepter l'élément <url> dans les paramètres <source> et <cible>. La capacité est identifiée par les arguments URL qui indiquent les schémas d'URL pris en charge.

8.8.2 Dépendances

Aucune.

8.8.3 Identifiant de capacité

La capacité :url est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:url:1.0?scheme={name,...}

L'URI de capacité :url DOIT contenir une liste de noms de schémas d'arguments "scheme" séparés par des virgules qui indique quels schémas l'homologue NETCONF prend en charge. Par exemple :

urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file

8.8.4 Nouvelles opérations

Aucune.

8.8.5. Modifications aux opérations existantes

8.8.5.1 <edit-config>

La capacité :url capability modifie l'opération <edit-config> pour qu'elle accepte l'élément <url> comme solution de remplacement au paramètre <config>. Si l'élément <url> est spécifié, il devrait alors identifier un fichier de configuration local.

8.8.5.2 <copy-config>

La capacité :url modifie l'opération <copy-config> pour qu'elle accepte l'élément <url> comme valeur des paramètres <source> et <cible>.

8.8.5.3 <delete-config>

La capacité :url modifie l'opération <delete-config> pour qu'elle accepte l'élément <url> comme valeur du paramètre <cible>. Si ce paramètre contient un URL, il devrait alors identifier un fichier de configuration local.

8.8.5.4 <validate>

La capacité :url modifies l'opération <validate> pour qu'elle accepte l'élément <url> comme valeur du paramètre <source>.

8.9 Capacité XPath

8.9.1 Description

La capacité XPath indique que l'homologue NETCONF prend en charge l'utilisation des expressions XPath dans l'élément <filter>. XPath est décrit dans [XPath].

L'expression XPath doit retourner un ensemble de nœuds.

L'expression XPath est évaluée dans un contexte où le nœud contexte est le nœud racine, et l'ensemble des déclarations d'espace de noms est dans la portée de l'élément filtre, incluant l'espace de noms par défaut.

8.9.2 Dépendances

Aucune.

8.9.3 Identifiant de capacité

La capacité :xpath est identifiée par la chaîne de capacités suivante :

urn:ietf:params:netconf:capability:xpath:1.0

8.9.4 Nouvelles opérations

Aucune.

8.9.5 Modifications aux opérations existantes

8.9.5.1 <get-config> et <get>

La capacité :xpath modifie les opérations <get> et <get-config> pour qu'elles acceptent la valeur "xpath" dans l'attribut de type de l'élément filtre. Quand l'attribut de type est réglé à "xpath", un attribut select DOIT être présent sur l'élément filtre. L'attribut select va être traité comme une expression XPath et utilisé pour filtrer les données retournées. L'élément filtre lui-même DOIT être vide dans ce cas.

Par exemple :

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source><courant/></source>
    <!-- obtient l'utilisateur nommé fred -->
    <filter type="xpath" select="top/users/user[name='fred']"/>
  </get-config>
</rpc>
```

9. Considérations sur la sécurité

Le présent document ne spécifie pas un schéma d'autorisation, car un tel schéma devrait être lié à un modèle de méta-données ou un modèle de données. Les mises en œuvre DEVRAIENT fournir un schéma d'autorisation complet avec NETCONF.

L'autorisation des utilisateurs individuels via le serveur NETCONF peut ou non se transposer de façon bijective sur d'autres interfaces. D'abord, les modèles de données peuvent être incompatibles. Ensuite, il peut être souhaitable d'autoriser sur la base des mécanismes disponibles dans la couche de protocole de transport (TELNET, SSH, etc).

De plus, les opérations sur les configurations peuvent avoir des conséquences inattendues si ces opérations ne sont pas

aussi gardées par le verrou global sur les fichiers ou objets sur lesquels on opère. Par exemple, une liste d'accès partiellement complète pourrait être engagée à partir d'une configuration candidate inconnue du propriétaire du verrouillage de la configuration candidate, conduisant à un appareil non sûr ou inaccessible si le verrouillage sur la configuration candidate ne s'applique pas aussi à l'opération <copy-config> quand elle s'applique à lui.

Les informations de configuration sont par nature sensibles. Leur transmission en clair et sans vérification de l'intégrité laisse les appareils ouverts aux attaques d'espionnage classiques. Les informations de configuration contiennent souvent des mots de passe, des noms d'utilisateur, des descriptions de service, et des informations topologiques, qui sont toutes sensibles. À cause de cela, ce protocole devrait être mis en œuvre avec une attention adéquate à toutes les manières d'attaques qu'on peut s'attendre à subir avec les autres interfaces de gestion.

Le protocole doit donc prendre en charge au minimum les options pour la confidentialité et l'authentification. Il est prévu que le protocole sous-jacent (SSH, BEEP, etc) va assurer la confidentialité et l'authentification, comme exigé. On s'attend, de plus, à ce que l'identité de chaque extrémité d'une session NETCONF va être disponible à l'autre afin de déterminer l'autorisation pour toute demande. On pourrait aussi aisément envisager des informations supplémentaires, comme des méthodes de transport et de chiffrement, rendues disponibles pour les besoins de l'autorisation. NETCONF lui-même ne fournit pas de moyens de ré authentifier, encore moins d'authentifier. Toutes ces actions se font aux couches inférieures.

Des environnements différents peuvent bien permettre des droits différents avant et après l'authentification. Donc, un modèle d'autorisation n'est pas spécifié dans ce document. Quand une opération n'est pas autorisée de façon appropriée, un simple "accès refusé" est suffisant. Noter que les informations d'autorisation peuvent être échangées sous la forme d'informations de configuration, qui est une raison de plus d'assurer la sécurité de la connexion.

Ceci dit, il est important de reconnaître que certaines opérations sont par nature clairement plus sensibles que d'autres. Par exemple, <copy-config> aux configurations de démarrage ou courantes est clairement une opération de provisionnement anormale, tandis que <edit-config> est normal. De telles opérations globales DOIVENT interdire le changement des informations que des individus n'ont pas l'autorisation d'effectuer. Par exemple, si un utilisateur A n'a pas la permission de configurer une adresse IP sur une interface mais si l'utilisateur B a configuré une adresse IP sur une interface dans la configuration <candidate>, l'utilisateur A ne doit pas avoir la permission d'engager la configuration <candidate>.

De même, juste parce que quelqu'un dit "va écrire une configuration par la capacité URL à un endroit particulier", cela ne signifie pas qu'un élément devrait le faire sans l'autorisation appropriée.

L'opération <lock> va montrer que NETCONF est destiné à être utilisé par des systèmes qui ont au moins une certaine confiance dans l'administrateur. Comme spécifié dans ce document, il est possible de verrouiller des portions d'une configuration à laquelle un principal n'aurait autrement pas accès. Après tout, la configuration entière est verrouillée. Pour atténuer ce problème, il y a deux approches. Il est possible de tuer une autre session NETCONF de façon programmatique à partir de l'intérieur de NETCONF si on connaît l'identifiant de session de la session dérangeante. L'autre façon possible de casser un verrou est de fournir une fonction au sein de l'interface d'utilisateur native de l'appareil. Ces deux mécanismes souffrent d'une condition de concurrence qui peut être améliorée en supprimant l'utilisateur dérangeant d'un serveur AAA. Cependant, une telle solution n'est pas utile dans tous les scénarios de déploiement, comme ceux où des paires de clés SSH publique/privées sont utilisées.

10. Considérations relatives à l'IANA

10.1 Espace de noms XML NETCONFdeux

Le présent document enregistre un URI pour l'espace de noms NETCONF XML dans le registre XML de l'IETF [RFC3688].

Suivant le format de la RFC 3688, l'IANA a fait l'enregistrement suivant.

URI : urn:ietf:params:xml:ns:netconf:base:1.0
Contact d'enregistrement : IESG.
XML : N/A, l'URI demandé est un espace de noms XML.

10.2 Schéma XML NETCONF

Le présent document enregistre un URI pour le schéma XML NETCONF dans le registre XML de l'IETF [RFC3688].

Suivant le format de la RFC 3688, l'IANA a fait l'enregistrement suivant.

URI : urn:ietf:params:xml:schema:netconf
Contact d'enregistrement : IESG.
XML : voir l'Appendice B de ce document.

10.3 URN de capacité NETCONF

Le présent document crée un registre qui alloue les identifiants de capacités NETCONF. Les ajouts à ce registre exigent une action de normalisation de l'IETF.

Le contenu initial du registre contient la capacité URN définie à la Section 8.

Suivant les lignes directrices de la [RFC3553], l'IANA a alloué un sous espace de noms NETCONF comme suit :

Nom du registre : netconf
Spécification : Section 8 de ce document.
Répertoire : le tableau suivant.

Indice	**Identifiant de capacité**
:writable-courant	urn:ietf:params:netconf:capability:writable-courant:1.0
:candidate	urn:ietf:params:netconf:capability:candidate :1.0
:confirmed-commit	urn:ietf:params:netconf:capability:confirmed -commit:1.0
:rollback-on-error	urn:ietf:params:netconf:capability:rollback-on-error:1.0
:validate	urn:ietf:params:netconf:capability:validate: 1.0
:startup	urn:ietf:params:netconf:capability:startup:1.0
:url	urn:ietf:params:netconf:capability:url:1.0
:xpath	urn:ietf:params:netconf:capability:xpath:1.0

Valeur d'indice : le nom de la capacité.

11. Auteurs et remerciements

Le présent document a été écrit par :
Andy Bierman
Ken Crozier, Cisco Systems
Rob Enns, Juniper Networks
Ted Goddard, IceSoft
Eliot Lear, Cisco Systems
Phil Shafer, Juniper Networks
Steve Waldbusser
Margaret Wasserman, ThingMagic

Les auteurs tiennent à remercier les membres du groupe de travail NETCONF. En particulier, merci à Wes Hardaker pour sa persévérance et sa patience dans son aide sur les considérations de sécurité. Merci aussi à Randy Presuhn, Sharon Chisholm, Juergen Schoenwalder, Glenn Waters, David Perkins, Weijing Chen, Simon Leinen, Keith Allen, et Dave Harrington de tous leurs précieux conseils.

12. Références

12.1 Références normatives

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC3553] M. Mealling et autres, "Sous-espace de noms d'URN de l'IETF pour les paramètres de protocole enregistrés", juin 2003. (BCP0073)

[RFC3688] M. Mealling, "Registre XML de l'IETF", BCP 81, janvier 2004.

[RFC3986] T. Berners-Lee, R. Fielding et L. Masinter, "Identifiant de ressource uniforme (URI) : Syntaxe générique", STD 66, janvier 2005. *(P.S. ; MàJ par* RFC8820)

[RFC4742] M. Wasserman, T. Goddard, "Utilisation du protocole de configuration NETCONF sur Secure SHell (SSH)", décembre 2006. *(P.S.)*

[XML] Sperberg-McQueen, C., Paoli, J., Maler, E., et T. Bray, "Extensible Markup Language (XML) 1.0 (Second Edition)", World Wide Web Consortium, http://www.w3.org/TR/2000/REC-xml-20001006 , octobre 2000.

[XPath] Clark, J. et S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation, http://www.w3.org/TR/1999/REC-xpath-19991116 , novembre 1999.

12.2 Références pour information

[RFC2865] C. Rigney et autres, "Service d'authentification à distance de l'utilisateur appelant (RADIUS)", juin 2000. (*MàJ par* RFC2868, RFC3575, RFC5080, RFC8044) *(D.S.)*

[RFC3470] S. Hollenbeck, M. Rose, L. Masinter, "Lignes directrices pour l'utilisation du langage de balisage extensible (XML) dans les protocoles de l'IETF", janvier 2003. (BCP0070)

[RFC4251] T. Ylonen et C. Lonvick, "Architecture du protocole Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8308)

[RFC4346] T. Dierks et E. Rescorla, "Protocole de sécurité de la couche Transport (TLS) version 1.1", avril 2006. *(Remplace* RFC2246 *; Remplacée par* RFC5246 *; MàJ par* RFC4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919)

[XSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116 , novembre 1999.

Appendice A. Liste d'erreurs NETCONF

Étiquette : in-use *(déjà utilisée)*
Type d'erreur : protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande exige une ressource qui est déjà utilisée.

Étiquette : invalid-value *(valeur invalide)*
Type d'erreur : protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande spécifie une valeur inacceptable pour un ou plusieurs paramètres.

Étiquette : too-big *(trop gros)*
Type d'erreur : transport, rpc, protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande ou réponse (qui aurait été générée) est trop grosse pour que la mise en œuvre la traite.

Étiquette : missing-attribute *(attribut manquant)*
Type d'erreur : rpc, protocole, application

Sévérité : erreur
Informations d'erreur : <bad-attribute> : nom de l'attribut manquant
<bad-element> : nom de l'élément qui devrait contenir l'attribut manquant
Description : un attribut attendu est manquant.

Étiquette : bad-attribute *(mauvais attribut)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-attribute> : nom de l'attribut qui a une mauvaise valeur
<bad-element> : nom de l'élément qui contient l'attribut avec la mauvaise valeur
Description : une valeur d'attribut n'est pas correcte ; par exemple, mauvais type, hors gamme, discordance de schéma.

Étiquette : unknown-attribute *(attribut inconnu)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-attribute> : nom de l'attribut inattendu
<bad-element> : nom de l'élément qui contient l'attribut inattendu
Description : un attribut inattendu est présent.

Étiquette : missing-element *(élément manquant)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-element> : nom de l'élément manquant
Description : un élément attendu est manquant.

Étiquette :bad-element *(mauvais élément)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-element> : nom de l'élément qui a une mauvaise valeur
Description : une valeur d'élément n'est pas correcte ; par exemple, mauvais type, hors gamme, discordance de schéma.

Étiquette : unknown-element *(élément inconnu)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-element> : nom de l'élément inattendu
Description : un élément inattendu est présent.

Étiquette : unknown-namespace *(espace de noms inconnu)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : <bad-element> : nom de l'élément qui contient l'espace de noms inattendu
<bad-namespace> : nom de l'espace de noms inattendu
Description : un espace de noms inattendu est présent.

Étiquette : access-denied *(accès refusé)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : l'accès au RPC, opération de protocole, ou modèle de données demandé est refusé parce que l'autorisation a échoué.

Étiquette : lock-denied *(verrou refusé)*
Type d'erreur : protocole
Sévérité : erreur
Informations d'erreur : <session-id> : identifiant de la session ID qui détient le verrou demandé, ou zéro pour indiquer qu'une entité non NETCONF détient le verrouillage
Description : l'accès au verrou demandé est refusé parce que le verrouillage est actuellement détenu par une autre entité.

Étiquette : resource-denied *(ressource refusée)*
Type d'erreur : transport, rpc, protocole, application
Sévérité : erreur

Informations d'erreur : aucune
Description : la demande ne peut être satisfaite parce que les ressources sont insuffisantes.

Étiquette : rollback-failed *(échec de roulement)*
Type d'erreur : protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande de revenir sur un changement de configuration (via l'opération rollback-on-error ou discard-changes) n'a pas été réalisé pour une raison quelconque.

Étiquette : data-exists *(les données existent déjà)*
Type d'erreur : application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande ne peut pas être satisfaite parce que le contenu de modèle de données pertinent existe déjà. Par exemple, une opération 'create' a été tentée sur des données qui existent déjà.

Étiquette : data-missing *(données manquantes)*
Type d'erreur : application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande ne peut pas être satisfaite parce que le contenu de modèle de données pertinent n'existe pas. Par exemple, une opération 'replace' ou 'delete' a été tentée sur des données qui n'existent pas.

Étiquette : operation-not-supported *(opération non prise en charge)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande ne peut pas être satisfaite parce que l'opération demandée n'est pas prise en charge par cette mise en œuvre.

Étiquette : operation-failed *(échec de l'opération)*
Type d'erreur : rpc, protocole, application
Sévérité : erreur
Informations d'erreur : aucune
Description : la demande ne peut pas être satisfaite parce que l'opération demandée a échoué pour une raison non couverte par une autre condition d'erreur.

Étiquette : partial-operation *(opération partielle)*
Type d'erreur : application
Sévérité : erreur
Informations d'erreur :
<ok-element> : identifie un élément dans le modèle de données pour lequel l'opération demandée a été réalisée pour ce nœud et tous ses nœuds fils. Cet élément peut apparaître zéro, une ou plusieurs fois dans le conteneur <error-info>.
<err-element> : identifie un élément dans le modèle de données pour lequel l'opération demandée a échoué pour ce nœud et tous ses nœuds fils. Cet élément peut apparaître zéro, une ou plusieurs fois dans le conteneur <error-info>.
<noop-element> : identifie un élément dans le modèle de données pour lequel l'opération demandée n'a pas été tenté pour ce nœud et tous ses nœuds fils. Cet élément peut apparaître zéro, une ou plusieurs fois dans le conteneur <error-info>.
Description : une partie de l'opération demandée a échoué ou n'a pas été tentée pour une raison quelconque. Un nettoyage complet n'a pas été effectué (par exemple, roulement non pris en charge) par le serveur. Le conteneur error-info est utilisé pour identifier quelles portions du contenu du modèle de données d'application pour lequel l'opération demandée ont réussi (<ok-element>), échoué (<bad-element>), ou n'ont pas été tentées (<noop-element>).

Appendice B. Schéma XML pour NETCONF RPC et les opérations de protocole

DÉBUT

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
        targetNamespace="urn:ietf:params:xml:ns:netconf:base:1.0"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        xml:lang="fr">
<!--  importation des définitions XML standard -->
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
          schemaLocation="http://www.w3.org/2001/xml.xsd">
    <xs:annotation>
     <xs:documentation>
      Cette importation accède aux groupes d'attributs xml: pour le xml:lang comme déclaré dans l'élément error-message.
     </xs:documentation>
    </xs:annotation>
   </xs:import>
<!-- attribut message-id -->
    <xs:simpleType name="messageIdType">
     <xs:restriction base="xs:string">
      <xs:maxLength value="4095"/>
     </xs:restriction>
    </xs:simpleType>
<!-- Types utilisé pour session-id -->
    <xs:simpleType name="SessionId">
     <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="SessionIdOrZero">
     <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
<!-- élément <rpc> -->
    <xs:complexType name="rpcType">
     <xs:sequence>
      <xs:element ref="rpcOperation"/>
     </xs:sequence>
     <xs:attribute name="message-id" type="messageIdType"
      use="required"/>
<!-- Des attributs arbitraires peuvent être fournis avec l'élément <rpc>. -->
     <xs:anyAttribute processContents="lax"/>
    </xs:complexType>
    <xs:element name="rpc" type="rpcType"/>
<!-- Types et éléments de données utilisés pour construire rpc-errors -->
    <xs:simpleType name="ErrorType">
     <xs:restriction base="xs:string">
      <xs:enumeration value="transport"/>
      <xs:enumeration value="rpc"/>
      <xs:enumeration value="protocol"/>
      <xs:enumeration value="application"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ErrorTag">
     <xs:restriction base="xs:string">
      <xs:enumeration value="in-use"/>
      <xs:enumeration value="invalid-value"/>
      <xs:enumeration value="too-big"/>
```

```
        <xs:enumeration value="missing-attribute"/>
        <xs:enumeration value="bad-attribute"/>
        <xs:enumeration value="unknown-attribute"/>
        <xs:enumeration value="missing-element"/>
        <xs:enumeration value="bad-element"/>
        <xs:enumeration value="unknown-element"/>
        <xs:enumeration value="unknown-namespace"/>
        <xs:enumeration value="access-denied"/>
        <xs:enumeration value="lock-denied"/>
        <xs:enumeration value="resource-denied"/>
        <xs:enumeration value="rollback-failed"/>
        <xs:enumeration value="data-exists"/>
        <xs:enumeration value="data-missing"/>
        <xs:enumeration value="operation-not-supported"/>
        <xs:enumeration value="operation-failed"/>
        <xs:enumeration value="partial-operation"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ErrorSeverity">
      <xs:restriction base="xs:string">
        <xs:enumeration value="error"/>
        <xs:enumeration value="warning"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="errorInfoType">
      <xs:sequence>
        <xs:choice>
          <xs:element name="session-id" type="SessionIdOrZero"/>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:sequence>
              <xs:element name="bad-attribute" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
              <xs:element name="bad-element" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
              <xs:element name="ok-element" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
              <xs:element name="err-element" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
              <xs:element name="noop-element" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
              <xs:element name="bad-namespace" type="xs:QName"
                minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
          </xs:sequence>
        </xs:choice>
<!-- Des éléments provenant de tout autre espace de noms sont aussi permis derrière des éléments NETCONF -->
        <xs:any namespace="##autre"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rpcErrorType">
      <xs:sequence>
        <xs:element name="error-type" type="ErrorType"/>
        <xs:element name="error-tag" type="ErrorTag"/>
        <xs:element name="error-severity" type="ErrorSeverity"/>
        <xs:element name="error-app-tag" type="xs:string" minOccurs="0"/>
        <xs:element name="error-path" type="xs:string" minOccurs="0"/>
        <xs:element name="error-message" minOccurs="0">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
```

```
            <xs:attribute ref="xml:lang" use="optional"/>
           </xs:extension>
          </xs:simpleContent>
         </xs:complexType>
        </xs:element>
        <xs:element name="error-info" type="errorInfoType"
         minOccurs="0"/>
       </xs:sequence>
      </xs:complexType>
<!-- Élément <rpc-reply> -->
      <xs:complexType name="rpcReplyType">
       <xs:choice>
        <xs:element name="ok"/>
        <xs:group ref="rpcResponse"/>
       </xs:choice>
       <xs:attribute name="message-id" type="messageIdType"
        use="optional"/>
<!-- Tous les attributs fournis avec l'élément <rpc> doivent être retournés sur <rpc-reply>. -->
       <xs:anyAttribute processContents="lax"/>
      </xs:complexType>
      <xs:group name="rpcResponse">
       <xs:sequence>
        <xs:element name="rpc-error" type="rpcErrorType"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="data" type="dataInlineType" minOccurs="0"/>
       </xs:sequence>
      </xs:group>
      <xs:element name="rpc-reply" type="rpcReplyType"/>
<!-- Type pour le paramètre <test-option> à <edit-config> -->
      <xs:simpleType name="testOptionType">
       <xs:restriction base="xs:string">
        <xs:enumeration value="test-then-set"/>
        <xs:enumeration value="set"/>
       </xs:restriction>
      </xs:simpleType>
<!-- Type pour le paramètre <error-option> à <edit-config> -->
      <xs:simpleType name="errorOptionType">
       <xs:restriction base="xs:string">
        <xs:annotation>
         <xs:documentation>
          L'utilisation de la valeur rollback-on-error exige la capacitée :rollback-on-error.
         </xs:documentation>
        </xs:annotation>
        <xs:enumeration value="stop-on-error"/>
        <xs:enumeration value="continue-on-error"/>
        <xs:enumeration value="rollback-on-error"/>
       </xs:restriction>
      </xs:simpleType>
<!-- rpcOperationType : utilisé comme type de base pour toutes les opérations NETCONF -->
      <xs:complexType name="rpcOperationType"/>
      <xs:element name="rpcOperation" type="rpcOperationType" abstract="true"/>
<!-- Type pour l'élément <config> -->
      <xs:complexType name="configInlineType">
       <xs:complexContent>
        <xs:extension base="xs:anyType"/>
       </xs:complexContent>
      </xs:complexType>
<!-- Type pour l'élément <data> -->
      <xs:complexType name="dataInlineType">
       <xs:complexContent>
        <xs:extension base="xs:anyType"/>
```

```
      </xs:complexContent>
    </xs:complexType>
<!-- Type pour l'élément <filter> -->
    <xs:simpleType name="FilterType">
      <xs:restriction base="xs:string">
        <xs:annotation>
          <xs:documentation>
            L'utilisation de la valeur xpath exige la capacité :xpath.
          </xs:documentation>
        </xs:annotation>
        <xs:enumeration value="subtree"/>
        <xs:enumeration value="xpath"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="filterInlineType">
      <xs:complexContent>
        <xs:extension base="xs:anyType">
          <xs:attribute name="type"
                        type="FilterType" default="subtree"/>
<!-- si type="xpath", l'expression xpath apparaît dans l'élément select -->
          <xs:attribute name="select"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
<!-- noms des magasins de données de configuration -->
    <xs:annotation>
      <xs:documentation>
        Le magasin de données de démarrage ne peut être utilisé que si la capacité :startup est annoncée. Le magasin de
        données candidat ne peut être utilisé que si le magasin de données :candidate est annoncé.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType name="configNameType"/>
    <xs:element name="config-name"
                type="configNameType" abstract="true"/>
    <xs:element name="startup" type="configNameType"
                substitutionGroup="config-name"/>
    <xs:element name="candidate" type="configNameType"
                substitutionGroup="config-name"/>
    <xs:element name="courant" type="configNameType"
                substitutionGroup="config-name"/>
<!-- operation attribute utilisée dans <edit-config> -->
    <xs:simpleType name="editOperationType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="merge"/>
        <xs:enumeration value="replace"/>
        <xs:enumeration value="create"/>
        <xs:enumeration value="delete"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:attribute name="operation"
                  type="editOperationType" default="merge"/>
<!-- élément <default-operation> -->
    <xs:simpleType name="defaultOperationType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="merge"/>
        <xs:enumeration value="replace"/>
        <xs:enumeration value="aucune"/>
      </xs:restriction>
    </xs:simpleType>
<!-- élément <url> -->
    <xs:complexType name="configURIType">
```

```
      <xs:annotation>
       <xs:documentation>
        L'utilisation de l'élément url exige la capacité :url.
       </xs:documentation>
      </xs:annotation>
      <xs:simpleContent>
       <xs:extension base="xs:anyURI"/>
      </xs:simpleContent>
     </xs:complexType>
<!-- Type pour l'élément <source> (sauf <get-config>) -->
     <xs:complexType name="rpcOperationSourceType">
      <xs:choice>
       <xs:element name="config" type="configInlineType"/>
       <xs:element ref="config-name"/>
       <xs:element name="url" type="configURIType"/>
      </xs:choice>
     </xs:complexType>
<!-- Type pour l'élément <source> dans <get-config>-->
     <xs:complexType name="getConfigSourceType">
      <xs:choice>
       <xs:element ref="config-name"/>
       <xs:element name="url" type="configURIType"/>
      </xs:choice>
     </xs:complexType>
<!-- Type pour l'élément <cible> -->
     <xs:complexType name="rpcOperationTargetType">
      <xs:choice>
       <xs:element ref="config-name"/>
       <xs:element name="url" type="configURIType"/>
      </xs:choice>
     </xs:complexType>
<!-- Opération <get-config> -->
     <xs:complexType name="getConfigType">
      <xs:complexContent>
       <xs:extension base="rpcOperationType">
        <xs:sequence>
         <xs:element name="source"
                  type="getConfigSourceType"/>
         <xs:element name="filter"
                  type="filterInlineType" minOccurs="0"/>
        </xs:sequence>
       </xs:extension>
      </xs:complexContent>
     </xs:complexType>
     <xs:element name="get-config" type="getConfigType"
           substitutionGroup="rpcOperation"/>
<!-- Opération <edit-config> -->
     <xs:complexType name="editConfigType">
      <xs:complexContent>
       <xs:extension base="rpcOperationType">
        <xs:sequence>
         <xs:annotation>
          <xs:documentation>
           L'utilisation de l'élément test-option exige la capacité :validate. L'utilisation de l'élément url exige la capacité :url.
          </xs:documentation>
         </xs:annotation>
         <xs:element name="cible"
                  type="rpcOperationTargetType"/>
         <xs:element name="default-operation"
                  type="defaultOperationType"
                  minOccurs="0"/>
```

```
            <xs:element name="test-option"
                    type="testOptionType"
                    minOccurs="0"/>
            <xs:element name="error-option"
                    type="errorOptionType"
                    minOccurs="0"/>
            <xs:choice>
              <xs:element name="config"
                      type="configInlineType"/>
              <xs:element name="url"
                      type="configURIType"/>
            </xs:choice>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="edit-config" type="editConfigType" substitutionGroup="rpcOperation"/>
<!-- Opération <copy-config> -->
    <xs:complexType name="copyConfigType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType">
          <xs:sequence>
            <xs:element name="cible" type="rpcOperationTargetType"/>
            <xs:element name="source" type="rpcOperationSourceType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="copy-config" type="copyConfigType" substitutionGroup="rpcOperation"/>
<!-- Opération <delete-config> -->
    <xs:complexType name="deleteConfigType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType">
          <xs:sequence>
            <xs:element name="cible" type="rpcOperationTargetType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="delete-config" type="deleteConfigType" substitutionGroup="rpcOperation"/>
<!-- Opération <get> -->
    <xs:complexType name="getType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType">
          <xs:sequence>
            <xs:element name="filter" type="filterInlineType" minOccurs="0"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="get" type="getType" substitutionGroup="rpcOperation"/>
<!-- Opération <lock> -->
    <xs:complexType name="lockType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType">
          <xs:sequence>
            <xs:element name="cible" type="rpcOperationTargetType"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

```
    <xs:element name="lock" type="lockType" substitutionGroup="rpcOperation"/>
<!-- Opération <unlock> -->
    <xs:complexType name="unlockType">
     <xs:complexContent>
      <xs:extension base="rpcOperationType">
       <xs:sequence>
        <xs:element name="cible" type="rpcOperationTargetType"/>
       </xs:sequence>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
    <xs:element name="unlock" type="unlockType" substitutionGroup="rpcOperation"/>
<!--Opération <validate> -->
    <xs:complexType name="validateType">
     <xs:annotation>
      <xs:documentation>
       L'opération validate exige la capacité :validate.
      </xs:documentation>
     </xs:annotation>
     <xs:complexContent>
      <xs:extension base="rpcOperationType">
       <xs:sequence>
        <xs:element name="source" type="rpcOperationSourceType"/>
       </xs:sequence>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
    <xs:element name="validate" type="validateType" substitutionGroup="rpcOperation"/>
<!-- Opération <commit> -->
    <xs:simpleType name="confirmTimeoutType">
     <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="commitType">
     <xs:annotation>
      <xs:documentation>
       L'opération commit exige la capacité :candidate.
      </xs:documentation>
     </xs:annotation>
     <xs:complexContent>
      <xs:extension base="rpcOperationType">
       <xs:sequence>
        <xs:annotation>
         <xs:documentation>
          L'utilisation des éléments confirmed et confirm-timeout exige la capacité :confirmed-commit.
         </xs:documentation>
        </xs:annotation>
        <xs:element name="confirmed" minOccurs="0"/>
        <xs:element name="confirm-timeout" type="confirmTimeoutType" minOccurs="0"/>
       </xs:sequence>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
    <xs:element name="commit" type="commitType" substitutionGroup="rpcOperation"/>
<!-- Opération <discard-changes> -->
    <xs:complexType name="discardChangesType">
     <xs:annotation>
      <xs:documentation>
       L'opération discard-changes exige la capacité :candidate.
      </xs:documentation>
```

```
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="rpcOperationType"/>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="discard-changes" type="discardChangesType" substitutionGroup="rpcOperation"/>
<!-- Opération <close-session> -->
    <xs:complexType name="closeSessionType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType"/>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="close-session" type="closeSessionType" substitutionGroup="rpcOperation"/>
<!-- Opération <kill-session> -->
    <xs:complexType name="killSessionType">
      <xs:complexContent>
        <xs:extension base="rpcOperationType">
          <xs:sequence>
            <xs:element name="session-id" type="SessionId" minOccurs="1"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="kill-session" type="killSessionType" substitutionGroup="rpcOperation"/>
<!-- Élément <hello> -->
    <xs:element name="hello">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="capabilities">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="capability" type="xs:anyURI"
                  maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="session-id" type="SessionId" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

FIN

Appendice C. Gabarit de capacité

C.1 Nom de capacité (gabarit)
C.1.1 Vue d'ensemble
C.1.2 Dépendances
C.1.3 Identifiant de capacité
Le {nom} de capacité est identifié par la chaîne de capacité suivante :
{uri de capacité}
C.1.4 Nouvelles opérations
C.1.4.1 <nom d'opération>
C.1.5 Modifications aux opérations existantes
C.1.5.1 <nom d'opération>
Si les opérations existantes ne sont pas modifiées par cette capacité, ce paragraphe peut être omis.
C.1.6 Interactions avec les autres capacités
Si cette capacité n'interagit pas avec les autres capacités, ce paragraphe peut être omis.

Appendice D. Configuration de plusieurs appareils avec NETCONF

D.1 Opérations sur les appareils individuels

On considère le travail impliqué pour effectuer une mise à jour de configuration sur un seul appareil individuel. Pour faire un changement de configuration, l'application a besoin de construire la confiance que ce changement a été fait correctement et qu'il n'a pas impacté le fonctionnement de l'appareil. L'application (et l'utilisateur de l'application) devrait avoir confiance que le changement n'a pas endommagé le réseau.

Protéger chaque appareil individuel consiste en un certain nombre d'étapes :
- o Acquérir le verrou de configuration.
- o Charger la mise à jour.
- o Valider la configuration entrante.
- o Verifier la configuration courante.
- o Changer la configuration courante.
- o Essayer la nouvelle configuration.
- o Rendre le changement permanent (si c'est désiré).
- o Libérer le verrou de configuration.

Examinons les détails de chaque étape.

D.1.1 Acquisition du verrou de configuration

Un verrou devrait être acquis pour empêcher des mises à jour simultanées provenant de multiples sources. Si plusieurs sources affectent l'appareil, l'application est gênée à la fois pour vérifier son changement de la configuration et pour récupérer si la mise à jour devait échouer. Acquérir un verrou de courte durée est une défense simple pour empêcher d'autres parties d'introduire des changements sans rapport avec celui désiré.
Le verrou peut être acquis en utilisant l'opération <lock>.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <cible>
      <courant/>
    </cible>
  </lock>
</rpc>
```

D.1.2 Chargement de la mise à jour

La configuration peut être chargée sur l'appareil sans impacter le système courant. Si la capacité :url est prise en charge et donne "file" comme schéma pris en charge, les changements entrants peuvent être placés dans un fichier local.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <cible>
      <url>file://incoming.conf</url>
    </cible>
    <source>
      <config>
        <!-- placer ici la configuration entrante -->
      </config>
    </source>
  </copy-config>
</rpc>
```

Si la capacité :candidate est prise en charge, la configuration candidate peut être utilisée.

```
<rpc message-id="101"
```

```
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
   <cible>
    <candidate/>
   </cible>
   <config>
    <!-- placer ici la configuration entrante -->
   </config>
  </edit-config>
</rpc>
```

Si la mise à jour échoue, le fichier d'utilisateur peut être supprimé en utilisant l'opération <delete-config>, ou on peut revenir à la configuration candidate en utilisant l'opération <discard-changes>.

D.1.3 Validation de la configuration entrante

Avant d'appliquer la configuration entrante, il est souvent utile de la valider. La validation permet à l'application d'avoir confiance que le changement va réussir et simplifie la récupération si elle ne réussit pas.

Si l'appareil prend en charge la capacité :url et mentionne "file" comme schéma pris en charge, on utilise l'opération <validate> avec le paramètre <source> réglé au fichier d'utilisateur approprié:

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
   <source>
    <url>file://incoming.conf</url>
   </source>
  </validate>
</rpc>
```

Si l'appareil prend en charge la capacité :candidate, une certaine validation va être effectuée au titre du chargement de la configuration entrante dans la candidate. Pour une pleine validation, on passe le paramètre <validate> durant l'étape <edit-config> donnée ci-dessus, ou on utilise l'opération <validate> avec le paramètre <source> réglé à <candidate>.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
   <source>
    <candidate/>
   </source>
  </validate>
</rpc>
```

D.1.4 Vérification de la configuration courante

La configuration courante peut être sauvegardée dans un fichier local comme point de vérification avant de charger la nouvelle configuration. Si la mise à jour échoue, la configuration peut être restaurée en rechargeant le fichier du point de vérification.

Le fichier du point de vérification peut être créé en utilisant l'opération <copy-config>.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
   <cible>
    <url>file://checkpoint.conf</url>
   </cible>
   <source>
```

```
      <courant/>
    </source>
  </copy-config>
</rpc>
```

Pour restaurer le fichier de point de vérification, on inverse les paramètres source et cible.

D.1.5 Changement de la configuration courante

Quand la configuration entrante a été chargée en toute sécurité sur l'appareil et validée, elle est prête à impacter le système courant.

Si l'appareil prend en charge la capacité :url et mentionne "file" comme schéma pris en charge, on utilise l'opération <edit-config> pour fusionner la configuration entrante dans la configuration courante.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <cible>
      <courant/>
    </cible>
    <config>
      <url>file://incoming.conf</url>
    </config>
  </edit-config>
</rpc>
```

Si l'appareil prend en charge la capacité :candidate, on utilise l'opération <commit> pour régler la configuration courante à la configuration candidate. On utilise le paramètre <confirmed> pour permettre une inversion automatique de la configuration d'origine si la connexité à l'appareil a une défaillance.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <confirmed/>
    <confirm-timeout>120</confirm-timeout>
  </commit>
</rpc>
```

D.1.6 Essai de la nouvelle configuration

Maintenant que la configuration entrante a été intégrée dans la configuration courante, l'application a besoin de s'assurer que le changement affecte l'appareil de la façon prévue et sans effets secondaires négatifs.

Pour obtenir cette assurance, l'application peut conduire des essais de l'état de fonctionnement de l'appareil. La nature de l'essai dépend de la nature du changement et sort du domaine d'application de ce document. De tels essais peuvent inclure l'accessibilité de l'application à partir du système courant (avec un ping) des changements de l'accessibilité au reste du réseau (en comparant le tableau d'acheminement de l'appareil) ou l'inspection du changement particulier (en cherchant un preuve opérationnelle de l'homologue BGP qui vient d'être ajouté).

D.1.7 Rendre le changement permanent

Quand le changement de configuration est en place et que l'application a une confiance suffisante dans le bon fonctionnement de ce changement, l'application devrait rendre le changement permanent.

Si l'appareil prend en charge la capacité :startup, la configuration courante peut être sauvegardée à la configuration de démarrage en utilisant la configuration de démarrage comme cible de l'opération <copy-config>.

```
<rpc message-id="101"
```

```
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
   <cible>
    <startup/>
   </cible>
   <source>
    <courant/>
   </source>
  </copy-config>
</rpc>
```

Si l'appareil prend en charge la capacité :candidate et si un engagement confirmé a été demandé, l'engagement de confirmation doit être envoyé avant l'expiration du temporisateur.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>
```

D.1.8 Libération du verrou de configuration

Quand la mise à jour de configuration est achevée, le verrouillage doit être libéré, permettant à d'autres applications d'accéder à la configuration.

On utilise l'opération <unlock> pour libérer le verrou de la configuration.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
   <cible>
    <courant/>
   </cible>
  </unlock>
</rpc>
```

D.2 Opérations sur plusieurs appareils

Quand un changement de configuration exige des mises à jour sur un certain nombre d'appareils, on devrait veiller à fournir la sémantique de transaction requise. Le protocole NETCONF contient des primitives suffisantes pour construire les opérations orientées sur la transaction. Fournir une sémantique de transaction complète sur plusieurs appareils est d'un coût prohibitif, mais la taille et le nombre de fenêtres pour des scénarios d'échec peut être réduit.

Il y a deux classes d'opérations sur plusieurs appareils. La première classe permet à l'opération d'échouer sur des appareils individuels sans exiger que tous les appareils reviennent à leur état d'origine. L'opération peut être ré-essayée ultérieurement, ou son échec simplement rapporté à l'utilisateur. Un exemple de cette classe pourrait être d'ajouter un serveur NTP. Pour cette classe d'opérations, l'évitement de défaillance et la récupération sont centrés sur l'appareil individuel. Cela signifie la récupération de l'appareil, le rapport de la défaillance, et peut-être de programmer une autre tentative.

La second classe est plus intéressante, exigeant que l'opération soit réalisée sur tous les appareils ou être entièrement inversée. Le réseau devrait soit être transformé en un nouvel état, soit être remis dans son état d'origine. Par exemple, un changement à un VPN peut exiger des mises à jour à un certain nombre d'appareils. Un autre exemple de ceci serait d'ajouter une définition de classe de services. Laisser le réseau dans un état où seulement une portion des appareils ont été mis à jour avec la nouvelle définition va conduire à de futures défaillances quand la définition est référencée.

Pour donner une sémantique de transaction, les mêmes étapes utilisées dans les opérations d'un seul appareil mentionnées ci-dessus sont utilisées, mais sont effectuées en parallèle sur tous les appareils. Les verrous de configuration devraient être acquis sur tous les appareils cibles et conservés jusqu'à ce que tous les appareils soient mis à jour et que les changements soient rendus permanents. Les changements de configuration devraient être chargés et leur validation effectuée sur tous les

appareils. Les points de vérification devraient être faits sur chaque appareil. Ensuite, la configuration courante peut être changée, vérifiée, et rendue permanente. Si une de ces étapes échoue, les configurations précédentes peuvent être restaurées sur tous les appareils sur lesquels elles ont été changées. Après que les changements ont été complètement mis en œuvre ou complètement éliminés, les verrous de chaque appareil peuvent être libérés.

Appendice E. Caractéristiques différées

Les caractéristiques suivantes ont été différées à une future révision de ce document.

- o Verrouillage granulaire des objets de configuration.
- o Fichiers/magasins de données de configuration nommés.
- o Prise en charge de plusieurs canaux NETCONF.
- o Notifications asynchrones.
- o Prise en charge explicite par le protocole du repositionnement des changements de configuration sur les versions antérieures.

Adresse de l'éditeur

Rob Enns
Juniper Networks
1194 North Mathilda Ave
Sunnyvale, CA 94089
US

mél : rpe@juniper.net

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est fourni par l'activité de soutien administratif (IASA) de l'IETF.