April 8, 2014
PARC

Reliable Content Exchange of a CCN pipelined flow

Marc Mosko[1]*

Abstract
In CCN, a client may ask a remote service to store content on its behalf. To do this, the client sends an Interest to the service asking it to retrieve the content to be stored. Prior protocols made no guarantee about transfer, in particular that the service downloads the entire object using a pipelined stream. The new CCN Reliable Content Exchange (RCX) protocol allows the service provider to use an efficient sliding window Interest pipeline so the client can discard acknowledged chunks of the transferred data.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

Reliable Content Exchange (RCX) of a pipelined flow means that a client asks a service to pull content from it using an efficient windowed retrieval using a protocol that guarantees transfer, if possible. RCX uses a control channel between the client and service provider to manage the service provider's pull of content from the client.

RCX may be used with simple chunked user data or with more sophisticated secure Manifests. Chunked user data conforms to the Chunking Protocol, where a base name, such as `/foo/bar/slides.pdf` is extended for each Content Object in a series, to form name such as `/foo/bar/slides.pdf/c0`, ..., `/foo/bar/slides.pdf/c19`, where the terminal element indicates the chunk number. In a manifest approach, a chunked object called the Manifest enumerates all the constituent content objects to be transferred, including their Content Object Hash. The manifest is usually hash-chained between chunks of the manifest.

1. Protocol

The protocol uses a command channel between the client and service provider, which manages a sliding window. RCX allows two types of transfer: ChunkedName and ManifestName. In a ChunkedName transfer, the client provides the base name of a set of chunked content objects and the service transfers them with a pipeline of Interests. In a ManifestName transfer, the client provides the name of a Manifest, and the service transfers the Manifest and all items pointed to by the manifest. Data transfer occurs over normal data channels via Interests for chunked names or names derived from a chunked manifest. The sliding window uses a Cumulative ACK Name (CAN) to indicate the largest in-order name received.

The CAN follows the natural order of the objects being transferred. In the case of a ChunkedName transfer, the CAN follows the chunk number ordering. In the case of a ManifestName transfer, the CAN is derived from the natural order of the manifests. We define this order as the first manifest chunk followed by all of its items in order followed by the next manifest chunk followed by all of its items in order, and so on. Of course, the service may transfer items out of order – especially the manifest chunks – but for the purposes of reporting a Cumulative ACK Name, we need to impose a total order on all the content objects.

The CAN indicates the *up to and including* name transferred. We chose this approach because CCN names have a well-defined minimum (the empty name), but no well-defined "next" name except in the special case of a chunked stream using sequence numbers. This approach is different than many other sliding window protocols that choose to use the *up to but not including* approach. Therefore, when a transfer begins, the initial CAN would be the 0-length empty name. This has a further benefit, as it avoids distinguished values and because the empty name cannot actually be transferred on the wire, it does not conflict with any potential transfer name. For convenience of writing, we will indicate this as `EMPTY`.

The service provider manages its receive window by controlling the number of outstanding Interests it has in flight. It may manage this window as appropriate for the current network conditions. We expect some form of co-operative flow control, such as a TCP-like algorithm, but the exact algorithm is beyond the scope of this paper.

The client manages its send window by sending NEXT messages. This message indicates the name of the next element to be published, possibly including a KeyId or Content Object Hash field. The client continues updating the NEXT message as it publishes more content. In the case of a ChunkedName transfer, it is the name of the next content object in the stream. In the case of a ManifestName transfer, the NEXT name is next chunk *of the manifest* to be published.

1.1 Session Initialization

The client requests session initialization from the network service. This may involve providing user credentials, a password, or proving identity via cryptographic keys. The result of the session initialization is a Session ID (SID), and optionally a session key (SK). The session key is securely transferred to the client and maintained via a key rotation protocol, beyond the scope of this document. The session key encrypts user data in names and payloads on the control channel. The encryption algorithm may introduce nonces as necessary, which are not described in the following.q

Within the context of a session, a client may request one or more Exchanges, identified by a client-generated Exchange ID (XID). An Exchange is a specific request for the service to transfer named content. The named content may be either ChunkedNamed, which conforms to the standard chunking protocol. The named content may also be a Manifest (secure catalog), which identifies the exact Content Object names, including hash values, to transfer.

1.2 Exchange Initialization

A client generates a unique Exchange ID (XID), within the context of the current session. The client sends an Interest to the service provider as shown, the encryption is optional:

(1) `/provider/SID/E_sk({XID,ChunkedName, [KeyId|Hash],START})`

In Interest 1, the prefix `/provider` identifies the service. It contains a routable prefix plus the name of the service. The next element identifies the session, which implies a Session Key. The next element is a marshaled data structure, optionally encrypted with the session key. An implementation may use, for example, a TLV binary format or XDR or some other transfer encoding. The Exchange ID (XID) is generated by the client and can be any unique number, though we recommend using a random number. The ChunkedNamed conforms to the standard chunking protocol. If it is a base name without a chunk number (e.g. `/foo/bar`), the service provider will open a pipeline to transfer all the chunks (e.g. `/foo/bar/c0`, ..., `/foo/bar/c99`). It may also be a single chunk name (e.g. `/foo/bar/c33`) to transfer an individual chunk, though this usage is less commonly used.

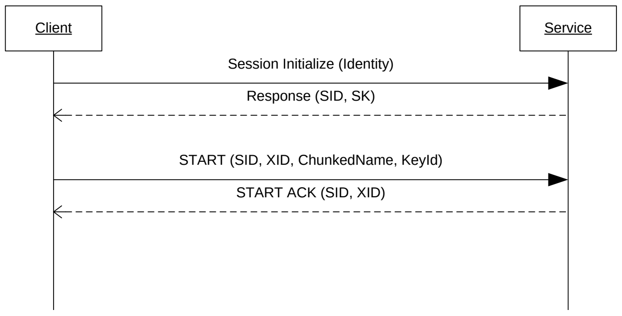
Client
Service
Session Initialize (Identity)
Response (SID, SK)
START (SID, XID, ChunkedName, KeyId)
START ACK (SID, XID)

Figure 1. Chunked Name START

The optional KeyId specifies the KeyId restriction to include in the Interest streams. The optional Hash is the Content Object Hash restriction to use in the Interest, though this field is only useful for transferring a single chunk.

The final element START indicates that this is a request to start the exchange.

Fig. 1 illustrates starting a transfer for a chunked content object, without showing the optional name and data encryption via the session key (SK). The client first associates with the service and obtains a SID and SK. It then requests that a transfer begin for `ChunkedName`, providing the SID and XID. Because this is a valid session, the service sends a START ACK back indicating it will commence the transfer.

(2) `/provider/SID/E_sk({XID,ManifestName, [KeyId|Hash],START})`

In Interest 2, the element ManifestName identifies a secure catalog that enumerates all other items to transfer. The ManifestName always complies with the chunking protocol. The KeyId is the KeyId restriction for use in the Interest for the manifest. The Hash is the ContentObjectHash restriction for chunk 0 of the manifest, which we assume is then hash chained to later manifest chunks. The service may begin downloading manifest items before receiving the entire manifest.

Upon receiving a START request from an authenticated Session, the service will send back an Acknowledgement (ACK) Content Object to the client informing the client that it is beginning. If the client does not receive an ACK, it should retransmit the START using the same XID each time to provide idempotence.

1.3 Content Transfer

After sending the START ACK, the service will begin transferring content using a multiplicity of Interests. In the case of transferring a ChunkedName, the interests are for the chunks of the single object. In the case of transferring a ManifestName, the interests are for a mixture of the Manifest and the items enumerated in the Manifest.

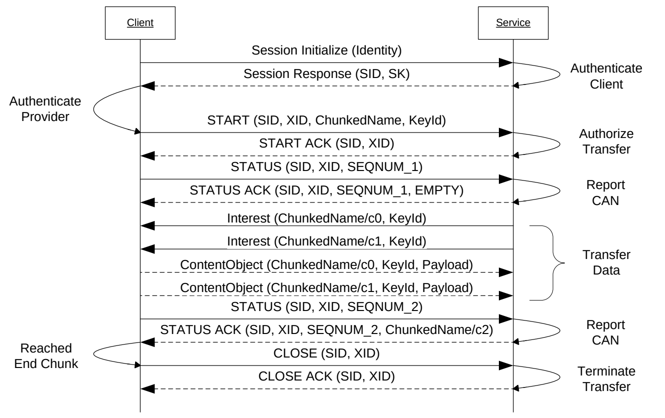
Client
Service
Session Initialize (Identity)
Session Response (SID, SK)
Authenticate Client
Authenticate Provider
START (SID, XID, ChunkedName, KeyId)
START ACK (SID, XID)
Authorize Transfer
STATUS (SID, XID, SEQNUM_1)
STATUS ACK (SID, XID, SEQNUM_1, EMPTY)
Report CAN
Interest (ChunkedName/c0, KeyId)
Interest (ChunkedName/c1, KeyId)
ContentObject (ChunkedName/c0, KeyId, Payload)
ContentObject (ChunkedName/c1, KeyId, Payload)
Transfer Data
STATUS (SID, XID, SEQNUM_2)
STATUS ACK (SID, XID, SEQNUM_2, ChunkedName/c2)
Report CAN
Reached End Chunk
CLOSE (SID, XID)
CLOSE ACK (SID, XID)
Terminate Transfer

Figure 2. Example ChunkedName transfer

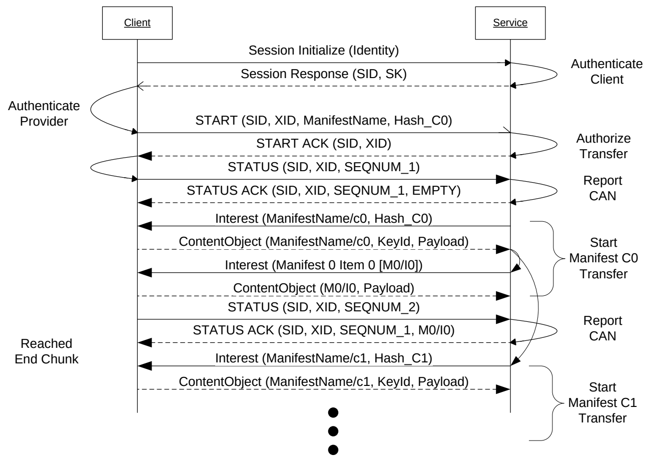
Client
Service
Session Initialize (Identity)
Session Response (SID, SK)
Authenticate Client
Authenticate Provider
START (SID, XID, ManifestName, Hash_C0)
START ACK (SID, XID)
Authorize Transfer
STATUS (SID, XID, SEQNUM_1)
STATUS ACK (SID, XID, SEQNUM_1, EMPTY)
Report CAN
Interest (ManifestName/c0, Hash_C0)
ContentObject (ManifestName/c0, KeyId, Payload)
Interest (Manifest 0 Item 0 [M0/I0])
ContentObject (M0/I0, Payload)
Start Manifest C0 Transfer
STATUS (SID, XID, SEQNUM_2)
STATUS ACK (SID, XID, SEQNUM_1, M0/I0)
Report CAN
Reached End Chunk
Interest (ManifestName/c1, Hash_C1)
ContentObject (ManifestName/c1, KeyId, Payload)
Start Manifest C1 Transfer

Figure 3. Example of ManifestName transfer

The client begins issuing STATUS Interests to the service, as shown in Interest 3. The field SeqNum is a sequence number for the status requests, to differentiate the requests and allow the matching of replies to requests. The field STATUS identifies the message as a status request.

(3) `/provider/SID/E_sk({XID,SeqNum,STATUS})`

As the service transfers content objects in their natural order, it tracks the minimum in-order name not transferred. This is the Cumulative ACK Name (CAN) value sent in response to a STATUS request. For example, suppose a client requests to transfer `/foo/bar` as a Chunked Name. At first, the service has not transferred any content objects, so the CAN to `EMPTY`, meaning it has received up to and including the empty name (i.e. nothing). The CAN is included in a normal Content Object sent in response to a specific STATUS Interest.

The service provider should respond promptly when it receives a STATUS message. The client may use the lack of response as an indication that the service is non-functional. The service provider may also request that the client send STATUS messages within a certain rate window to determine if the client is still connected. If the service provider wishes to set lower and upper rate bounds on STATUS messages, it should be done in the SETUP ACK or in STATUS responses.

Once the client receives a Cumulative ACK Name in a status response, it may discard content objects that occur up to an including the CAN in the natural order of the exchange. For example, if an application is streaming a large file, the CCN stack does not need to retain chunks of the file that have been successfully transfered.

Fig. 2 illustrates a transfer of a chunked object with 2 chunks. We do not show specific cryptographic fields used for trusted authentication, such as nonces, to prevent replay attacks or other attacks. The client first associates with the service provider, who authenticates the client and issues a SID. The Response is a CCN signed content object, so the client can authenticate the service provider. Within the context of that authenticated session, the client asks the service provider to transfer a ChunkedName. The service provider may apply access controls or other measures to authorize the SID for the transfer, and on success returns a START ACK. Once the client has a START ACK, it can expect Interests from the service provider to transfer the data. The client should begin issuing STATUS queries. The figure shows a first status query returning EMPTY because the service provider has not received any content. After it has retrieved the content, the client issues another STATUS query, to which the service provider responds `ChunkedName/c2`. Because that is the last chunk, the client closes the exchange with a CLOSE message, for which the service provider sends a CLOSE ACK. At this point, the service provider reliably transferred the client's data and both sides have cleanly closed the exchange.

Fig. 3 illustrates using a ManifestName transfer. As in the ChunkedName transfer, the client first establishes a SID and XID with the service provider. The XID is created with a ManifestName, and the provided `HASH_C0` is the Content Object Hash of the first chunk of the manifest. Therefore, the service provider transfers the first chunk of the manifest by both name and hash value. Immediately after receiving the first chunk of the manifest, the service provider can begin transferring the items listed in the manifest, such as the first item of chunk 0 of the manifest, denoted as “Manifest 0 Item 0” (`M0/I0`). This transfer continues until all items listed in that manifest are transferred. The service provider may also download the next chunks of the Manifest itself while downloading the items of the first manifest. As shown, the service provider begins downloading chunk 1 of the manifest while it is still downloading the contents of chunk 0 of the manifest. The hash name of chunk 1 of the manifest comes from the hash chain of manifest chunks, if available.

1.4 Detecting End of Exchange

If a Chunked object has an EndChunkNumber in the series of chunks, the service provider will use that to limit the number of Interests it expresses. Otherwise, the producer (client) can

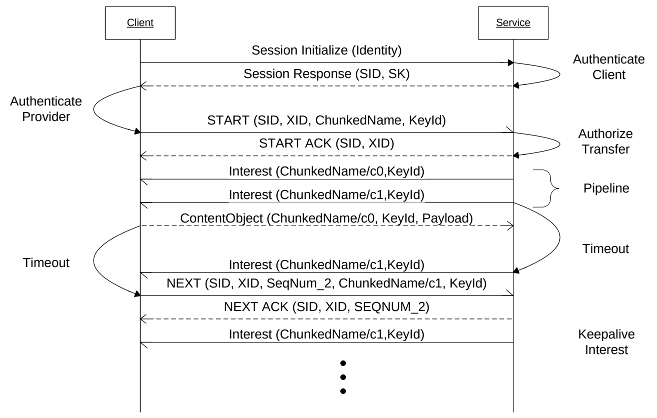
Client
Service
Session Initialize (Identity)
Authenticate Client
Session Response (SID, SK)
Authenticate Provider
START (SID, XID, ChunkedName, KeyId)
Authorize Transfer
START ACK (SID, XID)
Interest (ChunkedName/c0,KeyId)
Pipeline
Interest (ChunkedName/c1,KeyId)
ContentObject (ChunkedName/c0, KeyId, Payload)
Timeout
Timeout
Interest (ChunkedName/c1,KeyId)
NEXT (SID, XID, SeqNum_2, ChunkedName/c1, KeyId)
NEXT ACK (SID, XID, SEQNUM_2)
Interest (ChunkedName/c1,KeyId)
Keepalive Interest

Figure 4. Example of live stream with send window control

limit the send window with NEXT messages. Using NEXT messages tells the service provider the name of the next chunk of a ChunkedName or ManifestName that it can transfer. If an interest for that name goes without an answer, it means the client is stalled producing the content and the publisher should keep trying. If the stream is stalled for a significant duration, the client should keep issuing NEXT messages, which the service provider ACKs, in addition to sending an Interest for the NEXT name. This assures both parties that the transfer is still alive, but stalled.

If a chunked object does not have an EndChunkNumber, such as a live stream, the client may indicate the last published chunk number to the service provider, so the provider can detect the difference between a stalled stream due to no more content and an error condition. It does this by sending a NEXT message, as shown in Interest 4. The NextName is the full name (i.e. with chunk number) of the next chunk to be published, so the service provider can keep at least one interest alive waiting for more content.

(4) `/provider/SID/E_sk({XID,SegNum, NextName,NEXT})`

In the case of a ManifestName transfer, the NEXT message's NextName indicates the next Manifest chunk to be published, which would then dereference to more content objects.

Once the service provider receives the NextName in response to it's Interest, it should begin pipelining the stream again, unless it receives a new NEXT message.

The SegNum field of the NEXT message must be increasing with each new value of NextName. This is to ensure that the service provider only considers the most current message.

The service provider will respond to a NEXT message with an ACK. If the client does not receive an ACK, it should retransmit the NEXT message.

Fig. 4 illustrates a live stream that uses sender window flow control. After starting the transfer the service provider starts a pipeline transfer of the chunked named by issuing interests for chunks `C0` and `C1`. The first chunk was published and returned to the service provider, but the second chunk has not yet been published, so the first timeout for `C1` times out. The service provider tries again, until the client also times out and sends a NEXT notification so the service provider knows that `C1` is the next one to be published.

Note that the client may use different strategies on when to send a NEXT message. In Fig. **??**, the client used a timeout after publishing `C0` before sending the NEXT message. A client, however, could use a different strategy such as sending a NEXT message with each STATUS message.

1.5 Aborting Exchange

At any time, the client may send an ABORT message to the service, as shown in Interest 5, or the service may response to a STATUS message with an ABORT. The ABORT indicates that the exchange is over.

(5) `/provider/SID/E_sk({XID,ABORT})`

1.6 Exchange Termination

Upon the client sending a STATUS message and the service responding with a cumulative ack END, the client shall send an Exchange Close message to inform the service that it acknowledges the exchange is at its end, as shown in Interest 6.

(6) `/provider/SID/E_sk({XID,CLOSE})`

The service provider will return an ACK Content Object to the Close.

1.7 Session Termination

If the session terminates for any reason, all XIDs within that session become invalid. The service provider will cease all exchanges. If the client sends a STATUS, ABORT, CLOSE, or any other message, the service provider will respond with a NACK stating the SID does not exist.

1.8 Combining Messages

The client may combine messages into a single Interest using a structured message format. For example, if an implementation uses a TLV format for messages, the client may aggregate several messages together, such as a STATUS and LAST message. Likewise, the service provider may aggregate several responses together using a structured message format.

2. Conclusion

The Reliable Content Exchange (RCX) protocol uses a control channel between a client and a service provider to reliably transfer either a chunked name or a chunked manifest and its contents. The control channel allows the service provider to authenticate and authorize a client for transfers and the client can authenticate the service provider. The protocol allows the participants to encrypt the names and contents of the control channel using a session key.