Workgroup: OPSAWG
Internet-Draft:
draft-quilbeuf-opsawg-configuration-tracing-00
Published: 20 October 2022
Intended Status: Standards Track
Expires: 23 April 2023
Authors: J. Quilbeuf, Huawei; B. Claise, Huawei; T. Graf, Swisscom; D. Lopez, Telefonica I+D; Q. Sun, China Telecom

External Transaction ID for Configuration Tracing

Abstract

Network equipments are often configured by a variety of network management systems (NMS), protocols, and people. If a network issue arises because of a wrong configuration modification, it's important to quickly identify the specific service request and obtain the reason for pushing that modification. Another potential network issue can stem from concurrent NMS's with overlapping intent, each having their own tasks to perform: in such a case, it's important to map the respective modifications to its originating NMS. This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request, in the context of NETCONF. Such a mechanism is required for autonomous networks, to trace the reason of a particular configuration change that lead to an anomaly detection or a broken SLA. This mechanism facilitates the troubleshooting, the post mortem analysis, and in the end the closed loop automation required for self-healing networks. The specifications contain a new YANG module mapping a local configuration change to the corresponding northbound transaction, up to the controller or even the orchestrator.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/JeanQuilbeufHuawei/draft-quilbeuf-opsawg-configuration-tracing.

Status of This Memo

This Internet-Draft will expire on 23 April 2023.

Copyright Notice

Table of Contents

1. Introduction
2. Terminology
3. Use cases
   3.1. Configuration Mistakes
   3.2. Concurrent NMS Configuration
   3.3. Conflicting Intents
4. Relying on Transaction-id to Trace Configuration Modifications
   4.1. Instantiating the YANG module
   4.2. Using the YANG module
5. YANG module
   5.1. Overview
   5.2. YANG module ietf-external-transaction-id
6. Security Considerations
7. IANA Considerations
8. Contributors
9. Open Issues / TODO
   9.1. Possibility of setting the transaction Id from the client
10. Normative References
11. Informative References
Appendix A. Changes between revisions
Appendix B. Tracing configuration changes
Acknowledgements

Authors' Addresses

1. Introduction

Issues arising in the network, for instance violation of some SLAs, might be due to some configuration modification. In the context of automated networks, the assurance system needs not only to identify and revert the problematic configuration modification, but also to make sure that it won't happen again and that the fix will not disrupt other services. To cover the last two points, it is imperative to understand the cause of the problematic configuration change. Indeed, the first point, making sure that the configuration modification will not be repeated, cannot be ensured if the cause for pushing the modification in the first place is not known. Ensuring the second point, not disrupting other services, requires as well knowing if the configuration modification was pushed in order to support new services. Therefore, we need to be able to trace a configuration modification on a device back to the reason that triggered that modification, for instance in a NMS, whether the controller or the orchestrator.

This specification focuses only on configuration pushed via NETCONF [RFC6241]. The rationale for this choice is that NETCONF is better suited for normalization than other protocols (SNMP, CLI). Another reason is that the notion of transaction ID, useful to track configuration modification, is already defined in [I-D.lindblad-netconf-transaction-id] and comes from RESTCONF [RFC8040].

The same network element, or NETCONF [RFC6241] server, can be configured by different NMSs or NETCONF clients. If an issue arises, one of the starting points for investigation is the configuration modification on the devices supporting the impacted service. In the best case, there is a dedicated user for each client and the timestamp of the modification allows tracing the problematic modification to its cause. In the worst case, everything is done by the same user and some more tricks must be done to trace the problematic modification to its source.

This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request. Practically, this mechanism annotates configuration changes on the configured element with sufficient information to unambiguously identify the corresponding transaction, if any, on the element that requested the configuration modification. It reuses the concept of a NETCONF transaction ID from [I-D.lindblad-netconf-transaction-id] and augment it with an ID for the client. The information needed to do the actual configuration tracing is stored in a new YANG module that maps a local configuration change to the corresponding northbound transaction, up to the controller or even the orchestrator. In case of a controller, the local configuration modification ID to both corresponding north- and southbound transaction ID. Additionally, for northbound transactions, we store the ID of the client.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the terms client and server from [RFC6241].

This document uses the terms transaction and transaction id from [I-D.lindblad-netconf-transaction-id].

3. Use cases

This document was written with autonomous networks in mind. We assume that an existing monitoring or assurance system, such as described in [I-D.ietf-opsawg-service-assurance-architecture], is able to detect and report network anomalies , e.g. SLA violations, intent violations, network failure, or simply a customer issue. Here are the use cases for the proposed YANG module.

3.1. Configuration Mistakes

Taking into account that many network anomalies are due to configuration mistakes, this mechanism allows to find out whether the offending configuration modification was triggered by a tracing-enabled client/NMS. In such as case, we can map the offending configuration modification id on a server/NE to a local configuration modification id on the client/NMS. Assuming that this mechanism (the YANG module) is implemented on the controller, we can recursively find, in the orchestrator, the latest (set of of) service request(s) that triggered the configuration modification. Whether this/those service request(s) are actually the root cause needs to be investigated. However, they are a good starting point for troubleshooting, post mortem analysis, and in the end the closed loop automation, which is absolutely required for for self-healing networks.

3.2. Concurrent NMS Configuration

Building on the previous use case is the situation where two NMS', unaware of the each other, configuring routers, each believing that they are the only NMS for specific device. So one configuration executed by the NMS1 is overwritten by the NMS2, which in turn is overwritten by NMS1, etc.

3.3. Conflicting Intents

Autonomous networks will be solved first by assuring intent per specific domain; for example data center, core, cloud, etc. This last use case is a more specific "Concurrent NMS configuration" use case where assuring domain intent breaks the entire end to end service, even if the domain-specific controllers are aware of each other.

4. Relying on Transaction-id to Trace Configuration Modifications

4.1. Instantiating the YANG module

In [I-D.lindblad-netconf-transaction-id], the concept of a NETCONF transaction ID is proposed, to match the same mechanism from RESTCONF [RFC8040]. The goal of this document is to speed up the re-synchronization process between a client and a server, by using a common transaction ID. If the current transaction ID on the server is the same as the transaction ID known by the client, then both are synchronized. Otherwise, the client has to fetch again the configuration. The transaction ID can be applied to the whole configuration or to so-called versioned nodes. In the latter case, only versioned nodes for which the transaction ID differs need to be updated.

```
 +---------------+
 | Orchestrator  |
 +---------------+
        | tx-1
        v
 +---------------+
 |   Controller  |
 +---------------+
   | tx-2      | tx-3
   v           v
+-----+     +-----+
| NE1 |     | NE2 |
+-----+     +-----+
```

Figure 1: Example of Hierarchical Configuration. tx: transaction

A server considers as a northbound transaction a transaction that modifies its configuration. On Figure 1, tx-1 is a northbound transaction for the Controller.

A client considers as a southbound transaction the modification of a server configuration. On Figure 1, tx-2 and tx-3 are southbound transactions for the Controller.

If the set-tx-id feature is enabled (see open issue in Section 9.1), the client can specify its own transaction ID when sending the configuration ID for the server. In that case, the Controller in Figure 1 could use the same transaction-id for both tx-2 and tx-3 and save a single southbound transaction ID for that commit. Otherwise, the server is the one generating the ID for the transaction between the client and the server. If the client has to configure several servers, for instance to enable a network service, then each of the configured servers might return a different ID. Therefore, for a configuration modification on the client might be implemented via several southbound transactions and thus might have several southbound transaction ID.

Our proposed solution is to store, on the server, a mapping between the existing local commit id and the northbound and southbound transactions related to that local configuration change. The mapping is read only and populated by the server at configuration time as follows:

*Northbound transaction: If the set-tx-id feature is available (see Section 9.1), the server MUST accept a transaction-ID and a client ID from client supporting configuration tracing. The server MUST store both entries as respectively northbound transaction ID and northbound client ID, associated to the local configuration ID. If the set-tx-id feature is not available, the server MUST accept the client ID, generate a transaction ID, save both the transaction ID as northbound transaction id and the client ID as northbound client ID, and send back the transaction ID to the client. If the client does not support configuration tracing, none of these entries are populated. In Figure 1, for the Controller, the northbound transaction ID is the ID of tx-1.

*Southbound transaction: If the set-tx-id feature is available (see Section 9.1), when a client has to configure servers in response to a local configuration change, then it MUST generate a transaction ID, send it along with its ID to the configured servers, and save it as a southbound transaction ID. If the set-tx-id feature is not available, it MUST sent its own ID with the configuration, receive back the transaction ID from each server, and save all of them as southbound transaction ID. In Figure 1, for the Controller, the southbound transaction IDs are the IDs of tx-2 and tx-3.

The two cases above are not mutually exclusive. A Controller can be configured by an Orchestrator and configure network equipment in turn, as shown in Figure 1. In that case, both the northbound transaction ID, shared with the Orchestrator and the southbound transaction IDs, shared with the network equipments, are stored in the Controller. They are both associated to the corresponding configuration commit in the Controller.

It is technically possible that several clients push configuration to the candidate configuration datastore and only one of them commits the changes to the running configuration datastore. From the running configuration datastore perspective, which is the effective one, there is a single modification, but caused by several clients, which means that this modification should have several northbound transaction id. Although, this case is technically possible, it is a bad practice. We won't cover it in this document. In other terms, we assume that a given configuration modification on a server is caused by a single northbound transaction, and thus has a single corresponding northbound transaction ID.

4.2. Using the YANG module

The YANG module defined below enables tracing a configuration change in a Network Equipment back to its origin, for instance a service request in an orchestrator. To do so, the Anomaly Detection System (ADS) should have for each NMS ID (as stored in northbound-client-id), access to some credentials enabling read access to the model. It should as well have access to the network equipment in which an issue is detected.

```
                                               +---------------+
  .----------------[5]match SB tx-1----------->|               |
  |                                            | Orchestrator  |
  | ----------------[6]commit-id---------------|               |
  | |                                          +---------------+
  | |                                                | tx-1
  | |                                                v
  | |                                          +---------------+
  | |   .-----------[3] match SB tx-2--------->|               |
  | |   |                                      |   Controller  |
  | |   | .-----------[4] NB-tx-id tx-1--------|               |
  | |   | |                                    +---------------+
  | |   | |                                          | tx-2
  | v   | v                                          v
+-----------+                                      +----+
| Anomaly   |--[1] match commit-id before time t-->|    |
| Detection |                                      | NE |
| System    |<--------- [2] NB-tx-id tx-2 ----- ---|    |
+----------+                                       +----+
```

Figure 2: Example of Configuration Tracing. tx: transaction, NB: northbound, SB: southbound. The number between square brackets refer to steps in the listing below.

The steps for a software to trace a configuration modification in a Network Equipment back to a service request are illustrated in Figure 2. They are detailed below.

1. The Anomaly Detection System identifies the commit id that created an issue, for instance by looking for the last commit-id occuring before the issue was detected. The ADS queries the NE for the northbound transaction-id and northbound client id associated to the commit-id.

2. The ADS receives the northbound transaction Id. In Figure 2, that step would recieve the id of tx-2 and the id of the Controller as a result. If they are no results, or no associated northbound-transaction-id, the change was not done by a client compatible with the present draft, and the investigation stops here.

3. The ADS queries the client identified by the northbound-client-id found at the previous step, looking for a match of the northbound-transaction-id from the previous step with a southbound-transaction-id in the client version of the YANG model. In Figure 2, for that step, the software would look for the id of tx-2 in the southbound transaction IDs stored in the Controller.

4. From that query, the ADS knows the local-commit-id on the client (Controller in our case). Since the local-commit-id is associated to a northbound-transaction-id, namely the id of tx-1, the ADS continues the investigation. The client to query is identified by the northbound-client-id, in our case the Orchestrator.

5. The ADS queries the Orchestrator, trying to find a match for the Id of tx-1 as a southbound transaction ID.

6. Finally, the ADS receives the commit-id from the Orchestrator that ultimately caused the issue in the NE. Since there is no associated northbound transaction id, the investigation stops here. The modification associated to the commit-id, for instance a service request, is now available for further manual or automated analysis, such as analyzing the root cause of the issue.

Note that step 5 and 6 are actually a repetition of step 3 and 4. The general algorithm is to continue looking for a client until no more client (no more northbound-transaction-id) can be found in the current element.

5. YANG module

We present in this section the YANG module for modelling the information about the configuration modifications.

5.1. Overview

The tree representation [RFC8340] of our YANG module is depicted in Figure 3

```
module: ietf-external-transaction-id
  +--ro external-transactions-id
     +--ro configuration-change* [local-commit-id]
        +--ro local-commit-id              string
        +--ro northbound-transaction-id?   ietf-netconf-txid:etag-t
        +--ro northbound-client-id         string
        +--ro southbound-transaction-id*   ietf-netconf-txid:etag-t
```

Figure 3: Tree representation of ietf-external-transaction-id YANG module

The local-commit-id represents the local id of the configuration changes. It can be used to retrieve the local configuration changes that happened during that transaction.

The northbound-transaction-id should be present when the server is configured by a client supporting the external transaction ID. In that case, the northbound-client-id is mandatory. The value of both fields are sent by the client whenever it sends the configuration that trigger the changes associated to the local-commit-id.

The southbound-transaction-id should be present when the current configuration change leads to the configuration of other devices. In that case, the southbound-transaction-id should be generated by the server (and unique among other southbound-transaction-id fields generated on this server), sent to the configured devices and saved in that field. If the configured server do not support having a forced transaction id, then the transaction IDs resulting of the configuration of the servers must be stored in that list.

Even if this document focuses only on NETCONF, the use cases defined in Section 3 are not specific to NETCONF and the mechanism described in this document could be adapted to other configuration mechanisms. For instance, a configuration modification pushed via CLI can be identified via a label. As such cases are difficult to standardize, we won't cover them in this document. However, our model could be extended to support such mechanism for instance by using a configuration label instead of the northbound transaction ID.

5.2. YANG module ietf-external-transaction-id

```
<CODE BEGINS> file "ietf-external-transaction-id@2021-11-03.yang"

module ietf-external-transaction-id {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-external-transaction-id";
  prefix ext-txid;

  import ietf-netconf-txid {
    prefix ietf-netconf-txid;
  }

  organization
    "IETF OPSAWG Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Author:   Benoit Claise  <mailto:benoit.claise@huawei.com>
     Author:   Jean Quilbeuf  <mailto:jean.quilbeuf@huawei.com>";
  description
    "This module enable tracing of configuration changes in an
     automated network. It stores the ID of the northbound
     transaction when the local device is configured by an enabled
     NMS, and the southbound transaction ID when the local device
     configures other devices.

     The main usage of this module is to map a local configuration
     change to a northbound transaction ID that can be retrieved as
     southbound transaction ID on the configuring NMS, or to map a
     southbound transaction ID to a northbound transaction ID on
     devices that are both configured and configuring other devices.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
     'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
     are to be interpreted as described in BCP 14 (RFC 2119)
     (RFC 8174) when, and only when, they appear in all
     capitals, as shown here.

     Copyright (c) 2021 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
     This version of this YANG module is part of RFC XXXX; see the
```

```
     RFC itself for full legal notices.  ";

  revision 2021-11-03 {
    description
      "Initial revision";
    reference
      "RFC xxxx: Title to be completed";
  }

  container external-transactions-id {
    config false;
    description
      "Contains the IDs of configuration transactions that are
       external to the current device.";
    list configuration-change {
      key "local-commit-id";
      description
        "List of configuration changes, identified by their
         local-commit-id";
      leaf local-commit-id {
        type string;
        description
          "Id as saved by the server. Can be used to retrieve
           the corresponding changes using the server mechanism
           if available.";
      }
      leaf northbound-transaction-id {
        type ietf-netconf-txid:etag-t;
        description
          "External transaction ID, sent by the client, corresponding
           to a change initiated by a northbound NMS. There should be
           a corresponding entry on the NMS as a
           southbound-transaction-id that maps to the actual
           configuration commit that triggered the configuration of
           this server.

           This field is present only when the configuration was
           pushed by a compatible system.";
      }
      leaf northbound-client-id {
        when '../northbound-transaction-id';
        type string;
        mandatory true;
        description
          "ID of the client doing the modification, to further query
           information about the corresponding change.";
      }
      leaf-list southbound-transaction-id {
        type ietf-netconf-txid:etag-t;
```

```
        description
          "Transaction ID transmitted to southbound devices
           configured following the configuration change
           corresponding to local-commit-id. ";
      }
    }
  }
}

<CODE ENDS>
```

6. Security Considerations

7. IANA Considerations

This document includes no request to IANA.

8. Contributors

9. Open Issues / TODO

*Evaluate risk of collision between transaction ids in the southbound-transaction id. Example scenario: 1) client configures server 1 and server 2 for commit-id (client) 1 the southbound transaction IDs are A (server 1) B (server 2) 2) client configures server 1 and server 2 for commit-id (client) 2 the southbound transaction IDs are B (server 1) C (server 2) 3) the last configuration of server 1 causes an issue, when looking for southbound transaction id B, it's not clear whether the issue comes from commit 1 or commit 2 in the client

9.1. Possibility of setting the transaction Id from the client

In the -00 version of [I-D.lindblad-netconf-transaction-id], there is the possibility for the client to set the transaction id when sending the configuration to the server. This feature has been removed in subsequent versions. In this draft, we call this feature set-tx-id. Such a feature would simplify the present draft, therefore we try to present two versions, one with the feature set-tx-id available and one without.

10. Normative References

[I-D.lindblad-netconf-transaction-id]
Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-lindblad-netconf-transaction-id-02, 8 June 2022, <https://www.ietf.org/archive/id/draft-lindblad-netconf-transaction-id-02.txt>.

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

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

11. Informative References

[I-D.ietf-opsawg-service-assurance-architecture]
Claise, B., Quilbeuf, J., Lopez, D., Voyer, D., and T. Arumugam, "Service Assurance for Intent-based Networking Architecture", Work in Progress, Internet-Draft, draft-ietf-opsawg-service-assurance-architecture-11, 18 October 2022, <https://www.ietf.org/archive/id/draft-ietf-opsawg-service-assurance-architecture-11.txt>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Appendix A. Changes between revisions

Initial version

Appendix B. Tracing configuration changes

Acknowledgements

Authors' Addresses

Jean Quilbeuf
Huawei

Email: jean.quilbeuf@huawei.com

Benoit Claise
Huawei

Email: benoit.claise@huawei.com

Thomas Graf
Swisscom
Binzring 17
CH-8045 Zurich
Switzerland

Email: thomas.graf@swisscom.com

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain

Email: diego.r.lopez@telefonica.com

Qiong Sun
China Telecom

Email: sunqiong@chinatelecom.cn