

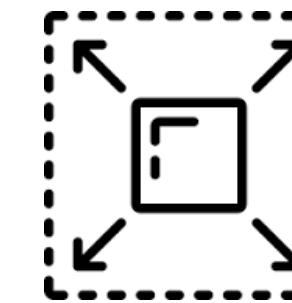
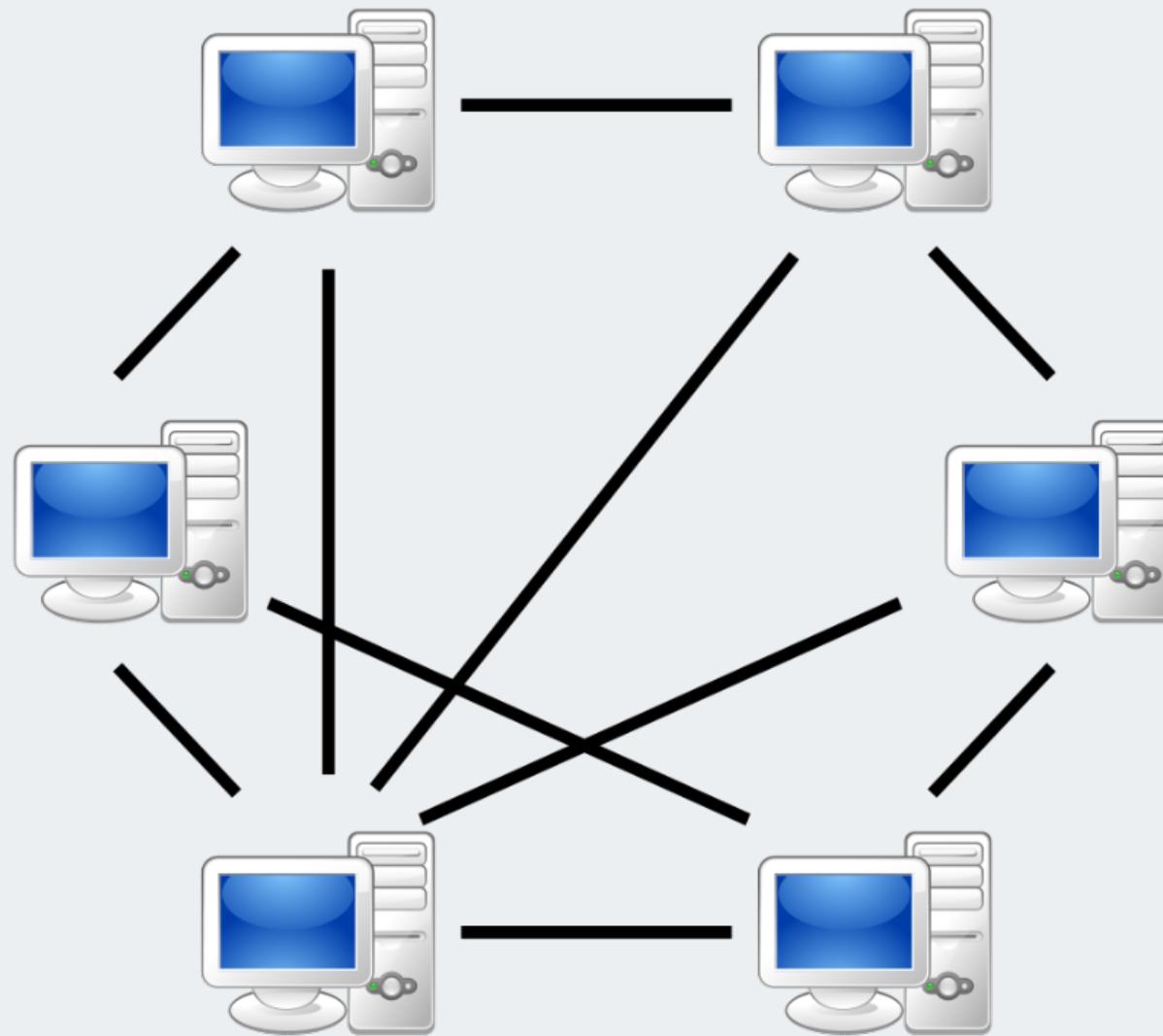
Unstructured P2P network

HYBRID 노드 탐색 알고리즘

사이버보안학과 202220664
허한빈



peer-to-peer network



Scalability

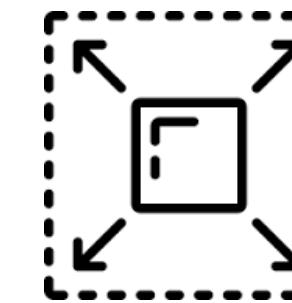
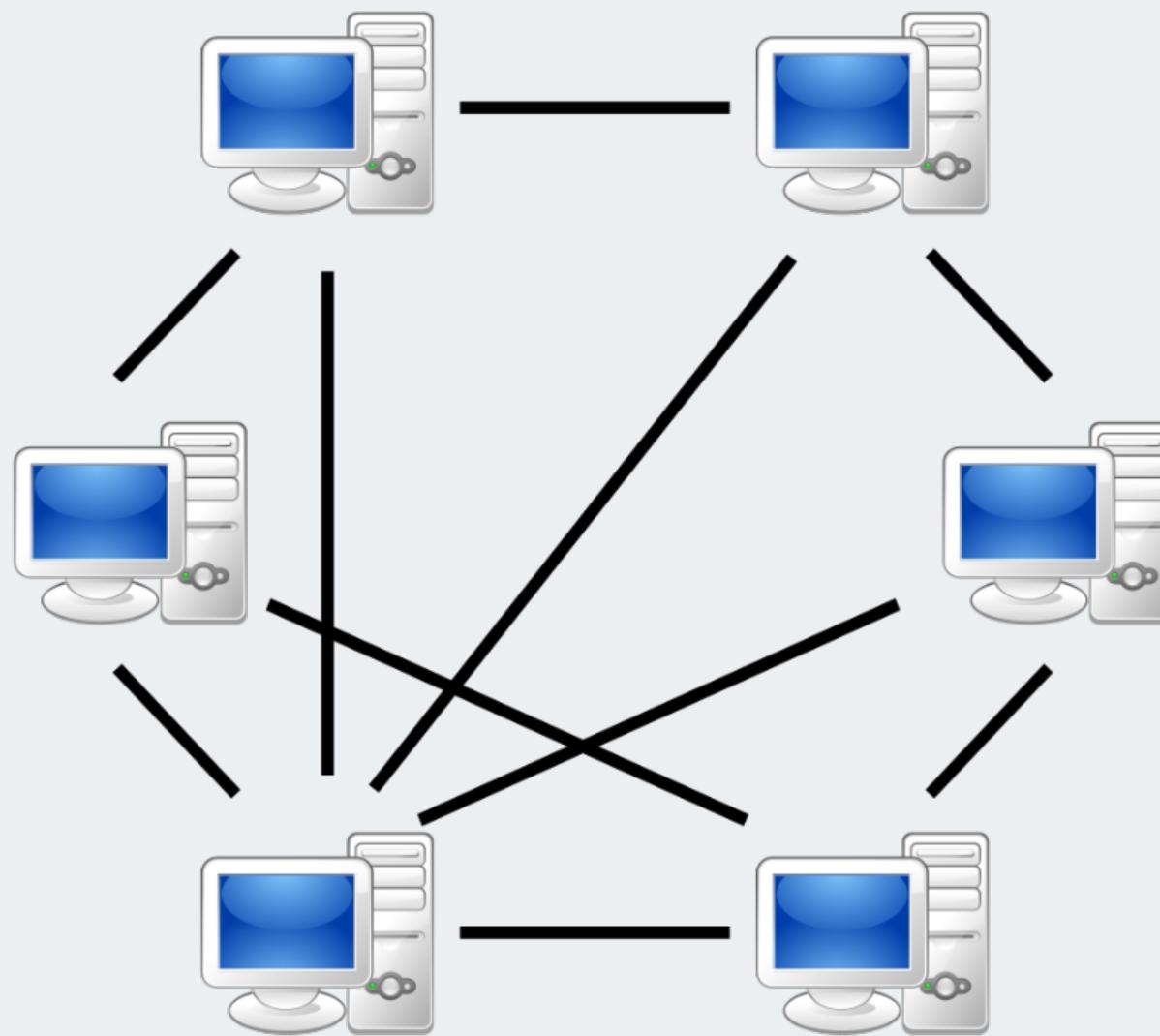
새로운 노드가 네트워크에 연결될 때



Unstructured P2P

모든 노드는 본인의 hop 영역, 이웃 노드를 제외하고
네트워크 전체 노드를 알지 못한다

peer-to-peer network



Scalability

새로운 노드가 네트워크에 연결될 때



Unstructured P2P

모든 노드는 본인의 hop 영역, 이웃 노드를 제외하고
네트워크 전체 노드를 알지 못한다

노드 간 message 송수신을 위해서

노드 Discovery 단계 가 선행되어야 한다.

노드 discovery

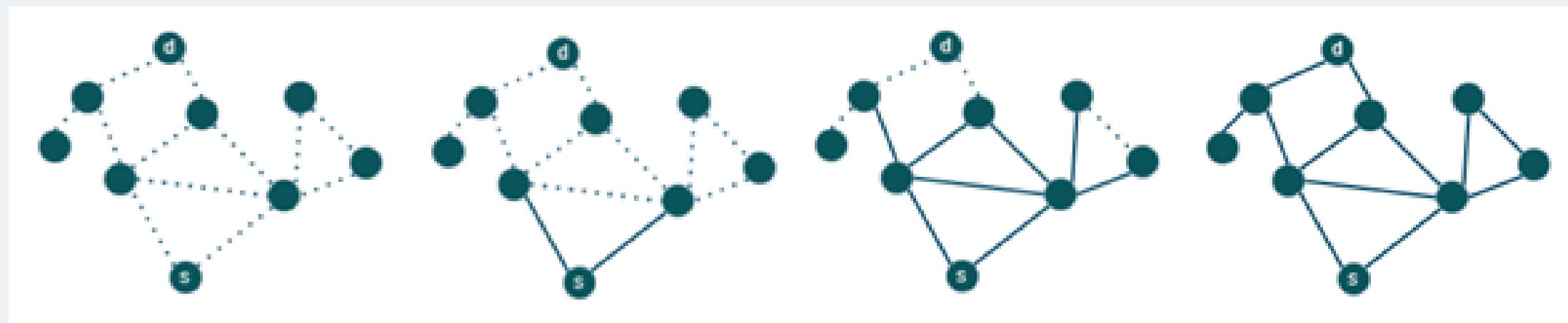
기존 알고리즘

네트워크 정보가 없을 때 네트워크 상의 주소를 이용하여
목적지까지 경로를 체계적으로 결정

Flooding

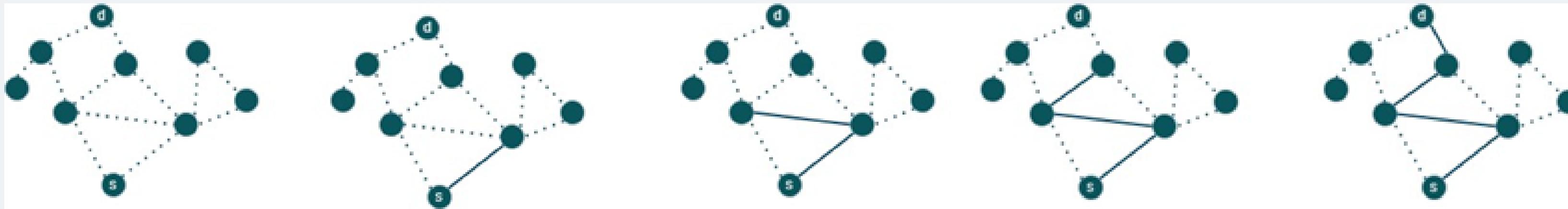
무제어 포트 배정

query를 보낸 peer를 제외한 모든 이웃 peer에게 검색 query를 전달



Random Walk

하나 이상의 이웃을 무작위로 선택하여 query를 전달



노드 discovery

기존 알고리즘

네트워크 정보가 없을 때 네트워크 상의 주소를 이용하여 목적지까지 경로를 체계적으로 결정

Flooding

빠른 탐색 가능

자원과 대역폭의
상당한 소모

Random Walk

대역폭 소모 적음

arrival time 늦음

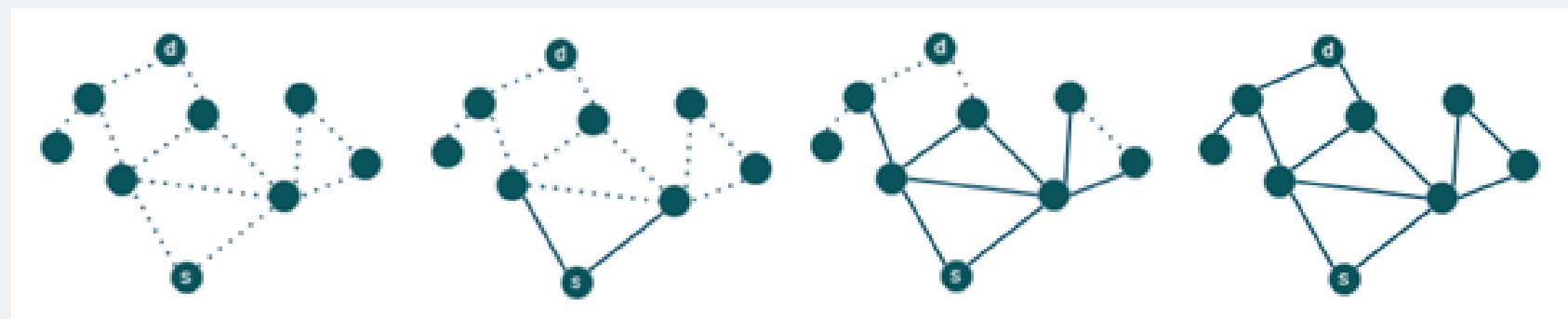
Random Walk with Neighbors Table

이웃 peer가 소유한 파일 목록이 포함된 테이블을 가지고 있는 경우
해당 테이블에서 요청된 리소스에 대한 항목을 찾는 추가 step 포함

노드 discovery

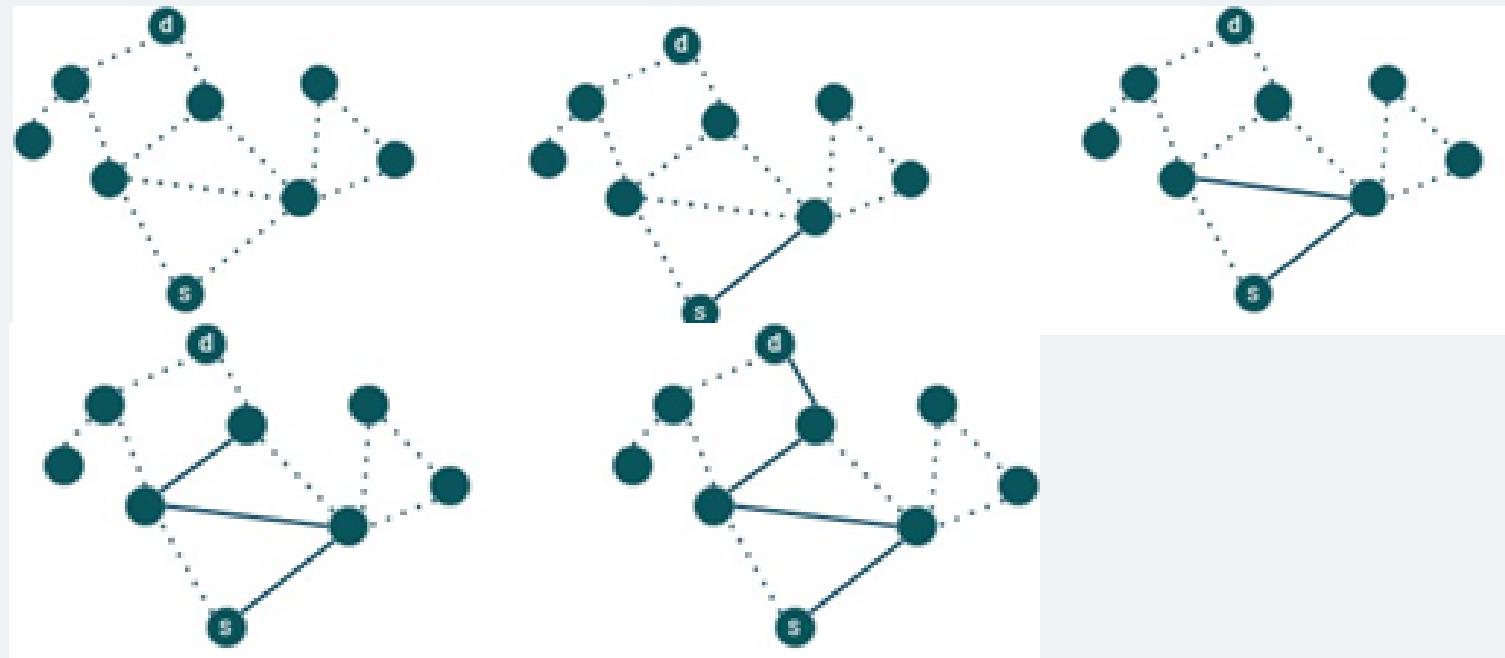
기존 알고리즘 문제점

Flooding



목표 노드(d)와 시작 노드 (s)의 거리가
근본적으로 먼 경우

Random Walk



목표 노드를 지나쳐버리는 경우

새 알고리즘

노드 간의 거리가 먼 경우

1. Start 노드에서의
Random walk 수행

2. endpoint의 발견

Query를 중복하지 않고 수신할 수 없는
상태의 노드

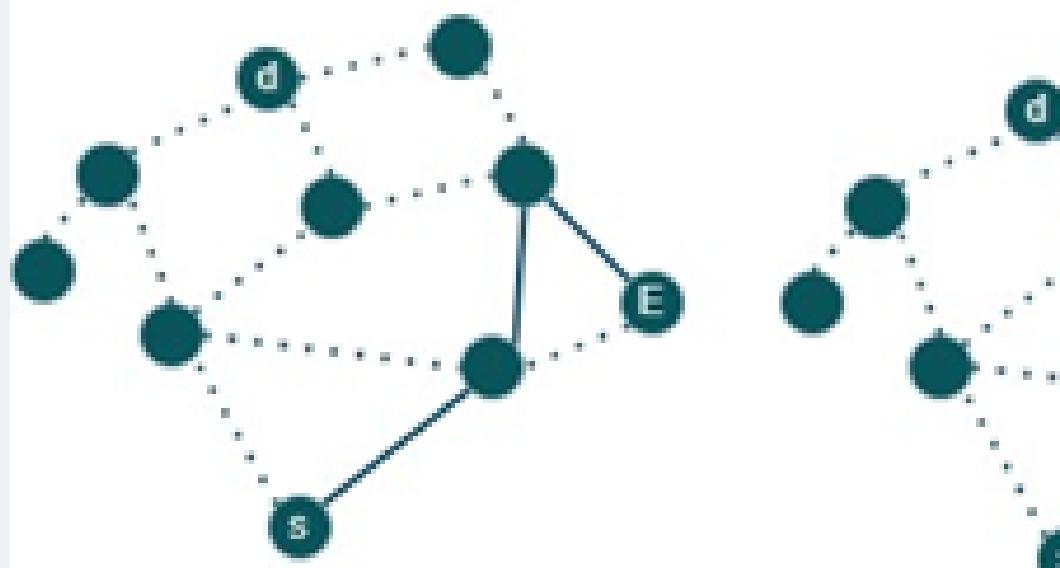
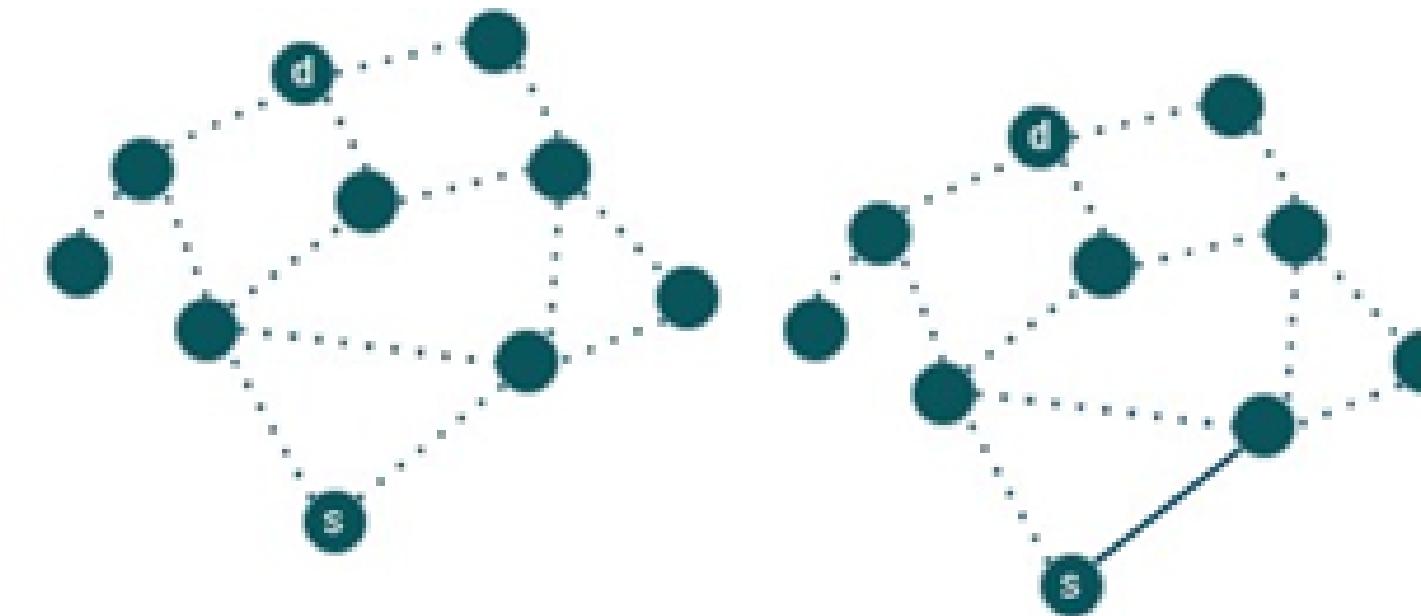
3. router를 기점으로 한
flooding 알고리즘 수행

endpoint에게 query를
수신한 노드

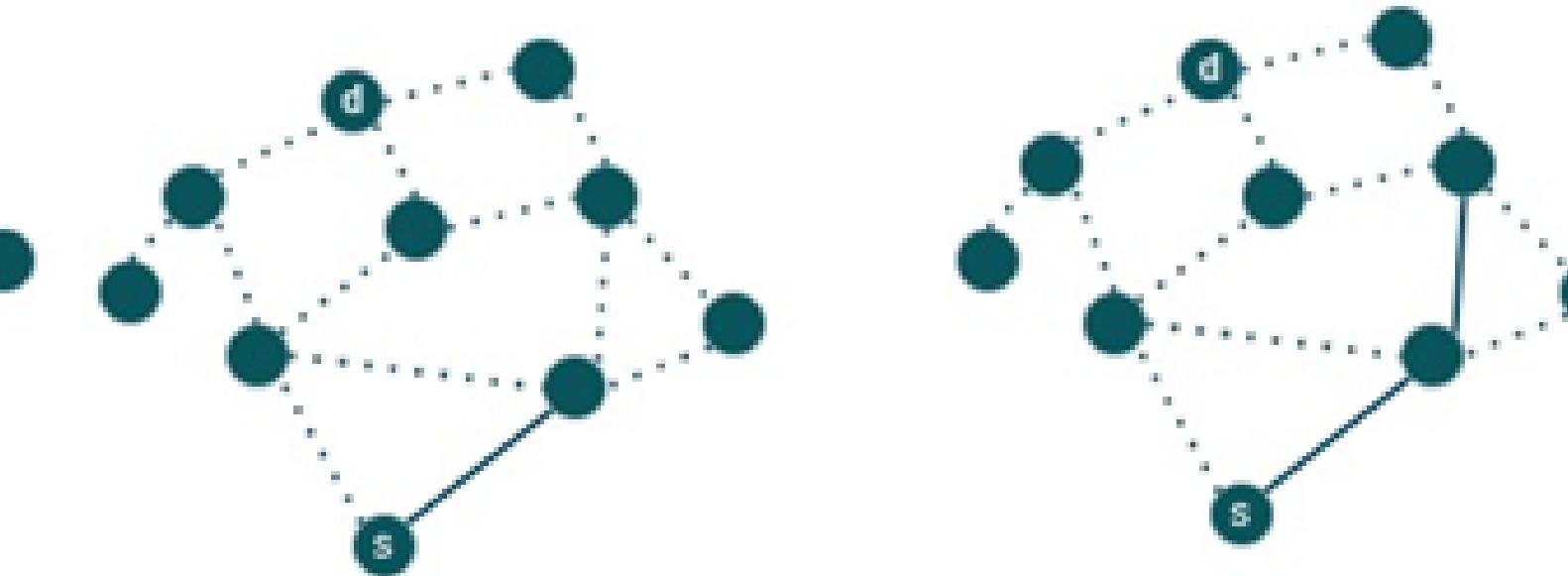
random walk를 통한 end point의 탐지와 router의 flooding 방식.

노드 discovery

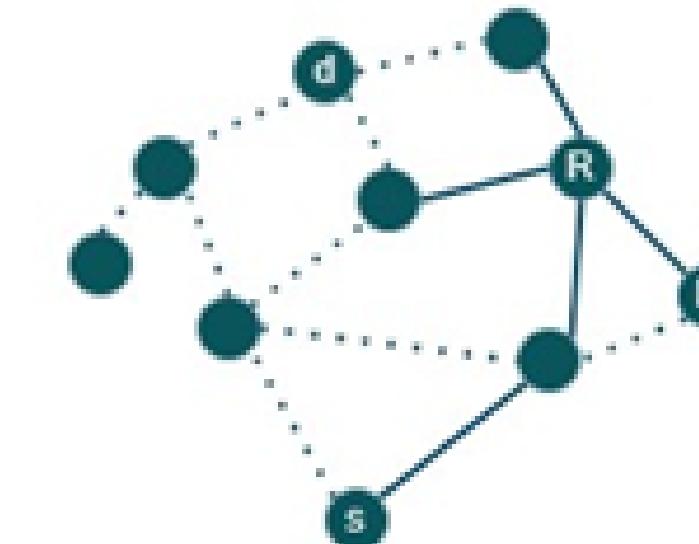
알고리즘 예



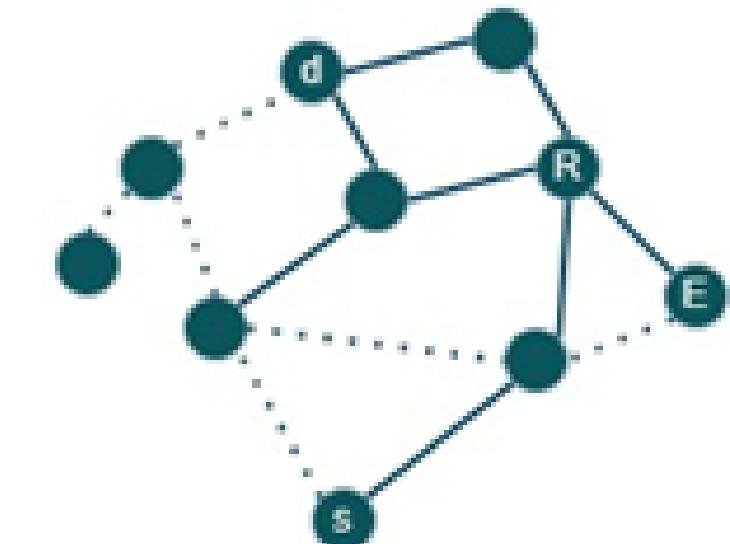
노드 E는 수신할 노드가 존재
하지 않을 때



노드 R을 라우팅로,
노드 E를 end point로 정함



노드 R을 기준으로
Flooding 진행



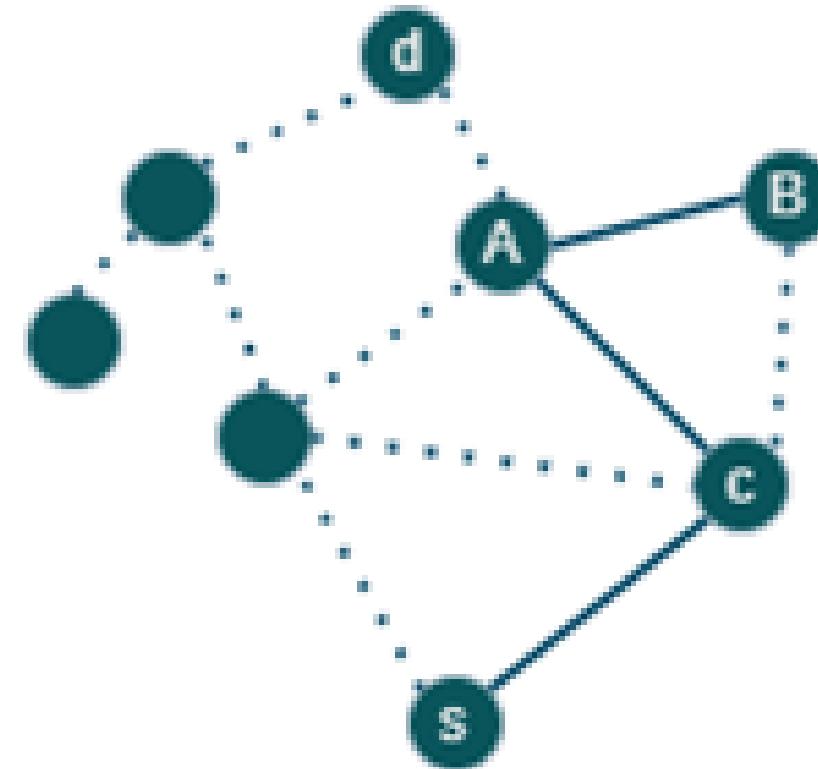
router와 end point의 정의

router A end point B

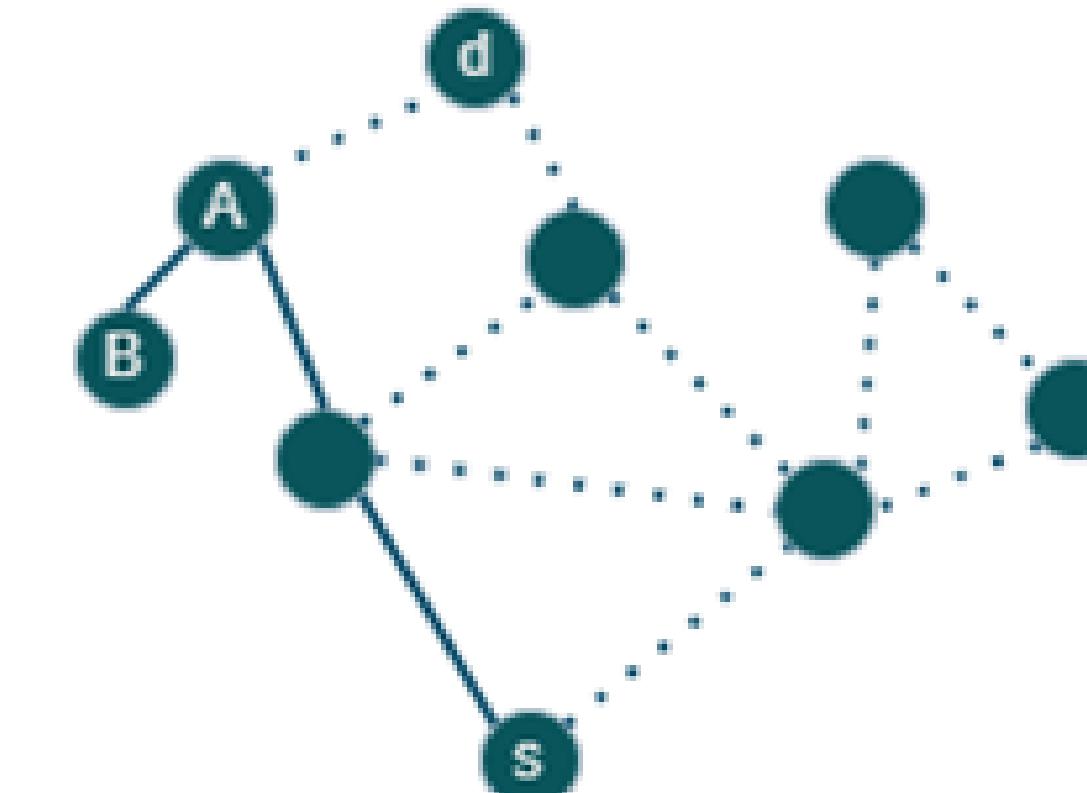
child 역할을 하는 end device A
parent 역할을 하는 라우터 B

노드 A가 노드 B에 도달할 수 있는 유일한 경로
송신할 노드 경로가 더 이상 존재하지 않는 경우

B가 수신할 노드가 없는 경우



B에 도달할 수 있는 유일한 노드 A



Peer가 동일한 고유번호의 query를 이미 받았다면,
해당 query는 버려진다

노드 discovery

성능 측정

Flooding

After k steps, some $R(k) = d \cdot (d-1)^{k-1}$

Random walk

$$S = \sum_{k=1}^N k \cdot \mathbb{P}[k] = \sum_{k=1}^N k \cdot \frac{r}{N} \left(1 - \frac{r}{N}\right)^{k-1} \approx N/r \text{ for } 1 \ll r \leq N.$$

$$T_{\text{random_walk}} = O(N_{\text{visited}})$$

$$T_{\text{is_endpoint}} = O(d)$$

$$T_{\text{flooding}} = O(V+E)$$

우선 연결, join & leave 노드 개념 도입

노드 1000개. 랜덤 배정

노드 discovery

알고리즘 구현

```

# endpoint 판별
def is_endpoint(node):
    neighbors = list(G.neighbors(node))
    for neighbor in neighbors:
        if not set(G.neighbors(neighbor)).issubset(neighbors + [node]):
            return False
    return True

# endpoint 발견 기점으로 알고리를 전환
def perform_discovery(start_node):
    endpoint = random_walk(start_node)
    if is_endpoint(endpoint):
        flooding(endpoint)

#τj = 조인 프로세스 중에 하위 그래프에 포함할 무작위로 선택된 노드에서 떨어진 흔 수
#연결할 노드를 찾기 위해 무선 연결 프로세스가 어느 정도까지 진행되는지
#τl = 탈퇴 프로세스 동안 하위 그래프에서 고려할 네트워크를 떠나는 노드에서 떨어진 흔 수
#나가는 노드에 연결된 노드를 다시 연결하기 위해 네트워크가 얼마나 멀리 떨어져야 하는지

# Example Usage
if __name__ == "__main__":
    grow(Ntarget, τj=2, τl=2)
    start_node = random.randint(1, N)
    perform_discovery(start_node)

```

우선 연결, join & leave 노드 개념 도입

노드 1000개, 랜덤 배정

```

import networkx as nx
import random

#global 변수
m = 2 # 최소 degree
μ = 0.1 # 노드가 네트워크를 떠날 확률
N = 0 # 기존 네트워크의 초기 노드 ID
G = nx.Graph() # 기존 네트워크 그래프
Ntarget = 100 # 목표 노드수

# 네트워크 생성(노드 추가, 우선연결)
def grow(Ntarget, τj, τl):
    global N
    for i in range(m+1, Ntarget):
        join(N, τj)
        num = random.random()
        if N == Ntarget:
            break
        if num < μ:
            Ndel = random.randint(1, N)
            leave(Ndel, τl)

#G2의 노드를 사용하여 G1에 우선 연결을 수행.
def preferential_attachment(G1, G2):
    #성공한 새 링크 수를 반환
    new_links = 0
    for node in G2:
        if node in G1:
            G1.add_edge(random.choice(list(G1.nodes)), node)
            new_links += 1
    return new_links

# Nrand의 이웃 노드를 포함한 부분 그래프를 최대 hop 거리로 가져옴.
def get_subgraph(Nrand, hops):
    # TTL 결정 판단 기준
    return nx.ego_graph(G, Nrand, radius=hops)

```

```

def join(i, τj):
    global N
    N += 1
    numoflinks = 0
    while numoflinks < m:
        Nrand = random.randint(1, N)
        myG = get_subgraph(Nrand, τj)
        numoflinks += preferential_attachment(G, myG)

def leave(i, τl):
    myG = get_subgraph(i, τl)
    G.remove_node(i)
    preferential_attachment(G, myG)

# Random walk 알고리즘
def random_walk(node):
    visited = set()
    while True:
        neighbors = list(G.neighbors(node))
        if not neighbors:
            break
        next_node = random.choice(neighbors)
        if next_node in visited:
            return node
        visited.add(next_node)
        node = next_node
    return node

#Flooding 알고리즘
def flooding(node):
    queue = [node]
    visited = set()
    while queue:
        current = queue.pop(0)
        if current in visited:
            continue
        visited.add(current)
        for neighbor in G.neighbors(current):
            if neighbor not in visited:
                queue.append(neighbor)
    return visited

```

노드 discovery

알고리즘 구현

```
# endpoint 판별
def is_endpoint(node):
    neighbors = list(G.neighbors(node))
    for neighbor in neighbors:
        if not set(G.neighbors(neighbor)).issubset(neighbors + [node]):
            return False
    return True

# endpoint 발견 기점으로 알고리를 전환
def perform_discovery(start_node):
    endpoint = random_walk(start_node)
    if is_endpoint(endpoint):
        flooding(endpoint)

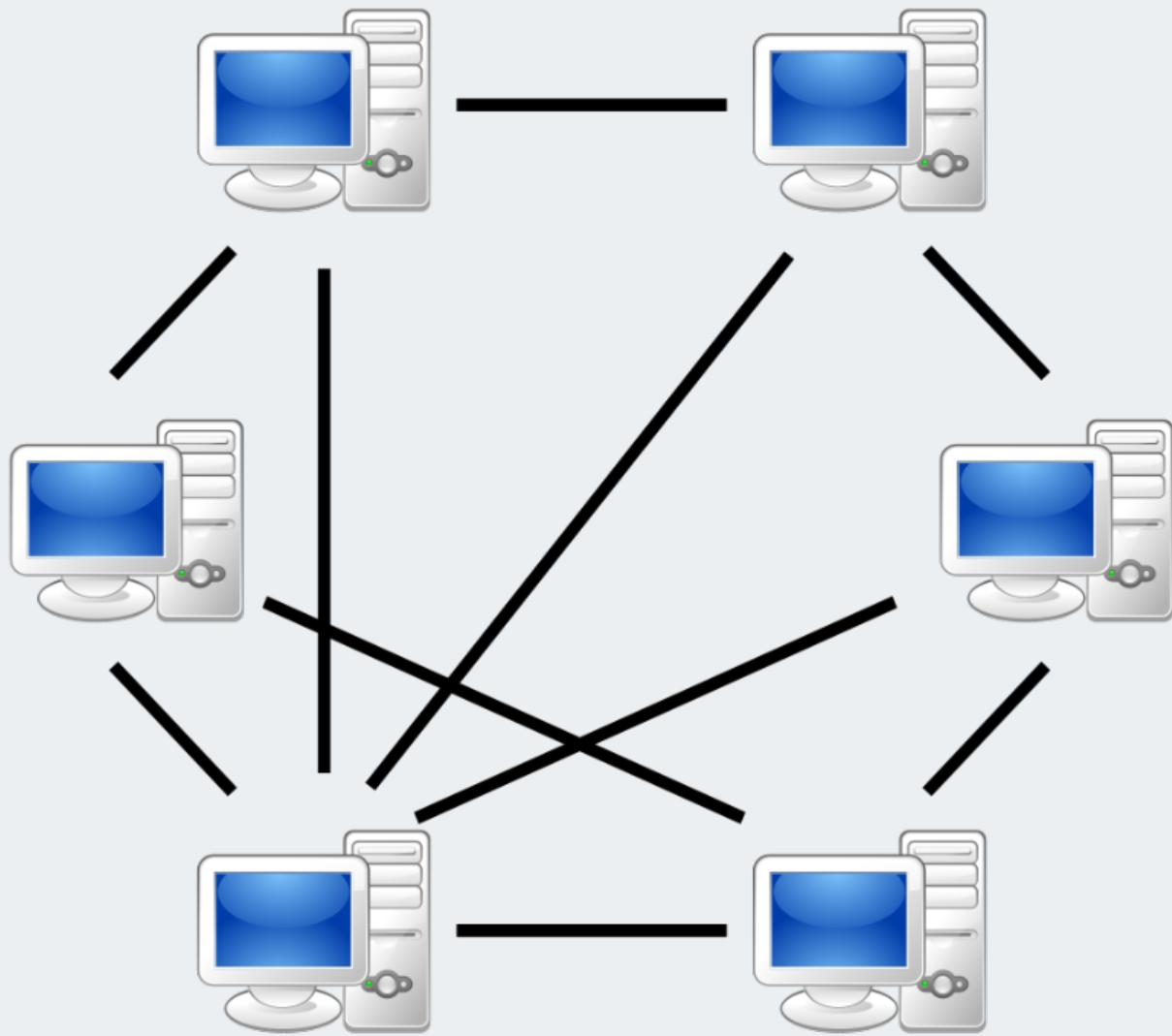
# τj = 조인 프로세스 중에 하위 그래프에 포함할 무작위로 선택된 노드에서 떨어진 흔적 수
# 연결할 노드를 찾기 위해 무선 연결 프로세스가 어느 정도까지 진행되는지
# τl = 탈퇴 프로세스 중에 하위 그래프에서 고려할 네트워크를 떠나는 노드에서 떨어진 흔적 수
# 나가는 노드에 연결된 노드를 다시 연결하기 위해 네트워크가 얼마나 멀리 떨어져야 하는지

# Example Usage
if __name__ == "__main__":
    gnm(Ntarget, τj=2, τl=2)
    start_node = random.randint(1, N)
    perform_discovery(start_node)
```

process	arrival time
p1	139ms
flooding	144ms
random walk	160ms

수많은 시행 중 하나

기대 효과



Dynamic Adaptation

현재 네트워크 토플로지 및 리소스 가용성을 기반

Targeted Flooding

홀 네트워크 그룹 내의 Flooding: 관련 노드 수를 최소화

효율적인 resource 사용

A dark, grainy photograph of a city street scene. In the foreground, there's a utility pole with a speed limit sign that reads "60". Behind it are several modern skyscrapers with glass facades. One building has a distinctive wavy or undulating pattern on its facade. The sky is overcast and grey.

THANK YOU