YANG-based Data Modeling Techniques for the Content Layer of NETCONF to Improve Query Throughput

YangMin Lee, JaeKee Lee

Department of Computer Engineering, Dong-A University, South Korea
{yangwenry, jklee}@dau.ac.kr

Abstract

NETCONF was originally proposed as a protocol for updating and managing configuration data on complex heterogeneous network equipment that forms part of a larger current network. There is particularly the problem of processing efficiency in the Operation and Content layers. Currently, NETCONF uses YANG to generate an XML document as a data model. The unique operations of NETCONF are used to change the configuration of the equipment. However, when there are multiple managers, the standard NETCONF exhibits relatively low data modeling flexibility, and this decreases the query processing throughput. In this paper, we propose the use of a YANG XML document that describes a data block generation method based on the dependencies of equipment configuration data. The proposed technique can be used to generate a logical structure of the equipment configuration data, which may be stored and grouped as a set of independent equipment data in the physical memory. Hence, when multiple network administrators modify different data blocks, the processes can be performed concurrently. We performed experiment for various factors and confirmed that our improve NETCONF is outperform for existing method or protocol.

Keywords: NETCONF, Management, Data modeling, Subtree filtering

1 Introduction

The management of a modern network comprising heterogeneous equipment is a complex task that requires a protocol for monitoring and controlling equipment information. The SNMP is an example of a protocol that can be used for management, but it used monitoring purpose only. So, it has many limitations in functions for purpose of network management. The installation of a modern network comprising heterogeneous equipment units with diverse configuration data is dispersed, and this requires a protocol that can be used to directly change the equipment configuration data. The protocol should also be able to perform the task in a centralized manner and from a remote location. To meet the requirements for performing diverse network management functions and achieve better network productivity, a Network Configuration Protocol (NETCONF) standard was enacted in 2006. In 2011, a revised version of NETCONF with many new features was released [1]. However, additional improvement is still required because the layers still contain inadequacies and some parts of the operation process and data modeling remain inefficient [2].

In this paper, we propose a technique for improving the structure of the data generated by YANG (Yet another Next Generation) for use as the current data modeling language of NETCONF. The proposed technique improves the operating efficiency of the Operation layer. We also propose a technique for expanding the <partial-lock> operation. An improved data structure that enables efficient modification and storage of the equipment configuration data provided by the Operation layer via the Remote Procedure Call (RPC) layer was also developed [3-4]. Lastly, experiments were performed to evaluate the effectiveness of a NETCONF that uses the proposed improvements, with particular focus on the Content layer, to which was applied a data structure that affords efficient configuration data update [3]. The results were compared with those of a standard NETCONF [1] and a data-structure-based NETCONF [3], with particular focus on the query throughput and process delay. The NETCONF with the proposed improvements was confirmed to be superior.

2 Related Studies

2.1 NETCONF-related Studies

Since the introduction of the NETCONF standard, diverse studies have been conducted on the application and use of the protocol. In [5], a NETCONF-based network management system comprising three parts, namely, a manager referred to as BUPT-NET, a server, and a module group, was developed. In [6], a YANG

[*] Corresponding Author: JaeKee Lee; E-mail: jklee@dau.ac.kr

DOI: 10.3966/160792642018121907004

and NETCONF-based management system was built for network management. YANG is presently used for many NETCONF studies [7-9]. Some previous studies have also analyzed NETCONFs and developed a system known as XCMS for NETCONF-compatible IP sharing equipment [10]. [11] reports the development of a structure for managing a network that uses NETCONF, and also describes the structure of a NETCONF-based network management system and its movement elements.

Loureiro et al. [12] proposed the development of the NETCONF agent function for link state monitoring, in comparison with other management technologies, such as SNMP. [13] emphasized the security and scalability advantages of NETCONF over SNMP and Representational State Transfer (REST), however, there was no consideration for multiple administrators and data modeling. [14] considers resource use of SNMP and NETCONF protocol implemented in embedded device. [15] compared the performance of the NETCONF, Constrained application protocol (CoAP), and SNMP protocols within the Fog computing architecture. [16] and [17] compared NETCONF with SNMP in wireless network management scenarios. These studies did not consider multiple managers, and in the aspect of data modeling, utilized standard techniques using YANG. [18] used the ProVerif cryptographic protocol verifier to analyze the NETCONF protocol that relies on the transport layer for authentication, and to identify the manager of the device to be authenticated. The focus of [18] was on NETCONF security. Wallin and Wikstrom [19] verified that NETCONF and YANG greatly simplified the configuration management of devices and services, and still provided superior performance. Performance tests were executed on a cloud that managed 2000 devices. This study does not consider special data modeling techniques as a study on the use of YANG and its performance and standardization.

2.2 Studies on Improvement of the Operation Layer and the Data Modeling of the Content Layer

Several studies have been conducted on the improvement of NETCONF, with the major goal being to increase the performance of each layer of the protocol. The layers that have been most studied for this purpose are the RPC and Operation layers [4, 10-11]. Several studies have particularly been conducted to improve the efficiency of the Operation layer and reduce the overall processing delay by improving the processing method of the RPC layer. Studies on improving the efficiency of the Operation layer mainly considered issues related to the memory of the network equipment, such as the <lock> operation, and were aimed at increasing the speed of the writing and reading processes in the equipment memory by modifying the operations [3, 20].

The modeling method employed in the Content layer is an important subject because it affects the overall performance of a NETCONF network management system.

YANG is a data-modeling language specifically designed for application to NETCONF [7-8]. It is the de facto standard that is applied to the Content layer of NETCONF. YANG uses XML to store and exchange equipment data, and supports the interoperability of the heterogeneous equipment that is used together with NETCONF [7-8]. However, YANG cannot be used to create a data-dependency model structure for multiple users. [20] presents a technology that can be used to lock a portion of the data store in various cases. A partial lock mechanism affects only the configuration data and running data store, and not the start-up data store. Using [20], we were able to implement a new command, namely, <partial-lock> that expanded the capabilities of NETCONF.

[21] and [22] proposed a YANG model for describing sliceable transponders to provide variable speed, code, modulation format, and monitoring functions. This was applied in NETCONF, and experimentally demonstrated on a test bed. [23] introduced the YANG model for describing services in network virtualization and explained that NETCONF can be applied even when the network is virtualized. These studies used the YANG model for transponder description or service description, and were the latest NETCONF technologies. However, they were not modeling techniques for multiple managers and partial locks.

[24] suggested a tree-based association rule-mining technique that provided the necessary information to enable a quick query and response for XML file structures and content. However, it was not developed for use in NETCONF. [25] has extended the extensible access control markup language (XACML) and implemented it in the NETCONF network management system (BUPT-NEP) to make NETCONF more secure. In this research, subtree filtering was used to represent resources; however, the flexibility of resource representation was lower than XPath. [26] proposed Region-based Labeling (ReLab), which is a subtree-based labeling scheme that generates labels using depth-first traversal. However, [26] was not a labeling technique that considered partial-locking.

2.3 Problem Definition

We analyzed several previously proposed technologies and found that many of them were aimed at improving the performance of NETCONF. There are some standard modeling languages such as YANG that are applied to the Content layer but cannot be used to generate data that are based on the dependency of the network equipment configuration data. We thus needed a more flexible partial locking mechanism because standard partial locking technology only enables the locking of specific portions that is used for low-level element in XML documents, and also has many restrictions. We therefore determined that for more than one network manager to efficiently perform corrective equipment reconfiguration, the <partial-lock> operation of the Operation layer should be flexible and efficient [3-4].

To solve these problems, first, we need to expand the <partial-lock> operation. Second, we needed a data modeling technique that could be applied to the Content layer to extend the <partial-lock> operation. We therefore used YANG to perform the data modeling for the network equipment. We then modified the generated data model. In other words, we developed a method for using a subtree technique to create a data structure based on the dependency information of the network equipment. The method can be used to generate logical equipment configuration data separate from the physical structure that can be directly stored in the memory.

3 NETCONF Agent and Manager to which the Improvement Technique Was Applied

3.1 Outline of the Proposed System

In this study, the subtree-based data modeling technique was applied to the Content layer of NETCONF to improve the operation efficiency of the Operation layer. We employed a structure of the NETCONF agent and manger that uses a method for fast updating of the proposed data modeling technique and equipment configuration data. The agent and manager of the NETCONF are illustrated in Figure 1 and Figure 2 to describe the positions and functions of the proposed improvement techniques within the NETCONF.

The NETCONF agent used in this study incorporates features employed in previous studies on improving the XML storage data structure for transmission/receipt of equipment configuration command and updating of the configuration data [4], and rules for checking and selecting candidate documents [3]. The details of the employed data extraction process using VTD-XML (Virtual Token Descriptor) and XPath, as well as the data modeling technique used in the Content layer, are available in [27-29]. The data modeling techniques used in this paper are described in the last part of 3.2.1, and the overview and application of VTD-XML are described in the last part of 3.2.3. The operating sequence of the agent is presented in Table 1.

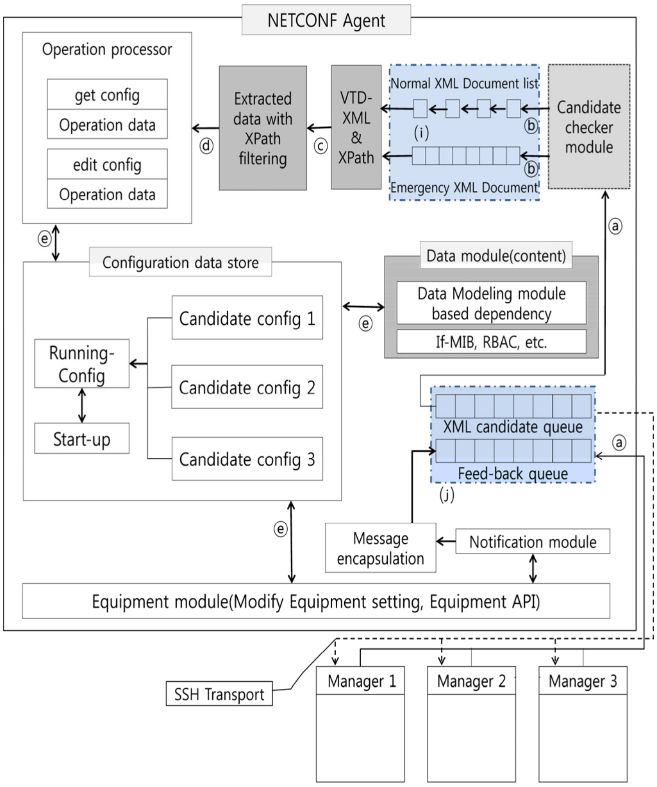


Figure 1. Structural diagram of the improved NETCONF agent

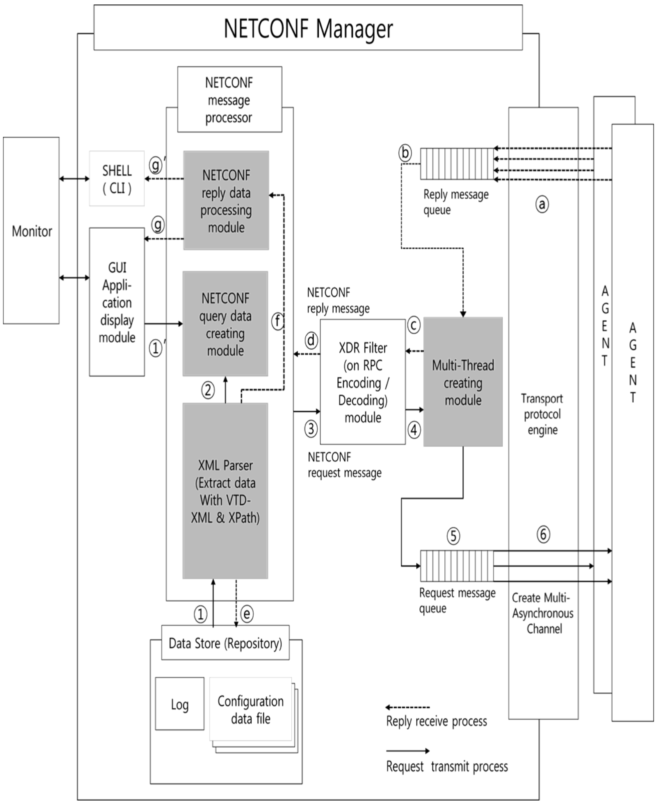

NETCONF Manager
NETCONF message processor
Monitor
SHELL (CLI)
GUI Appli-cation display module
NETCONF reply data processing module
NETCONF query data creating module
XML Parser (Extract data With VTD-XML & XPath)
ⓖ'
ⓖ
ⓕ
①'
①
②
③
④
⑤
⑥
ⓐ
ⓑ
ⓒ
ⓓ
ⓔ
NETCONF reply message
NETCONF request message
XDR Filter (on RPC Encoding / Decoding) module
Multi-Thread creating module
Reply message queue
Request message queue
Transport protocol engine
Create Multi-Asynchronous Channel
AGENT
AGENT
Data Store (Repository)
Log
Configuration data file
Reply receive process
Request transmit process

Figure 2. Structural diagram of the improved NETCONF manager

Table 1. Operation sequence of NETCONF agent

step	action
1	An XML document containing an operation is generated and sent by a manager to a queue that stores candidate XML documents. The content of the queue is then handed over to the module that checks the candidate documents. (Marked ⓐ)
2	The candidate XML documents are classified into emergency and normal documents based on the results of a check of their priority using the rules proposed in [3-4], within the check module (dotted line box). (Marked ⓑ)
3	All the XML documents, both emergency and normal (which have different processing sequences), are expressed in a binary array using VTD-XML, and the positions and contents of all the elements of the XML documents are then stored. (Marked ⓒ)
4	The required data are extracted through an XPath query using the content of the array. (Marked ⓓ)
5	An operation of the Operation Layer is performed using the extracted information, and the result is sent to the equipment configuration information storage and used by the data module to model the configuration information. This is done based on the dependency between equipment configuration information and their periodic communication with each other. (Marked ⓔ)
6	The modeling of the initial configuration information of the equipment is done using the address of the XML element within the array generated by the VTD-XML. The dependency between equipment configuration information is particularly expressed based on the relative path that can be expressed in XPath. A table is then generated where the dependent information is organized as one subtree.

Figure 1 illustrates the structure and process sequence of the NETCONF agent, including the module position. Figure 2 shows the block diagram of the NETCONF manager developed in this study. A multithread is used to process the queries and the VTD-XML is employed for extraction of the required data. The most important module of the NETCONF manager is the NETCONF message processor. The other modules are as follows:

- NETCONF query data creation module: This module generates the query messages using the information inputted through the GUI or data store, and sends them to the XDR filter.
- NETCONF reply data processing module: This module extracts the required data from the received reply messages using the XML parser and stores the reply result in the data store or hands it over to the GUI.

The numbered continuous lines with arrows in Figure 2 represent the process of transmitting the request of the network manager. The dotted lines marked with alphabets represent the process of receiving the reply, while the lines with two-way arrows represent the common processes. The process of transmitting a request by a manager is presented in Table 2, while the process by which the manager receives the reply is presented in Table 3.

Table 2. Process of the transmission of a request in NETCONF manager

Process of the transmission of request in NETCONF Manager	
step	action
1	The manager receives data from the GUI module or data store. (Process ①’, ①)
2	The manager generates a query message in the NETCONF query data creation module using the needed part of the input data. (Process ②)
3	The generated NETCONF request message is handed over to the XDR filter through process ③ and the coded.
4	The operation of the multi-thread generation module is commenced. (Process ④)
5	The message is enqueued in the request message queue through the generated thread (Process ⑤)
6	A multi-asynchronous channel is generated in the transport protocol engine through process ⑥ for parallel communication with the agent.

Table 3. Process of receives the reply by NETCONF manager

Process of receives the reply by NETCONF Manager	
step	action
1	The manager receives data from the established multi-channel and enqueues it in the received message queue. (Process ⓐ)
2	While dequeuing the data in the queue, the manager hands over the query data to the multi-thread generation module. (Process ⓑ)
3	The decoded reply message is handed over to the XDR filter. (Process ⓒ)
4	The XDR filter hands over the reply message th to NETCONF message processor (process ⓓ). The message enters the XML parser inside the message processor.
5	The NETCONF message processor extracts the required data through the XML parser and proceeds to process ⓔ whereby it records the data in the configuration storage file.
6	The NETCONF message processor outputs the data through process ⓕ to the NETCONF reply data processing module, and then to the GUI application through process ⓖ, or the CLI through process ⓖ’.

3.2 Improvement of the Operation Efficiency of the Operation Layer

3.2.1 Expanded Partial Locking Technique for Improvement of the <partial-lock> Operation

The previous NETCONF standard (RFC 4741, 2006) uses a global lock to avoid inconsistency among memories when multiple managers access the same memory concurrently. As noted in a related work, the 2011 NETCONF standards include <partial-lock> and enable the simultaneous operation of multiple managers. However, there are some limitations [20]. For example, although partial memory locking is possible, the hardware-centric <partial-lock> operation including the global lock is still inefficient. In other words, to increase the speed of changing the equipment configuration and the productivity of the network managers, an expanded operation that locks only a specific part of the data store is required. In the present study, the equipment configuration data dependency verification technique was employed [3]. This technique divides the equipment configuration data into groups in advance of storing them, and does the storing in a classified manner.

Figure 3 illustrates the concept of a <partial-lock> operation that partially locks the data store. If there is interdependency among the equipment configuration data, the storage would be modeled as a subtree. Another subtree may recursively exist in the model subtree. Each shaded rectangle in Figure 3 represents a group of equipment configuration data classified in advance of storage. This structure enables the application of the <partial-lock> operation to a subtree. This method is different from that used by nodes for classification, and partial locking of XML elements is only achieved using the XPath expressions described in [1, 20].

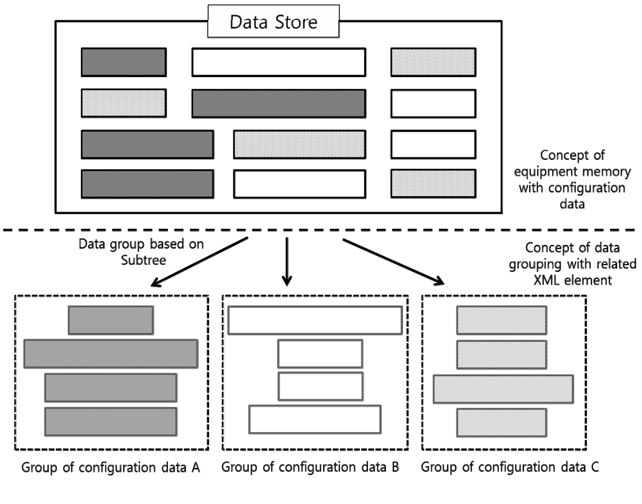

Data Store
Concept of equipment memory with configuration data
Data group based on Subtree
Concept of data grouping with related XML element
Group of configuration data A
Group of configuration data B
Group of configuration data C

Figure 3. Concept of the <partial-lock> operation for the partial locking of the data store and grouping based on their interdependency

The method used by XPath to identify other nodes apart from the current one is based on the respective locations of the two nodes. Both the relationship between the nodes and their attribute values can be used for data grouping, as was done in the present study for detailed inspection [3-4].

The methods used in previous studies are described here in more detail. The equipment configuration data in the network equipment memory is organized in the form shown at the top of Figure 3. The information is intricately entangled; however, it can be classified into groups based on equipment configuration data that is interdependent or related to each other as shown at the bottom of Figure 3. This is done by labeling and subtree-based storing for the XML document element.

A grouped data set can be considered as a subtree. If the data are connected vertically in a certain direction and are interdependent, the top node of a vertical data group would be the root of the subtree.

In more detail, the leftmost dotted line box in Figure 3 corresponds to the subtree represented by the root node, r1, in Figure 4. The middle dotted line box corresponds to r2, and the right dotted line box corresponds to the subtree, r3. Node n1 in subtree r1 is an ancestor of n2 and n3, and is uniquely identified in the entire data. Moreover, the root nodes, r1, r2, and r3 of each subtree, are uniquely identified, and each is independent. Here, when accessing interface1, and changing ip4-address or macaddr included in the same subtree, interface1 is entirely locked. However, if the manager accesses the system to modify the information related to 'name' or 'subnet', it does not lock the subtree that includes interface1. Instead, it locks the labeled range based on r2 or r3 (subnet) of the below subtree. That is, it is possible to lock both r2 and r3 based on r1, or lock them on the basis of each root.

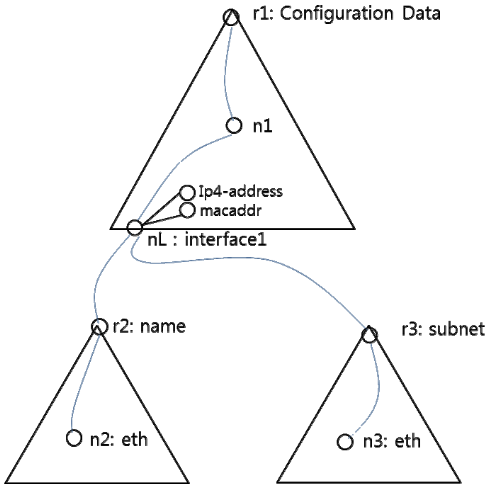


Figure 4. Data modeling based on their interdependency

3.2.2 Improved Equipment Configuration Data Update Using Additional Data Structure

If multiple operations simultaneously approach the same memory at the same time, an efficiency problem may arise. This problem was solved in the present study by filtering. For this purpose, the candidate check module was used in such a way that only adequate candidate documents were left after filtering multiple XML documents that contained equipment configuration update information [3-4]. The candidate documents are processed by the agent of the equipment, and a number of candidate documents equal to the number of network managers can be generated and maintained within the limits of the memory. Furthermore, for efficient update of candidate XML documents for the running-config, a rule was developed to give priority to each network manager in accordance with its rating and authority, as reflected in the query packet. In addition, in the data structure used to process and efficiently handle candidate XML documents, there is an agent XML candidate document queue that exclusively receives queries from the manager, separate from the queue of notifications (reply and feedback). Separate lists of queues that handle emergency documents and queues that handle normal documents are also allocated in accordance with the candidate document processing rules.

Figure 5 illustrates the handling of the candidate XML documents around rectangles (i) and (j) (broken/dotted lines) in Figure 1. The candidate documents that pass through the candidate check module can then update the running-config of the equipment. For efficiency, the equipment configuration update documents may be arranged in accordance with their urgency as indicated by the XML document list. This is done by creating an emergency XML queue separate from the XML document list. The feedback queue below the XML candidate document queue is used to send quick replies to the network manager.

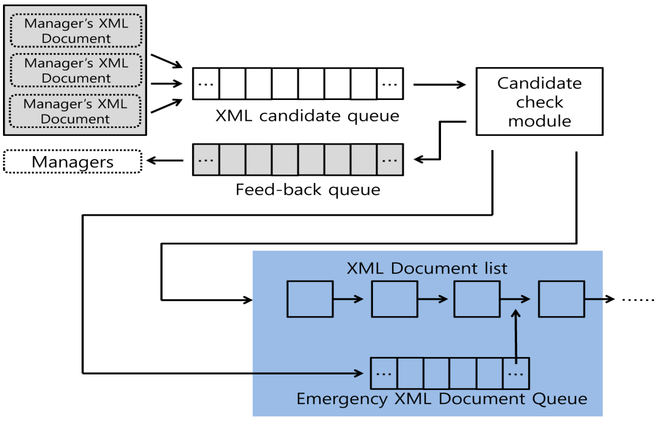
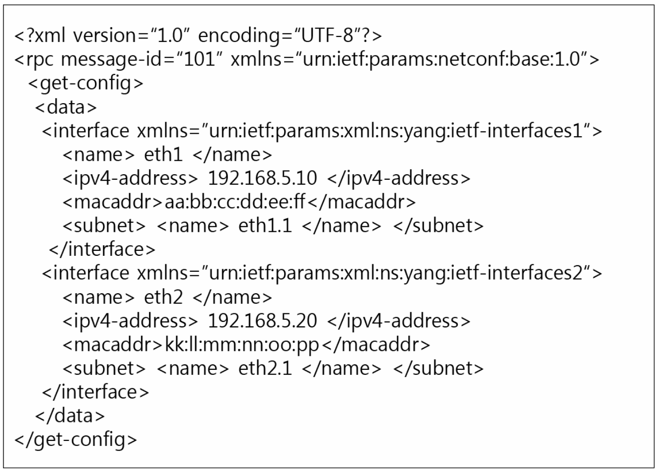


Figure 5. Operation process of the candidate check module and the improved data store structure

3.2.3 Proposed Data Modeling Technique of the Content Layer

Subtree-based XPath data storage method. The parsing of XML documents and extracting or storing of the required data are the most time consuming operations of a NETCONF. Parsing and a technique for extracting and storing the necessary information are required to process the XML document as shown in Figure 6. This involves a request for the performance of a <get-config> operation, and the name, IP address, MAC address, and other information are structurally inserted into the <interface> element. Furthermore, the <interface> element is presented twice, although with different attribute values. The data dependency is as discussed in subsection 3.2.1. The equipment configuration data is stored in the form of a subtree using a subtree-based XPath process, by which the data in Figure 6 is converted to the form shown in Figure 7.



```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101" xmlns="urn:ietf:params:netconf:base:1.0">
  <get-config>
   <data>
    <interface xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces1">
       <name> eth1 </name>
       <ipv4-address> 192.168.5.10 </ipv4-address>
       <macaddr>aa:bb:cc:dd:ee:ff</macaddr>
       <subnet> <name> eth1.1 </name> </subnet>
     </interface>
    <interface xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces2">
       <name> eth2 </name>
       <ipv4-address> 192.168.5.20 </ipv4-address>
       <macaddr>kk:ll:mm:nn:oo:pp</macaddr>
       <subnet> <name> eth2.1 </name> </subnet>
    </interface>
   </data>
</get-config>
```

Figure 6. XML query document

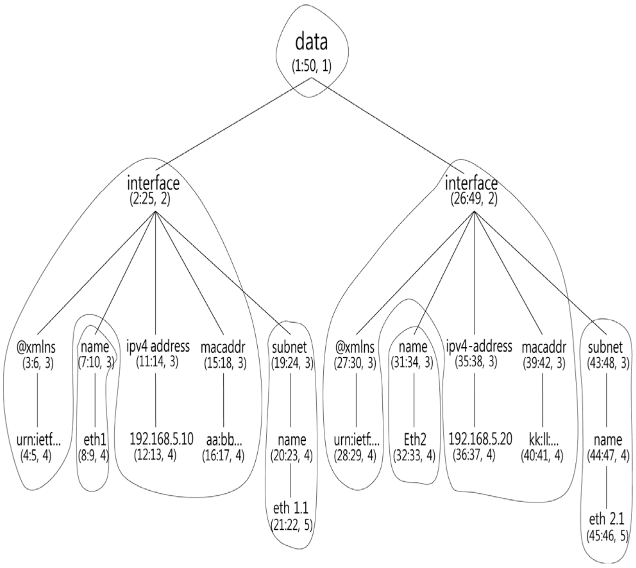


Figure 7. Labeling of XML document tree

The latest technique for storing XML data is known as XML node labeling. In applying this technique, the first interface in Figure 6 can be expressed as (2:52, 2) using a label like that in Figure 7. The first two numbers separated by a colon in the node labels represent the range (start : end), while the third number indicates the depth of the node in the tree. The parts surrounded by the free-hand lines are respectively the subtrees of the XML tree, and each subtree can be stored as a storage unit.

The task of storing XML data in subtrees and, to skip depending on the form of the XPath query, the parsing used to extract the internal data of the subtrees, an XML fragmentation schema is required. Such a path fragmentation schema can be generated from the tree in Figure 7 and expressed as an XPFS tree, as shown in Figure 8. The number beside each node (prefixed with #) is the path ID of the node, N, which is allocated based on the type of the path from the root. Table 4 describes the storage of the equipment configuration data, including the XPFS subtree information in Figure 8 and the labeling information in Figure 7. In the present study, the interdependency between the configuration data and the subtree-based storage method was taken into consideration at the time when the equipment configuration data was created or stored for the first time using the XML subtree-based storage method [3, 27].

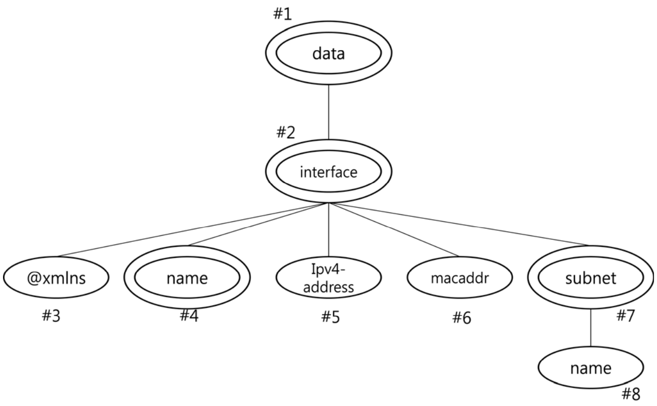


Figure 8. XPFS (XML path fragmentation schema) tree of the document tree in “Figire 6”

Table 4. Storage based on the subtree

no	name	path ID	begin	end	value
(1)	data	#1	1	50	<data> $$ </data>
(2)	interface	#2	2	25	<interface xmlns=“urn:ietf.params:sml:”>
(3)	name	#4	7	10	<name> eth1 </name>
(4)	subnet	#7	19	24	<subnet> <name> eth1.1 </name> <subnet>
(5)	interface	#2	26	49	<interface xmlns=“urn:ieft:params: xml:ns.yang:ieth-interfaces2”> <ipv4-address>192.158.5.20 </<ipv4-address> <macaddr> kk:ll:mm:nn:oo:pp </macaddr> $$ </interface>
(6)	name	#4	31	34	<name> eth2 </name>
(7)	subnet	#7	43	48	<subnet> <name> eth2.1 </name> <subnet>

The safety of subtree-based XPath processing. In this section, we present the basis for the structural join of each subtree to show the safety of the subtree-based storage method shown in Figure 7, Figure 8, and Table 4 [27].

The path type of the XML document D can be defined as a sequence of element tags/attribute names. This corresponds to the path of D in the XPFS tree. If there are two path types, one path type can be a prefix of another path type. For example, in Figure 8, path type ‘/data/interface’ is a prefix, as in path type ‘/data/interface/name’ and ‘/data/interface/subnet/name’. For XML node n in XML document D, P (n) represents the path type from the root of D to n. For example, in Figure 7, P (interface1) = /data/interface. For XML node n, range (n) represents the range (begin: end). Assuming that range(t) represents range(r) for the stored subtree t, whose root node is r, then with a subtree-based XML storage of XML document D derived by the XPFS tree, with the basic XML fragmentation rules, we get the following rules [27].

-Rule 1-

For two XML nodes n_1 and n_2, belonging to subtrees t_1 and t_2 of XML document D, stored in a subtree-based storage derived by the XPFS tree of D, n_1 is an ancestor of n_2 if the following two conditions are satisfied.

(1) range (t_1) contains or equals range (t_2)

(2) P (n_1) is a prefix of P (n_2)

Proof for Rule 1 is omitted in this paper. More importantly, according to Rule 1, the independent subtrees may be separated from each other and locked. It is also possible to concurrently lock the descendant subtree linked to the ancestor subtree by a structural join. This denotes that the subtree-based storage method excludes the risk of multiple managers accessing data at the same time and can be used safely.

Data modeling with integrated data storage using XPath processing and the dependency of the equipment configuration data. If the subtree XML data storage method is integrated with the data modeling of the dependency of the equipment configuration data, the data used to model the network equipment configuration data may be considered as a block. When an interface task or a task related to a subnet on the interface is to be performed using a NETCONF, the XPath query with the specific command expressed in an XML document may be in the form of either of the following XPath expressions:

/data/interface[@xmlns=“urn:ietf:params:xml:ns:yang: ietf-interfaces1”]//name **(1)**

/data/interface[@xmlns=“urn:ietf:params:xml:ns:yang: ietf-interfaces2”]//name **(2)**

Let us consider the case in which change the name of a specific interface and the interface name of a subnet for the relative address expressions (1) and (2), or retrieve it from the equipment configuration data and return it. If the data modeling is done using the XML storage structure proposed in this paper, a logical data structure of the form shown in Figure 9 would be generated for such a query.

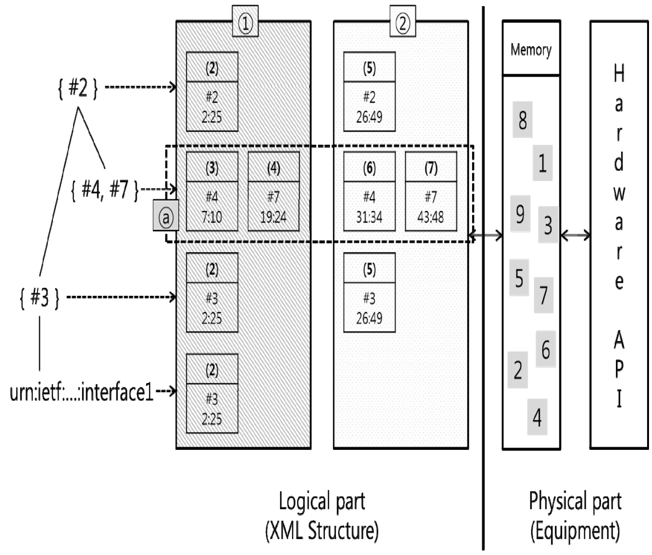


Figure 9. Structure of stored data for NETCONF query

For Query (1), the network equipment memory can be approached using the subtree in rectangle ① in Figure 9 as a data block, in which case the target of the approach would be the subtree root. Similarly, for Query (2), the subtree in rectangle ② with the same structure but a different storage range (label) can be considered as a data block. Accordingly, the interface names of the two queries and the name of the interface that uses the subnet can be searched for, with the memory of the subtree unit locked when two or more managers approach the same equipment.

Another sample query is that for the returning of all the names of the equipment interfaces, in which case the data can be extracted using the dotted line rectangle ⓐ. That is, if the subtree-based equipment configuration storage technique is used, the data block should be created based on the equipment configuration data dependency. It is therefore possible to extend the standard <partial-lock> operation [20]. This enables multiple managers to concurrently access each square to process the configuration setup. In the event of two or more managers approaching an equipment unit at the same time, a partial <lock> operation would be performed to separately lock each rectangle.

The memory construction state for Query 1 can be expressed in detail as shown in Figure 10.

The free-hand lines between the physical and logical parts of the figure represent the pointer connections between the actual and logical memories. In the execution of Query 1, the network equipment memory is approached through rectangle ① in Figure 10. When using Table 4, which gives the modeling result, Query 1 is processed with only the white part of rectangular ① locked. The shaded Box ② inside box ① is independently maintained as another subtree. That is, even if box ① is locked, box ② may still be approached. This is because the logical and physical parts of the data structure are entirely separated. Hence, what is shown in Figure 10 is the conceptual data grouping indicated in Figure 3 and Figure 4 and is implemented in the actual memory. With the memory organized in this manner, when the physical part is abstracted from the viewpoint of the network manager, the equipment configuration can be organized such that only the subtree storage structure of the XML document would be seen. Moreover, it would be possible to execute partial <lock> operations.

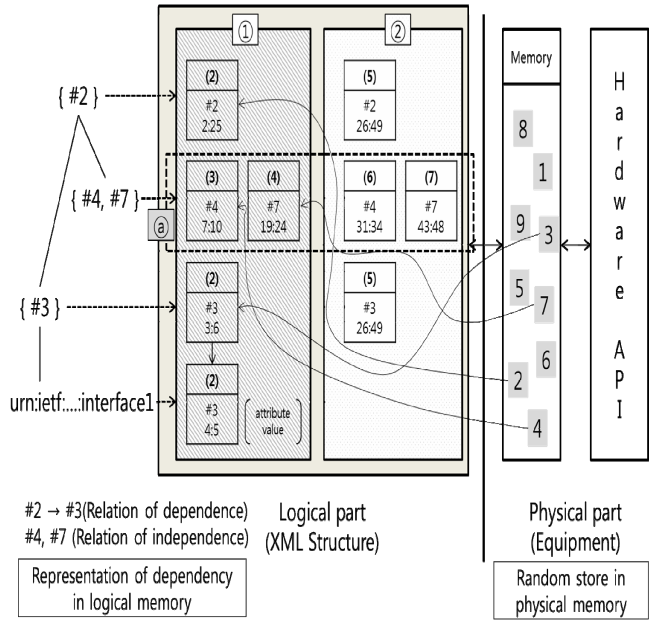


Figure 10. Relation between XML document structure and physical memory for NETCONF query

Data Extraction Using VTD-XML and XPath. VTD-XML is a parsing technology that operates in a non-extractive manner. In this paper, an expanded VTD-XML was also used to process XPath queries. Non-extraction denotes that parsing of the entire subtree is not required to extract its internal elements. In this case, an XML document is stored in the memory in a binary form, and the positions of the document elements are analyzed in a byte array to record the information. This is different from the method in which an object is generated within the memory by extracting the document parts, such as the Document Object Model (DOM). When VTD-XML is used, the XML documents can be quickly analyzed and processed [28]. In addition, when XPath processing uses only VTD-XML to extract the required data, the expression flexibility of XPath is maintained. This is, however, accompanied by the disadvantage of XPath being slower than other general subtree methods, although compensation could be made for this. We extract useful data in the form of an integrated XML document. With VTD-XML and XPath, applications can bind only relevant data items, avoiding the creation of useless objects. Because XPath can be applied to a parsed tree of XML, VTD-XML is used to create a tree-like table. Therefore, VTD-XML is suitable for the generation of a data structure of the type in Table 4, in which the root is within the tree structure [28-31].

In this paper, we use an open-source VTD-XML Java API, which consists of VTDGen (VTD generator), VTDNav (VTD explorer), and the AutoPilot Class. To use extended VTD-XML and XPath, we must include the ‘com.ximpleware.extended’ package.

4 Experimental Implementation Results and Performance Comparison

4.1 Experimental Environment

Experiments were performed to assess the performance of the NETCONF agent and manager using the proposed improvement techniques. All the improvement techniques applied to each layer of the NETCONF were modules prepared in C language and were implemented inside the agent and manager. The experiments were performed using a virtual switch because of the difficulty of implementing changes inside the agent in an actual network unit. The dependency of equipment configuration data was then modeled from the binary array. The experimental environments are shown in Table 5.

Table 5. Specifications of experimental environment

Category	Name
Operation System	Windows 7/64 bit
Development Language	Java, C
Date Modeling Language	YANG
RAM	8G Byte
CPU	Inter® core i7-3770
Software	Yenca, Cisco Nexus 1000V, Extended VTD-XML
Compared Factors	VTD-XML vs DOM: Throughput Speed, Memory Consumption Subtree-based vs Node-based: XPath Processing Time (Query), Space Requirement
Performed Operation	<edit-config> for interface configuration change with XML form
Network Composition	10 virtual switch (support NETCONF) 10 Host (for MAanager)
Platform	JVM 64 bit
Compared Protocol	Standard NETCONF (RFC 6241), data-structure-based NETCONF ([3] based on RFC 6241)

Experiments are divided into two categories. The first category is an experiment on the processing performance of the method applied to the improved NETCONF. Here, we compared the throughput speed and memory consumption while using VTD-XML and DOM. Moreover, we compared the processing time and space requirements of XPath processing using subtree-based storage with that of node-based storage. The second category is the comparisons between the improved NETCONF and data-structure-based NETCONF, and standard NETCONF. The comparison factors are query throughput and query processing delay. The improved NETCONF additionally uses a subtree-based storage technique according to VTD-XML, and an equipment data dependency compared with data-structure-based NETCONF [3], which is our previous study.

4.2 VTD-XML vs DOM

Five representative test data were used for the analysis (see Figure 11). In this experiment, we measured the performance of VTD-based parsing, in terms of both parsing throughput and memory consumption. The parsing throughput was used to measure the speed (amount of documents parsed per unit time (MB), MB/S). The memory consumption was measured as a ratio, which was total memory usage/original file size.

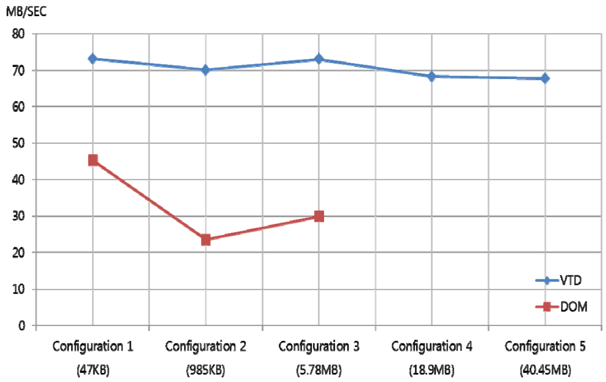


Figure 11. Throughput comparisons for VTD and DOM model

4.2.1 Throughput

As can be seen in Figure 11, when parsing small XML documents, it is far better to use the DOM, which quickly converts XML documents into a tree structure. However, VTD is relatively stable in terms of processing speed, regardless of the XML file size. The superiority of the VTD parsing model is that it shows almost similar processing speeds when the document size increases. On the other hand, the DOM model cannot handle file sizes over 5.78 MB because the DOM process structure consumes a large amount of memory.

4.2.2 Memory Consumption

Figure 12 shows that the memory usage of the VTD-XML parsing model matches the official specification. The memory usage ratio is close to the stable range (1.2~1.4) as the document size increases and is not affected by the document complexity (depth, schema). Therefore, when compared with the DOM model, it can be seen that VTD has stable throughput and relatively low memory consumption.

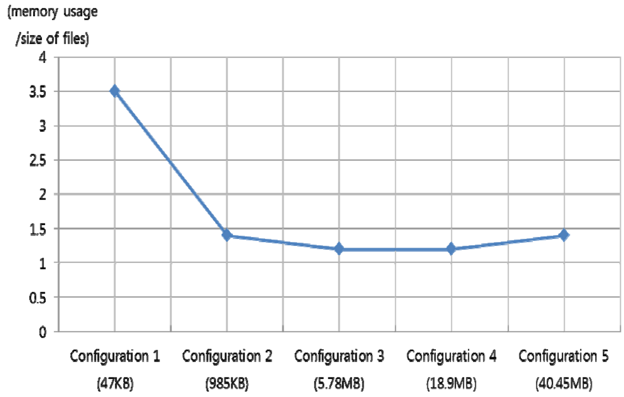


Figure 12. Memory consumption of the VTD model

4.3 Subtree-based vs. Node-based

In order to show the superiority of the subtree-based XML storage performance, we compared it with the node-based XML storage method. For the experiment, node-based XML storage, XPath processing (Process/Node) subtree based XML storage, and XPath processing (Process/Subtree) were constructed. In the Process/Node, each node is stored as a table record in the NODE, while in the Process/Subtree, each subtree is stored as a record in the SUBTREE table. The XPath expressions used are shown in Table 6.

Table 6. Query (XPath Expression)

Query	XPath Expressions
Query 1	/data/interface[@xmlns]/protocol [@type]/ip4-address
Query 2	/data/interface[@xmlns]//subnet[number] /name[@type]/protocol[/source/running]
Query 3	/data/validate/source/candidate

4.3.1 XPath Query Processing Time

For all XPath expressions in Table 6, the query performance of Process/Subtree was compared with the performance of Process/Node. The configuration2.xml file was used. We experimented, respectively, with setting the final result XML node of all queries as a descendant of the subtree root, and setting the final result XML node as the root of the subtree for all queries. In many cases, the network equipment configuration data could be retrieved or the necessary data modified, if we knew the root information of the subtree. Moreover, the performance of the Process/Subtree depends on whether the inside of the subtree should be parsed.

The left side of Table 7 indicates the processing speed when the query result consists of the child for subtree root. The right side shows the processing speed when the query result is made the root of the subtree. The Process/Subtree takes a longer time than the Process/Node owing to the parsing overhead, as shown on the left side of Table 7. However, Process/Subtree outperforms Process/Node, as shown on the right side of Table 7. There are several reasons for this result, but the biggest is that the size of the table NODE is much larger than the SUBTREE. Therefore, the time it takes to retrieve the node list for a twig node in the Process/Node is much longer than the time it takes to retrieve the subtree list for a twig node in the Process/Subtree.

Table 7. Query Processing Time (ms)

Query	Result: inside the subtree		Result: root the subtree	
	Process/ Node (ms)	Process/ Subtree (ms)	Process/ Node (ms)	Process/ Subtree (ms)
Query 1	48.62	82.08	49.23	18.17
Query 2	182.78	273.92	183.11	21.96
Query 3	59.52	62.48	59.45	24.58

4.3.2 Space Requirement

As mentioned in the previous experiment, subtree-based XML storage is much smaller than node-based storage. In the first experiment, if the size of configuration2.xml is 985 KB, the NODE table stores about 20,000 records, and the SUBTREE table stores about 4,700 records. NODE and SUBTREE take up approximately 0.98 MB and 0.51 MB of space in the database table, saving space if we use subtree-based XML storage. An XPFS tree is essentially required in the Process/Subtree, however, it is optional in the Process/Node [27]. Therefore, the XPFS tree sizes were measured for the five configuration files used in our experiment. Table 8 shows that the space required for the XPFS tree is a negligible rate, compared to these five XML document sizes.

Table 8. Space overhead rate for XPFS

Size of Configuration file (KB): S_c	Size of XPFS Tree (KB): S_{Ttc}	(S_c/S_T)*100 (%)
47	33.37	7.1
985	46.3	4.7
5,781	62.8	0.11
18,972	65.5	0.03
40,521	65.5	0.01

4.4 Comparison of NETCONF Protocols

4.4.1 Comparison of Query Throughputs

There are two experimental methods for evaluating the query throughput with changing equipment configuration data:

The first method involves the measurement of the ratio of the number of queries actually processed to the varying total number of queries. The variation of the total number of queries induces the NETCONF manger to change or read the equipment configuration data provided by the agent. As the number of queries was increased from 100 to 1,000 in the present study, the number of received command execution success messages (OK messages) was measured.

In the second experimental method, the number of received command execution success messages is measured with increasing number of managers from 1 to 10 for a fixed number of queries per manager. The query types used in the present study were <get-config> and <edit-config>.

Figure 13 shows the results of the first experiment. The improved NETCONF had a processing success rate of no less than 93%, even when the number of queries reached 1,000. The average query throughput was 97 %. By comparison, the successful throughputs of the NETCONF with data structure [3] and the standard NETCONF [1] began to decrease when the number of queries reached 700, the values being 93 % and 86% respectively. Data-structure-based NETCONF has a structure that can efficiently process concurrent query transactions and use multiple XML documents. However, data-structure-based NETCONF uses a standard YANG-generated data model, and does not use subtree-based storage. Thus, it is slow to access certain data and has an insufficient partial lock function based on data dependency. This causes a decrease of query throughput.

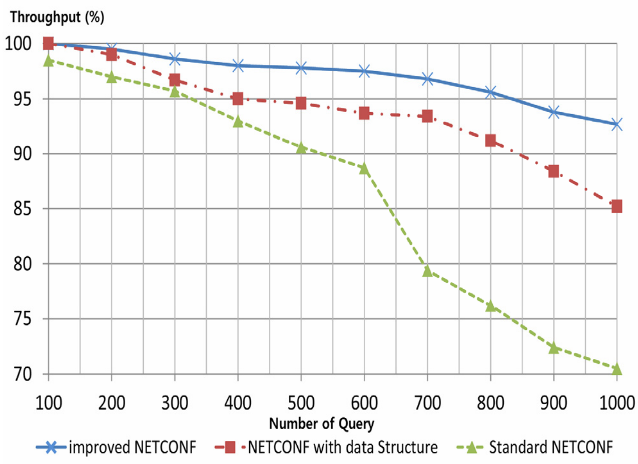


Figure 13. Throughput with respect to number of queries

In the case of the standard NETCONF, the probability of the discarding of XML documents that were handed over via queries was relatively high because of the absence of a data structure that enabled smooth processing of XML queries. In addition, if a specific manager does not complete a transaction, all the queries of the other managers would be discarded, even if they are operating in <partial-lock> mode [20].

In the second experiment, the priorities of the managers were introduced and each manager was set to generate 10 queries. Because the queries were processed in an XML document, there were also priorities among them. Two particular managers had relatively high priorities compared to the other eight. In addition, among queries of the same priority rating, one that arrived earlier was accorded higher priority. The experimental parameters are given in Table 9. Because no document was discarded by any of the three types of NETCONFs when the number of managers was 1, as shown in Figure 14, the throughputs were all 100%. However, beginning at when two managers were used, the throughput of the standard NETCONF fell below 100% for packets to which priorities were attached.

Table 9. Throughput experimental parameters

Basis parameter	Considered value
Number of Query	Increased 100 to 1000
Number of Document per Client	10
Size of one XML Dcoument	Below 3K byter
Existence of Priority between Client	Exist (for Increasing Client)
CPU usage rate	Below 17% (All kind of NETCONF are same)

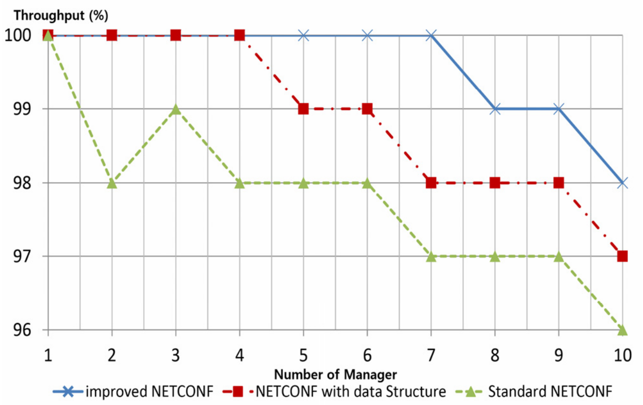


Figure 14. Throughput with respect to the number of managers

Standard NETCONF and data-structure-based NETCONF may appear to be operated in the <partial-lock> mode, but are actually operated in the full lock mode by high-priority managers. According to the rules suggested in [20], if a manager with a higher priority performs a memory lock or global lock function, <partial-lock> would fail.

In the case of the improved NETCONF, some of the queries were not processed, with some occasionally discarded, when the number of managers reached 7. However, the throughputs of all the three types of NETCONFs did not significantly decrease until the number of managers was 10. The trivial reduction at this point was because the number of queries that could actually be processed was only 100. If the number of managers increases, the proposed NETCONF creates a data structure for partial locking. Because it uses a labeling by equipment data dependency, subtree-based storage method, and VTD-XML, it is obvious that the partial locking function would be available for such situations where multiple managers exist and the query speed is increased.

4.4.2 Comparison of Query Processing Delay

The query processing delay, which changes the equipment configuration data, can be measured by two methods. The changing factor in the experiment is the same as in 4.4.1.

The processing delay is determined by the time between then a query is sent by the manager and when the reply for its successful processing by the agent is received. There are two important parameters that should be noted in the results of an experiment to evaluate the query processing delay. The first is the XPath delay, which is the time required to process the XPath expression of the equipment configuration data stored in the form of an XML document. The other is how fast an update can be performed after a query to change the configuration of the network equipment is received when there are more than two managers. In the case of the improved NETCONF, the equipment configuration data is stored as indicated in Table 4, with an XML document expressed in the form of an address. In addition, it includes a queue and list data structure, which may comprise two or more candidate XML documents, and corrects the equipment configuration data in parallel as need arises. This experiment was performed under the condition wherein the initial configuration data was generated inside the existing network equipment without provision for the time required to create the initial data structure given in Table 4. The experimental parameters are presented in Table 10.

In the case of the experiments performed using increasing number of queries, for queries less than 100, the results presented is the value obtained by multiplying the average delay for a single query by the number of queries. As shown in Figure 15, for up to 200 queries, the improved NETCONF, the NETCONF with data structure [3], and the standard NETCONF [1] consumed a little more time than that obtained by multiplying 20ms (the single query delay [29]) by the number of queries. That is, the actual delay was a little more than 20ms × N, where N is the number of queries.

Table 10. Delay experimental parameters

Basis parameter	Considered value
Process Delay per <edit-config>	Average: about 21 millisecond (for 3K byte XML)
Process Delay per <get-config>	Average: about 23 millisecond (for 3K byte XML)
Number of Query	Increased 100 to 1000
Number of Agent	Increased 1 to 10, 1 (for Increasing Query)
Number of Document per Manager	10
Size of one XML Document	Below 3K byte
Existence of Priority between Manager	Exist (for Increasing Manager)
CPU usage rate	Below 17% (All kind of NETCONF are same)

The delay of the standard NETCONF began to increase significantly when the number of queries reached 300, and again when the number reached 600. The delay of the NETCONF with data structure began to differ from that of the improved NETCONF when the number of queries reached 500. In the case of the improved NETCONF, the processing required only 24ms, even when the number of queries reached 1,000. This is almost the same as the single query delay and was enabled by the fact that when using the improved NETCONF, dependent data could be directly found by checking up to the root of the subtree when approaching the equipment configuration data. In addition, the subtree enabled rapid partial lock. The cause of the longer delay times of data-structure-based and standard NETCONFs is much similar to that discussed in Section 4.4.1.

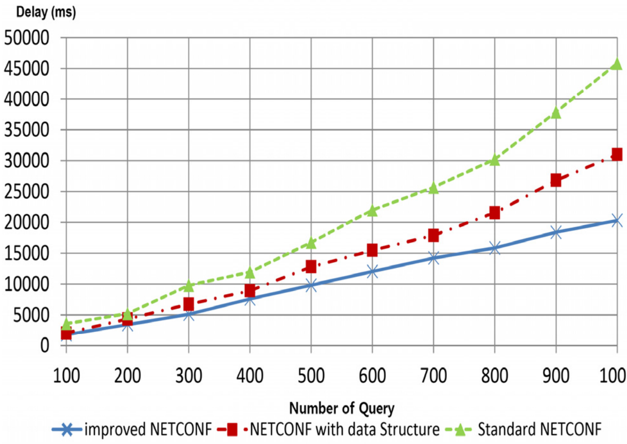


Figure 15. Process delay with respect to the number of queries

In the second experiment, the delay was measured for increasing number of managers with varying priorities, each of which generated 10 queries. Figure 16 compares the results for the different NETCONFs.

Because the improved NETCONF could have two or more candidate configuration XML documents and could parallel change the equipment configuration data with respect to the condition of the memory approach, its response time was shorter than that of the standard NETCONF, in which one manager had to wait until the memory approach of other manager had been completed. Although the XML document processing efficiency of the NETCONF with data structure was also high, its speed was slower than that of the improved NETCONF because of the incapability of parallel processing.

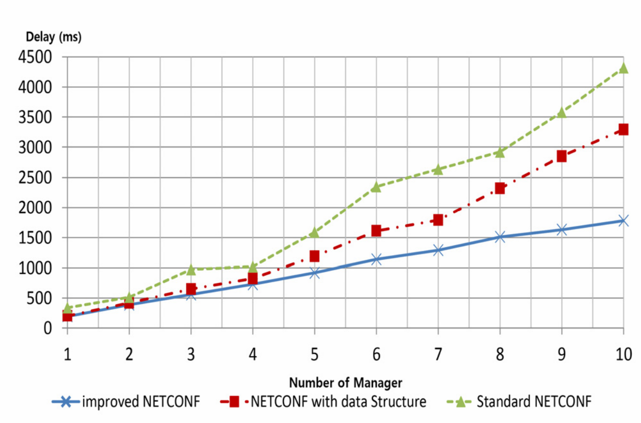


Figure 16. Process delay with respect to the number of managers

5 Conclusion and Future Study

The NETCONF standard was introduced to meet the management requirements of modern networks and increase the productivity of the network management. Although it has been a long time since the first NETCONF standard was announced, and some corrections and enhancements have been introduced along the way, there is still much more to be improved in the protocol.

We propose three methods to overcome the shortcomings of NETCONF.

The first method uses VTD-XML, which does not parse the entire tree in order to process XML documents containing configuration data at a high speed. This method can process XML documents of a much larger size and handle more than 5 MB of data using much less memory as compared to the DOM model.

The second method proposed is the use of subtree-based XPath data storage for fast processing and stability of XPath queries. In this method, it is not necessary to parse redundant data, and all data in the tree can be accessed by referencing the root information. Compared to the node-based processing method, the speed of retrieving the data within the tree is relatively slow using this method. However, we have experimentally confirmed that the results of the query are retrieved much faster than in case of the node-based method in case of the subtree root. Moreover, the memory overhead is much lower than in case of node-based processing.

Finally, we propose a data modeling method for the Content layer of NETCONF and an efficient data storage technique for improving the efficiency of the Operation layer. The proposed methods involve a modification of the model structure generated using the YANG XML data model to take the data dependencies of the utilized equipment into account.

The improvement of the Operation layer efficiency was particularly accomplished by enhancing the data modeling of the Content layer. The modeling technique of equipment configuration data and grouping method employed in the Content layer were developed to enable interlocking with the Operation layer. In addition, the efficiency of the expanded <partial-lock> operation in the Operation layer was improved by a data grouping procedure based on the interdependency of the equipment configuration data. Furthermore, a data structure that enables more efficient processing of XML queries, even when two or more managers are present, was proposed.

To quantitatively verify the performance of the proposed methods, the query throughputs and query processing delays of the improved NETCONF, the data-structure-based NETCONF [3], and the standard NETCONF [1] were compared.

We have confirmed that the improved NETCONF performs better than the other protocols by using efficient YANG-based data modeling when there are multiple network managers and by applying a dual queue data structure for simultaneous transaction processing. The throughput is the best among the three protocols for an increasing number of queries and an increasing number of managers. The processing delay was also found to be the lowest among the three protocols because of reasons similar to those for the best throughput. Thus, the experimental results indicate that the improved NETCONF can increase the productivity of network management and the processing efficiency.

Further study using more elaborate experiments and considering more parameters is required. Additional research is also required for finding the most appropriate form for constructing equipment configuration data in a subtree form.

Finally, it is necessary to compare the proposed method with other NETCONF improvement models and apply our proposed improved NETCONF in a practical network environment.

Acknowledgments

This work was supported by the Dong-A University research fund.

References

[1] R. Enns, M. Bjorlund, J. Schoenwaelder, A. Bierman, *Network Configuration Protocol (NETCONF)*, RFC 6241, June, 2011.

[2] J. Yu, I. A. Ajarmeh, An Empirical Study of the NETCONF Protocol, *2010 Sixth International Conference on Networking and Services*, Cancun, Mexico, 2010, pp. 253-258.

[3] Y. M. Lee, J. K. Lee, Improving and Optimizing the Operation Layer Algorithm of NETCONF Protocol, *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, Canada, 2014, pp. 449-455.

[4] Y. M. Lee, M. Y. Cha, J. K. Lee, Development of Update Methods for Configuration Data of NETCONF Protocol Considering Multiple Network Administrators, *Journal of Internet Computing and Services*, Vol. 14, No. 5, pp. 27-38, October, 2013.

[5] H. Ji, B. Zhang, G. Li, X. Gao, Y. Li, Challenges to the New Network Management Protocol: NETCONF, *2009 First International Workshop on Education Technology and Computer Science*, Wuhan, Hubei, China, 2009, pp. 832-836.

[6] J. Schonwalder, M. Bjorklund, P. Shafer, Network Configuration Management Using NETCONF and YANG, *IEEE Communications Magazine*, Vol. 48, No. 9, pp. 166-173, September, 2010.

[7] M. Bjorklund, *YANG: A Data Modeling Language for the Network Configuration Protocol*, RFC 6020, October, 2010.

[8] M. Scott, M. Bjorklund, *YANG Module for NETCONF Monitoring*, RFC 6022, October, 2010.

[9] J. Akhtar, YANG Modeling of Network Elements for the Management and Monitoring of Elastic Optical Networks, *2015 IEEE International Conference on Telecommunications and Photonics*, Dhaka, Bangladesh, 2015, pp. 1-5.

[10] M. S. Lee, S. M. Yoo, H. T. Ju, J. W. Hong, Performance Improvement of Systems and Network Configuration Management Based on NETCONF, *The Journal of Korean Institute of Communications and Information Sciences*, Vol. 33, No. 9B, pp. 786-794, September, 2008.

[11] Y. Chang, D. Xiao, H. Xu, L. Chen, Design and Implementation of NETCONF-based Network Management System, *2008 Second International Conference on Future Generation Communication and Networking*, Hainan Island, China, 2008, pp. 256-259.

[12] D. Loureiro, P. Gonçalves, A. Nogueira, NETCONF Agent for Link State Monitoring, *2012 IEEE International Conference on Communications*, Ottawa, ON, Canada, 2012, pp. 6565-6569.

[13] P. R. da P. F. Santos, R. P. Esteves, L. Z. Granville, Evaluating SNMP, NETCONF, and RESTful Web Services for Router Virtualization Management, *2015 IFIP/IEEE International Symposium on Integrated Network Management*, Ottawa, ON, Canada, 2015, pp. 122-130.

[14] A. Sehgal, V. Perelman, S. Kuryla, J. Schonwalder, Management of Resource Constrained Devices in the Internet of Things, *IEEE Communications Magazine*, Vol. 50, No. 12, pp. 144-149, December, 2012.

[15] M. Slabicki, K. Grochla, Performance Evaluation of CoAP, SNMP and NETCONF Protocols in Fog Computing Architecture, *2016 IEEE/ IFIP Network Operations and Management Symposium*, Istanbul, Turkey, 2016, pp. 1315-1319.

[16] M. Słabicki, K. Grochla, Performance Evaluation of SNMP, NETCONF and CWMP Management Protocols in Wireless Network, *4th International Conference on Electronics, Communications and Networks*, Beijing, China, 2014, pp. 377-382.

[17] M. Slabicki, K. Grochla, Influence of the Management Protocols on the LTE Self-configuration Procedures' Performance, *30th International Sympos- iumon Information Sciences and Systems*, London, UK, 2015, pp. 439-446.

[18] F. Izadi, H. S. Shahhoseini, Automated Formal Analysis of NETCONF Protocol for Authentication Properties, *IEEE Conference 6th International Symposium on Telecommunications*, Tehran, Iran, 2012, pp. 1055-1059.

[19] S. Wallin, C. Wikstrom, Automating Network and Service Configuration Using NETCONF and YANG, *25th International Conference on Large Installation System Administration*, Boston, MA, 2011, pp. 22-35.

[20] B. Lengyel, M. Bjorklund, *Partial Lock Remote Procedure Call (RPC) for NETCONF*, RFC 5717, December, 2009.

[21] M. Dallaglio, N. Sambo, F. Cugini, P. Castoldi, Management of Sliceable Transponder with NETCONF and YANG, *2016 International Conference on Optical Network Design and Modeling*, Cartagena, Spain, 2016, pp. 1-6.

[22] M. Dallaglio, N. Sambo, J. Akhtar, F. Cugini, P. Castoldi, YANG Model and NETCONF Protocol for Control and Management of Elastic Optical Networks, *2016 Optical Fiber Communications Conference and Exhibition*, Anaheim, CA, 2016, pp. 1-3.

[23] S. Mehraghdam, H. Karl, Placement of Services with Flexible Structures Specified by a YANG Data Model, *2016 IEEE NetSoft Conference and Workshops*, Seoul, South Korea, 2016, pp. 184-192.

[24] J. G. Cho, A Extracting Information of Tree-Based Association Rules from XML Documents, *The Journal of Korean Institute of Information Technology*, Vol. 12, No. 11, pp. 173-180, November, 2014.

[25] J. Wang, B. Zhang, G. Li, Y. Li, X. Gao, Improvement of XACML Access Control Mechanism Based on NETCONF Subtree Filtering RPC, *2nd IEEE International Conference on Network Infrastructure and Digital Content*, Beijing, China, 2010, pp. 1000-1004.

[26] S. Subramaniam, S. C. Haw, L.-K. Soon, ReLab: A Subtree based Labeling Scheme for Efficient XML Query Processing, *2014 IEEE 2nd International Symposium on Telecommunication Technologies*, Langkawi, Malaysia, 2014, pp. 121-125.

[27] K. H. Shin, H. C. Kang, Subtree-based XML Storage and XPath Processing, *KSII Transactions on Internet and Information Systems*, Vol. 4, No. 5, pp. 877-895, October, 2010.

[28] C. Subhashini, A. Arya, A Framework for Extracting

Information from Web Using VTD-XML's XPath, *International Journal on Computer Science and Engineering*, Vol. 4, No. 3, pp. 463-468, March, 2012.

[29] Y. Gao, B. Zhang, G. Li, Y. Li, X. Gao, The Comparison and Analysis of Tree Data Model and Table-like Data Model based on NETCONF, *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics*, Wuhan, China, 2010, pp. 75-78.

[30] P. R. K. S. Bhama, R. Senthilkumar, P. Varshinee, XQUICK: An Efficient Path-Based XML Storage Scheme for Fast Query Processing and Update, *Journal of Internet Technology*, Vol. 18, No. 2, pp. 261-270, March, 2017.

[31] J. T. Chen, S. R. Tsai, A Content-Based Event Brokering System Embedding Lively Contents in XML Objects using Publish/Subscribe Communication Scheme, *Journal of Internet Technology*, Vol. 11, No. 2, pp. 227-236, March, 2010.

Biographies

YangMin Lee received the B.S., M.S. and Ph.D. degrees in computer engineering from Dong-A University, Busan, Korea, in 2000, 2002, and 2006 respectively. Since 2015, he has been an assistant professor at Dong-A University. His major research interests include ad hoc network, IoT, network management protocol, and software defined network.

JaeKee Lee received M.S. degree in electronic computing from Yeungnam University in 1983. And he received Ph.D. degrees in electronic information engineering from University of Tokyo, Japan, in 1990. Since 1990, he has been a professor at Dept. of computer engineering in Dong-A University. His major research interests network management, IoT.