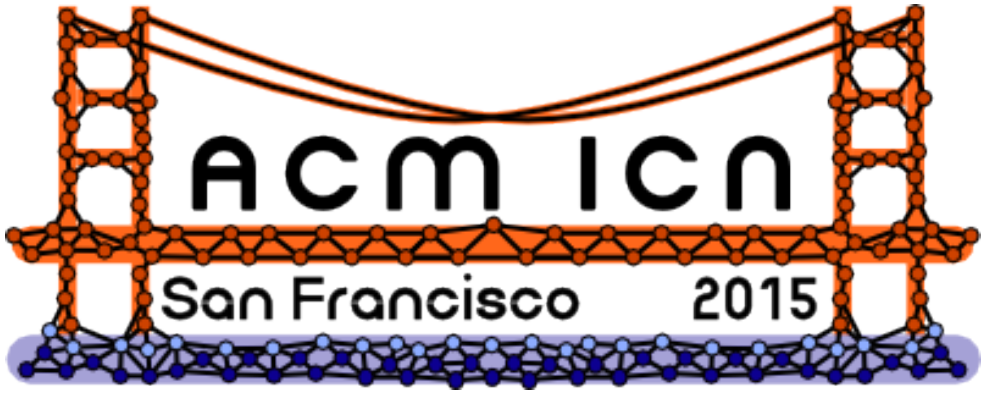
Consumer / Producer Communication with Application Level Framing in Named Data Networking

Ilya Moiseenko
UCLA

Lijing Wang
Tsinghua University

Lixia Zhang
UCLA

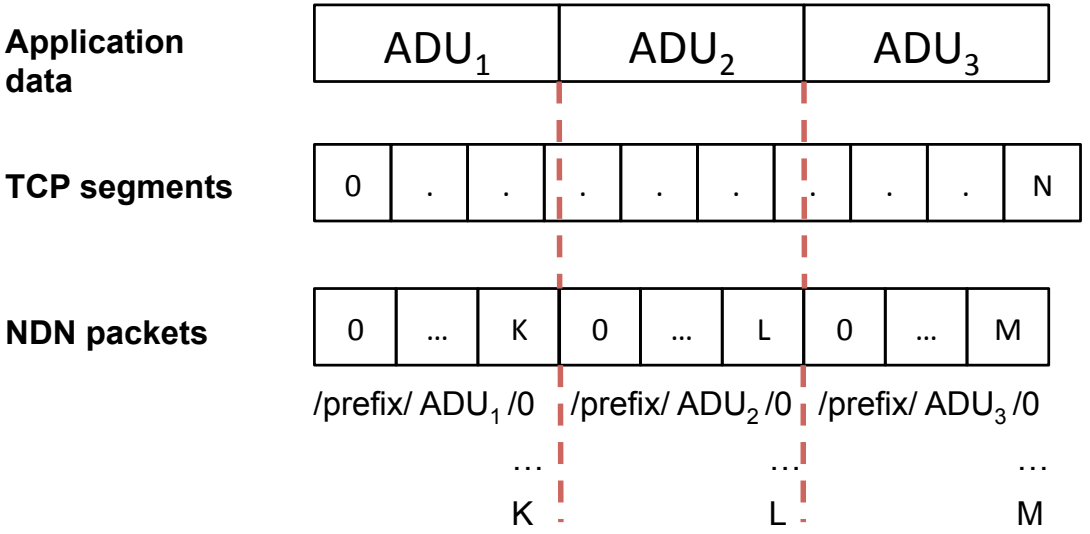


Interest / Data communication model

Up until now, NDN applications had to be implemented on top of Interest / Data API.

- time consuming
- needs a lot of expertise

Application Level Framing



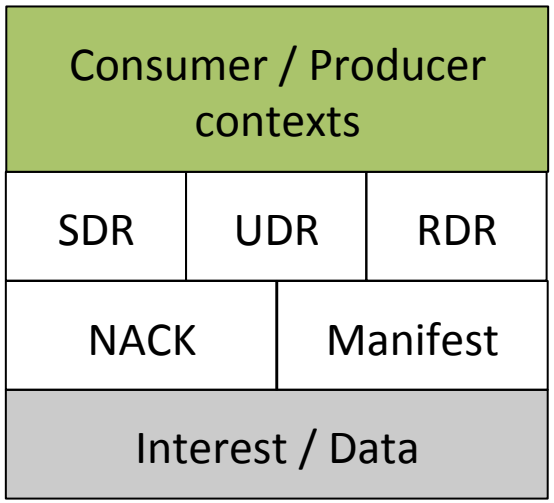
- Network applications work with Application Data Units (ADU)
- ADUs must be segmented, because the packet size is limited by network MTU
- Applications must handle ADU segmentation/reassembly

Research objective

A comprehensive design of a communication abstraction that

- supports ADU principle
- has built-in content fetching and segmentation protocols
- can support a broad range of NDN applications
- significantly simplifies application development

Communication abstractions



Need for two NDN abstractions

- Producer-specific state
  - Data name construction
    - {location dependent prefix, application prefix, ADU suffix}
  - Data segmentation and signing
  - Temporary caching / permanent storage

- Consumer-specific state
  - Interest pipelining and flow control
    - Similar to the sender in TCP
  - Loss and error correction
  - ADU reassembly and verification
  - Interest Selectors

Identified content publication patterns

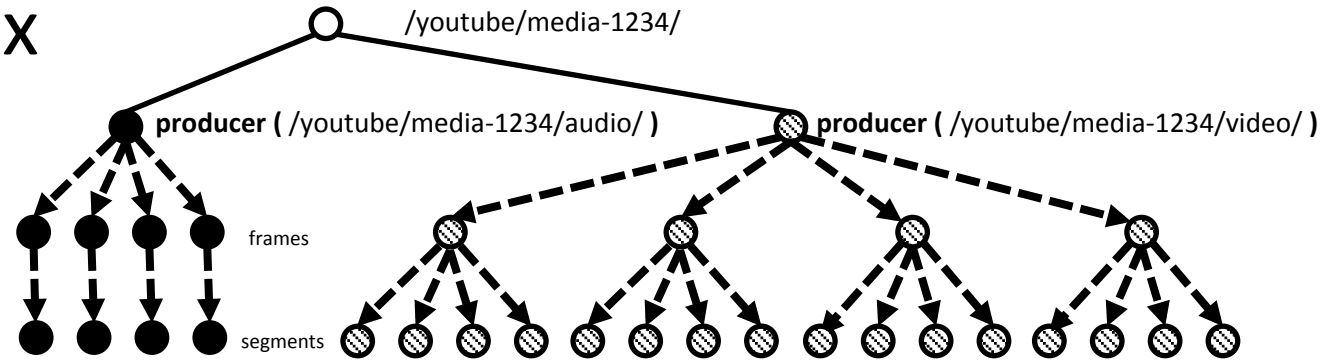
- **Realtime** ADU publishing and consumption
  - Publishers may need to ``wait for pull'' and keep the ADUs in memory temporarily to handle a possible mismatch between production and consumption timing.

- ADU publishing to stable storage, to support potentially **large asynchronies** between ADU publishing and consumption.

- ADU publishing to **remote stable storage**, to support mobile publishers and IoT publishers.

Producer context

Represents the state of data transfer for a specific prefix



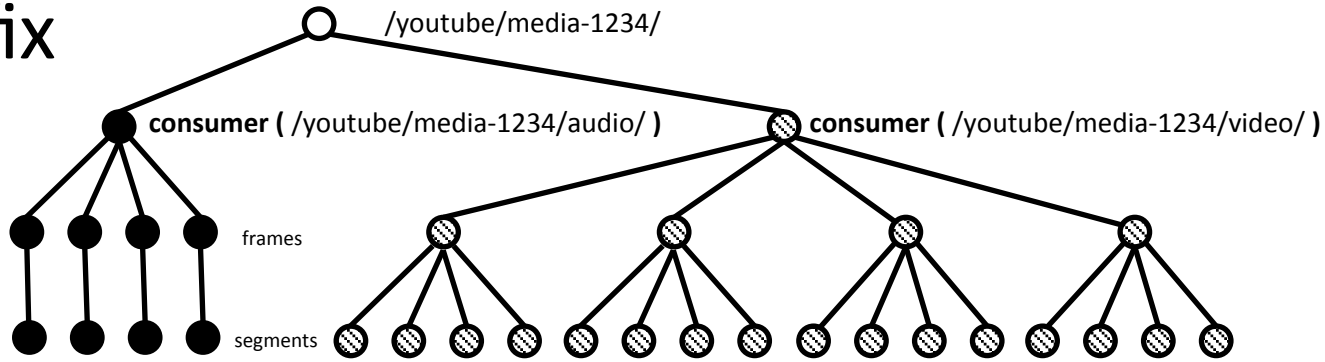
Initialization	**producer** (name prefix) ➔ context
Primitives	**attach** (context) **produce** (context, name suffix, content) **nack** (context, negative acknowledgement) **delete** (context) **setcontextopt** (context, option name, value) **getcontextopt** (context, option name)

Identified content consumption patterns

- **Sequential** fetching of ADUs, with allowance of missing any ADU in the stream if necessary.
- **Parallel** fetching of ADUs to speed up content transfer.
- Fetching of individual, **dynamically generated** ADUs.

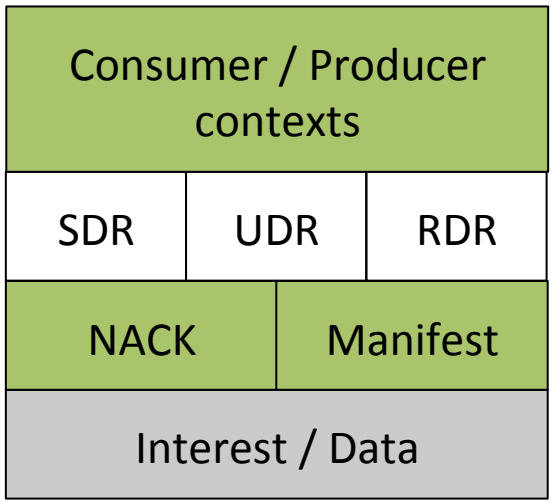
Consumer context

Represents the state of data transfer for a specific prefix



Initialization	**consumer** (name prefix, retrieval protocol) ➔ context
Primitives	**consume** (context, name suffix) **stop** (context) **delete** (context) **setcontextopt** (context, option name, value) **getcontextopt** (context, option name)

Supporting mechanisms



Asynchrony between data production and consumption

- In IP, sockets are synced in time; TCP syncs transmission rate inside the connection
- In NDN, data production and consumption can be at different time and with different rates
- May require careful coordination about availability and specifics of the data
  - Need explicit support in API (e.g. **nack()**)

Data availability problem

- Consumer does not know when exactly content becomes available

- The problem of polling
  - in HTTP, client can long poll
    - requires server to keep the client's state
  - in NDN, there is no underlying TCP to guarantee Interest delivery to the server
    - long poll method is not going to work

NACK as a coordination mechanism

- Application NACK
  - Retry after
  - No data
- Network NACK
  - Congestion
  - No route to destination

Retry-After NACK

Contains a retry timer to be set by the consumer.

- Consumer retransmits interest when timer expires
- Producer doesn't have to keep any state
- Signed and cachable
- NACK must expire in cache before consumer retransmits
  - Packet can be cached at each node during its lifetime period (e.g. link latency is not subtracted from the lifetime)
  - There are multiple nodes on the path, so the NACK at the closest to the consumer node can potentially block retransmissions for a longer period than retry timer.

Data specifics problem

Producer knows a lot about the data, consumer does not, but needs to know the meta-information about ADU or related ADUs in order to fetch desired data most efficiently

- Other versions of content
- Leftmost and rightmost ADU sibling name
- Digests of the Data packets
- etc.

Manifest as a coordination mechanism

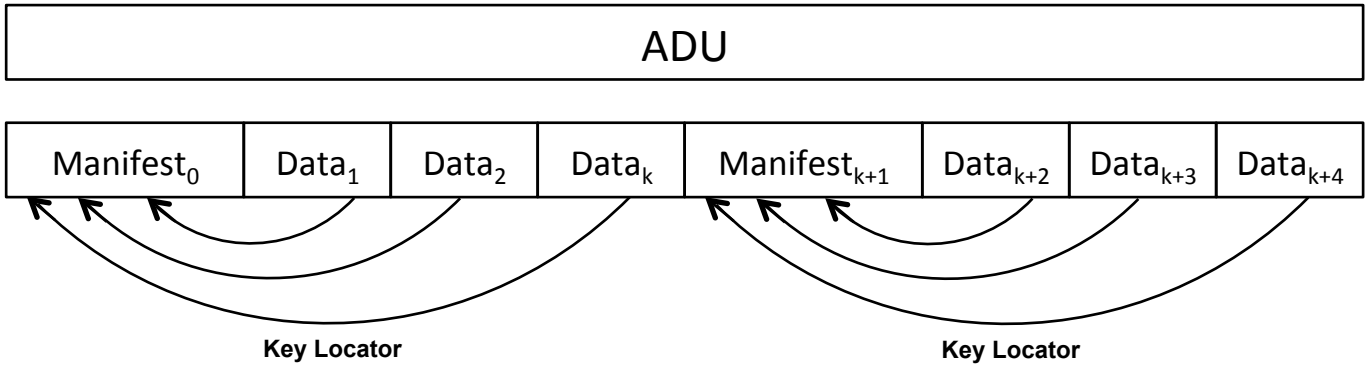
Manifest contains meta-information

- Key-value pairs expressing meta-data information
- Catalogue of names

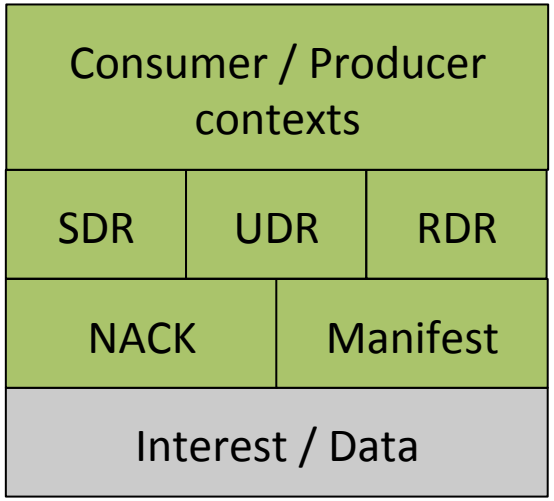
Manifest embedding in ADU

- The original design of catalogues for NDN (M. Baugher, D. Oran 2012)
  - Hash is computed for each Data packet
  - Hash is placed in the catalogue, which is signed
  - Catalogue is fetched prior to Data packets

- Manifest embedding
  - Interleave manifest segments with ADU segments
  - No 1 RTT penalty, because packets are fetched simultaneously
  - 32x speed increase when signing, 17% when verifying



Data Retrieval Protocols



Data retrieval protocols

- Simple Data Retrieval (SDR)
  - single Interest – single Data
- Unreliable Data Retrieval (UDR)
- Reliable Data Retrieval (RDR)

UDR summary

- Unreliable and unordered delivery of a single- or multi-segment ADU
- No ADU reassembly
- Fast retransmission after three out-of-order Data packets
- NACKs passed up to the application
- Flow control optimized for finite size information objects instead of long streams

RDR summary

- Reliable fetching of a single- or multi-segment ADU
  - pre-generated ahead of time
  - dynamically generated upon Interest arrival
- Ordering of ADU segments and reassembly of ADU
- Persistent recovery from Data verification failures
- Understands NACK and Manifest

RDR

So far identified 4 types of errors:

1. the Interest is lost in transit before it reaches the data, which may reside in cache, or need to be produced
   - **Interest retransmission timers**
2. the returning Data packet is lost
   - **Interest retransmission timers**

RDR

So far identified 4 types of errors:

3. the Interest reaches the producer-application but the application does not respond due to various reasons
   - **Negative acknowledgement**
4. the returning Data packet fails the signature validation (e.g. content is poisoned, etc.)
   - **Interest retransmission with exclusion by digest**

Available communication patterns

1. Realtime publishing and consumption
2. Publishing with embedded manifests
3. Publishing into the local Repo (persistent storage)
4. Publishing into the remote Repo
5. Changing forwarding strategy (e.g. broadcast)
6. Sequential fetching (e.g TCP-like stream)
7. Parallel fetching (e.g. QUIC-like stream)

Pilot app #1: NDNtube

- Backend
  - Three producers
  - Publishing in the Repo
- Player
  - More than 50 consumers
  - Parallel fetching with RDR to achieve high throughput

Technical report: http://named-data.net/wp-content/uploads/2015/05/ndn-0031-1-ndnlive-ndntube.pdf

Pilot app #2: NDNlive

- Backend
  - Four producers
  - Realtime publishing with producer's send buffer
- Player
  - More than 50 consumers
  - Parallel fetching with RDR
  - Current video and audio frame name discovery via SDR

Technical report: http://named-data.net/wp-content/uploads/2015/05/ndn-0031-1-ndnlive-ndntube.pdf

Conclusions

- Consumer / Producer model solves the problems that socket model cannot solve
  - enables desirable communication patterns
- Consumer / Producer API abstracts away many difficult things related to NDN programming
- Pilot apps demonstrated that there is no need to revert back to the plain Interest / Data
- More functional than CCNx Portal API

Future work

- Offer more data retrieval protocols
  - InfoMax is in progress at UIUC
- Integration with trust models
  - select trust model in a similar way to how the data retrieval protocol is selected
- Versioning models (integration or built on top)
  - Publishing without name collisions and consuming multi-versioned content

Q / A

Thank you