Consumer-Producer API for Named Data Networking

Ilya Moiseenko
UCLA
iliamo@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

This paper presents a new network programming interface to NDN communication protocols and architectural modules. This new API is made of (1) a consumer context which associates a name prefix with consumer-specific data fetch parameters controlling Interest transmission and Data packet processing, and (2) a producer context which associates a name prefix with producer-specific data transfer parameters controlling Interests demultiplexing and Data packet production. Both API contexts are extensible to new functionalities once they are identified.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems; D.2 [**SOFTWARE ENGINEERING**]: Software Libraries

Keywords

NDN; API; data producer and consumer

1. INTRODUCTION

As a new architecture, NDN requires a new API. Today's socket API cannot be reused for NDN communication because its foundational concept is point-to-point virtual channel that does not exist in NDN. The NDN architecture development has been following an application-driven approach by going through the cycles of design $\rightarrow$ experimenting with pilot applications $\rightarrow$ revision. Our experience with pilot NDN applications [1], [2], [3] has provided us with enough hints to sketch a new NDN API, then we can put it back to application development to verify and validate.

Unlike TCP/IP's point-to-point data delivery, where data transfer parameters are the properties of the channel between two endpoints, in NDN network, data transfer parameters are the properties of the namespace and the node that produces/consumes the data in that namespace. Note that producer and consumer applications of the same namespace do not directly talk to each other, thus they do not share the same set of data transfer parameters as the two endpoints do in TCP/IP networks.

ICN'14, September 24–26, 2014, Paris, France.
ACM 978-1-4503-3206-4/14/09.
http://dx.doi.org/10.1145/2660129.2660158.

Our proposed NDN API has two programming abstractions: consumer context and producer context. A context keeps all necessary state of ongoing data transfer related to a specific name prefix. A contexts allow the following operations:

$$setOption(option,\ value)$$
$$consume(name\,prefix),\,or$$
$$produce(name\,prefix,\ content)$$

Consumer context is an abstraction that assists application designer to perform unreliable or reliable retrieval of potentially multi-segment content of a given name prefix. The context can perform packet ordering, packet reassembly, as well as give access to raw Interest and Data packets. It also provides event notifications to enable application designers to closely monitor data delivery progress and various errors that may occur in the process.

Producer context is an abstraction that assists application designer to publish single or multi-segment content under a specified name prefix.

Both consumer and producer contexts allow application designers to plug in user-defined, content-based security actions to secure outgoing Interests or Data packets, and verify incoming packets.

2. CONSUMER CONTEXT

Consumer context associates a name prefix with a set of data fetching, transmission, and verification parameters, and integrates processing of Interest and Data packets on the consumer side. An application designer interacts with consumer context by calling API primitives listed in Table 1 and supplying callback functions to process events that may be triggered by the consumer context.

consumer(name prefix, type, sequencing) ➔ handle
consume (handle, name suffix) **stop** (handle) **close** (handle) **setcontextopt** (handle, option name, value) **getcontextopt** (handle, option name) ➔ value

Table 1: API primitives for consuming data.

The first thing an application designer must do is to initialize a consumer context with a desired name prefix and data transfer parameters. The name prefix is a meaningful application-specific name that is expected to bring Data packet(s) back. The required data transfer parameters specify what type of protocol machinery is to be used inside the consumer context. For example, specifying the pair *(UNRELIABLE, DATAGRAM)* instructs consumer context to send a single Interest and receive a single Data packet, whereas the pair *(RELIABLE, SEQUENCE)* results in consumer context to involve necessary machinery to send multiple Interest packets, perform Interest retransmission when needed, manage flow control window size, and reassemble received Data packets.

Any parameter of the consumer context can be obtained or modified using *get/setcontextopt()* API primitives. An application designer can specify what Interest selectors, what flow & congestion control parameters, what size of receive and send buffers to be used by the consumer context during the data transfer. In addition, callback functions can be passed as an argument to the *setcontextopt()* primitive to plug in user defined actions in packet processing pipeline. For example, when consumer context has reassembled enough content from incoming Data packets, it executes *ContentCallback* to return the content to the application. As another example, *VerificationCallback* accepts a Data packet to perform customized Data verification operations. Other callbacks can be activated to monitor events such as Interest timeouts, Data packet arrival, etc.

When all context parameters are set, an application designer can start data transfer using *consume()* primitive that accepts name suffix which augments name prefix of the consumer context. Name suffix, such as version component, provides the flexibility of fetching multiple data objects without having to recreate a consumer context for every object.

Data fetching in the consumer context can progress in non-blocking way. An application designer can terminate the transfer at any moment by calling the *stop()* primitive, which will reset the consumer context to its initial state.

When a consumer context is no longer needed, an application designer can release all its associated resources by executing *close()* primitive.

3. PRODUCER CONTEXT

Producer context associates a name prefix with a set of packet framing, caching, content-based security, and namespace registration parameters, and integrates processing of Interest and Data packets on the producer side. An application designer interacts with producer context by calling API primitives listed in Table 2 and supplying callback functions to process events that may be triggered by the producer context.

producer (name prefix) ➔ handle
produce (handle, name suffix, content)
setup (handle)
close (handle)
setcontextopt (handle, option name, value)
getcontextopt (handle, option name) ➔ value

Table 2: API primitives for producing data.

The application designer must first initialize a producer context with a desired name prefix and parameters for data publishing. The name prefix is to be used for publishing content under it, and demultiplexing incoming Interest packets.

Any parameter of the consumer context can be obtained and modified using *get/setcontextopt()* API primitives. An application designer can specify the size, freshness and security properties of Data packets. In addition, callback functions can be passed as an argument to the *setcontextopt()* primitive to plug in user defined actions in the packet processing pipeline.

Prior to publishing any content, *setup()* primitive must be called in order to set up Interest demultiplexing by name prefix, and to acquire a routable prefix using the built-in prefix discovery/registration protocol (similar to [4]) in cases when Interest packets need to be routed to the producer.

An application designer can seamlessly transform any raw content (e.g. memory buffer) into Data packets with *produce()* primitive. The producer context will use its own parameters to package the content in a right number of Data segments (packets) that fully conform with naming and other packet conventions.

When a producer context is no longer needed, an application designer executes *close()* primitive.

4. USING NDN API CONTEXTS

We use NDN FileSync as a use case to illustrate the new API. NDN Filesync is a distributed peer-to-peer application to support file synchronization in a shared directory [3]. As one of the simple pilot NDN applications, it requires reliable data delivery service, but does not have an elaborate security model.

The application's Interest packets contain a name of the file to be downloaded from any other peer. When an Interest is received, the application parses the name to locate the file on the disk, then packages the file in Data packets. Sample data packet name: */broadcast/apps/filesync/class217/Reports/Report.pdf/<timestamp>*.

Pseudocode 1 Sharing a file

```
 1: $h \leftarrow$ producer("/broadcast/apps/filesync")
 2: setcontextopt($h$, packet_size, 16KB)
 3: setcontextopt($h$, interest_callback, ProcessInterest)
 4: setup($h$)
 5: function ProcessInterest(Interest i)
 6:     $Name\ suffix \leftarrow$ extract file name from i.name to understand what file is needed
 7:     $content \leftarrow$ read file from disk
 8:     $Name\ suffix \leftarrow$ append current time stamp
 9:     produce($h$, $Name\ suffix$, $content$)
10: end function
```

Pseudocode 2 Downloading a file

```
1: $h \leftarrow$ consumer("/broadcast/apps/filesync", RELIABLE, SEQUENCE)
2: setcontextopt($h$, receive_buffer_size, 10MB)
3: setcontextopt($h$, content_callback, ProcessContent)
4: consume($h$, "/class217/Reports/Report.pdf")
5: function ProcessContent(byte[] content)
6:     $file \leftarrow$ read content
7:     Save $file$ on disk
8: end function
```

Interested readers can find other use cases, such as streaming live video (NDNvideo [1]) and building automation system (NDNlighting [2]), from the full version of this paper [5].

5. REFERENCES

[1] D. Kulinski and J. Burke, "NDN Video: Live and Prerecorded Streaming over NDN," NDN, Tech. Rep., 2012.

[2] J. Burke, A. Horn, A. Marianantoni, "Authenticated Lighting Control Using Named Data Networking," NDN, Tech. Rep., 2012. johnbernando@gmail.com

[3] J. Lindblom, M. Huang, J. Burke, L. Zhang, "FileSync/NDN: Peer-to-Peer File Sync over Named Data Networking," NDN, Tech. Rep., 2013.

[4] [Online]. Available: https://www.ccnx.org/releases/latest/doc/technical/Registration.html

[5] I. Moiseenko and L. Zhang, "Consumer-Producer API for Named Data Networking," NDN, Tech. Rep., February 2014. [Online]. Available: http://named-data.net/publications/techreports/tr17-consumer-producer-api/