https://doi.org/10.7840/kics.2022.47.9.1360

SDN 정책엔진의 사용자 모듈을 위한 분석 요청 정의 언어

이 승 준•, 윤 대 건*, 오 상 윤°

Request Definition Language for the User Module in the SDN Policy Engine

Seungjun Lee•, Daegun Yoon*, Sangyoon Oh°

요 약

현대전에서 작전 수행은 네트워크 중심전의 양상을 띄고 있으며, 이에 따라 군 전술망 자원을 효과적으로 사용하기 위한 여러 연구가 진행되고 있다. 단일 연구 결과가 아닌 여러 연구의 결과를 복합적으로 적용했을 때의 효과를 분석하기 위한 노력의 일환으로 통합 테스트베드가 구축되고, 여기에서 여러 네트워크 알고리즘을 동시에 수행하고 성능을 분석하기 위한 정책 엔진도 설계되었다. 하지만 이종 네트워크 환경에서는 사용자들이 요구하는 서른 다른 데이터 구조의 성능 지표와 이를 처리할 각 알고리즘의 서로 다른 실행 환경에 적응적으로 대응하기 어려운 문제가 있었다. 이에 본 연구에서는 프로그래밍 언어와 실행 환경 등 특정 기술에 종속되지 않는 정책 엔진을 위한 XML 기반의 인터페이스 포맷을 정의하고 그 스키마를 제안한다. 제안된 스키마를 사용하여 메시지는 특정 프로그래밍 언어에 종속되지 않고 인코딩과 디코딩을 할 수 있으며 Open Container Initiative 표준을 기반으로 실행 환경을 정의하는 컨테이너를 기술할 수 있다.

키워드 : 인터페이스 정의 언어, 정책 엔진, 테스트베드
Key Words : interface definition language, policy engine, testbed

ABSTRACT

In modern warfare environment, the well defined networks becomes important to the operations. Thus, researchers study on how to use the military tactical network resources effectively. To analyze effectiveness of the results from multiple studies together, an integrated testbed is critical as well as the design and the implementation of a policy engine that performs multiple network algorithms and analyze performance simultaneously. However, when the network environment is heterogeneous, it is hard to respond adaptively to the performance indicators of the different data structures and the different execution environments of each user algorithm. To address this issue, we propose an XML-based interface format and its schema for the policy engine, which is independent from specific technologies such as programming languages and execution environments. A message from and to the policy engine and the testbed can be encoded and decoded regardless of the programming language. Furthermore, it can describe containers of the execution environment based on the Open Container Initiative standard.

※ 본 연구는 방위사업청과 국방과학연구소가 지원하는 미래전투체계 네트워크기술 특화연구센터 사업의 일환으로 수행되었습니다 (UD190033ED).
• First Author : Ajou University Department of Artificial Intelligence, henry174@ajou..ac.kr, 정회원
° Corresponding Author : Ajou University Department of Computer Engineering, syoh@ajou.ac.kr, 정회원
* Ajou University Department of Artificial Intelligence, kljp@ajou.ac.kr, 정회원
논문번호 : 202205-080-0-SE, Received April 30, 2022; Revised June 8, 2022; Accepted June 12, 2022

Ⅰ. 서 론

현대전에서의 작전 수행은 기존의 무기 중심 전투 체계와 달리 컴퓨팅 자원과 잘 갖추어진 네트워크에 기반을 두고 네트워크 중심의 전쟁(Net-Centric Warfare) 양상을 보여준다[1]. 이에 따라 군 전술망의 중요성과 망 크기가 매우 증가하고 있으며[2], 대규모 군 전술망을 이용하면서 네트워크 자원을 효율적으로 사용할 수 있는 성능 개선점을 찾아내어 주어진 네트워크 자원을 효과적으로 사용하는 것이 매우 중요하다.

군 전술망 자원을 효과적으로 사용하기 위해서 연구들이 수행되고 있으며, 연구 결과들에 대한 효과성을 입증하는 결과들도 소개가 되고 있다. 그러나, 여러 연구 결과를 복합적으로 적용했을 때의 효과성 분석에 대해서는 아직 많은 사례가 보고되지 않았으며, 연구그룹 별로 다양한 연구 결과를 적용하고 성능을 분석할 수 있도록 해 주는 통합 테스트베드[3]를 구축하고 이를 중심으로 다양한 분석들이 이루어지고 있는 상태이다.

본 연구팀 또한 통합테스트베드를 구축하고, 이를 기반으로 여러 네트워크 알고리즘이 동시에 수행되는 상황에서의 네트워크 성능의 통합 분석과 이를 통한 성능 개선점을 도출하는 성능 분석을 위한 정책 엔진[4]을 설계하였다 (특허 출원 10-2022-00006572). 정책 엔진은 사용자들로부터 성능 지표 데이터를 입력받고 이를 성능 개선 알고리즘으로 처리해 성능 개량 정보를 계산할 수 있는 환경을 제공한다.

하지만 네트워크, 즉 망 규모가 크고 다양한 장치로 구성된 이종 환경인 경우, 기존의 정책 엔진은 사용자들이 사용하는 서로 다른 데이터 구조의 성능 지표와 이를 처리할 각자의 알고리즘을 가지고 있는 환경에 대해 적응적 대응이 어려울 수 있다. 따라서 효과적인 성능 분석을 위해서는 사전에 정의된 성능 지표와 알고리즘이 아닌 사용자 정의 성능 지표와 알고리즘을 입력받아 성능 분석 서비스를 제공해 줄 수 있어야 한다.

본 연구진은 프로그래밍 언어와 실행환경 등 특정 기술에 종속되지 않는 정책 엔진 구현을 위해 인터페이스 포맷을 위한 메시지 스키마를 설계하였다. 본 논문에 설명하는 메시지 스키마는 XML 기반으로 인터페이스의 포맷을 기술하도록 지원하며, 실행환경의 설명을 위한 인터페이스에 Open Container Initiative (OCI)와 Container Runtime Interface(CRI) 표준을 따르는 컨테이너 시스템을 사용하도록 하여 프로그래밍 언어와 플랫폼에 중립적인 정책 엔진의 구현이 가능하게 한다.

본 논문의 구성은 다음과 같다. 2장에서는 네트워크상에서 문서를 교환하기 위한 배경지식들을 알아본다. 3장에서는 이기종 환경을 통합하기 위한 XML 기반 언어와 성능 지표 처리 시스템에 대한 기존 연구들을 살펴본다. 4장에서 제안하는 인터페이스의 스키마를 살펴보고 5장에서 제안된 인터페이스의 표현력을 검증한 다음 마지막으로 결론을 짓고 끝낸다.

Ⅱ. 배 경

2.1 인터페이스 정의 언어 (Interface Definition Language)

두 소프트웨어 시스템이 상호작용을 하기 위해서는 상호 간 사용할 인터페이스에 관한 내용이 합의되어야 한다. 인터페이스 정의 언어(interface definition language, IDL)는 특정 프로그래밍 언어나 기술에 종속되지 않고 중립적으로 방법으로 인터페이스의 형태를 기술하는 명세서를 작성하는 데에 사용되는 언어 체계이다.

IDL의 대표적인 예로 SWIG(Simplified Wrapper and Interface Generator)[5]가 존재한다. SWIG는 C와 C++로 작성된 컴포넌트를 다른 언어에서 사용할 수 있도록 인터페이스를 정의하는 데에 사용되는 언어이다. SWIG는 C 함수에 대해 C 언어 스타일로 인터페이스를 정의 파일을 만들고 이를 다른 언어로 작성된 프로그램이 읽어 C 언어로 작성된 함수의 입출력 형태를 알아내어 호출하는 방식으로 작동한다.

언어 간 상호 참조가 가능하도록 인터페이스를 정의해 주는 IDL의 다른 예로는 스리프트(Thrift)[6,7]가 있다. 스리프트는 원격 프로시저 호출 (remote procedure call, RPC) 프레임워크로서 프로시저들의 입출력 인터페이스를 스리프트 언어로 정의하여 모든 서비스가 동일한 인터페이스로 통신을 진행할 수 있게 해 준다.

위 IDL들은 각자 설계된 언어로 인터페이스를 정의하고 있다. 이러한 언어들은 목적에 맞춰 효율적으로 작성될 수 있으나 이를 해석하기 위해서는 해당 언어를 작성하는 데에 사용된 전용 프로그램을 사용해야 하는 단점이 있었다. 정보 교환을 위해 폭넓게 사용되고 있는 확장성 있는 다목적 마크업 언어인 XML(eXtensible Markup Language)[8]은 넓은 범위의 정보 포매팅 능력을 갖추고 있어 다양한 형태의 정보를 XML로 포매팅할 수 있고 XML 문서 전달에서도 HTTP와 같이 널리 사용되는 프로토콜을 통해 전달할

수 있기에 높은 호환성을 가져 대부분의 기기에서 이용할 수 있다. 웹 서비스의 인터페이스를 정의하기 위한 IDL인 Web Service Description Language (WSDL)[9]는 XML을 IDL의 포맷으로 사용하여 높은 호환성을 제공해 주고 있다. WSDL은 서비스 제공자 측이 제공하는 웹 서비스의 정보를 XML로 기술한 규약으로 서비스가 어느 엔드 포인트에서 제공되는지, 사용해야 할 프로토콜이 어떻게 되는지, 어떤 서비스가 있는지, 주고받는 메시지의 포맷이 어떻게 되는지를 XML로 정의한다. 웹 서비스에 연결하고자 하는 클라이언트는 웹 서비스 제공자로(서버)부터 WSDL 문서를 읽어 들여 서버에 어떠한 조작이 가능한지 확인할 수 있고 WSDL의 데이터 타입 명세대로 데이터 타입을 맞춰 메시지를 작성한 다음 WSDL의 서비스 명세대로 서버가 지원하는 프로토콜을 통해 정의된 엔드 포인트에 연결한다.

WSDL 명세는 서비스의 엔드 포인트 정보는 물론 해당 서비스가 제공하는 기능에 대한 추상화를 제공한다. 이 추상화는 클라이언트와 서버가 진행할 상호작용에 대한 정보를 제공하는데 먼저 데이터 유형을 정의하고, 서버와 클라이언트가 주고받을 수 있는 메시지의 포맷을 정의하고 마지막으로 서버와 클라이언트 사이의 상호작용 순서를 정의한다. 또한 WSDL에서는 네트워크 프로토콜과 인터페이스 추상화를 분리하여 정의한다. 이는 서버와 클라이언트가 메시지를 주고받기 위해 사용하는 프로토콜에 상관없이 항상 같은 인터페이스를 사용할 수 있게 하며 서비스가 프로토콜에 독립적으로 제공될 수 있게 해 준다 [10].

2.2 메시지 전달 패턴

웹 서비스는 XML 메시지를 주고받기 위한 용도로 SOAP(Simple Object Access Protocol)을 주로 사용한다[10]. SOAP는 HTTP나 SMTP와 같이 자주 사용되는 프로토콜을 통해 XML 기반의 메시지를 두 기기 간 교환하기 위한 프로토콜이다. 교환에 사용되는 XML 문서는 헤더와 보디(body)로 구성되어 있다. 헤더는 보안 및 트랜잭션을 위한 메타 정보를 기술하며 생략할 수 있다. 보디의 경우 애플리케이션이 필요로 하는 데이터를 기술한다. 많은 경우 SOAP는 RPC (Remote Procedure Call) 패턴으로 통신을 진행한다.

RPC 패턴 또는 서버-클라이언트 패턴에서는 클라이언트가 먼저 서버에게 요청을 보내고 서버는 해당 요청을 처리하여 그에 따른 응답을 클라이언트에게 회신한다. 이 패턴은 동기식 방식으로 메시지의 수신자가 정확히 지정되어야 하며 송신 측과 수신 측은 서로 같은 시간에 송신과 수신을 진행하여야 한다. 간단하게 구현할 수 있으나 송신 측과 수신 측이 서로의 엔드 포인트를 정확히 알고 있어야 하기에 결합도가 높다는 단점이 있다.

이와 반대로 발행-구독 (publish-subscribe) 패턴은 비동기 방식의 메시지 전달 패턴으로 이 방식에서는 네트워크에 데이터를 발행하여 전송하는 "발행자"와 네트워크에 발행된 데이터를 구독하여 수신하는 "구독자"로 나누어진다. 발행자는 데이터를 발행할 때 데이터를 수신할 노드를 특정하지 않고 토픽이라고 불리는 데이터의 분야만을 정해 발행한다. 구독자 측에서 수신할 때도 발신자의 엔드 포인트를 지정하지 않고 수신할 토픽을 메시징 시스템에 알려 해당 토픽에 발행된 데이터만을 받는다. 이렇게 함으로써 발행자와 구독자는 서로의 존재를 정확히 모르고 토픽만으로 연결되기에 시스템의 결합도가 낮아지는 장점이 있다. 발행-구독 패턴을 유지하기 위해서는 발행자와 구독자 중간에서 메시지의 전달을 매개하는 미들웨어인 메시지 브로커가 필요하지만 메시지 브로커가 메시지를 잠시 보관하는 것으로 발행자와 구독자가 같은 시간에 활성화되지 않고도 통신을 할 수 있다는 장점이 생긴다. 발행-구독 패턴을 사용하는 메시지 교환 체계로 ISO 표준 발행-구독 방식의 메시지 전달 프로토콜인 MQTT(Message Queuing Telemetry Transport)[11]가 존재한다.

본 논문의 정책 엔진은 통합 테스트베드와 메시지를 주고받기 위해 메시지 형식은 XML 포맷을 사용하고 전달 프로토콜은 MQTT를 사용한다. 이는 XML의 플랫폼 독립적인 특징과 발행-구독 패턴의 낮은 결합도 속성을 이용하여 특정 기술에 종속되지 않는 정책 엔진 구현에 도움을 주기 위해서이다.

Ⅲ. 관련 연구

이 장에서는 이기종 환경으로 구성된 네트워크의 통합을 위한 표준화 작업에 XML을 사용한 기존 연구와 사용자 정의 알고리즘을 위한 인터페이스를 정의한 기존 연구들에 관해서 설명한다.

3.1 XML을 통한 네트워크망 관리

IP 네트워크에서 네트워크를 구성하는 다양한 장치들로부터 기기 상태 정보를 수집 및 관리하고 동작을 조정하기 위해 간이 망 관리 프로토콜(Simple Network Management Protocol, SNMP)[12]이 많이 사용된다. SNMP는 구현이 쉽고 호환성이 좋다는 장점이 있어

초기 인터넷 환경에서 많이 사용되었으나, 이후 SNMP의 제한된 확장성과 효율성의 문제가 대두되었고, 이에 SNMP MIB의 트리 구조 정보가 XML로 쉽게 변환될 수 있다는 것에 착안하여 확장성을 가지는 XML의 장점을 이용하여 해당 문제를 해결하고자 하는 연구[13]가 진행되었다.

해당 연구에서는 확장성이 높은 XML을 이용하여 네트워크망을 관리하면서도 기존의 SNMP를 사용하는 기존 시스템과의 호환성을 위해 정보 손실 없이 XML과 SNMP MIB의 포맷 변환을 수행하는 알고리즘을 제안하였다. SNMP MIB의 관리 대상 정보는 트리 구조로 정의되며 이는 XML의 트리 구조를 사용해 문서 구조의 변경 없이 변환이 가능하다. 따라서 해당 연구에서는 SNMP MIB의 각 노드를 XML 엘리멘트로 매핑하고 MIB 노드의 내부 clause들을 그 엘리멘트의 속성으로 매핑하는 것으로 상호간 변환을 수행한다.

3.2 사용자 정의 알고리즘을 통한 유연한 네트워크 성능 측정

네트워크에서 발생하는 문제들을 빠르게 진단하고 대응하기 위해서 네트워크 성능 모니터링은 필수적이다. 모니터링을 위해 사용하던 기존 방식은 네트워크 단말에서 정보를 수집하거나 종단 간 프로브(probe)를 수행하는 것이었으나 네트워크 스택 내부 상황에 대한 정보를 수집하기에는 부족하였다. SDN(Software Defined Network)이 등장하면서 programmable 스위치가 사용되고 스위치 기반 모니터링이 가능해졌지만 샘플링[14], 미러링[15], 카운팅[16]과 같은 제한된 모니터링 기능으로 모든 패킷에 대한 정보를 수집하는 데에는 어려움이 있었다.

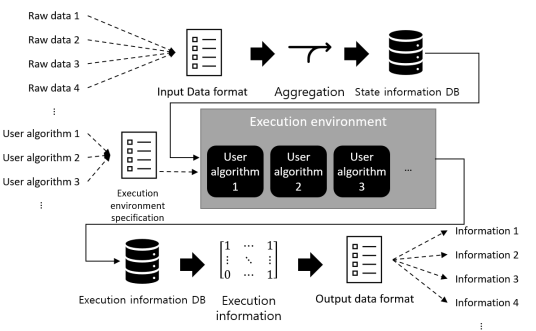
Narayana의 연구[17]에서는 다목적으로 쓰일 수 있는 범용 함수를 정의하고 이들의 조합으로 사용자가 원하는 정보를 추출할 수 있는 패킷 처리 언어를 제안하였다. 해당 언어는 패킷 스트림에서 사용자의 관심사에 속하는 패킷만을 추출하는 filter 연산, 패킷 단위 전처리를 진행하는 map 연산, 패킷들 조건에 따라 그룹 짓는 group 연산, 두 패킷의 정보를 병합하는 zip 연산을 제공하고 사용자는 최종적으로 자신이 원하는 정보가 도출되도록 위 연산을 조합하여 시스템에 입력한다. 이 연구에서는 추가로 평균이나 카운팅과 같이 모든 패킷으로부터 집계(aggregation)되는 정보를 빠르게 처리할 수 있도록 집계 함수를 기존 정보와 새 정보 간 병합이 가능한 linear-in-state 함수로 정의하고 캐시를 이용하여 빠르게 처리한다.

Ⅳ. 사용자 모듈을 위한 분석 요청 정의 언어

이 장에서는 본 연구에서 제안하는 "사용자가 정책 엔진에 자신의 모듈(사용자 알고리즘) 등록을 요청하는 경우 사용하는 API의 인터페이스를 스키마로서 정의하는 수단으로서의 XML 기반 언어인 ARDL(Analysis Request Definition Language)"에 대해 설명한다. 이 스키마를 기반으로 사용자는 정책 엔진에 적용할 분석 알고리즘을 등록할 수 있으며, 제안 스키마가 가지는 플랫폼 중립성을 통해 플랫폼에 상관없이 사용할 수 있다.

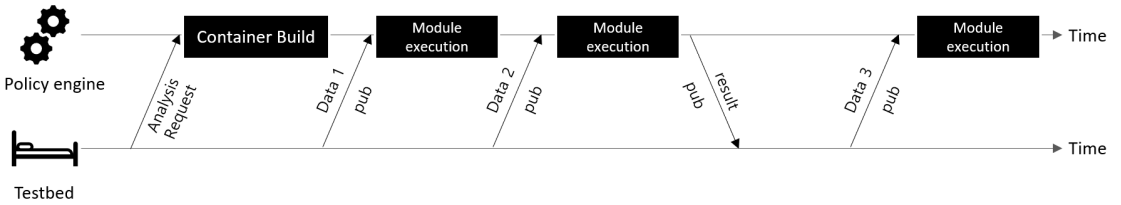
4.1 정책 엔진의 작동 구조

그림 1은 본 논문에서 사용하는 정책 엔진의 구조와 동작을 설명한 그림이다. 통합 테스트베드에서 생성된 성능 지표는 포매팅 과정을 거쳐 정책 엔진의 저장소에 저장된다. 정책 엔진은 입력으로 들어온 성능 지표 데이터에 대해 aggregation 연산을 수행한다. 이는 사용자의 알고리즘에서 통계 정보를 생성하기 위해 매번 모든 데이터를 읽어서 발생하는 오버헤드 대비 낮은 오버헤드만을 가지기 위함이다. 생성된 통계 데이터는 원본 성능 지표와 함께 상태 정보 DB에 기록된다. 사용자는 성능 지표 데이터에 대한 처리 알고리즘을 알고리즘 실행 환경 정보와 함께 기술하여 정책 엔진에 요청한다. 정책 엔진은 환경 정보의 명세에 따라 사용자의 알고리즘을 실행할 환경을 컨테이너에 구현한다. 이후 사용자로부터 데이터가 입력될 때마다 통계 정보와 함께 해당 입력을 알고리즘의 입력으로 주어 알고리즘을 실행시킨다. 알고리즘은 실행 정보를 도출하며 도출된 결과는 정책 엔진 내부의 실행 환경



정책 엔진은 사용자로부터 처리할 데이터와 데이터를 처리하는 방법을 기술한 알고리즘을 받아 데이터를 처리하고 그 결과를 사용자에게 응답한다.

그림 1. 정책 엔진 구조 및 동작 개요
Fig. 1. Overview of policy engine structure and execution



통합 테스트베드는 성능 지표 분석 처리를 요청하기 위해 정책 엔진에 자신이 전송할 성능 지표의 구조와 알고리즘 실행환경을 정의하여 요청한다. 분석 요청을 받은 정책 엔진은 사용자가 요청한 환경을 컨테이너로 구현하여 알고리즘 실행환경을 준비한다. 본 예제에서는 정책 엔진과 통합 테스트베드는 구독-발행 패턴의 네트워크로 연결되어 있다. 통합 테스트베드에서 성능 지표가 발행되면 정책 엔진이 이를 수신하여 사용자의 알고리즘을 실행한다. 사용자가 처리된 결과를 요청하면 사용자가 정의했던 포맷대로 맞춰 알고리즘의 결과를 응답한다.

그림 2. 성능 지표 분석 과정
Fig. 2. Performance indicator processing

정보 DB에 기록된 뒤 사용자의 요청이 있을 때 DB에서 꺼내져 사용자가 요청한 포맷에 맞춰 구조를 바꾼 후 응답으로서 전달된다.

그림 2는 통합 테스트베드가 정책 엔진에 분석을 요청한 뒤 성능 지표가 전달되고 처리된 결과를 받아오는 과정을 보여 주는 예이다. 이 예제에서는 정책 엔진과 통합 테스트베드가 MQTT와 같은 구독-발행 패턴을 사용하는 통신 미들웨어로 연결되어 있다고 가정하였다. 그림 2의 "Analysis Request"는 사용자가 정책 엔진에 새로운 분석을 요청하는 과정 개요를 나타낸다. 이 요청을 위한 메시지에는 사용자가 앞으로 전송할 성능 지표 데이터의 포맷과 결과로 받을 데이터의 포맷이 정의되고 그 데이터를 처리할 알고리즘을 위한 실행환경이 기술된다. 분석 요청 이후 통합 테스트베드에서 자신이 분석 요청을 보낼 때 기술했던 포맷에 맞추어 데이터를 전송하고 이후 결과 요청 메시지를 보내 결과를 확인한다.

4.2 분석 요청 정의 언어 스키마

본 장에서는 사용자가 정책 엔진에 분석을 요청하는 데 사용하는 언어인 Analysis Request Definition Language(ARDL)의 구조를 설명한다. ARDL은 사용자의 플랫폼과 프로그래밍 언어에 중립적인, 즉 어떠한 플랫폼과 언어를 사용하더라도 요청을 보낼 수 있도록 하는 메시지 포맷으로서, 메시지 스키마 정의를 XSD(XML Schema Definition)로 기술한다.

ARDL에서 정의하는 정보는 MQTT 구독, 입력 데이터 포맷과 출력 데이터 포맷, aggregation 정의, 사용자 모듈 실행 환경 정보로, 구성되는 모듈의 등록에 필요한 기본 네 가지 정보와 ARDL 인터페이스의 버전을 기술하는 버전 정보를 더해 총 5개의 정보를 기술한다. 그림 3은 ARDL XML 메시지의 root 엘리먼트의 구조를 XSD 스키마로 정의한 문서이다. 이하의

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:ARDL="namespace/ARDL"
    targetNamespace="namespace/ARDL"
    elementFormDefault="qualified">

    <xs:element name="ARDL">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="version" type="xs:integer" />

                <xs:element name="MQTT" type="ARDL:MQTTInformation" />
                <xs:element name="dataFormat" type="ARDL:DataFormat" />
                <xs:element name="aggregation" type="ARDL:Aggregation" />
                <xs:element name="moduleEnvironment" type="ARDL:ModuleEnvironment"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- named complexTypes define section -->
</xs:schema>
```

ARDL XML 문서의 root 엘리먼트 구조를 XSD 스키마로 정의한다. 네임스페이스로 ARDL 키워드를 사용한다. 스키마에서 정의된 named complexType은 생략되어 표기된 그림이다.

그림 3. ARDL root 엘리먼트 스키마 정의
Fig. 3. The schema for the root element of ARDL
ARDL XML 문서의 root 엘리먼트 구조를 XSD 스키마로

장에서는 각 엘리먼트의 세부 구조를 정의하고 설명한다. ARDL 전체 스키마는 "https://github.com/henry174Ajou/ARDL"에서 확인할 수 있다.

4.2.1 MQTT 구독 정보 스키마

정책 엔진은 구독-발행 메시지 브로커인 MQTT를 통신 미들웨어로 사용하며, 정책 엔진과 통합 테스트베드(사용자)는 이미 MQTT 네트워크에 참여하고 있는 것으로 가정한다. 즉, 통합 테스트베드와 정책 엔진은 각자에게 발행자이자 구독자 관계이다. 성능 지표 데이터를 정책 엔진에 입력할 때는 통합 테스트베드 측이 발행자가 되며, 실행 정보를 받아올 때는 정책 엔진 측이 발행자가 된다. 따라서 통합 테스트베드 사용자는 발신을 위한 topic과 수신을 위한 topic을 정의하여 정책 엔진에게 알려 주어야 한다. 통합 테스트베드는 성능 지표를 발신 topic으로 발행하고 정책 엔진은 이 topic을 구독하여 데이터를 입력받는다. 반대로 모듈의 실행 결과는 정책 엔진이 사용자가 요청한 수신 topic으로 발행하고 통합 테스트베드가 이를 구독하여 수신한다. MQTT에서는 메시지 브로커가 제

```
<xs:element name="MQTT" type="ARDL:MQTTInformation"/>

<xs:complexType name="MQTTInformation">
    <xs:sequence>
        <xs:element name="upstream" type="ARDL:MQTTSubscribe"/>
        <xs:element name="downstream" type="ARDL:MQTTSubscribe"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="MQTTSubscribe">
    <xs:all>
        <xs:element name="topic" type="xs:string"/>
        <xs:element name="QoS" type="xs:integer"/>
    </xs:all>
</xs:complexType>
```

그림 4. MQTT 구독 정보를 기술하는 MQTT 엘리먼트의 XSD 스키마
Fig. 4. The schema of the MQTT element that describes subscription information about MQTT

공할 QoS 수준을 1, 2, 3의 정수로 지정하여 설정할 수 있다. Topic 이름에 더해 추가로 이 정보도 MQTT 구독 정보에 포함된다.

위 내용을 표현하기 위해 MQTT 구독 정보를 나타내는 엘리먼트에 내부 엘리먼트로 upstream 엘리먼트와 downstream 엘리먼트를 기술한다. up/down stream 엘리먼트는 각각 내부에 topic 엘리먼트와 QoS 엘리먼트를 원소로 가진다. upstream 엘리먼트는 통합 테스트베드가 정책 엔진으로 보내는 데이터를 위한 구독 정보를 기술하고 downstream 엘리먼트는 정책 엔진이 보내는 결괏값을 위한 구독 정보를 기술한다. 그림 4는 MQTT 엘리먼트의 XSD 스키마를 보여준다.

4.2.2 데이터 포맷 정의 스키마

“dataFormat” 엘리먼트는 통합 테스트베드가 어떤 구조의 데이터를 정책 엔진으로 보내고, 그 결괏값을 어떤 구조로 받을 것인지를 정의한다. 통합 테스트베드가 보내는 데이터의 포맷 정의는 “input” 엘리먼트에 기술하고 정책 엔진이 반환할 데이터의 포맷은 “output” 엘리먼트에 기술한다. 대부분의 프로그래밍 언어에서 구문분석이 가능하도록 ARDL은 프로그래밍 언어들이 공통으로 지원하는 정수(int), 실수(double), 문자열(string)의 원시 데이터 타입과, 구조체(structure)와 리스트(list)의 복합 데이터 구조만을 지원한다. 통합 테스트베드는 분석 요청 단계에서 포맷을 정의하여 보내고, 이후 실제 데이터를 보낼 때는 정의한 포맷에 맞게 XML 형태로 데이터를 포매팅하여 전송해야 한다.

원시 데이터 타입은 “value” 엘리먼트로 기술한다. value 엘리먼트의 “name” 속성으로 해당 값의 참조 이름을 정의하고 “type”으로 이 값이 정수일지 실수일지 문자열일지 정의한다. value 엘리먼트는 하위 엘리먼트를 가지지 않는다.

구조체 복합 데이터 타입은 여러 원시 데이터 타입 및 다른 복합 데이터 구조를 묶어서 호칭하기 위한 타입이다. “structure” 엘리먼트로 기술하고 “name” 속성에 해당 구조체의 이름을 정의한다. 하위 엘리먼트로 value나 structure 또는 list 엘리먼트를 적어도 한 개 가져야 한다. 내부 엘리먼트의 순서는 고려하지 않으며 정의된 포맷을 따르는 데이터에서도 내부 엘리먼트의 순서에는 의미가 없다.

리스트 복합 데이터 타입은 여러 개의 같은 타입 데이터를 묶어서 호칭하기 위한 타입이다. “list” 엘리먼트로 기술하고 “name” 속성에 해당 리스트의 이름을 정의한다. 하위 엘리먼트로 value, structure, list 중 하나를 정확히 1개만 가져야 한다. list 엘리먼트는 정의된 포맷을 따르는 데이터에서 내부 엘리먼트의 순서가 의미가 있다. 그림 5는 ARDL의 데이터 포맷 정의 부분의 스키마를 보여 준다.

모듈 분석 요청이 끝난 후 통합 테스트베드에서 정책 엔진에 데이터를 보낼 때는 위에서 정의한 포맷에 맞춰 XML 데이터를 보내야 한다. 이때 포맷의 정의된 value, structural, list의 name 속성은 실데이터의

```
<xs:element name="dataFormat" type="ARDL:DataFormat" />

<xs:complexType name="DataFormat">
        <xs:sequence>
            <xs:element name="input" type="ARDL:DataDefinition"/>
            <xs:element name="output" type="ARDL:DataDefinition"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="DataDefinition">
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="value" type="ARDL:PrimitiveValue" />
            <xs:element name="structure" type="ARDL:Structure"/>
            <xs:element name="list" type="ARDL:List"/>
        </xs:choice>
    </xs:complexType>

    <xs:complexType name="PrimitiveValue">
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="type">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="int"/>
                    <xs:enumeration value="double"/>
                    <xs:enumeration value="string"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>

    <xs:complexType name="Structure">
        <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="value" type="ARDL:PrimitiveValue" />
            <xs:element name="structure" type="ARDL:Structure"/>
            <xs:element name="list" type="ARDL:List"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="List">
        <xs:choice>
            <xs:element name="value" type="ARDL:PrimitiveValue" />
            <xs:element name="structure" type="ARDL:Structure"/>
            <xs:element name="list" type="ARDL:List"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>
```

그림 5. 입출력 데이터의 형태를 정의하는 dataFormat 엘리먼트의 XSD 스키마
Fig. 5. The XSD schema for dataFormat element that define input/output data format.

```
<ARDL:input>
    <ARDL:structure name="routingInformation">
        <ARDL:value name="routingDistanceMax" type="double"/>
        <ARDL:value name="routingDiatanceAverage" type="double"/>
        <ARDL:structure name="position">
            <ARDL:value name="x" type="int"/>
            <ARDL:value name="y" type="int"/>
        </ARDL:structure>
        <ARDL:list name="linkList">
            <ARDL:value name="value" type="int"/>
        </ARDL:list>
    </ARDL:structure>
</ARDL:input>
```

(A)

```
<routingInformation>
    <routingDistanceMax>3.5</routingDistanceMax>
    <routingDiatanceAverage>1.5</routingDiatanceAverage>
    <position>
        <x>7</x>
        <y>8</y>
    </position>
    <linkList>
        <value>1</value>
        <value>8</value>
        <value>7</value>
        <value>5</value>
    </linkList>
</routingInformation>
```

(B)

(A) ARDL 데이터 정의 스키마를 따르는 데이터 포맷 정의 예시 (B) 데이터 포맷을 따르는 입력 데이터의 예시

그림 6. 데이터 포맷 정의의 예시와 그 사용 예
Fig. 6. Example of data format definition and it's usage

엘리먼트 이름이 된다. 그림 6의 A는 그림 2의 "Analysis Request" 단계에서 통합 테스트베드가 정책 엔진에 보내는 ARDL 메시지의 입력 데이터 포맷 정의 부분을 예시로 보여 준다. 이후 정책 엔진이 데이터를 실제로 전송할 때는 그림 6의 B처럼 그림 6의 A에 정의한 대로 포맷을 맞춰 보낸다.

4.2.3 Aggregation 정의 스키마

"aggregation" 엘리먼트에서는 사용자 모듈이 정책 엔진에서 계산해 주기를 요청하는 통계 정보를 기술한다. 사용자가 aggregation을 지정하면 정책 엔진은 매 데이터 입력마다 지정된 aggregation을 계산하여 업데이트한다. 이후 aggregation 데이터는 입력 데이터와 함께 사용자 알고리즘의 입력으로 주어진다. 현재의 ARDL 버전 1에서는 sum, average, count 3개의 aggregation을 지원한다. 각각의 aggregation은 대상으로 하는 값에 대해 합을 구하거나 평균을 구하거나 개수를 구한다.

sum aggregation은 "sum" 엘리먼트로, average aggregation은 "average" 엘리먼트로 count aggregation은 "count" 엘리먼트로 기술하며 3개의 엘리먼트 모두 "name" 속성과 "target" 속성을 가진다. name 속성은 계산된 aggregation의 값을 참조할 때 사용될 이름을 지정한다. 알고리즘의 input에 aggregation 값이 입력될 때 name 속성에 지정된 이름으로 입력된다. target 속성은 aggregation을 진행할 데이터의 위치를 기술한다. 데이터의 위치는 Ⅳ.2.2장에서 정의된 입력 데이터 포맷에 따라 aggregation 하고자 하는 값이 있는 위치까지 존재하는 엘리먼트의 name 속성값을 "."을 구분자로 하여 (deliminator

```
<xs:element name="aggregation" type="ARDL:Aggregation" />

<xs:complexType name="Aggregation">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="sum">
            <xs:complexType>
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="target" type="xs:string"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="average">
            <xs:complexType>
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="target" type="xs:string"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="count">
            <xs:complexType>
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="target" type="xs:string"/>
            </xs:complexType>
        </xs:element>
    </xs:choice>
</xs:complexType>
```

(A)

```
<ARDL:aggregation>
    <average name="averageOfMax" target="routingInformation.routingDistanceMax"/>
</ARDL:aggregation>
```

(B)

(A) Aggregation을 정의하는 aggregation 엘리먼트의 XSD 스키마 (B) 그림 6의 routingDistanceMax 값의 평균을 구하는 aggregation을 정의한 예시

그림 7. Aggregation을 정의하는 aggregation 엘리먼트 XSD 스키마와 aggregation 정의 예시
Fig. 7. The schema of aggregation element that defines aggregations and an example of aggregation definition

로 하여) 나열한다. 그림 7의 (A)는 aggregation 엘리먼트의 스키마를 XSD로 정의한 모습을 보여주며 그림 7의 (B)는 그림 6의 데이터 포맷 예시에서 routingDistanceMax의 평균을 계산하는 aggregation을 정의하는 예제이다.

4.2.4 모듈 환경 정의 스키마

사용자 알고리즘이 가지는 다양한 의존성과 요구사항을 해결하기 위해, 알고리즘은 정책 엔진 내부에서 컨테이너 단위로 실행되며, 컨테이너는 알고리즘을 실행하는 데 필요한 환경 및 의존성을 모두 포함하고 있다. 통합 테스트베드에서 정책 엔진으로 컨테이너 이미지를 전달하기 위해 "image manifest"를 전달하는 방식을 사용한다. Image manifest는 컨테이너 이미지 내부의 레이어 구조를 기술하는 양식으로 JSON으로 표현되며 이미지의 레이어 구조와 아키텍처에 대한 정보를 포함한다. ARDL은 image manifest의 양식으로 OCI 표준 사양[18]을 사용하여 정책 엔진과 통합 테스트베드 사용자가 사용하는 컨테이너 런타임에 상관없이 이미지 전달이 가능하게 한다. "enviornment" 엘리먼트 내부에 "OCI" 엘리먼트로 OCI 포맷의 image manifest를 기술한다. 그림 8은 모듈 실행환경

```
<xs:element name="moduleEnviornment" type="ARDL:ModuleEnviornment"/>

<xs:complexType name="ModuleEnviornment">
    <xs:choice>
        <xs:element name="OCI" type="xs:string"/>
    </xs:choice>
</xs:complexType>
```

그림 8. 모듈 실행환경을 컨테이너 이미지로 정의하는 moduleEnvironment의 XSD 스키마
Fig. 8. The XSD schema of moduleEnvironment element that define execution environment for user's module as a container image.

을 정의하는 moduleEnviornment 엘리먼트의 스키마를 XSD로 정의한 것이다. 현재의 ARDL 버전 1에서는 image manifest 포맷을 OCI만을 정의하지만 추후 확장할 수 있다.

Ⅴ. ARDL의 표현력 평가

ARDL에서는 데이터 구조, aggregation, 모듈 환경의 세 가지 데이터에 대해 사용자가 자유롭게 정의할 수 있다. 본 장에서는 세 가지 데이터의 표현에 대한 ARDL의 표현력을 평가한 결과를 제시한다.

5.1 데이터 구조 표현

Ⅱ.1장에서 설명한 것과 같이 웹 서비스의 인터페이스의 명세를 중립적으로 기술하는 언어인 WSDL에서는 클라이언트와 서버 간 주고받을 데이터를 XML 형식으로 정의하고, 정의한 포맷을 WSDL 내에 XSD 스키마로 기술하고 있다. WSDL 수신 후 서비스를 요청할 때 보내지는 실제 데이터는 XSD로 정의된 스키마에 맞게 인코딩하여 보내게 된다. WSDL 1.1 명세서에 따르면 서버와 클라이언트 간 주고받을 XML 데이터는 엘리먼트에 속성(attribute)을 정의하지 않으며, XML 데이터가 배열을 표현할 수 있어야 하고, 타입을 따로 기술하지 않는 any type을 지원해야 한다[9].

ARDL에서도 WSDL과 같이 정책 엔진(서버)과 통합 테스트베트(클라이언트) 간 XML로 메시지를 주고받도록 지정하며 이 XML 메시지의 모든 엘리먼트에는 속성값이 정의되지 않는다. 추가로 ARDL의 “list” 엘리먼트를 통해 순서 있는 배열을 표현할 수 있고, “structure”을 통해 순서 없는 배열 또는 dictionary로 볼 수 있는 구조체를 표현할 수 있다. 다만 ARDL에서는 데이터의 타입을 생략하는 any type을 허용하지는 않지만, string 타입을 지원하기에 타입을 기술하지 않을 필요가 있는 데이터의 경우 문자열로 인코딩하여 보내고 받는 쪽에서 디코딩하여 복원하는 것이 가능하다. 따라서 ARDL에서는 타입 생략을 불허하고 복합 데이터 구조를 표현하는 전용 엘리먼트를 추가함으로써, 범용적으로 XML의 구조를 정의하는 XSD를 사용한 WSDL과 달리, ARDL은 서버와 클라이언트가 주고받는 XML 데이터가 프로그램에서 사용되는 데이터 표현과 가깝게 나타나도록 구조를 정의하고 있다.

5.2 Aggregation 표현

SDN 환경에서는 현재 네트워크의 상태를 모니터링 할 때 programmable 스위치를 이용하여 필요한 성능 수치를 계산해 낼 수 있다. 성능 수치를 계산하는 과정 또는 방법을 기술하기 위한 언어로 Marple[17]이라고 불리는 언어 체계가 연구되었다. Marple에서는 filter, zip, group, map등의 단순하고 기본적인 aggregation을 수행하는 단위 연산을 제공한다. 사용자는 단위 연산을 조합하는 것으로 더 복잡한 aggregation을 기술할 수 있다.

본 논문에서 제안하는 ARDL 버전 1에서는 sum, average, count 등의 aggregation만 지원하여 표현력이 다소 제한적일 수 있으나, 언어 설계의 전제조건에 따라 간단한 aggregation은 정책 엔진에서 미리 계산하고 이후 복잡한 aggregation은 사용자 정의 모듈을 통해 사용자의 요구 사항대로 알고리즘을 실행할 수 있게 하여 다양하고 복잡한 처리가 가능하다.

5.3 모듈 실행환경 표현

Docker는 사실상 표준 (de facto standard) 컨테이너 런타임으로 현재 가장 유명한 컨테이너 런타임 중 하나이다[19,20]. Docker에서는 이미지를 빌드하기 위해 docker file이라는 사용자 친화적인 이미지 빌드 형식을 제공한다. 사용자는 docker file 형식으로 이미지 빌드 과정을 기술하고 docker는 제공된 docker file로부터 이미지를 빌드한다. Docker에서 빌드된 이미지는 docker image manifest 작성되는데 이 형식은 OCI 표준 image manifest와 직접적으로 호환되지는 아니한다. 하지만 buildah[21]와 같은 도구를 사용하여 docker file로부터 OCI 이미지를 빌드한 후 OCI image manifest 포맷으로 ARDL 메시지에 첨부할 수 있다. 따라서 ARDL은 OCI 표준 이미지를 지원하는 것뿐만이 아니라 널리 사용되는 docker 이미지도 약간의 변환을 통해 지원할 수 있다.

5.4 향후 연구

본 연구는 SDN 정책엔진의 사용자 묘듈을 위한 분석 요청 정의 언어의 사전 연구로 이번 연구에서는 언어의 표현력에 대해서 살펴보았다. 제안된 언어의 차별성을 보여 주기 위한 실험은 향후 연구로 남겨 둔다.

ARDL은 Marple[17]과 같은 기존 언어 체계와 달리 가상 환경을 제공하여 사용자 정의 알고리즘을 자유롭게 작성하고 실행시킬 수 있는 것이 특징이다. 따라서 향후 연구에서는 기존 알고리즘을 SDN/정책 엔진에서 실행하기 위해 제공되는 언어의 포맷에 맞게 변환해야 할 때 수정되어야 하는 코드의 양을 기존 기법과 정성적으로 비교할 것이다. 또한 ARDL은 가상

화 환경을 제공하는 컨테이너에 OCI 표준외 다른 제한을 두지 않으므로 여러 컨테이너 런타임에 따른 알고리즘 등록 처리 시간 및 알고리즘 수행 시간을 비교해 볼 수 있다.

Ⅵ. 결 론

본 연구에서는 프로그래밍 언어와 실행환경 등 특정 기술에 종속되지 않는 정책 엔진 구현을 위해 인터페이스 포맷을 위한 메시지 스키마인 ARDL을 설계하고 제안하였다. 본 논문에서 제안한 메시지 스키마는 XML을 사용하여 통합 테스트베드와 정책 엔진 상호 간 서로 사용할 통신 메커니즘과 데이터 포맷을 합의할 수 있도록 지원하고, 사용자 정의 알고리즘의 실행환경을 OCI 표준 명세를 따라 정의하여 프로그래밍 언어와 플랫폼에 중립적인 정책 엔진의 구현이 가능하게 하였다. ARDL은 상호 간 교환할 데이터의 구조를 프로그램의 데이터 구조와 유사하도록 정의하여 다양한 구조의 데이터를 간단히 정의할 수 있었으며, 사용자 정의 알고리즘을 지원하기 위해 OCI 포맷은 물론 사실상 표준(de facto standard)인 docker 포맷을 지원함으로써 다양한 컨테이너의 정보를 전달할 수 있었다.

ARDL을 통해 통합 테스트베드와 정책 엔진 간 종속 수준을 낮추고 특정 컨테이너 환경에 종속되지 않게 함으로써 쉬운 유지 보수와 컴포넌트 교체가 가능할 것으로 기대한다. 향후 연구에서는 MQTT 이외의 더 다양한 통신 프로토콜로 확장하는 것을 목표로 해 볼 수 있다.

References

[1] A. R. Pushkar, "Network-centric warfare: Its origin and future," *Accel. World's Res.*, vol. 124, pp. 28-35, 1998.

[2] S. H. Lee, S. Lee, H. Song, and H. S. Lee, "Wireless sensor network design for tactical military applications: Remote large-scale environments," in *MILCOM 2009*, pp. 1-7, Boston, USA, October 2009. (https://doi.org/10.1109/MILCOM.2009.5379900)

[3] B. Roh, H. Lee, and C. Lee, "Design and implementation of riverbed modeler based multi-domain integrated tactical SDN testbed," in *ICNGC2021*, pp. 333-335, Jeju, Korea, October 2021.

[4] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, pp. 279-291, 2011. (https://doi.org/10.1145/2034574.2034812)

[5] *SWIG*(2008), Retrieved Mar., 30, 2022, from http://www.swig.org/index.php.

[6] K. Rakowski, *Learning Apache Thrift*, Packt Publishing Ltd., 2015,

[7] M. Slee, A. Agarwal, and M. Kwiatkowski, "*Thrift: Scalable cross-language services implementation*," *Facebook white paper*, vol. 5, pp. 127, 2007.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan, *Extensible markup language (XML) 1.0*, W3C Recommendation, Oct. 2000.

[9] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, and others, *Web services description language (WSDL) 1.1*, Citeseer: 2001.

[10] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, pp. 86-93, March. 2002. (https://doi.org/10.1109/4236.991449)

[11] *MQ Telemetry Transport*(2022), Retrieved March. 30. 2022, from http://mqtt.org.

[12] JD. Case, *Simple Network Management Protocol (SNMP)*(1989), Retrieved September, 2, 2022, from https://www.rfc-editor.org/rfc/rfc1098

[13] H. T. Ju, J. H. Yoon, and J. W. Hong, "SNMP-XML translator and gateway for XML-based network managemet," *KNOM Rev.*, vol. 5, no. 1, pp. 1-17, 2002.

[14] *Cisco IOS NetFlow*, Retrieved March. 30. 2022, from https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[15] *Configuring SPAN*, Retrieved September. 2. 2022, from https://www.cisco.com/c/en/us/td/d

ocs/switches/datacenter/nexus5000/sw/configuration/guide/cli/CLIConfigurationGuide/Span.pdf.

[16] G. Cormode and S. Muthukrishnan, “An improved data stream summary: The count-min sketch and its applications,” *J. Algorithms*, vol. 55, p. 58-75, 2005. (https://doi.org/10.1016/j.jalgor.2003.12.001)

[17] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, “Language-directed hardware design for network performance monitoring,” in *Proc. Conf. ACM Special Interest Group on Data Commun.*, pp. 85-98, Los Angeles, USA, August 2017. (https://doi.org/10.1145/3098822.3098829)

[18] *OCI Image Format Specification(2022)*, Retrieved September. 2. 2022, from https://github.com/opencontainers/image-spec.

[19] S. S. Tadesse, F. Malandrino, and C. Chiasserini, “Energy consumption measurements in docker,” in *2017 IEEE 41st Annu. COMPSAC*, pp. 272-273, Turin, Italy, July 2017. (https://doi.org/10.1109/COMPSAC.2017.117)

[20] A. Farshteindiker and R. Puzis, “Leadership hijacking in docker swarm and its consequences,” *Entropy*, vol. 23, p. 914, May 2021. (https://doi.org/10.3390/e23070914)

[21] *Buildah,* Retrieved March. 30. 2022, from https://buildah.io/.

이 승 준 (Seungjun Lee)

2021년 2월 : 아주대학교 소프트웨어및컴퓨터공학 전공 학사 졸업

2021 3월~현재 : 아주대학교 인공지능학과 석사과정

<관심분야> 분산 시스템, Federated learning

[ORCID:0000-0003-0385-0724]

윤 대 건 (Daegun Yoon)

2018년 8월 : 아주대학교 소프트웨어학과 학사

2018년 9월~현재 : 아주대학교 인공지능학과 석박사통합과정

<관심분야> 분산 딥 러닝, GPU 기반 그래프 알고리즘

[ORCID:0000-0002-7520-1144]

오 상 윤 (Sangyoon Oh)

2006년 : 인디애나대학교 컴퓨터공학과 박사

2006년~2007년 : SK 텔레콤

2007년~현재 : 아주대학교 소프트웨어학과 교수

<관심분야> 분산/병렬 컴퓨팅, HPC, 빅데이터처리, 클라우드

[ORCID:0000-0001-5854-149X]