NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2020

M. Boucadair
Orange
Q. Wu
Z. Wang
Huawei
D. King
Lancaster University
C. Xie
China Telecom

November 3, 2019

Framework for Use of ECA (Event Condition Action) in Network Self Management
draft-bwd-netmod-eca-framework-00

Abstract

Event-driven management is meant to provide a useful method to monitor state change of managed objects and resources, and facilitate automatic triggering of a response to events, based on an established set of rules. This would provide rapid autonomic responses to specific conditions, enabling self-management behaviors, including: self-configuration, self-healing, self-optimization, and self-protection.

This document provides a framework that describes the architecture for supporting event-driven management of managed object state across devices. It does not describe specific protocols or protocol extensions needed to realize the objectives and capabilities discussed in the document.

Table of Contents

1. Introduction

Network management data objects can often take two different values: the value configured by the administrator or an application (configuration) and the value that the device is actually using (operational state). Particularly, these network management data objects can be fetched from various different YANG datastore

[RFC8342] by subscribing to continuous datastore updates [RFC8641] without needing to poll for data periodically.

YANG-Push mechanisms are used to select which data objects are of interest using filters and provide frequent or prompt updates of remote object state, thus allowing (client) applications to maintain a continuous view of operational data and state and enabling a network operator to optimize the system behavior across the whole network to meet objectives and provide some performance guarantees for network services.

Network management may rely upon one or multiple policies to influence management behavior within the system and make sure policies are enforced or executed correctly so that there will no conflict in policies and that the observed behavior is the expected one. Event-driven policy (i.e., ECA Policy [RFC8328]) enables actions being automatically triggered based on when certain events in the network occur while certain conditions hold. YANG Push subscription provides a source for such events.

It is often the case that where Event Condition Action (ECA) is defined is decoupled from where ECA is executed. ECA Engine in the management system or the network device defines one or multiple events corresponding to the workflow management, correlate these events with action triggers and create ECA policy. ECA policy can be enforced either at the management system or pushed to and executed by the network device. Alternative, some of these predefined events can be translated into filter in the YANG push subscription which is in turn used to select data objects that are of interest. When these data objects are streamed out to the destination, both the management system and network device check for the condition when the event is observed. If the condition is satisfied, the ECA script is executed.

Event-driven management (of states of managed objects) across a wide range of devices can be used to monitor state changes of managed objects or resource and automatic trigger of rules in response to events so as to better service assurance for customers and to provide rapid autonomic response that can exhibit self-management properties including self-configuration, self-healing, self-optimization, and self-protection.

This document provides a framework that describes the architecture for supporting such event-driven management.

This document does not describe specific protocols or protocol extensions needed to realize the objectives and capabilities discussed in the document.

1.1. 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.

2. Problem Statement

2.1. Defining Network Event and Network Control Logic

Datastores are used by network management protocols such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Operational state data objects, in the operational state datastore, provide network visibility to the actual state of the network, and ensure the network is running efficiently.

The network event are used to keep track of state of changes associated with one of multiple operational state data objects in the network device. Typical examples of network event include a fault, an alarm, a change in network state, network security threat, hardware malfunction, buffer utilization crossing a threshold, network connection setup, and an external input to the system.

To control which state a network device should be in or is allowed to be in at any given time, a set of conditions and actions are defined and correlated with network events, which constitute an event-driven policy or network control logic.

YANG Push subscription allows client applications to select which datastore nodes are of interest and provides source of network events. The NETCONF client can define event-based policy based on YANG Push subscription data source or some other data source.

2.2. Delegating Network Control Logic to Network Device

Usually the NETCONF clients subscribe to continuous datastore updates and rely on event notifications sent to the NETCONF client to check for the condition so that reaction to many network events may be very slow in the face of communication glitches between the client and the sever. Such solution doesn't scale well.

It is more desirable in many circumstances to delegate all event response behaviors (e.g., recover from network failure, instruct the network to control congestion) to the NETCONF server so that the network can react to network change as quickly as the event is detected.

The event response behaviors delegation can be done using YANG push subscription filter enhancements, e.g., define a new filter to allow the NETCONF client send updates only when the value falls within a certain range. Another example is to define a filter to allow the NETCONF client send updates only when the value exceeds a certain threshold for the first time but not again until the threshold is cleared. In the latter case, additional state is required.

In addition, the event response behavior delegation can be done by pushing ECA policy to the network device. Similar to YANG Push subscription filter, the ECA approach also includes filter and defines it as Event and Condition separately in the ECA policy model. Different from using YANG Push subscription filter, ECA allow a group of events to be observed, allow multiple actions to be triggered, e.g., sending log report notification, add or remove multiple YANG Push subscriptions.

2.3. Executing ECA Script in the Network Device

When the YANG Push subscription filter or ECA policy is pushed to the server, the server is expected to register the event conveyed in the YANG push subscription filter or Event-driven policy, generate server specific script. With a server specific script, the server can manipulate various network resources autonomously.

After the event registration, the server subscribes to its own publications encapsulated in the event notification with respect to all events that are associated with ECA Policy so that the publication is intercepted and all events specified in the ECA policy model are continuously monitored by the server before the publication is encapsluated in the event notification and sent to the YANG Push subscription's client. At the moment of event detection, the server loads the operational state data object filtered by the YANG Push subscription's filters or ECA policy into the auto-configured ECA's event and execute the ECA's associated condition-action chain.

The condition is associated with an ECA event and evaluated only within event threads triggered by the event detection, and the action corresponds to a set of statements that may trigger state changes in the device or publication content changes in the Event subscription and could be various different operations to be carried out by the server:

- Configuration data object reconfiguration;
- ECA Log report Notification;
- Add or remove one or multiple YANG Push Subscriptions;

- Invoke another Event in the same network device or different network device.

2.4. Event-Driven Notification Handling

ECA notifications are the only ECA actions that directly interact and hence need to be unambiguously understood by the client.

ECA notification can be sent when the client may find any interesting about the associated event with all the logic to compute said data (e.g., datastore content changes history, median values), and delegate computation task to the server via an ECA script.

When a "Send ECA notification" action is configured as an ECA Action, the client may receive different ECA notification associated with the same event or different events, YANG Push Publication will also be sent through Event notification. Therefore it is important for client to correlate of events and ECA notifications received from the server.

When ECA notification and YANG Push Publication are both pushed to the client, the client can execute client specific script generated in the same way as the server does and manipulate various network resources autonomously remotely. However the network resource can not be manipulated twice in both client and the server. Therefore policy conflict should be avoided or resolved.

2.5. Requisite State Information

A ECA policy rule is read as: When event occurs in a situation where condition is true, then action is executed. The ECA associated state is used to indicate when Events are triggered and what actions must be performed on the occurrence of an event.

A simple information model for one piece of the ECA associated state is as follows:

```
{
  event name;
  start time;
  end time;
  threshold value;
  event occurrence times
}
```

The event that is observed could be a fault, an alarm, a change in network state, network security threats, hardware malfunctions,

buffer utilization exceeding a threshold. For any of the aforementioned events, multiple actions may be triggered.

3. Architectural Concepts

3.1. What is Defined in ECA Policy?

ECA Event is a change of datastore operational state. Each policy rule consists of a set of conditions and a set of actions. Policy rules may be aggregated into policy groups. These groups may be nested, to represent a hierarchy of policies.

ECA Condition is evaluated to TRUE or FALSE logical expression. ECA condition is specified as a hierarchy of comparisons and logical combinations of thereof, which allows for configuring logical hierarchies. One of ECA condition example is logical hierarchies specified in a form of:

<target><relation><arg>
where target represent managed data object while arg represent either constant/enum/identity, Policy variable or pointed by XPath data store node or sub-tree,

relation is one of the comparison operations from the set: ==, !=, >, <, >=, <=

Logical calculation between multiple trigger conditions:

The YANG language cannot clearly describe complex logical operations between different condition lists under the same event, for example, (condition A & condition B) or condition C.

By default, the ECA model performs logic "AND" operation between different conditions in the same Event. That is, event is triggered when different conditions are met at the same time. For example,

Event A consists of two conditions:
o Condition A;
o Condition B;
If Condition A AND Condition B is met;
Event A is triggered;
Action A is executed.

For the logic "OR" operation between different conditions, the conditions can be defined in different events. If the corresponding event is triggered, the same action is executed. For example,

Event A is triggered on Condition A.
Event B is triggered on Condition B.
If Condition A is met;
Event A is triggered;
Action A is executed.
If Condition B is met;
Event B is triggered;
Action A is executed.

ECA Action is one of the following operations to be carried out by a server:

- Configuration data object reconfiguration
- ECA Log report Notification
- Add or remove one or multiple YANG Push Subscriptions
- Invoke another Event in the same network device or different network device

In case of one event triggering another event, a set of events can be grouped together and executed, in a coordinated manner. The action associated with the event can be executed in the same network device or in different network devices. In the latter case, events executed by different network devices need to coordinate as a group to fulfil a task, previously set.

3.2. Where is ECA Script and State Held?

The ECA state information described in Section 2.5 and associated ECA script has to be held somewhere. There are two locations where this applies:

- in a central controller where decisions about resource adjustment are made;
- in the network nodes where the resources exist.

The first of these locations have a good visibility of the whole network or information of the flow packets are going to take through the entire network, but requires a centralized, searchable repository of all network information that can be used for diagnostics, service assurance, maintenance or audit purposes. The response to network event can be slow since all monitored data objects from large amount of network devices need to be sent and correlated at the point where decisions about resource adjustment are made, less alone multiple network event triggering a single action.

Conversely, if the ECA state and associated ECA script is held in the network nodes, it makes policing of resource adjustment easier. It means many points in the network can have immediate response to network event. The limitation is the configurations and state of a particular device does not have the visibility of the whole network or information of the flow packets are going to take through the entire network, so they provide little insight into network level policy-related behavior.

3.3. What State is Held?

As already described, the network control logic associated with ECA script needs access to ECA state table. It stores network events pushed from YANG push subscription or ECA policy model, threshold value it set for observed network management data object.

In addition, when the event needs to be continuously monitored, the Event scheduling information such as start time, end time can be included.

In case of sending updates only when the value exceeds a certain threshold for the first time but not again until the threshold is cleared, a threshold clear flag is also needed.

In case of monitoring the data change or data change rate, for example, YANG Push On-Change mode [RFC8641] or ECA Threshold Test [I.D-wwx-netmod-event-yang], the ECA state table need to store history state to check the condition to be satisfied and determine the current state.

4. Architecture Overview

The architectural considerations and conclusions described in the previous section lead to the architecture described in this section and illustrated in Figure 1. The interfaces and interactions shown in the figure and labeled (a) through (f) are described in Section 4.1.

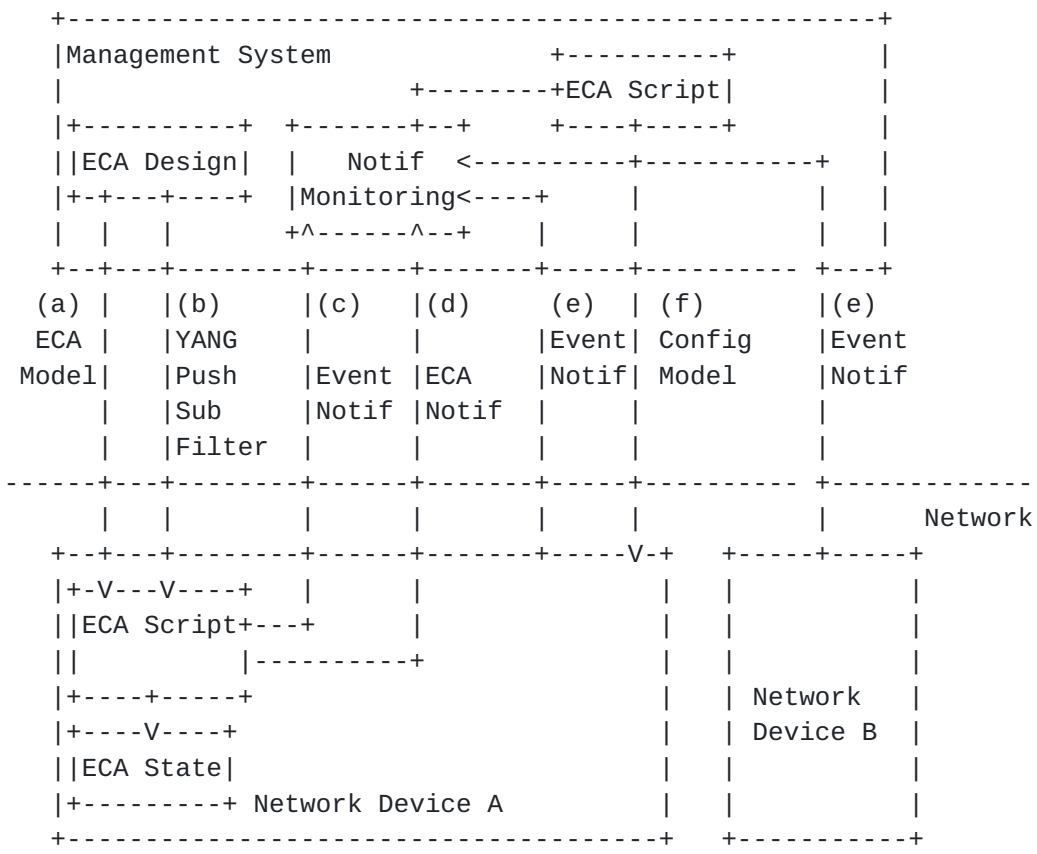

Figure 1.Reference Architecture for Use of ECA and Network Self Management

4.1. Telemetry Automation in the Network Device

As shown in Figure 1, some component in the management system defines and designs ECA Policy rule. This may be invoked by a Service assurance application or device fault self-management application. We show this component on the figure as the "ECA Design", and it extracts Event and Condition in the ECA model and fill into YANG Push Subscription as filter. When YANG Push subscription filter is pushed down to the network device, ECA script can be generated automatically from it (ECA script can also be generated in the management system and downloaded to the network device). The YANG Push subscription request, indicated on Figure 1 by the arrow (b), includes all of the parameters of network management data objects that the requester wishes to be supplied, such as filter node, threshold value, start time, and end time. Note that the requester in this case may be the management system shown in the figure or a distinct system such as data collector.

The network device registers network event that is corresponding to the filter carried in the YANG Push Subscription and enters the

network event in its ECA state and then the server subscribes to its own continuous datastore updates in the operational state datastore that is encapsulated in the event notification as publication to the YANG Push subscriber.

Upon the network event is detected, the server intercepts the publication of subscribed data and loads the operational state data object in the operational state datastore into the auto-configured ECA's event and execute the ECA's associated condition. When ECA Condition is evaluated to TRUE, the operational state data objects will be filtered and the remaining data objects will be entered back into the publication of subscribed data and encapsulated in the event notification (c) and sent to notification monitoring component in the management system.

The notification monitoring component may further derive some new ECA policy rule and fed into ECA Design component. The remaining procedure is same as the procedure starting from (b).

Alternatively, the ECA design component can push ECA model directly with additional actions included (a) to the network device, ECA script is generated automatically from ECA model. The ECA model request, indicated on Figure 1 by the arrow (a), includes additional action parameters besides one included in the YANG Push subscription request.

The network device register network event that is corresponding to the ECA carried in the ECA request and enter them in its ECA state and then the server subscribes to its own continuous datastore updates in the operational state datastore that is encapsulated in the event notification as publication to the YANG Push subscriber.

Upon the network event is detected, the server loads the operational state data object in the operational state datastore into the auto-configured ECA's event and execute the ECA's associated condition. Different from YANG Push subscription filter, the server will not intercept the publication of subscribed data. Instead, it allows the server to trigger a set of actions associated with the network event, e.g., send ECA log report notification, add/remove YANG push subscription, reconfigure the network management data object within the control of the server. After all actions are executed, one or multiple separate ECA notifications (d) can be sent to the notification monitoring component in the management system, the remaining procedure is same as YANG Push subscription case.

Conversely,when, network level policy-related behavior became necessary, once a subscription has been set up, event notification message associated with the subscription from different network

device will be sent to the notification monitoring component in the management system(e), which in turn trigger network behavior change on the network device via configuration model (f).

4.2. Detecting and Resolving Policy Conflict

There are two possible places where policy conflict can take places:

1. An event triggers multiple actions in the network device that cannot occur together as specified by the system administrator.

2. Multiple ECA notifications or multiple combination of ECA notification and Event notification lead to generate ECA policy that cannot occur together.

In both case, policy conflict can be addressed by policy conflict detection mechanism and Policy validation mechanism.

4.3. Chain Reaction of Coordinated Events

In some cases events executed by the same or different network devices can be executed in a coordinated manner. To make sure these network devices coordinate on some task or a group of events coordinate in the same network device, these events on the same or different network devices need to be pre-configured to work together. During capability negotiation phase, the management system should know what each network device supports, which event may take action, and what condition on which event. So ECA model with multiple events can be configured on the network device to allow one event be triggered by another event configured on the same network device.

5. Security Considerations

The framework described in this document for supporting autonomic event-driven self-management will require consideration of potential security and operational requirements, and ensure best security practices and methods are applied.

Key security considerations that will be discussed in future versions of this document, include:

- Authentication of ECA programming requests;

- Application of suitable authorization methods when enabling ECA functions;

- Securing ECA communication channels;

- Locking ECA device config and state databases;
- Mitigation, and negation, of ECA functional component attacks;
- Logging and auditing of ECA transactions;
- Maintaining ECA device confidentially.

6. Acknowledgements

This work has benefited from the discussions of ECA Policy over the years. In particular, the SUPA project [https://datatracker.ietf.org/wg/supa/about/] provided approaches to express high-level, possibly network-wide policies to a network management function (within a controller, an orchestrator, or a network element).

Igor Bryskin, Xufeng Liu, Alexander Clemm, Henk Birkholz, Tianran Zhou contributed to an earlier version of [GNCA]. We would like to thank the authors of that document on event response behaviors delegation for material that assisted in thinking that helped develop this document.

Finally, the authors would like to thank David Hutchison and Mehdi Bezahaf at Lancaster University, Phil Eardley and Andy Reid at British Telecom, for their input and applicability of ECA device self management to the Next Generation Converged Digital Infrastructure (NG-CDI) project.

7. References

7.1. Normative References

[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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

7.2. Informative References

[I-D.bryskin-netconf-automation-yang] Bryskin, I., Liu, X., Clemm, A., Birkholz, H., and T. Zhou, "Generalized Network Control Automation YANG Model", draft-bryskin-netconf-automation-yang-03 (work in progress), July 2019.

[I-D.clemm-netmod-push-smart-filters] Clemm, A., Voit, E., Liu, X., Bryskin, I., Zhou, T., Zheng, G., and H. Birkholz, "Smart Filters for Push Updates", draft-clemm-netmod-push-smart-filters-01 (work in progress), October 2018.

[I-D.clemm-nmrg-dist-intent] Clemm, A., Ciavaglia, L., Granville, L., and J. Tantsura, "Intent-Based Networking - Concepts and Overview", draft-clemm-nmrg-dist-intent-02 (work in progress), July 2019.

[I-D.wwx-netmod-event-yang] Wang, Z., WU, Q., Xie, C., Bryskin, I., Liu, X., Clemm, A., Birkholz, H., and T. Zhou, "A YANG Data model for ECA Policy Management", draft-wwx-netmod-event-yang-04 (work in progress), November 2019.

[RFC8328] Liu, W., Xie, C., Strassner, J., Karagiannis, G., Klyus, M., Bi, J., Cheng, Y., and D. Zhang, "Policy-Based Management Framework for the Simplified Use of Policy Abstractions (SUPA)", RFC 8328, DOI 10.17487/RFC8328, March 2018, <https://www.rfc-editor.org/info/rfc8328>.

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/info/rfc8572>.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Michael Wang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: wangzitao@huawei.com

Daniel King
Lancaster University
Bailrigg, Lancaster LA1 4YW
UK

Email: d.king@lancaster.ac.uk

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf.bri@chinatelecom.cn