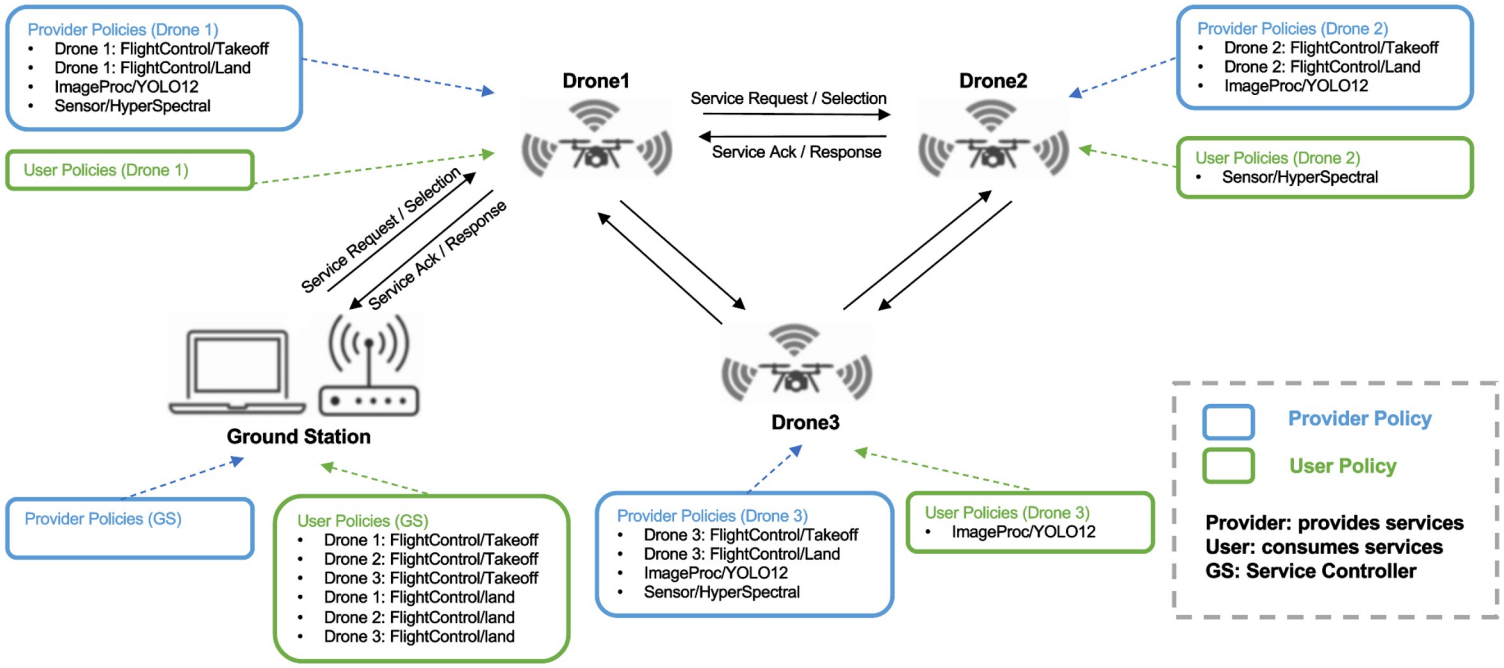
Named Data Network Service Framework (NDNSF)

Tianxing Ma, Eddie Jacobs, Lan Wang
University of Memphis

Motivation

- Goal: develop a High level NDN-based API that can be used to create secure service-oriented applications by people who know little about NDN



Introduction

NDNSF is a secure distributed service framework to support distributed service applications:

- service publication and discovery
- service request to one or multiple service providers
- service authorization for providers and access control for users
- authenticity and integrity of service messages

Background

- NDN Sync
  - To support distributed applications, sync protocols synchronize the data names of a shared dataset among a group of participants
  - We use the NDN-SVS (NDN State Vector Sync) protocol.

- NAC-ABE
  - an extension to Name-based Access Control (NAC)
  - uses an attribute-based encryption (ABE) scheme to support access control with improved scalability and flexibility.

Service Publication and Discovery

- NDNSF uses NDNSD to publish and discover services.
- Service providers publish their service information via NDN Sync.
- Service users join the Sync group to discover available services.
- Standard service information includes
  - the provider's name prefix
  - service/function name, lifetime, and description.

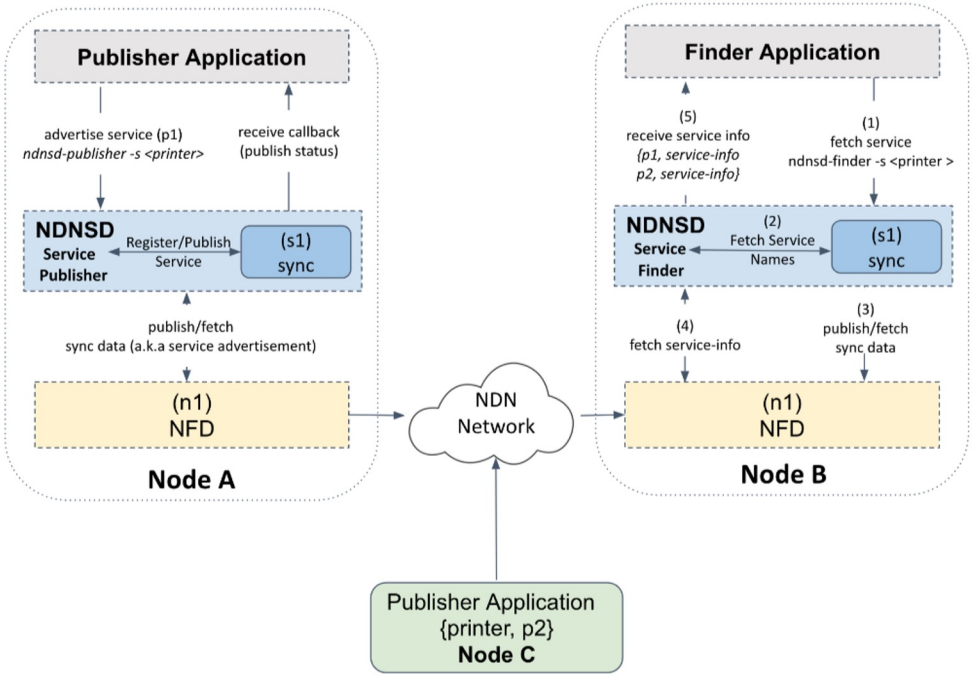
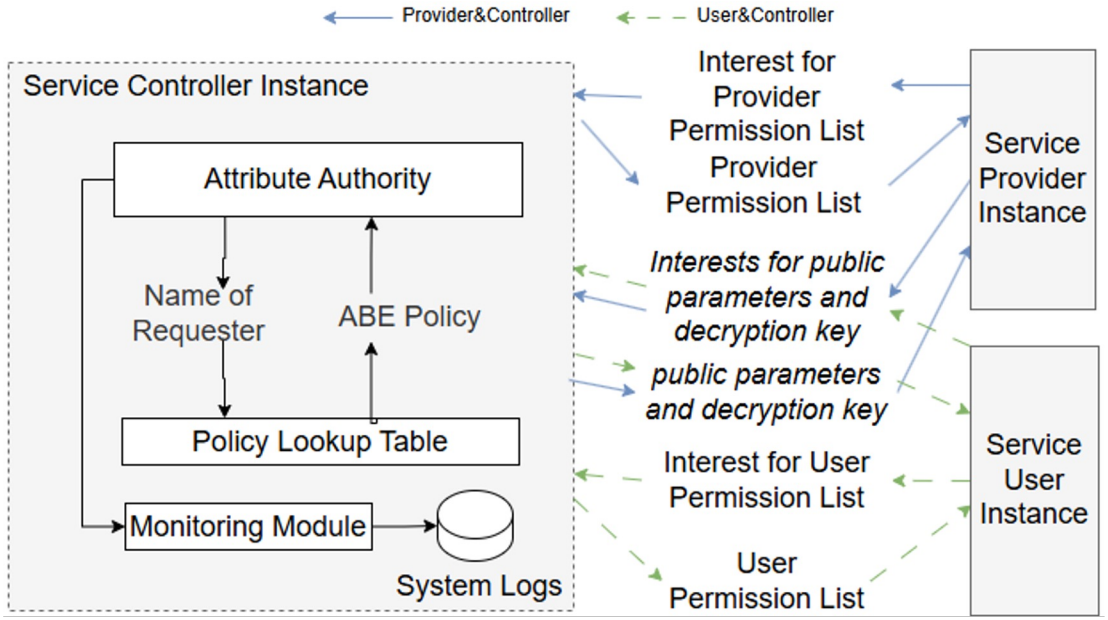


Fig. 2: Application workflow showing all layers involved in service publishing and discovery process

Service Controller



Name-based Policies

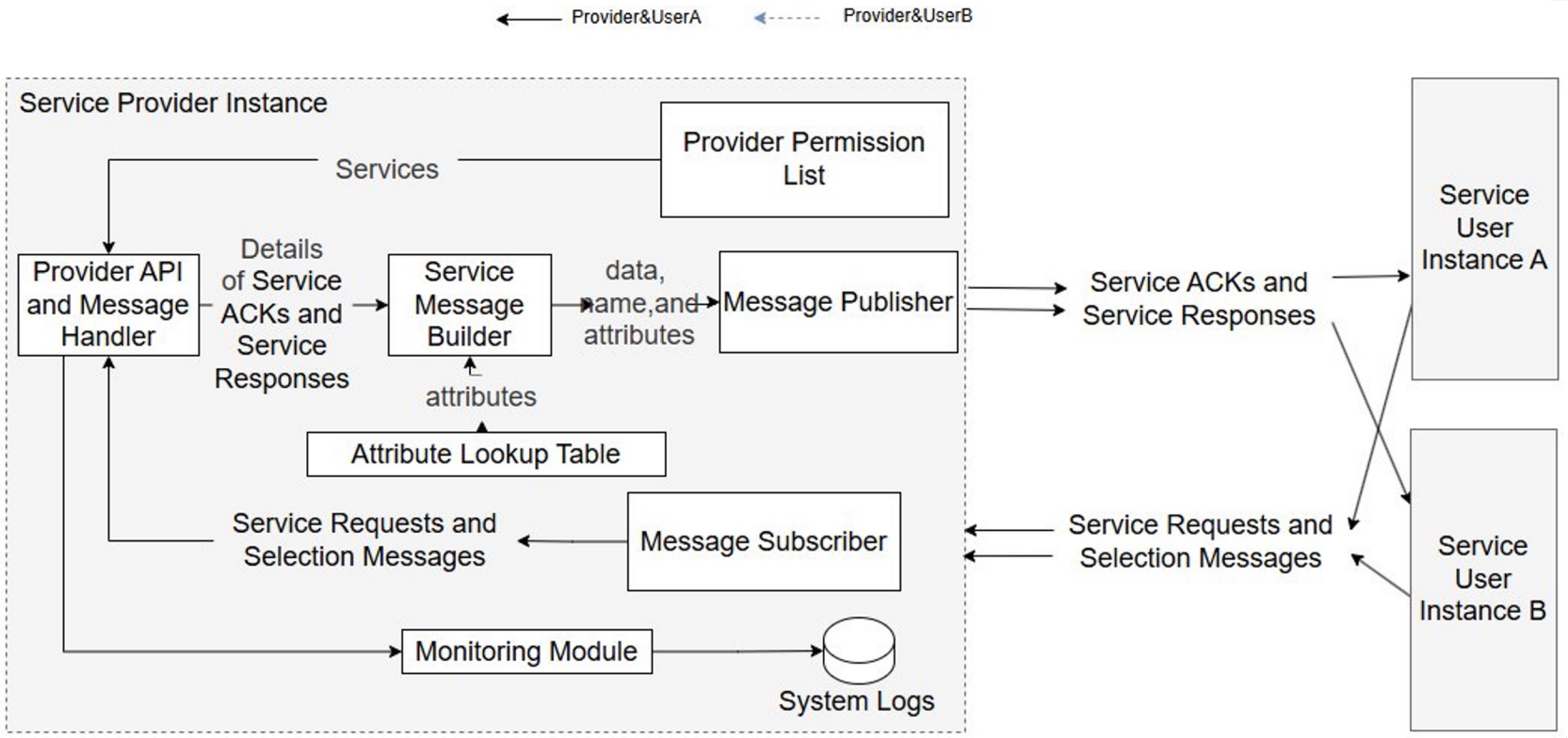
Provider Policies

```
provider-policies
{
    provider-policy
    {
        for /UAVNET/drone1
        allow
        {
            /UAVNET/drone1/FlightControl/Takeoff
            /UAVNET/drone1/FlightControl/Land
            /UAVNET/ImageProc/YOLO12
            /UAVNET/Sensor/HyperSpectral
        }
    }
    provider-policy
    {
        for /UAVNET/drone2
        allow
        {
            /UAVNET/drone2/FlightControl/Takeoff
            /UAVNET/drone2/FlightControl/Land
            /UAVNET/ImageProc/YOLO12
        }
    }
    provider-policy
    {
        for /UAVNET/drone3
        allow
        {
            /UAVNET/drone3/FlightControl/Takeoff
            /UAVNET/drone3/FlightControl/Land
            /UAVNET/Sensor/HyperSpectral
        }
    }
}
```

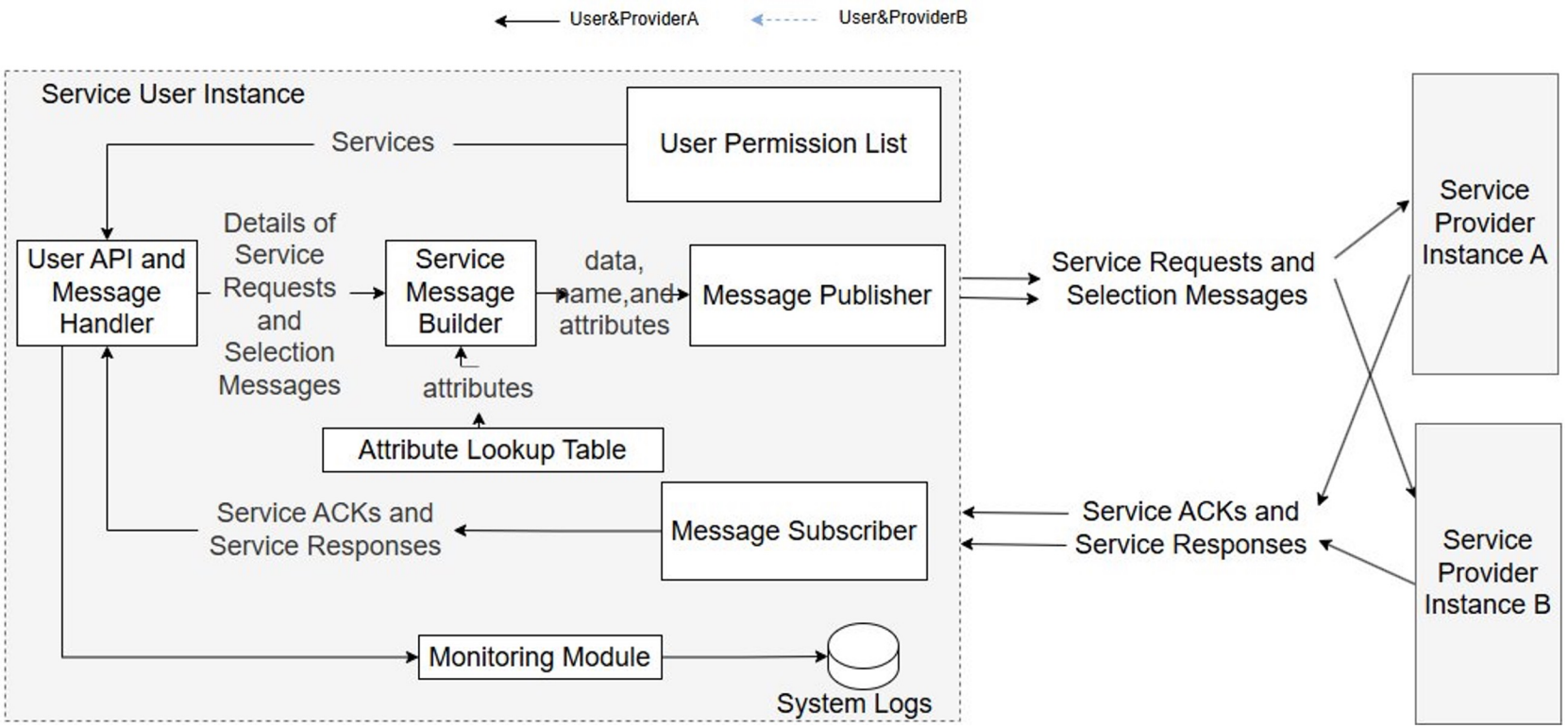
User Policies

```
user-policies
{
    user-policy
    {
        for /UAVNET/gs/
        allow
        {
            /UAVNET/drone1/FlightControl/Takeoff
            /UAVNET/drone1/FlightControl/Land
            /UAVNET/drone2/FlightControl/Takeoff
            /UAVNET/drone2/FlightControl/Land
            /UAVNET/drone3/FlightControl/Takeoff
            /UAVNET/drone3/FlightControl/Land
        }
    }
    user-policy
    {
        for /UAVNET/drone2/
        allow
        {
            /UAVNET/Sensor/HyperSpectral
        }
    }
    user-policy
    {
        for /UAVNET/drone3/
        allow
        {
            /UAVNET/ImageProc/YOLO12
        }
    }
}
```

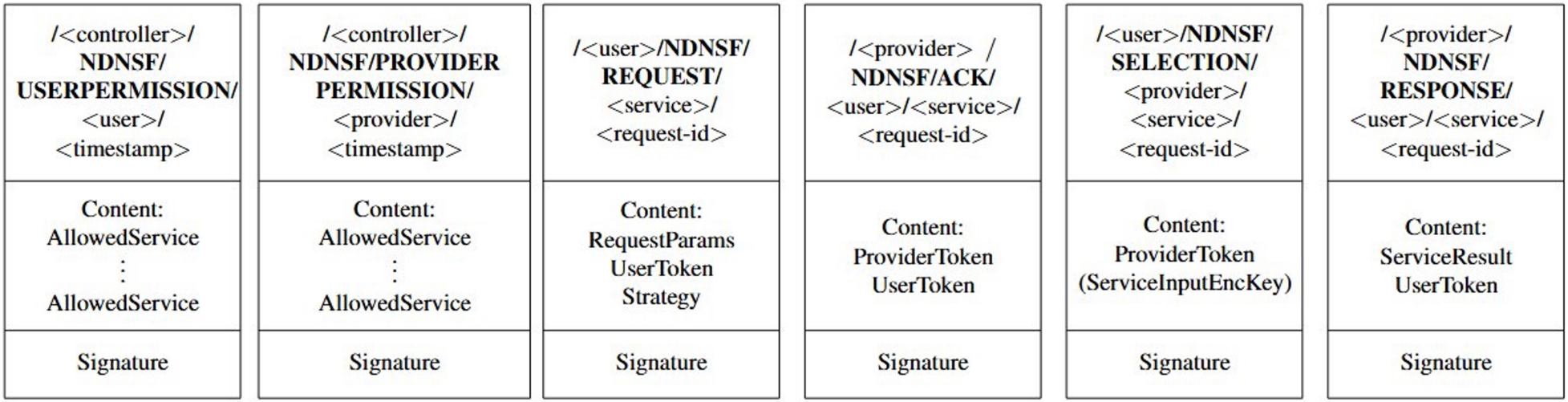
Service Provider



Service User



NDNSF Messages



(a) User Permission (b) Provider Permission (c) Request (d) ACK (e) Selection (f) Response

- NDNSF messages are published as data through Sync. Note: the Request message is not an Interest.
- NDN's signature mechanism ensures authenticity and integrity.

NDNSF Adaptive Admission Control

- **Excess workload is buffered locally rather than injected into an already congested control plane.**
- **User-Side Adaptive Admission Control**
  - New tasks first enter a local queue
  - The user monitors inflight requests and latency
  - If congestion increases, request publication slows down
  - If the system becomes overloaded, publishing temporarily pauses
  - Publishing resumes after inflight requests return to a safe level
- **Provider-Side Selective ACK Admission**
  - Providers monitor pending requests and runtime backlog
  - Overloaded providers suppress or delay ACK publication
  - Only requests likely to complete successfully are accepted
- **Result**
  - Timeout cascades are avoided
  - The system converges toward sustainable throughput automatically

NDN-SVS Patch: Non-Blocking and Parallelization

- **Problem:**
  - NDNSF uses NDN-SVS as its message synchronization protocol. However, under high-frequency NDNSF request workloads, the blocking APIs of NDN-SVS, together with its design of placing nearly all logic on the main I/O thread, significantly degrade the performance of NDNSF.

- **Solution:**
  - For Sync Interest handling, the design goal was to keep all Face and io_context operations on the main NDN thread, while moving CPU-heavy work off the main loop. The main thread now receives the Sync Interest, takes a safe snapshot of the needed state, and submits pure processing work to a bounded worker pool. Worker threads can parse/decode, compare state vectors, compute missing data, and prepare response metadata, but they do not call face.put, expressInterest, processEvents, prefix registration, or any other Face operation. Results are posted back to the main thread, where freshness/staleness is checked again before any final NDN operation happens.
  - For publish/production, we added an asynchronous publish path instead of forcing every caller to block while the publication is fully prepared. The main thread accepts the publish request, queues or coalesces pending production work, and worker threads prepare the expensive parts where safe. Final SVS/Face publication remains on the main thread. This gives NDNSF a nonblocking publish API so application/request handlers are less likely to stall the Face event loop.

Selective Ack Strategies

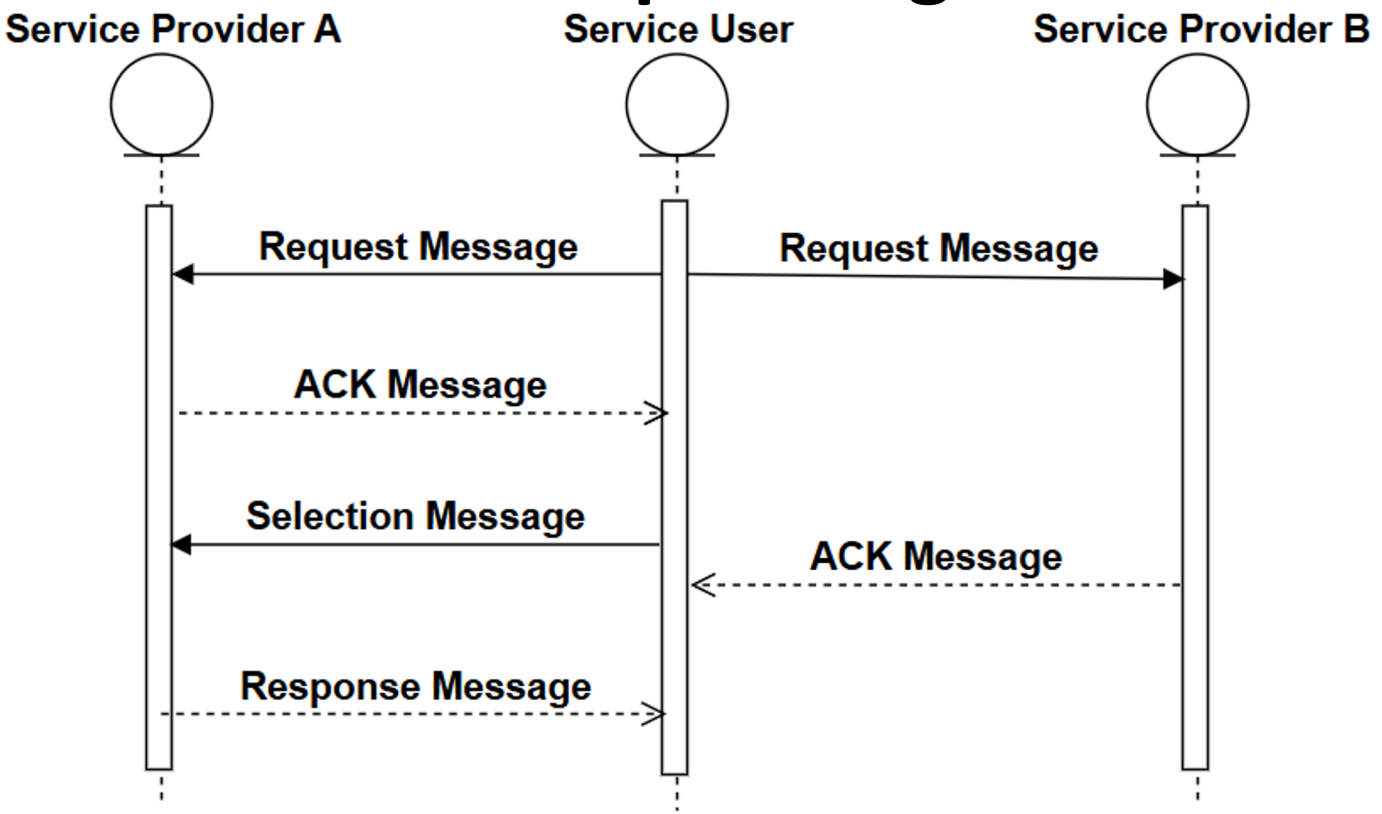
- **Ack-All:** the provider will ack all incoming requests

- **Custom Ack Strategy**: Service providers can autonomously decide whether to respond to incoming requests based on their current runtime status, such as CPU/GPU utilization, task queue size, and other resource conditions. When a provider is overloaded, choosing not to send an ACK can prevent users from experiencing unnecessary waiting delays.

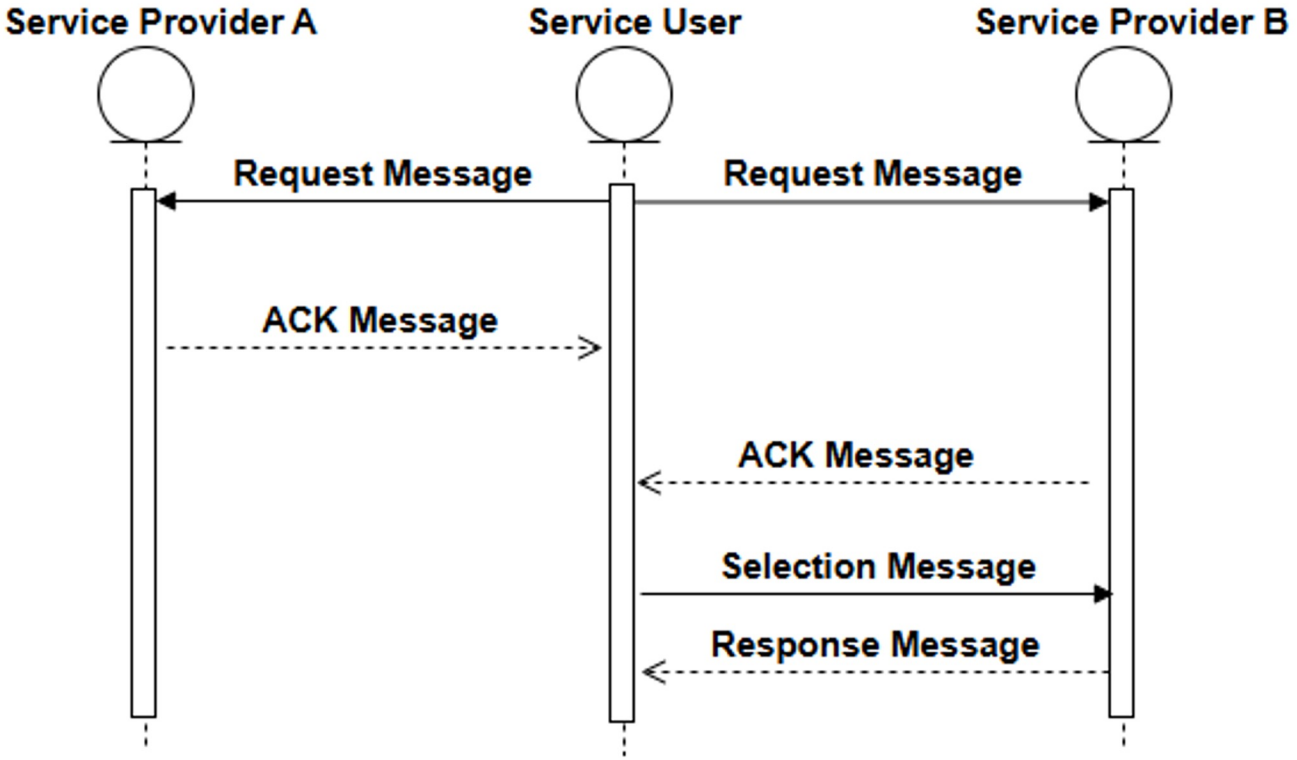
Service Selection Strategies

- **First Responding:** the service user will choose the first service provider that responded to handle the request.
- **Random Selection:** the service user will wait for a while, and then make a random selection from the providers that responded. If no response was received during the wait time, it will select the first responding provider.
- **All Responders:** the user will select all the service providers that returned an ACK to process the request
- **Custom Strategy**: the user will select one or more service providers among those that returned an ACK based on measurement information provided by NDNSF.

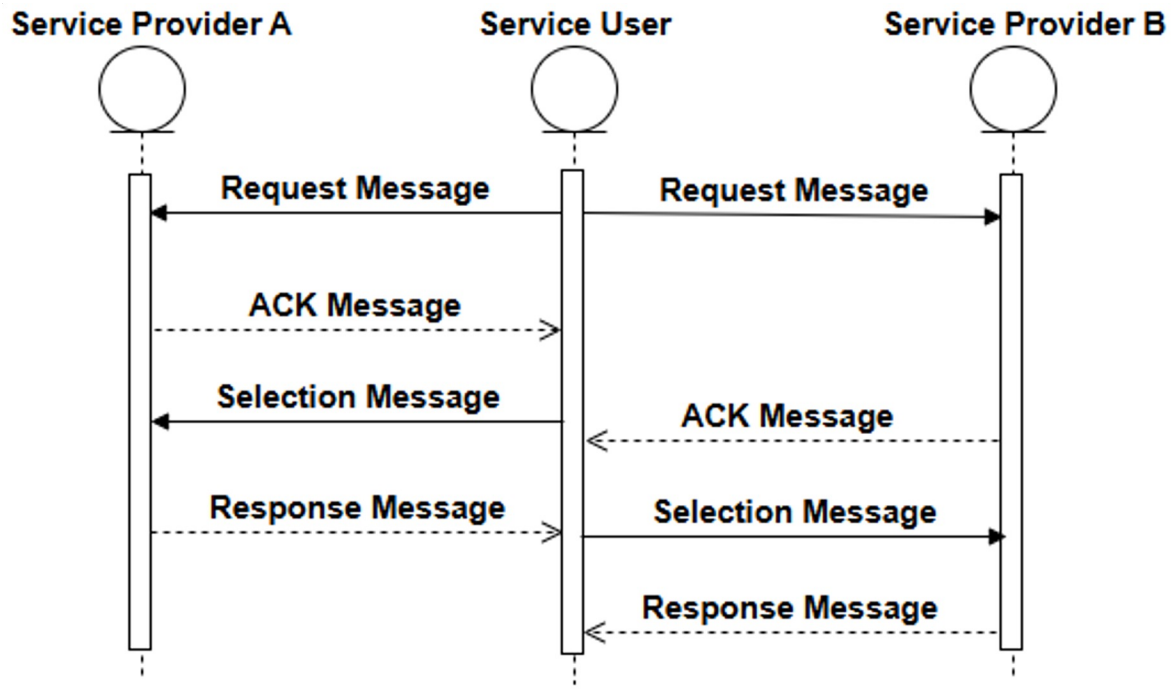
Service Selection Strategy: First Responding



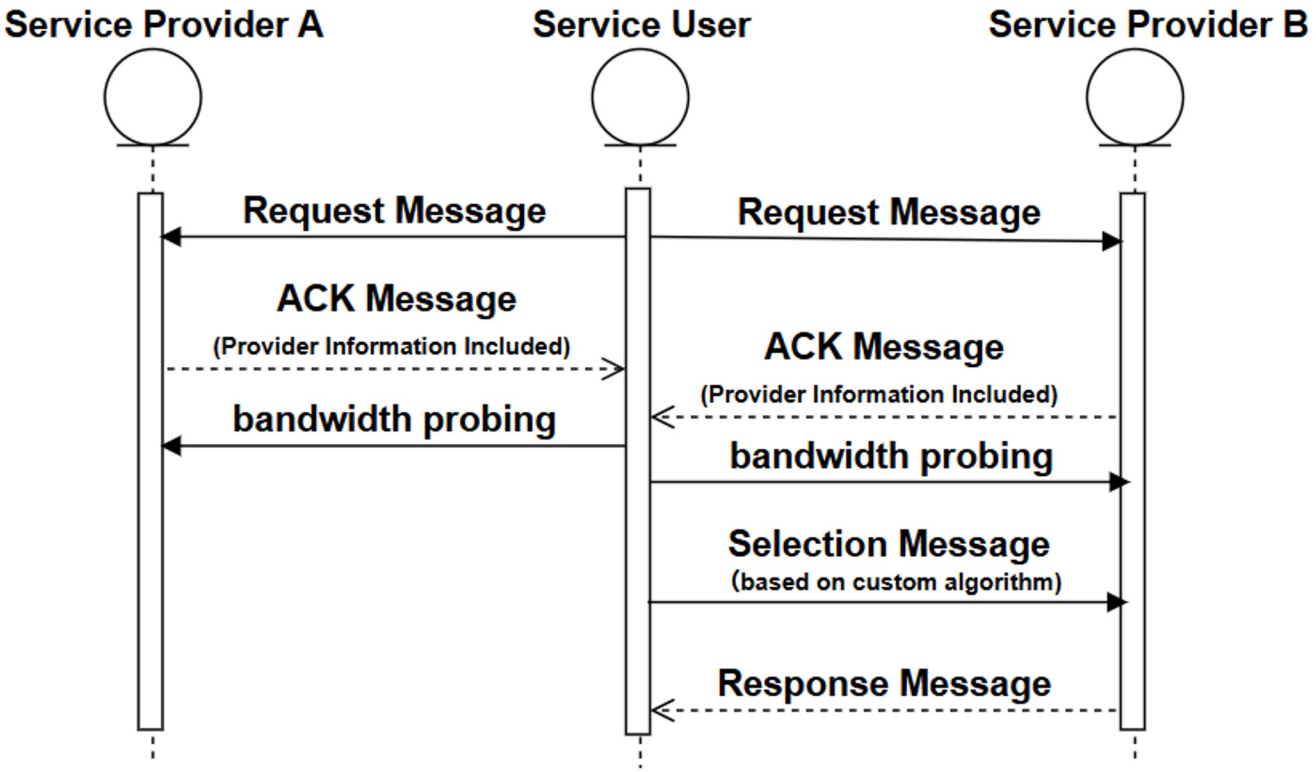
Service Selection Strategy: Random Selection



Service Selection Strategy: All Responders



Service Selection Strategy: Custom



Service Authorization and Access Control

- NDNSF uses attribute-based encryption so only authorized recipients can decrypt messages.
- Request/selection messages are encrypted with /SERVICE/<service> attributes so only providers offering that service can decrypt them.
- ACK/response messages are encrypted with /PERMISSION/<service> attributes so only permitted users can access them.
- One-time tokens are included to prevent replay attacks.

NDNSF App and API Example (1)

- We now use a simple flight control example to demonstrate how to build an application using NDNSF. In this example application, the ground station GS1 sends a takeoff command to the drone Drone1, and upon executing the command, Drone1 returns its current GPS information to GS1.
- First, the app defines the format of the request and response data, and then defines the takeoff service for Drone1:

```
message FlightControl_Takeoff_Request
{
  float latitude = 1;
  float longitude = 2;
}
message FlightControl_Takeoff_Response
{
  float latitude = 1;
  float longitude = 2;
}
```

```
serviceName: /muas/drone1/Takeoff
serviceDescription: FlightControl using MAVLINK
requestMessage: muas::FlightControl_Takeoff_Request
responseMessage: muas::FlightControl_Takeoff_Response
```

- The app.yml file defines services and their message types. By running **`sudo python NDNSFCodeGenerator.py`**, the script generates corresponding files in the Generated folder based on the templates defined in the Template folder.

NDNSF Service Info API

- Service providers use PublishServiceInfo to publish information about the current service. If information is published under the same service name, it will be updated.

```
namespace Provider {

    bool PublishServiceInfo(Name serviceName, std::string info_json);

}
```

- Users can first discover the services available in the system, and for a specific service name, they can retrieve the provider names along with detailed information

```
namespace User {

    std::vector<Name> DiscoverServices();

    std::map<Name, std::string> RetrieveServiceInfo(Name serviceName);

}
```

NDNSF App and API Example (2)

- The service provider needs to prepare two handlers for each service:
  - The onRequestHandler() determines whether to provide the service based on the provider's current state
  - The onSelectionHandler() processes the actual request and returns the result.

```
using OnRequestHandler = std::function<bool(const Name& serviceName,
                                            const Request& request)>;

using OnSelectionHandler = std::function<Response(const Name& serviceName,
                                                  const Request& request)>;

namespace Provider {

    bool RegisterService(const Name& serviceName,
                         OnRequestHandler onRequestHandler,
                         OnSelectionHandler onSelectionHandler);

}
```

NDNSF App and API Example (3)

- The service user needs to select one the built-in strategies or define a custom SelectionStrategy, which selects one or more service providers based on the received ACKs and the latest information about the providers.

```
using OnResponse = std::function<void(const Name& providerName,
                                      const Response& response)>;

using SelectionStrategy = std::function<std::vector<Name>(
    const std::vector<Ack>& ackList,
    const std::map<Name, std::string>& providerInfoMap)>;

namespace User {

    bool InvokeService(const Name& serviceName,
                       const Request& requestBody,
                       SelectionStrategy strategy,
                       OnResponse onResponse);

}
```

Experiments - A

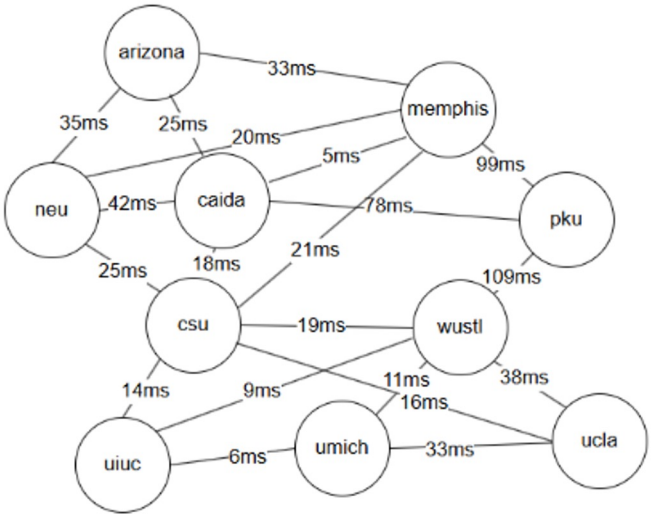


Fig. 11: Wired Network Topology

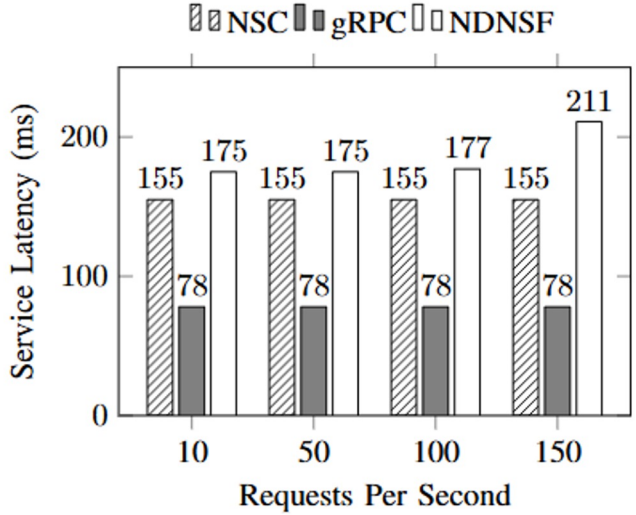


Fig. 13: Average Service Latency in Wired Network (Loss=0%); This experimental data was collected between two nodes: Memphis and UCLA.

Experiments - B

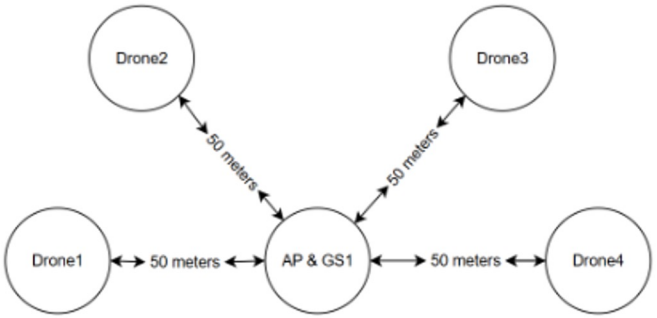


Fig. 12: Wireless Network Topology

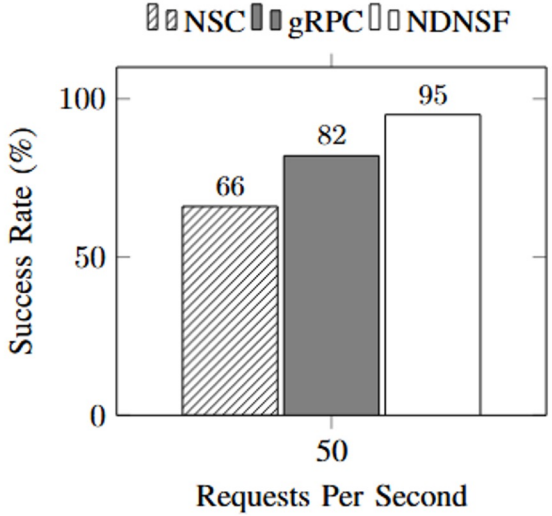


Fig. 17: Success Rate at 50 Requests per Second in a Wireless Mobile Network. All drone nodes provide identical services; however, due to continuous mobility, they may move out of the communication range of the ground station, making their services temporarily unavailable.

NDNSF as Distributed Inference Coordinator

- NDNSF extends from:
  - **single-provider** service invocation
  - to:
  - **multi-provider collaborative** inference
- NDNSF provides:
  - provider discovery
  - work group formation
  - work plan distribution
  - named data exchange
  - secure service coordination

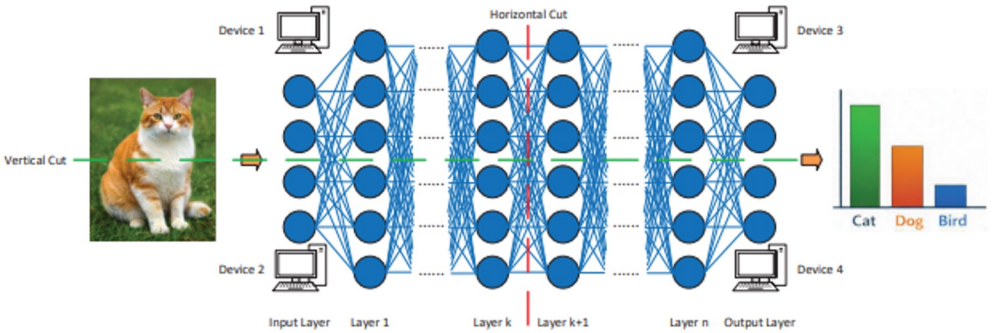


Figure 1: An Illustration of Horizontal and Vertical Cuttings on a Deep Learning Model for Decentralized Processing.

Work Group Formation

- NDNSF forms a work group for each inference request.
- A work group includes:
  - selected providers
  - assigned roles
  - data dependencies
  - communication relationships
- Selection is based on observed metrics:
  - available compute resources
  - model partition availability
  - queue delay
  - measured bandwidth
  - observed RTT

How NDNSF Supports Collaborative Distributed Inference

- Multiple devices collaborating
  - Performance-aware Work group formation
- Data exchange across stages
  - Named intermediate data
- Synchronization among nodes
  - Data-driven coordination
- Security
  - Per-stage access control

Summary

- NDNSF is a secure distributed service framework for Named Data Networking (NDN).
- NDNSF supports:
  - distributed service publication and discovery
  - secure service invocation
  - flexible service selection
  - multi-provider collaborative applications
- NDNSF introduces:
  - selective ACK strategies
  - information-aware service selection
  - adaptive admission control and overload mitigation
- Future work:
  - further optimization of control-plane scalability
  - extends to collaborative distributed inference