대규모 데이터 센터 네트워크를 위한 OpenFlow 기반 장애 복구 방법에 관한 연구

리건 O,1, 현종환 2, 유재형 2, 홍원기 2

1 포항공과대학교 정보전자융합공학부

2 포항공과대학교 컴퓨터공학과

{gunine, noraki, styoo, jwkhong}@postech.ac.kr

요 약

최근 들어, 다양한 형태의 모바일 및 클라우드 서비스들이 인기를 얻게 되면서 이런 서비스들을 제공하는 데이터 센터 네트워크들의 (DCNs) 규모 또한 급증하고 있다. DCN 은 IP 기반 네트워크와는 다른 기술 요구사항들을 가지고 있어 IP 기반 프로토콜을 바로 DCN 에 적용하는 것은 네트워크 운용 및 관리 측면에서 상당히 비효율적이다. 따라서 본 논문에서는 이에 대한 해결책으로 대규모 DCN 을 위한 확장 가능한 장애 복구 방법을 제안한다. DCN 의 확장성을 고려하여 지역 근사해(local sub-optimal solution)를 찾는 방식으로 대안 경로를 계산하였다. 또한 대상 네트워크의 규모에 상관없이 단일 링크 장애에 대하여 단 3 개 스위치의 플로우 테이블을 수정하는 것만으로 복구가 가능하도록 설계하였다. 제안한 장애 복구 방법은 Fat-Tree[1] 토폴로지를 고려하여 설계되었고 OpenFlow 기술을 사용하여 구현되었다. 실험을 통하여 제안한 방법이 만 개 이상의 호스트들로 구성된 대규모 DCN 에서도 효율적으로 작동하고 있음을 검증하였다.

1. 서론

클라우드 컴퓨팅 기술이 성숙되고 다양한 형태의 클라우드 기반 서비스들 출현함에 따라 현 DCN 들의 규모가 급증하고 있다. DCN 은 높은 확장성, 집중적 관리 및 DCN 에 최적화 된 사용자 정의 프로토콜 등 일반 IP 네트워크와는 다른 특성들을 가지고 있으며 이런 특성들을 잘 고려하여 DCN 을 위한 효율적인 네트워크 운용 및 관리 프로토콜들을 설계하여야 한다.

장애 복구는 가장 중요한 네트워크 관리 방법으로, 현 IP 네트워크에서는 Open Shortest Path First (OSPF) 프로토콜을 사용하여 장애를 복구하고 있다. OSPF 는 장애가 발생한 지점을 자동으로 탐지하고 장애 정보를 인접한 모든 스위치들에 순차적으로 통지하는 방식으로 구현되었다. 장애 정보는 제어 메시지에 포함되어 브로드캐스트 방식으로 전달된다. 이 때문에 대규모 DCN 에서 단일 장애라도 대량의 제어 메시지를 발생시켜 네트워크 혼잡을 야기시키고 네트워크 규모가 커질수록 장애를 복구하는데 더 많은 시간을 필요로 한다. 따라서 현 OSPF 기반 장애 복구 방법을 대규모 DCN 에 직접 적용하는 것은 상당히 비효율적이며 대규모 DCN 를 위한 확장성 높은 장애 복구 방법의 개발 및 적용이 시급하다.

DCN 에서 IP 기반 프로토콜들의 한계를 극복하고자 최근에는 소프트웨어 정의 네트워크 (Software-Defined Network, SDN) 기술이 급부상하고 있다. 이 기술을 활용하면 중앙집중적으로 네트워크 관리를 유연하게 할 수 있다. SDN 은 새로운 네트워크 패러다임으로 제어 평면과 데이터 평면의 분리를 통해 기존 IP 기반 프로토콜들로는 실현하기 어려웠거나 불가능했던 유연한 네트워크 관리를 가능하게 한다. 분리된 제어 평면 기능들은 중앙집중화된 컨트롤러에 위치하고 데이터 평면은 데이터 전송 기능만 수행한다. OpenFlow [2]는 SDN 의 하나의 구현 기술로 스탠포드대학에서 처음으로 설계 및 개발되었고 이미 Google 과 같은 클라우드 벤더들은 이 기술을 사용하여 DCN 를 위한 맞춤형 통신 프로토콜들과 네트워크 관리 기능들을 설계 및 구현하여 사용하고 있다 [3]. OpenFlow 기술을 사용함으로써 앞서 소개한 DCN 의 집중적 관리 및 맞춤형 토폴로지 및 프로토콜 설계 등 두 가지 특성은 만족시킬 수 있지만 이 기술 역시 중앙집중적인 관리 방식을 채택하고 있어 확장성 문제를 가지고 있다.

OpenFlow 를 사용한 장애 복구 시간은 크게 장애 탐지, 대안 경로 계산 및 플로우 설정(Flow Setup)을 하는데 드는 시간의 합으로 계산된다. 현존하는 Open-Flow 기반 장애복구 방법은 대안경로를 계산 함에 있어 전역 최적해 (global optimal solution)를 찾는 방식을 사용하고 있어 네트워크 규모가 커질 경우 대안경로의 경우수도 많아져 이를 계산하는데 걸리는 시간 또한 증가한다. 이는 결과적으로 확장성 문제를 야기하게 되는데 이와 달리 본고에서 제안한 장애 복구 방법은 지역 근사해를 계산하는 것으로 계산시간복잡도를 줄였고, 우회경로까지 고려하여

플로우 테이블 (Flow Table) 수정이 필요한 스위치의 개수를 최소화하였다. 제안한 방법은 장애가 발생한 지점으로부터 목적지까지의 최단 경로를 찾아주는 전역 최적화 방식에 비하여 최적성 (optimality) 측면에서 다소 저하를 가져오지만 계산시간이 짧고 플로우 테이블 수정이 필요한 스위치 수가 적어 대규모 DCN 를 위한 대안 경로 탐색에 적합하다.

본 논문은 다음과 같은 순서로 구성되었다. 2 장에서는 OpenFlow 기반 장애 복구와 관련된 연구들에 대하여 소개하고, 3 장에서는 제안한 장애 복구 방법에 대한 자세한 설계 과정을 보인다. 4 장에서는 3 장에서 소개한 장애 복구 방법의 구현 아키텍처 및 이를 바탕으로 구축한 테스트베드를 소개한다. 5 장에서는 구현한 장애 복구 방법을 4 장에서 소개한 테스트베드 상에서 시뮬레이션 방식으로 검증한 결과를 보이고 6 장에서는 결론을 내리고 향후 연구 계획에 대해 기술한다.

2. 관련연구

장애 복구 과정 중 컨트롤러의 개입 여부에 따라 OpenFlow 기반 장애 복구 기법을: 1) Restoration 및 2) Protection 두 가지 종류로 구분할 수 있다. Restoration 기법은, OpenFlow 스위치에서 링크 장애를 탐지할 경우 장애 탐지 메시지가 컨트롤러에 전송되고 컨트롤러는 순차적으로 플로우 설정 메시지를 계산된 대안 경로상에 위치한 스위치들로 전달한다. 이 과정을 통하여 장애로 인하여 영향 받은 Flow(플로우)들을 대안 경로로 리디렉션 (redirection) 한다. 플로우 설정 메시지의 전송이 순차적으로 진행되기 때문에 새로 계산된 경로에 더 많은 스위치가 포함될수록 플로우 설정 메시지 개수가 늘어나고 이는 결과적으로 장애 복구 시간의 증가로 이어진다. Protection 기법은, OpenFlow 스위치에서 링크 장애가 탐지될 경우 스위치는 곧바로 해당 장애로 인하여 영향을 받는 모든 플로우들을 작업 경로(working path) 로부터 백업 경로(backup path)로 리디렉션한다. Protection 기법의 장점은 장애 복구를 위하여 스위치들이 컨트롤러와 통신할 필요가 없기 때문에 상대적으로 적은 장애 복구 시간을 보이는 것이다. 하지만 이 방법을 사용하기 위하여 미리 정적인 방식으로 백업 경로를 각각의 OpenFlow 스위치에 설정해 주어야 하고, 동적으로 바뀌는 네트워크 토폴로지 변화에 빠르게 대응할 수 없다는 단점을 가지고 있다. 또한 스위치에서 전체 네트워크 상태를 알 수 없기 때문에 백업 경로의 설정순서에 따라 경로를 할당하여 네트워크의 특정 구간에 혼잡을 유발할 수도 있다. Restoration 기법을 사용하면 Protection 기법에서 발생할 수 있는 문제들로부터 자유로우며 전체 네트워크의 트래픽 로드에 근거하여 대안 경로를 계산할 수 있어 네트워크의 성능 저하를 막을 수 있다.

Protection 의 이런 문제점들 때문에 많은 연구들에서는 Restoration 의 장애 복구 시간을 줄이는데 초점을 맞추어 연구를 진행하고 있다. 본 연구에서도 Restoration 을 참조 모델로 삼아 확장성 있고 효율적인 장애 복구 방법을 제안한다. Restoration 의 장애 복구 시간은 주로 OpenFlow 스위치와 컨트롤러 사이에서 오가는 플로우 설정 메시지의 개수에 따라 결정되며, 장애 복구 시간을 줄이는데 있어 이 메시지 개수를 줄이는 것이 핵심이다. 논문 [4][5]에서 저자들은 장애를 탐지한 스위치에만 새로 계산된 전체 대안 경로 정보를 포함시켜 전송하는 것으로 스위치와 컨트롤러 사이 플로우 설정 메시지 개수를 최소화하였다. 이 방법들의 공통된 문제점은 OpenFlow 에서 표준에서 지정한 일부 필드([4]에서는 pad 필드, [5]에서는 이더넷 MAC 필드)들을 무작위로 사용하고 있어 현 OpenFlow 표준 프로토콜과 호환되지 않는다는 것이다. 이런 방법들과는 달리 본 논문에서는 장애가 발생한 스위치 주변에서 우회경로들을 검색하여 플로우를 할당하는 방식으로 수정해야 할 OpenFlow 스위치의 개수를 3 개로 최소화하였고 이를 통해 플로우 설정하는데 소요되는 시간을 대폭 줄였다. 또한 우회경로를 계산함에 있어 전역 최적해가 아닌 지역 근사해를 구하는 방식으로 대규모 DCN 에서도 신속하게 장애를 복구할 수 있도록 하였다.

3. 장애 복구 방법의 설계

Restoration 기법을 사용할 경우 장애 복구 시간은 주로 장애 탐지, 경로 계산 및 플로우 설정 시간에 의하여 결정된다. OpenFlow 기반 장애 탐지는 주로 토폴로지 탐색 프로토콜을 사용하여 실현하며 프로토콜의 종류에 따라 장애를 탐지하는데 상당히 큰 시간 차이를 보인다. 따라서 본 연구에서는 대안 경로 계산 및 플로우를 설정하는데 걸리는 시간을 줄이는 것에 초점을 맞추어 연구를 진행하였다.

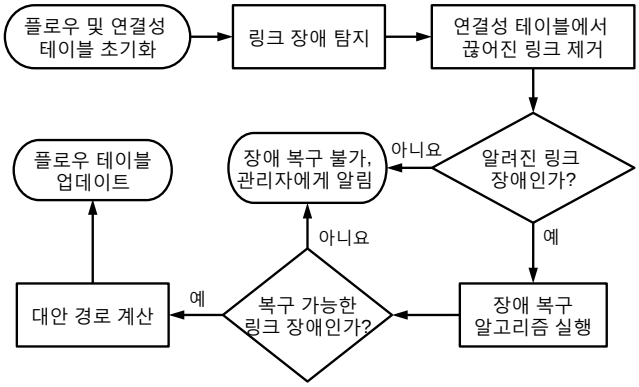

그림 1 장애 복구 동작 과정

그림 1 은 본 연구에서 제안한 장애 복구

방법의 전체 동작 과정을 도식화하고 있다. 초기 단계에서 플로우 테이블에 기본 라우팅 정보를 입력하여 라우팅을 수행하도록 하고, 스위치간 연결 정보를 관리하는 연결성 테이블 (Connectivity Table)을 초기화한다. 이후, OpenFlow 컨트롤러의 토폴로지 탐색 프로토콜을 사용하여 링크 장애를 탐지하고 장애가 탐지될 경우 끊어진 링크를 연결성 테이블에서 제거한다. 링크 장애가 알려진 링크 장애일 경우 장애 종류를 표시하는 매개변수와 함께 해당 링크에 연결된 스위치들 정보를 장애 복구 알고리즘의 입력으로 설정하고 실행한다. 너무 많은 링크가 끊어진 경우 비록 알려진 링크 장애일지라도 유휴 링크 자원으로는 복구가 불가능할 수 있으며 이때 해당 링크 장애는 복구가 불가능한 것으로 판단하여 네트워크 관리자에게 경고 메시지 형태로 발송된다. 복구 가능한 링크 장애로 판단된 경우 대안 경로를 계산하고 결과를 플로우 설정 메시지로 전환 후 관련된 스위치들에 전송한다. 스위치들은 플로우 설정 메시지를 받으면 플로우 테이블에 해당 항목을 업데이트하고 이를 통하여 그 동안 버퍼에 쌓인 패킷들과 새로 도착하는 패킷들은 새로 계산된 대안경로로 전달된다.

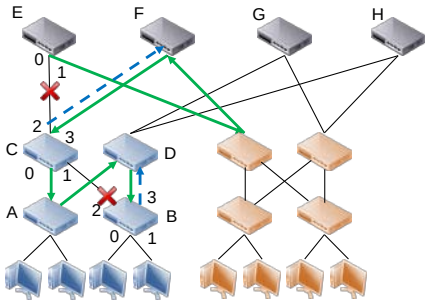

그림 2 링크 장애 복구 과정 예시

본 연구에서는 여러 DCN 토폴로지들 중에서 Fat-Tree 를 참조 토폴로지로 선정하여 장애복구 방법을 설계하였다. Fat-Tree 는 저렴한 DCN 구축 비용, 다수의 최소 비용 대안 경로 제공 및 SDN 을 활용한 쉬운 구현 등의 장점을 모두 지닌다. Fat-Tree 정적 라우팅은 2 계위 플로우 테이블(two-level Flow Table)을 사용하여, 상위 스위치에서 하위 스위치로 전달되는 다운링크(downlink) 플로우에 대해서는 상위 레벨 테이블의 prefix 엔트리에 매치시켜 전달하고, 하위 스위치에서 상위 스위치로 전달되는 업링크(uplink) 플로우에 대해서는 하위 레벨 테이블의 suffix 엔트리에 매치시켜 전달한다. Fat-Tree 의 정적 라우팅은 OpenFlow 1.0 의 priority 필드를 활용하면 여러 개의 플로우 테이블들을 사용하지 않고도 쉽게 구현 가능하다.

지역 근사해 방식의 한가지 가정은 링크 장애가 발생하였을 때 해당 링크와 연결된 스위치들이 항상 주변 스위치들의 도움을 받아 장애가 발생한 링크에 할당되었던 플로우를 다른 경로로 이전할 수 있다는 것이다. 즉, 다른 스위치들이 항상 전달받은 플로우들을 목적지까지 정상적으로 전달할 수 있기 때문에, 장애가 발생한 지점에서 정상적으로 작동하는 임의의 스위치에 플로우를 전달하면 해당 플로우는 항상 목적지까지 안전하게 도달할 수 있다.

DCN 에서 모든 링크들은 양방향성을 가지고 있어 단일 링크 장애에 대해 업링크 및 다운링크를 모두 복구해야 한다. 업링크 라우팅은 플로우 테이블의 suffix 엔트리들을 매칭하고 플로우를 여러 경로를 통하여 최상위 스위치로 전달해준다. 다운링크 라우팅은 플로우 테이블의 prefix 엔트리들을 매칭하고 상위 스위치로부터 호스트까지 하나의 경로만을 사용하여 전달해주는 특징이 있다. 따라서 업링크에서 발생한 장애는 임의의 상위 스위치로 플로우들을 리디렉션하는 것으로 쉽게 복구가 가능하고, 다운링크에서 발생한 장애는 별도의 중간 스위치들을 경유한 우회경로를 사용해야만 복구가 가능하다. 예를 들어, 그림 2 에서 스위치 B 와 C 사이의 링크가 끊어졌을 경우 B 는 C 로 향하는 모든 플로우들을 D 로 리디렉션(점선으로 표시)하여 업링크 장애를 복구한다. 이 과정은 B 의 플로우 테이블의 특정 플로우 엔트리를 업데이트하는 방식으로 구현한다. 또한 C 는 B 로 향하는 플로우들을 A 를 거쳐 D 로 전달해야만 다운링크 장애를 복구할 수 있어 C → A → D → B 와 같은 우회 경로(실선으로 표시)를 거친다. C 에서 B 로 향하는 모든 플로우들을 A 로 리디렉션하는 것은 C 의 플로우 테이블을 업데이트하는 방식으로 구현 가능하다. 목적지가 B 인 플로우들만 A 에서 D 로 리디렉션하는 것은 A 로 들어오는 목적지가 B 가 아닌 플로우들과 구분하여 처리해야 하기 때문에 A 의 플로우 테이블에 새로운 플로우 엔트리를 추가하는 방식으로 구현할 수 있다. 결과적으로 단일 링크를 복구하기 위하여 총 3 개 스위치에 대한 플로우 테이블 수정이 필요하고 다운링크 복구는 업링크 복구보다 하나의 플로우 설정 메시지가 더 필요하게 된다.

제안한 장애 복구 알고리즘을 알고리즘 1 에서 Pseudo Code 로 나타내었다. 이 알고리즘은 아래와 같은 4 개의 매개변수들을 입력으로 받고 있다.

- **연결성 메트릭스 테이블 (*conn_matrix*):** 전체 토폴로지의 연결 상태를 관리한다. 엔트리는 key, value 자료 구조를 가지고 있으며 key 는 스위치 ID, value 는 포트 엔트리로 구성된다. 각 엔트리는 *conn_matrix(switch_id)*와 같이 스위치 ID 로 조회가 가능하다. 포트 엔트리는 활성화된 포트 번호와 해당 포트와 연결된 스위치 ID 로 구성되고 *(port, sid)*로 표현한다.
- **트래픽 통계 테이블 (*traffic_stat*):** 스위치 포트 별 입출력 패킷 통계정보를 유지 및 관리한다.

- **링크 장애 발생 상위 스위치 (S_h):** 장애가 발생한 링크의 상위에 위치한 스위치를 나타낸다. 예를 들어, 그림 2 에서 B, C 사이 링크 장애일 경우, S_h는 스위치 C, S_l은 스위치 B 로 간주할 수 있다. 스위치는 스위치 아이디 ($S_h.id$)와 링크 장애로 다운된 포트 ($S_h.port$) 등 두 가지 속성을 가진다.
- **링크 장애 발생 하위 스위치 (S_l):** S_h와 유사하며, 장애가 발생한 링크의 하위 스위치 정보를 저장한다는 차이점이 있다.

알고리즘 1: Scalable failover

```
1  Remove (port : sid) pair from conn_matrix(S_h.id) entry
   by specifying downed port number S_h.port;
2  Remove (port : sid) pair from conn_matrix(S_l.id) entry by
   specifying downed port number S_l.port;
3  for (port : sid) ∈ conn_matrix(S_h.id) do
4      if conn_matrix(S_l.id) ⋂ conn_matrix(sid) ≠ φ then
5          isRecoverable ← TRUE; break;
6  if isRecoverable = FALSE then Exit(); ;
7  traffic_stat' ← traffic_stat;
   /* Fix uplink failure                                  */
8  while obtain the least loaded (port_up : sid) from
   traffic_stat'(S_l.id) do
9      if conn_matrix(sid) contains S_l.id then
10         UpdateFlowTable(S_l.id, S_l.port, port_up);
11         break;
12     Remove (port : sid) from traffic_stat'(S_l.id);
13 traffic_stat' ← traffic_stat;
   /* Fix downlink failure                                */
14 while obtain the least loaded (port_down : sid_inter_l) from
   traffic_stat'(S_h.id) do
15     if (port_up : sid_inter_h) ←
       conn_matrix(sid) ⋂ conn_matrix(S_l.id) ≠ φ then
16         AddFlowEntry(sid_inter_l, S_l.id, port_up, prefix);
17         UpdateFlowTable(S_h.id, S_h.port, port_down);
18         break;
19     Remove (port_down : sid_inter_l) from
       traffic_stat'(S_h.id);
```

알고리즘은 아래와 같은 네 가지 단계에 거쳐서 실행된다. 1) 다운된 포트 번호를 연결성 테이블에서 제거 (라인 1-2); 2) 정상인 링크들로 복구 가능한지 여부 체크 (라인 3-6); 3) 업링크 장애 복구 시도 (라인 8-12); 및 4) 다운링크 장애 복구 시도 (라인 14-19).

4. 장애 복구 방법의 구현

우리는 제안한 장애 복구 방법을 OpenFlow 애플리케이션 형태로 구현하였고 이를 장애 복구 관리자로 명명하였다 (그림 3 참조). 장애 복구 관리자는 OpenFlow 컨트롤러에서 제공하는 Northbound API 를 사용하여 전체 네트워크 상태를 모니터링하고 장애가 발생할 경우 두 스위치들 사이 대안 경로를 계산하며 계산된 결과를 관련된 스위치들의 플로우 테이블에 업데이트하는 등 일련의 과정들을 통하여 장애를 복구한다. 장애 복구 관리자는 다섯 가지 컴포넌트들로 구성되었다.

- **테이블 초기화 모듈:** 모든 스위치들의 플로우 테이블, 캐싱 목적으로 컨트롤러에서 생성한 내부 플로우 테이블 및 연결성 테이블들을 초기화 해준다.
- **링크 장애 탐지기:** OpenFlow 스위치들로부터 발생되는 포트다운 메시지를 실시간으로 모니터링하여 링크 장애 종류를 판단하고 장애가 발생한 링크와 연결된 스위치 포트를 연결성 테이블에서 제거해준다.
- **트래픽 통계 수집기:** 주기적으로 각 Open-Flow 스위치들에 질의하여 포트 별 트래픽 통계 수치를 수집하고 트래픽 통계 테이블에 저장한다.
- **경로 계산기:** 트래픽 통계 테이블과 연결성 테이블을 참조하여 장애가 발생한 스위치로부터 인접한 hop 의 스위치들로의 대안 경로를 계산한다. 3 장에서 제안한 장애 복구 알고리즘은 이 컴포넌트에 구현되어 있다.
- **플로우 테이블 갱신기:** 플로우 설정 메시지를 Open-Flow 스위치로 전송하여 장애로 인해 차단된 플로우를 새로 계산된 대안경로로 리디렉션 해준다.

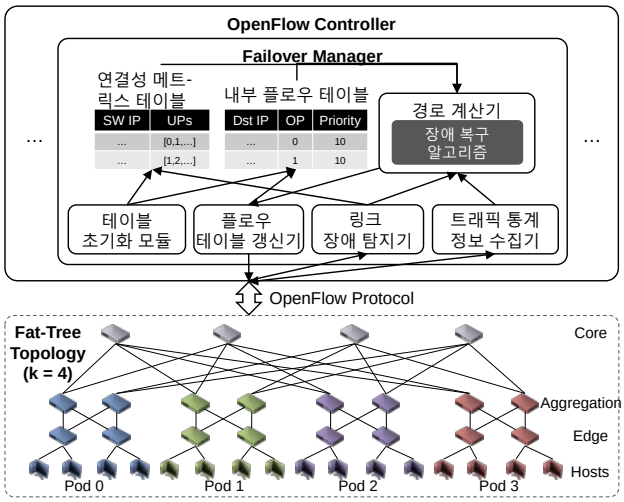

그림 3 장애 복구 관리자 구현 아키텍처

DCN 환경과 근접한 테스트베드를 그림 3 과 같이 구축하였다. Quad-Core 3.47GHz Intel Xeon CPU, 48GB DDR RAM 으로 구성된 고성능 서버를 사용하여 하드웨어 환경을 구축하였고 DCN 은 Mininet 네트워크 에뮬레이션 툴, SDN 컨트롤러는 Floodlight 을 사용하여 소프트웨어 환경을 구축하였다. Mininet 은 하나의 서버에 가상의 스위치 및 호스트들을 생성하여 임의의 네트워크 토폴로지를 손쉽게 구성할 수 있는 에뮬레이션 툴로써, 스위치는 오픈 소스인 Open vSwitch 를 사용하여 가상화하고, 호스트들은 프로세스 형태로 가상화한다. Open vSwitch 는 OpenFlow 표준 프로토콜을 지원하며 본 논문에서 사용한 Floodlight SDN 컨트롤러와 완벽히 호환된다. 본

연구에서는 Mininet 을 사용하여 Fat-Tree 기반 가상 DCN 네트워크를 구축하였고 이를 장애 복구 방법의 성능을 검증하는데 활용하였다.

5. 실험 및 검증

제안한 장애 복구 알고리즘의 확장성을 검증하기 위하여 4 장에서 소개한 테스트베드를 활용하여 실험을 진행하였다. 실험의 목적은 DCN 에서 단일 링크 장애가 발생하였을 경우 대안 경로를 계산하는데 걸리는 시간과 DCN 규모 사이의 상관관계를 살펴보고 이를 통하여 대안 경로 계산 과정의 확장성을 검증하는 것이다. 우리는 Fat-Tree 를 참조 토폴로지로 사용하였고 Fat-Tree 의 규모를 가늠하는 지표인 k 값을 4 (16 개 호스트들을 수용)부터 36 (11,664 개 호스트들을 수용)까지 순차적으로 증가시키면서 대안 경로 계산시간의 증가 추이를 살펴보았다. 비교를 위해 논문[6]에서 제안한 발견적 해결방법 (Heuristic Approach) 기반의 Traffic Engineering (TE) 알고리즘 - Bin Packing 을 선정하여 본 논문에서 제안한 대안 경로 계산시간과 비교하였다. Bin Packing 알고리즘은 장애가 발생한 링크를 토폴로지에서 제거한 후 얻어진 토폴로지에 대하여 트래픽 부하를 분산시키는 로직을 수행하는 알고리즘으로 본 논문에서 제안한 장애 복구 방법과 유사한 기능을 수행한다.

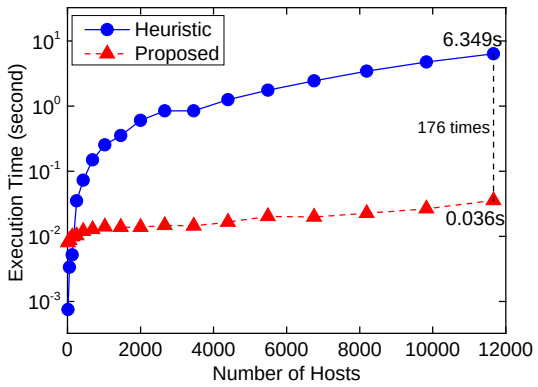

그림 4 Bin Packing Heuristic 과 제안한 장애 복구 방법의 대안 경로 계산 시간 비교

그림 4 는 Bin Packing 알고리즘의 수행시간과 본 논문에서 제안한 장애 복구 방법의 대안 경로 계산시간을 비교한 결과를 보여주고 있다. Bin Packing 알고리즘은 발견적 해결방법을 기반으로 하고 있어 소규모 (128 개 호스트들로 구성)의 DCN 에서는 더 짧은 계산 시간을 보이고 있지만 DCN 규모가 커질수록 계산 시간이 급속히 증가한다. 하지만 제안한 방법의 계산 시간 증가 추이는 가파르지 않으며 11,664 개 호스트를 수용하는 DCN 내에서 대안 경로를 계산하는데 36 ms 가 소요되어 Bin Packing 알고리즘의 계산시간보다 176 배 빠르다. 또한 Mininet 환경에서 플로우 설정 시간을 측정한 결과, 업링크와 다운링크를 복구하는 데 각각 15 ms, 30 ms 씩 걸려 대안 경로 계산 시간까지 고려하면 11,664 개의 호스트로 이루어진 대규모 DCN 에서 단일 링크의 장애 복구에 총 66 ms 가 소요되었다.

6. 결론 및 향후 연구

본 연구에서는 대규모 DCN 을 위한 효율적이고 확장성 높은 장애 복구 방법을 제안하였다. 제안된 장애 복구 방법은 대안 경로를 계산함에 있어 지역 근사해를 찾는 방식으로 확장성을 높였고, 장애가 발생한 링크의 주변 스위치들에만 플로우 설정 메시지를 발송하여 장애 복구에 소요되는 시간을 최소화하였다. 제안된 장애 복구 방법의 타당성을 검증하기 위하여 이를 Floodlight 애플리케이션 형태로 구현 및 Mininet 에 배포하여 실험을 진행하였다. 실험 결과, 11,664 개의 호스트로 이루어진 가상 DCN 환경에서 단일 링크 장애를 복구하는 데 66 ms 가 소요됨을 보여 제안된 방법의 확장성 및 효율성을 검증하였다. 향후 연구로는 Fat-Tree 가 아닌 다른 범용 DCN 토폴로지에서 링크 장애뿐만 아니라 스위치 장애까지 고려한 장애 복구 방법을 개발하고 이에 대한 성능분석을 진행할 계획이다.

7. 참고 문헌

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in ACM SIGCOMM Computer Communication Review, vol. 38, no. 4, 2008, pp. 63–74.

[2] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “OpenFlow: Meeting carrier-grade recovery requirements,” Computer Communications, 2012.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu et al., “B4: Experience with a globally-deployed software defined WAN,” in Proceedings of the ACM SIGCOMM, 2013, pp. 3–14.

[4] S. Kim, J.-M. Kang, S.-s. Seo, and J. W.-K. Hong, “A cognitive model-based approach for autonomic fault management in OpenFlow networks,” IJNM, vol. 23, no. 6, pp. 383–401, 2013.

[5] R. Ramos, M. Martinello, and C. Rothenberg, “Data center fault-tolerant routing and forwarding: An approach based on encoded paths,” in Latin-American Symposium on Dependable Computing, 2013.

[6] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, “ElasticTree: Saving energy in data center networks,” in Proc. 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10), pp. 1-16, Apr. 28-30, 2010.