Investigating REST API in NDN (Version 2)

Tai-Lin Chu

Feb 15, 2015

1 Introduction

If REST API is directly implemented in NDN, it will likely increase the interest name or number of round trips. Given that current REST architecture is built on top of HTTP, fitting it directly in NDN seems inappropriate. The goal of this project is to investigate software architecture that not only has properties of REST but also is native to NDN.

2 REST Constraint

In the original paper of REST architecture, the constraints guide the design.

1. client-server/separation UI from data
2. stateless communication
3. cache
4. uniform interface
    - resource id
    - manipulate resource through representation
    - self-descriptive msg
    - hypermedia as the engine of app state
5. layered system/encap
6. (optional) code on demand

3 Relate REST Constraints to NDN

1. The notion of client-server is replaced by data consumer-producer. The client is still one machine, but the server might be multiple machines.

2. Statelessness is a requirement of caching.
3. Caching is well-supported in NDN. No effort needs to be done.
4. The resource id is NDN name. It is also non-trivial how to manipulate resources through representation in NDN. NDN name is self-descriptive.
5. Hierarchical/layered system is a property of NDN naming convention. No effort needs to be done.

From this analysis, the problems to solve are:

1. client-server/consumer-producer communication
2. manipulate resource with NDN packet

4 Other constraints from NDN

4.1 efficiency

See Ilya's paper for issues related to *efficiency*. Here only new issues are discussed.

4.2 anonymity

Both client and server can be data producer and consumer. However, when NDN client becomes data producer, it still needs to be anonymous to the network. In other words, the client (normally data consumer) cannot register prefixes. Previous works do not address this issue.

4.3 minimum modification to NDN architecture

The solutions that modify packet processing, packet format and overall architecture should be avoided. If such modifications are needed, they should be minimal.

4.4 exploit statefulness of network

Use the statefulness property of NDN as an advantage.

5 Manipulate resource with NDN packet

Common manipulations are CRUD (CREATE, READ, UPDATE, DELETE). READ is supported already.

- CREATE in NDN implies the system needs to give an unique name to a newly allocated resource.
- UPDATE in NDN implies the system needs to give an unique and incrementing version number.

In either case, the system needs to be stateful in resource allocation, and the data flows from client to server.

6 Proposed solution: backward pit entry

Changes:

- new dataType: pendingData
- interest/data processing pipeline

When a client wants to send data to a server, it first sends an interest. All pit entries along the path are added as usual. When server responds to client, it sends **pendingData**, which is a data packet that has an unique name (**pendingData name**) and empty content. All pit entries along the path are erased when it satisfies the interest, but at the same time, backward pit entries are created along the path with *pendingData name*. Finally the client will send data with exact *pendingData name* to the server. The server then checks the client identity from the data packet.

This solution preserves the asymmetric communication model of NDN, and has no change to the packet format. The only change is that all NDN forwarder needs to add a processing rule to handle the new dataType: pendingData. It leverages static consumer (server) to bootstrap the communication.

This solution achieves anonymity by *not* requiring clients to register prefixes.

This solution also works around the problems presented in Ilya's paper.

6.1 DDOS on static consumer (server)?

It seems that the communication is susceptible to DDOS attack. But:

1. There are multiple servers that can answer interest from client.
2. The servers can decide whether to respond with pendingData depending on the workload.

7 Future work

PendingData is a viable solution with the following properties:

1. efficient
2. simple (minimal changes)
3. address REST constraints
4. anonymous

By using static consumer, producer can start sending data to consumer without routing update.