Improving microservices communication using Named data routing

Harshavardhan Kadiyala, Gu RainEr ,
Susmit Japhalekar , Ravneet

I. Introduction

Microservices architecture(MSA) [1] is a most used application architecture in modern cloud-native applications. MSA has replaced traditional monolithic applications, firstly because of its reduced complexity allows each service to be implemented independently in decentralized service model into manageable services. Secondly, independent implementation of services allows the designers to use new technologies in there new services without breaking other existing services.

A typical microservices application is a collection of interacting services, each can be launched in their own container. These containers are ephemeral in nature and are created, migrated or destroyed by a container orchestration system like kubernetics. When a container is created or migrated, its IP address will change. This will make it impossible for other services to communicate just using IP addresses. Currently, this discoverability problem in microservices is addressed by different service discovery mechanisms like distributed key-value store[cite] or DNS.

Service Discovery is an essential aspect of microservices architecture as it avoids early binding of clients to particular service instances. Removing such coupling provides greater flexibility for reconfiguration of the overall system. Other than just providing connectivity between services, service discovery will help in load balancing the flows between different services with decreasing end to end latency as its optimization goal.

In this project, we take a data-centric approach to service discovery. Named data networking(NDN) [2] is a new approach to interconnect systems together in which required data is directly requested from the network rather than connecting to a specific host. This model of network particularly helpful in interconnecting microservices as each microservice wants data from a service end-point. Using NDN for connecting microservices will not only provide service discovery but also improves security, reliability and scalability of the application as described in section IV .

In section II, we will introduce to existing solutions and NDN archtecture. In section III, we will give motivation for the problem. Section IV will contain description of project solution. finally section V will talk about how we will evaluvate our proposed system.

II. Related work

A. Service Discovery

There has been much work on service discovery in field of distributed systems. Etcd [3] is a distributed key-value store based on directories and consensus. It uses RAFT [4] protocol, primaraly for multiple nodes to maintain identical logs of state changing commands. Any node in a raft cluster may be treated as the master. It will coordinate with the others to agree on which order state changes should happen in. Another work is Consul [5] which is open source solution for service discovery and distributed configuration management. It has a distributed key, value store for storing service database that can be queried using DNS. Consul provides dynamic load balancing and comprehensive service health checking using both in-built solutions as well as user provided custom solutions. Unlike Consul, Synapse [6] was designed to support services running in Docker. At the core of Synapse is an HAProxy instance for routing requests from a service consumer on the application server to a service provider running in the cluster. HAProxy configuration in Synapse are made by watchers daemons that check for changes to the locations of services. Dynamic Service Discovery System with the help of Zookeeper [7], which provides an open source solution to the various coordination problems in large distributed systems. ZooKeeper, as a centralized coordination service, is distributed and highly reliable, running on a cluster of servers called a ZooKeeper ensemble. Distributed consensus, group management, presence protocols, and leader election are implemented by the Zookeeper ensemble.

The docker platform has Docker Swarm [8] and Docker Compose [9] that, when combined together, may provide a solution to the service discovery problem. Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. One can query every container running in the swarm through a DNS server embedded in the swarm. Mesos-DNS [10] provides service discovery through DNS. Mesos-DNS generates an service record for each application instances and translates these records to the IP address and port on the machine currently running each application.

It is important to distinguish the service discovery problem from orchestration - the process of deploying containers on a cluster of machines. CoreOS, Mesos/Mesosphere, OpenShift, CloudFoundry, Kubernetes, Brooklyn/Clocker, Shipyard, and Crane are just a few of the many available orchestration solutions, and some of the larger projects like Kubernetes and Mesosphere include some service discovery features. In general, orchestration solutions represent a fundamentally different approach to managing systems built with containers. Adoption of these solutions is often a very heavy undertaking requiring highly specialized deployment environments.

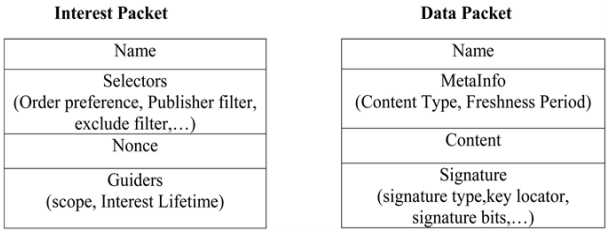

Fig. 1. NDN packets [2]

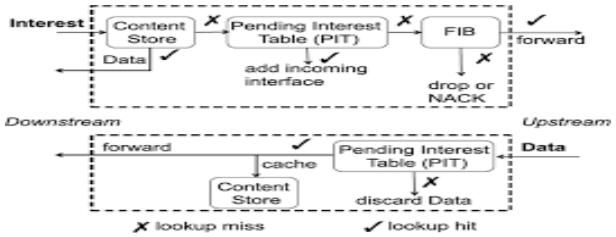

Fig. 2. Forwarding Process at an NDN Node [2]

B. Named Routing

Information-Centric Networking (ICN) has been proposed as a promising new Internet architecture. In ICN, the content is accessed by the name of the content rather than the IP address of the host machine that has the content. By separating content from location, ICN is expected to improve the network efficiency and reduce the communication cost for fetching contents. Similarly, Named data networking (NDN) [2] is a new networking paradigm using named data instead of named hosts for communication under the broad ICN umbrella. NDN architecture makes the original Internet design elegant and powerful. NDN is driven by the receiving end through the exchange of two types of packets: Interest and Data. Both types of packets carry a name that identifies a piece of data that can be transmitted in one Data packet. A consumer puts the name of a desired piece of data into an Interest packet and sends it to the network. Routers use this name to forward the Interest toward the data producer(s). Once the Interest reaches a node that has the requested data, the node will return a Data packet that contains both the name and the content, together with a signature by the producers key which binds the two (Fig 1). This Data packet follows in reverse the path taken by the Interest to get back to the requesting consumer.

To carry out the Interest and Data packet forwarding functions, each NDN router maintains three data structures: a Pending Interest Table (PIT), a Forwarding Information Base (FIB), and a Content Store (CS) (Fig 2), as well as a Forwarding Strategy module that determines whether, when and where to forward each Interest packet. The PIT stores all the Interests that a router has forwarded but not satisfied yet. The Content Store is a temporary cache of Data packets the router has received. The forwarding information base (FIB) table use the names to route a request to next hop routers that are closer to the data provider by performing longest-prefix matching. The hierarchy also allows name resolution and data routing information to be gathered across similar data source names, which is the key point to enhance scalability and flexibility of the network architecture.

III. Motivation

The solutions related to distributed reliable key-value store[3], [5], [6], [7] mentioned above essentially are based on the similar priciples and architecture which relys on a consistent key-value store as service registry that is part of the cluster implementation. However, the complexity of this approach is heavily dependent on the number of cluster nodes. Consider the situation when there are insufficient number of cluster nodes, the Raft-based[4] quorum can not be properly formed, hence the data consistency may not be achieved across the nodes. Also, the recovery might take very long time due to various failures and having to restart from a clean slate and recover from backup. On the other hand, when the number of cluster nodes is large, this approach results in heavy overhead for the synchronization of the distributed states among every cluster nodes, since these quorum will have to require high participation of the members through either the mulicating or broadcasting of their status.

The approaches based on DNS[8], [9], [10] although can be realily deployable through standard DNS infastructures and they do not suffer from the problems that the distributed consistent key-value stores have, they do have their own limitations using for service discovery since DNS was originally designed for a different purpose. Firstly, the DNS-based approaches do not support the pushing the changes of the services, so the service requester either has to periodically poll for the changes or implement extra level for caching. Secondly, DNS protocol runs over UDP, which makes the communications short of flow control. Even though such techniques could be implemented within the services with extra complexity, the advantage of using DNS can be diluted. Thirdly, DNS suffers from propagation delay and generation or updating of DNS records are also relatively slow, which can be problematic since service discovery always needs to be fast-paced.

Compared to DNS-based solutions and distributed key-value storage, ICN-based[11] solution emerged to be a better candidate for facilitating the efficient and timely delivery of service inforamtion to end users. In particular the hierarchical design of name space in NDN[2], [12] that allows name resolution and data routing information to be gathered across similar data source names, which is the keystone to improve the resilience and flexibility of the network architecture even when operating at large scale. Also this ICN-based solution not only allows facilitating changes and scalability of microservices but also avoids heavy communication for state control compared to existing solutions. Moreover, since NDN is an network paradigm, integrating flow control mechanism into the lower level network stack is possible, which can not be easily done in the case of DNS. Furthuremore, in NDN the data security check required by the protocol is embedded as a segment into the low level switching packet, which prevent the clients from various attacks that could happen to DNS such as DNS poisoning and spoofing.

The existing solutions to service discovery of MSA suffers from many problems that highly degrade the advantages of

building services with MSA. The service discovery not only require the fast-paced delivery of the entry points of the services but also the capability of supporting large number of requests with high availability. Therefore, a solution to address the naming and routing of microservices with low latency, flexibility, and high resilience at large scale is proposed based on the design of NDN network stack. This solution also provides load-balancing, and security of data packet through authentication.

IV. Proposed solution

In a typical microservices application, who is communicating is less important than what is the data being exchanged. In order to give priority to the data rather than to hosts, we will design an Information-centric network [11] between microservices using NDN network stack [12]. This new network design will facilitate various important services like load balancing, service discovery and security features to be provided as part of our new network.

A. Network services

1) **Service discovery**
   a) Each HTTP endpoint will have a unique global name in an application namespace.
   b) Every microservice inside an application namespace will have the capability to request objects of data from any other endpoint inside the same namespace.
   c) The network will have a capability to propagate requested objects through the network to the requester.
2) **Load balancing**
   a) We will provide a capability for in-network storage [12] of objects which can decrease latency and improve the reliability of the application.
   b) We will provide load balancing of flows across different objects of the same version.
   c) We will give capability for different versions of the same object to coexist [13].
3) **Security features**
   a) Enabling the formation of the trust domains [1] using the concept of application namespaces[ref] to protect microservices even when an intruder gets into the system (defence in depth [14]).
   b) We will implement service authentication as a network function for new microservices joining the network.
4) **Improved resilience**
   a) We will deliver the most recent object write from the local cache, in case of failure of an endpoint.

In order to achieve above-said goals, we describe the architecture of the new network model in next section.

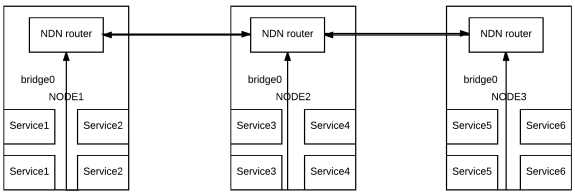

Fig. 3. Architecture of proposed network.

B. Implementation

1) Overview of design : Each microservice will have a client library which will help connect to the NDN network. Each physical node will have a virtual NDN router capable of routing NDN packets through the network (see figure 3) . This NDN router will be hosted on the top of DTCP as application layer protocol[ref], this will allow us to concentrate on microservices specific functionality as replacing stack is out of the scope of this project but is possible [12].

2) Naming schema: In our naming scheme, each HTTP endpoint is addressable. Every HTTP endpoint will have a globally unique name.

httpendpoint/instance_count/httpend_version

Example end point

This naming schema is simple enough but serves to achieve required features

1) By using only unique HTTP endpoint name and removing reference to a microservice name, we can decouple endpoints from hosting microservices. This will help to decrease downtime and unnecessary changes when endpoints are moved between services.
2) Having an instance count can help identify and load balance among different instances of same HTTP endpoint.
3) httpendversion helps in maintaining two versions of same end point available at the same time. this can felicitate rolling updates.

3) Initial handshake: Newly created microservice will broadcast in its own local container bridge [15] for local NDN router IP address. Once it receives IP address it will use Named Data Link State Routing Protocol to advertise all its objects to the local router. an NDN router will only advertise longest common name for all other routers. In this way, we can have few updates while maintaining load balance between distributed endpoints.

We will implement a layer on the top of the virtual router to facilitate the initial handshake of a new HTTP endpoint and token authorization for admission of new microservice into the network.

4) Routing: There will be a virtual NDN node router in every physical node (see figure 3). This router will be connected to all other containers in the same node using a bridge [15]. We will reuse NDN network stack in particularly named data network forwarding [16] and enhance its forwarding strategy to facilitate load balancing between different HTTP endpoint

instances and also for different versions of HTTP endpoints to co-exist.

We will use NLSR - Named Data Link State Routing Protocol [17] to transfer new HTTP endpoint information among router overlay network. we will only transfer routes longest matching subaddress.

5) In-network storage: We will implement a content cache [12] using memory cache in the router to facilitate in-network storage(see figure 2). This cache will also return most updated value of the HTTP endpoint in case of network failure.

6) Security features: We will use the signature section of NDN data packet [12] to sign with the public key supplied in the interest packet of the client. Upon receiving the requested object, the client can authenticate using the signature sent by the sender.

V. Evaluation

A. Testbed

We will create a 3 node test bed. Each node is a VM with 2 VCPUs and 8 GB ram. Microservices which will be deployed on this nodes will be synthetic and made only for testing.

B. Testing

1) We will develop a test microservice which will probe all the available HTTP endpoints in the application domain, measure latency and throughput.
2) We will repeat the experiment by placing the test microservice in all of the three nodes.

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *arXiv preprint arXiv:1606.04036*, 2016.

[2] "NSF Named Data Networking project online," http://www.named-data.net/, accessed: 2017-10-08.

[3] "etcd online," https://github.com/coreos/etcd, accessed: 2017-10-08.

[4] D. Ongaro and J. Ousterhout., "In search of an understandable consensus algorithm,," *Proc. USENIX Annual Technical Conference , pp. 305-320, June 2014.*, 2014.

[5] "Consul online," https://github.com/coreos/etcd, accessed: 2017-10-08.

[6] "Synapse online," https://github.com/airbnb/synapse, accessed: 2017-10-08.

[7] K. Ting, "Building an impenetrable zookeeper," *Strange Loop, 2012*, 2012.

[8] "Docker Swarm online," https://github.com/docker/swarm, accessed: 2017-10-08.

[9] "Docker Compose online," https://github.com/docker/compose, accessed: 2017-10-08.

[10] "Mesos DNS, howpublished = https://github.com/mesosphere/mesos-dns, note = Accessed: 2017-10-08."

[11] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[12] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[13] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[14] K. K. Stouffer and J. Falco, "Recommended practise: Improving industrial control systems cybersecurity with defense-in-depth strategies," *Department of Homeland Security, Control systems security program, national cyber security division*, 2009.

[15] "Docker container networking," https://docs.docker.com/engine/userguide/networking, accessed: 2017-10-08.

[16] H. Yuan, T. Song, and P. Crowley, "Scalable ndn forwarding: Concepts, issues and principles," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE, 2012, pp. 1–9.

[17] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.