Workgroup: OPSAWG
Internet-Draft:
draft-quilbeuf-opsawg-configuration-tracing-01
Published: 13 March 2023
Intended Status: Standards Track
Expires: 14 September 2023
Authors: J. Quilbeuf B. Claise T. Graf D. Lopez
Huawei Huawei Swisscom Telefonica I+D
Q. Sun
China Telecom

External Transaction ID for Configuration Tracing

Abstract

Network equipment are often configured by a variety of network management systems (NMS), protocols, and teams. If a network issue arises (e.g., because of a wrong configuration change), it is important to quickly identify the root cause and obtain the reason for pushing that modification. Another potential network issue can stem from concurrent NMSes with overlapping intents, each having their own tasks to perform. In such a case, it is important to map the respective modifications to its originating NMS.

This document specifies a NETCONF mechanism to automatically map the configuration modifications to their source, up to a specific NMS change request. Such a mechanism is required, in particular, for autonomous networks to trace the source of a particular configuration change that led to an anomaly detection. This mechanism facilitates the troubleshooting, the post mortem analysis, and in the end the closed loop automation required for self-healing networks. The specification also includes a YANG module that is meant to map a local configuration change to the corresponding change 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

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on 14 September 2023.

Table of Contents

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 correlations 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 additionally requires an ID for the client. The information needed to trace the configuration is stored in a new YANG module that maps a local configuration change to some additional metadata. In the server, this metadata includes the ID of the client triggering the configuration change as well as the transaction ID corresponding to that change in the client. In the client, the metadata includes the transaction ID for each server configured during the local configuration change. In case of a controller, which plays both the role of server (for an orchestrator) and client (for a network equipment), the metadata associated to both the server-side and the client-side of the translation are stored.

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].

Local Commit ID Identifier of a local configuration change on a Network Equipment, Controller, Orchestrator or any other device or software handling configuration. Such an identifier is usually present in devices that can show an history of the configuration changes, to identify one such configuration change.

Parent Transaction For a given NETCONF server, a transaction that changed the local configuration.

Child Transaction For a given NETCONF client, a transaction that required configuration changes to at least one server.

Client ID ID of a NETCONF client, must be unique among all NETCONF clients that configure the network.

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 a 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's, unaware of the each other, are configuring a common router, each believing that they are the only NMS for the common router. 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.

3.4. Not a use case: Onboarding

During onboarding, a newly added device is likely to receive a multiple configuration message, as it needs to be fully configured. Our use cases focus more on what happens after the initial configuration is done, i.e. when the "stable" configuration is modified.

4. Relying on Transaction ID to Trace Configuration Modifications

4.1. Existing configuration metadata on device

This document assumes that NETCONF clients or servers (orchestrators, controllers, devices, ...) have some kind of mechanism to record the modifications done to the configuration. For instance, routers typically have an history of configuration change

and this configuration associates a locally unique identifier to some metadata, such as the timestamp of the modification, the user doing the modification or the protocol used for the modification. Such a locally unique identifier is a Local Commit ID, we assume that it exists on the platform. This Local Commit ID is the link between the module presented in this draft and the device-specific way of storing configuration changes.

4.2. Client ID

This document assumes that each NETCONF client for which configuration must be traced (for instance orchestrator and controllers) has a unique client ID among the other NETCONF clients in the network. Such an ID could be an IP address or a host name. The mechanism for providing and defining this client ID is out of scope of the current document.

4.3. 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.

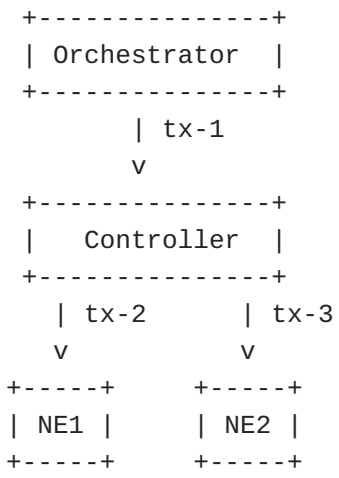

Figure 1: Example of Hierarchical Configuration. tx: transaction

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

A client considers as a Child Transaction the modification of a server configuration. On Figure 1, tx-2 and tx-3 are Child 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 Child 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 Child Transactions and thus might have several Child Transaction IDs.

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

* Parent 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 Parent Transaction ID and 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 Parent Transaction ID and the Client ID as 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 Parent Transaction ID is the ID of tx-1.

* Child 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 Child 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 Child Transaction ID. In Figure 1, for the Controller, the Child 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 Parent Transaction ID, shared with the Orchestrator and the Child 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 Parent 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 Parent Transaction, and thus has a single corresponding Parent Transaction ID.

4.4. 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 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.

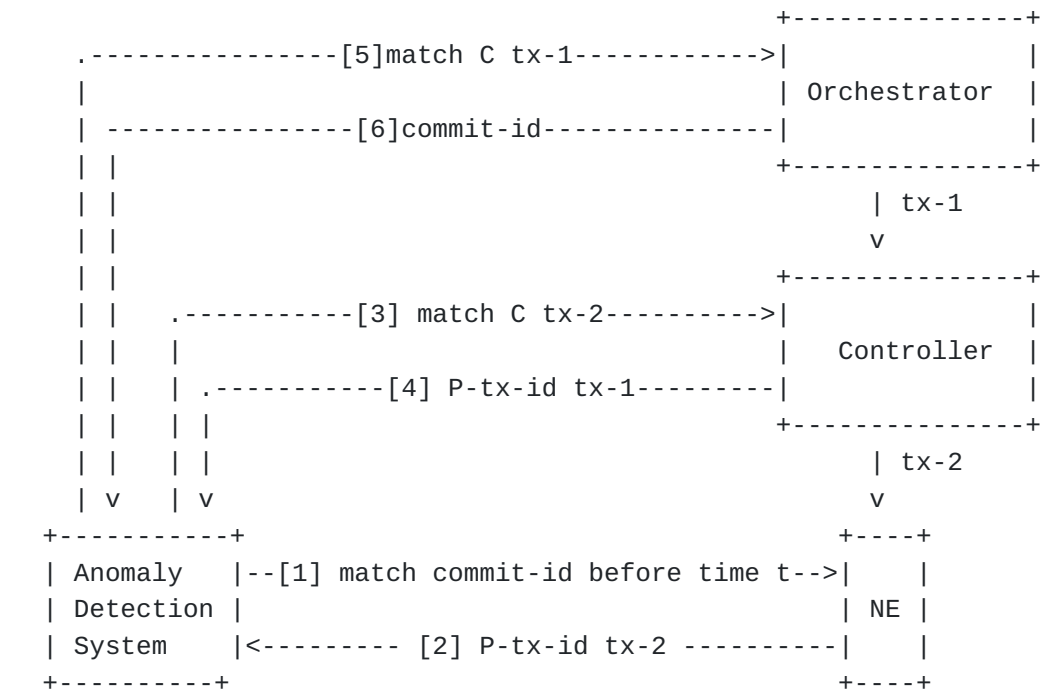

Figure 2: Example of Configuration Tracing. tx: transaction, P: parent, C: child. The number between square brackets refer to steps in the listing below.

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 (ADS) 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 Parent Transaction ID and client id associated to the commit-id.

2. The ADS receives the Parent 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 Parent 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 client-id found at the previous step, looking for a match of the Parent Transaction ID from the previous step with a Child 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 Child 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 Parent Transaction ID, namely the id of tx-1, the ADS continues the investigation. The client to query is identified by the 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 Child 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 Parent 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 Parent 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 timestamp?               yang:date-and-time
        +--ro parent-transaction-id?   ietf-netconf-txid:etag-t
        +--ro client-id                string
        +--ro child-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 Parent Transaction ID should be present when the server is configured by a client supporting the external transaction ID. In that case, the 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 Child Transaction ID should be present when the current configuration change leads to the configuration of other devices. In that case, the Child Transaction ID should be generated by the server (and unique among other Child 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 Parent 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-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }
  import ietf-netconf-txid {
    prefix ietf-netconf-txid;
    reference
      "draft-lindblad-netconf-transaction-id:
                     Transaction ID Mechanism for NETCONF";
  }

  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 enables tracing of configuration changes in a
     network for the sake of automated correlation between
     configuration changes and the external request that triggered
     that change.

     The module stores the identifier of the parent transaction
     that triggered the change in a device, and the child
     transaction ID when the local device originates in its turn a
     transaction.

     Copyright (c) 2022 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 Revised 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 2022-10-20 {
  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 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
        "Stores the identifier as saved by the server. Can be used
         to retrieve the corresponding changes using the server
         mechanism if available.";
    }
    leaf timestamp {
      type yang:date-and-time;
      description
        "A timestamp that can be used to further filter change
         events.";
    }
    leaf parent-transaction-id {
      type ietf-netconf-txid:etag-t;
      description
        "External transaction ID, sent by the client, corresponding
         to a change initiated by an external entity (e.g.,
         controller, orchestrator). There should be a corresponding
         entry on that external entity as a child-transaction-id
         that maps to the actual configuration commit that
         triggered the configuration of this server.

         This data node is present only when the configuration was
         pushed by a compatible system.";
    }
    leaf client-id {
      when '../parent-transaction-id';
      type string;
      mandatory true;
      description
```

```
          "ID of the client that originated the modification, to
           further query information about the corresponding
           change.";
      }
      leaf-list child-transaction-id {
        type ietf-netconf-txid:etag-t;
        description
          "Transaction ID transmitted to other 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 Child Transaction ID. Example scenario: 1) client configures server 1 and server 2 for commit-id (client) 1 the Child Transaction IDs are A (server 1) B (server 2) 2) client configures server 1 and server 2 for commit-id (client) 2 the Child Transaction IDs are B (server 1) C (server 2) 3) the last configuration of server 1 causes an issue, when looking for Child Transaction id B, it's not clear whether the issue comes from commit 1 or commit 2 in the client

* Indicate what to do with O-RAN apps, since each of them might be seen as a different client with a different client-id.

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://datatracker.ietf.org/doc/html/draft-lindblad-netconf-transaction-id-02>.

[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-13, 3 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-service-assurance-architecture-13>.

[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

00 -> 01

*Define Parent and Child Transaction

*Context for the "local-commit-id" concept

*Feedback from Med, both in text and YANG module

Appendix B. Tracing configuration changes

Acknowledgements

The authors would like to thank Mohamed Boucadair for his reviews and propositions.

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