```
ICN Research Group                                          R. Ravindran
Internet-Draft                                            A. Chakraborti
Intended status: Informational                                   S. Amin
Expires: January 17, 2018                            Huawei Technologies
                                                                 J. Chen
                                              Winlab, Rutgers University
                                                           July 16, 2017
```

Support for Notifications in CCN
draft-ravi-icnrg-ccn-notification-01

Abstract

This draft proposes a new packet primitive called Notification for CCN. Notification is a PUSH primitive and can be unicast or multicast to multiple listening points. Notifications do not expect a Content Object response hence only requires the use of FIB state in the CCN forwarder. Emulating Notification as a PULL has performance and routing implications. The draft first discusses the design choices associated with using current Interest/Data abstraction for achieving push and challenges associated with them. We follow this by proposing a new fixed header primitive called Notification and a CCN message encoding using Content Object primitive to transport Notifications. This discussion are presented in the context of CCNx1.0 [1] proposal. The draft also provides discussions on various aspects related to notification such as flow and congestion control, routing and reliability considerations, and use case scenarios.

Status of This Memo

Table of Contents

1. Introduction

Notification is a PUSH primitive used in the Internet today by many IoT and social applications. The nature of notifications varies with the application scenario, ranging from being mission critical to one that is best effort. Notifications can be unicast or multicast depending on whether the notification service is aware of all the consumers or not. A notification service is preceded by a consumer subscribing to a specific event such as, subscription to hash-tag feeds, health emergency notification service, or temperature sensor reading from a room in a building; following this subscription the service pushes notifications to consuming entities. It has to be noted that certain IoT applications expects notification end-to-end latency of few milliseconds [2]. Industrial IoT applications have more stringent requirement in terms of QoS, timeliness, and reliability of message delivery. Though we term it as a Notification, this primitive can also be used for transactional exchange between two points.

CCN optimizes networking around efficiently distributing already published content which the consumers learn through mechanisms like manifests containing the names of published content chunks and their locations. Applications relying on notifications requires event driven data to be pushed from multiple producers to multiple subscribers for which the current Interest/Data primitive is inefficient. This draft proposes to extend CCN's current primitives set with a new notification primitive that can be processed in a new way by the CCN forwarder to serve notification objectives. Notification here implies a PUSH semantic that is available with IP today and supported by other FIA architectures like MobilityFirst [3] and XIA [4].

2. Notification Requirements in CCN

General notification requirements and features have been discussed have been discussed in protocols such as CoAP's Observe proposal [5] to push notifications from the server to the clients. Here we discuss basic notification requirements from CCN's network layer perspective. Other requirements related to reliability, low latency, flow control can be engineered by the application or through more network layer state once the following requirements are met.

- Supporting PUSH Intent: CCN should provide efficient and scalable support for PUSH, where application's intent is to PUSH content to listening application without expecting any data in return. Efficiency relates to minimimizing control and forwarding overhead and scalability refers to support arbitrary number of producers and consumers participating in a general pub/sub or multicast service.

- Multicast Support: CCN network should be able to handle multicast notifications from a producer to multiple consumers.

- Security: Just as a content object in the context of Interest/Data primitive provides data authentication and privacy, similar features should also be offered by notification objects too.

o Routing/Forwarding Support: Name prefixes over which multicast notifications are managed should be handled in a different manner from the name prefixes over which Interest/Data primitive is used for content distribution in order to support the PUSH intent. This differentiation applies to the control as well as the forwarding plane.

o Minimizing Processing: Notification processing in the forwarder should be minimized considering the application’s intent to PUSH data to listening consumers.

3. Using Interest/Data Abstraction for PUSH

Recent CCN and NDN research [6][7] have studied the problem of handling notifications and have proposed several solutions to handle this. Here, we discuss several of them and point out their benefits and issues:

Long-lived Interest v.1: The most intuitive solution makes the assumption that the consumers know exactly the names of the contents that will be published in the future. Yet, it is not easy since the providers can give arbitrary names to each piece of content, even though the contents might share a common prefix (i.e., GROUP_PREFIX). To make it feasible, the providers can publish the contents with sequential ID, e.g., /GROUP_PREFIX/SENQUENTIAL_ID[/SEGMENT_ID], so that the consumers can query the contents with names /GROUP_ID/item_1, /GROUP_ID/item_2, ... (each name represents a content item). The consumers can pipeline the requests (always keep some unsatisfied requests in flight, similar to TCP) to better utilize the network capacity.

However, this solution has several issues, especially in the multi-provider scenario:

* Since it is unknown to the consumer (and the network) which provider will use which sequential ID, each request has to be forwarded to all the possible providers. This solution might use up a large amount of state (PIT entries) in the network, as each consumer can keep tens of requests (to all providers) in flight for each group.

* Since each sequential ID should only be used by one provider, many PIT entries will not be consumed until timeout (if there is a timeout mechanism). E.g., P1 and P2 are 2 providers of a group (/GROUP), the consumers have to send requests /GROUP/item_1, and /GROUP/item_2 to both providers. Assume that P1 publishes first so he uses the name /GROUP/item_1. The PIT

entries for /GROUP/item_1 towards P2 will not be consumed since P2 should now publish with name /GROUP/item_2.

* When the PIT entries form loops in the network (it can happen quite often in the multi-provider, multi-consumer scenario), the data packets can waste network traffic while following the loops and get discarded when redundancy happens.

* Other than the inefficiencies mentioned above, one major issue with this solution is the difficulty of provider synchronization. It is not easy to make sure that different providers would use different sequential IDs especially when the providers are publishing contents at the same time.

Polling v.1: To eliminate the requirement for a sequential ID when publishing (to address the synchronization issue), the solution Polling v.1 makes the providers publish contents with name format: /GROUP_ID/TIMESTAMP. While querying the contents, the consumer query using name /GROUP_ID/ with "exclude" field <Earliest version after Tx>, where Tx is the latest version the consumer has received. E.g., after receiving a content with name /GROUP_ID/ v_1234 (v_1234 is the timestamp of the publication time), the consumer would send a query with name /GROUP_ID/<Earliest after v_1234>. He might get the next piece with name /GROUP_ID/v_2345 (assuming that there is no content published between these two time stamps) without the need to know the exact names of the contents. The content providers do not have to be synchronized on the sequential IDs and use the timestamp instead.

While this solution is similar to the one used in NDN for getting the "latest" version under a prefix, it has several issues when we need to get "all" versions under a prefix:

* Ambiguity contents will appear when two providers of a same group publish at the same time.

* Consumers might miss messages when the clocks are not synchronized on the providers. E.g., one provider (with faster clock) might publish a content with name /GROUP_ID/v_2345 after v_1234. When the consumer queries for the earliest version after v_1234, he will get the content. Yet, another provider (with slower clock) would publish a content with name /GROUP_ID/v_2234 after the consumer gets v_2345. The consumer would miss the content with v_2234 as he will query for <Earliest after v_2345>.

* Consumers might miss messages due to different delivery latency (e.g., cache hit vs. no cache hit) even when the clocks on the

providers are perfectly synchronized (e.g., via GPS signals). E.g., when a client queries for content /GROUP_ID/<Earliest after v_1234>, and there are two pieces of content exist in the network (v_2234, and v_2345). It can happen that v_2345 is returned earlier (either due to a cache hit or because the provider is closer). The consumer would then query for <Earliest after v_2345> and miss v_2234 with this solution.

* Also just as with the previous approach, this mechanism also requires the producers to sync so that they don’t produce content using the same name.

Long-lived Interest v.2: To completely address the issues with multiple providers sharing a same prefix (e.g., synchronization in Long-lived Interest v.1, and clock synchronization in Polling v.1), Long-lived Interest v.2 gives a prefix to each provider. The providers in this solution provide contents with name /GROUP_ID/PROVIDER_ID/SEQUENTIAL_ID, and the consumers query the full names accordingly (similar to Long-lived Interest v.1 but with an extra prefix PROVIDER_ID). The consumer can still use pipelining to improve the throughput.

While this solution can avoid packet losses in the previous solution, it has several other issues:

* Consumers have to know all the potential providers, which might be difficult in some applications where every user can send messages in any group that he might be interested in.

* Compared to Long-lived Interest v.1, the consumers in this solution have to keep multiple pending queries per group per provider. It might consume even more states in the network, which makes the solution less scalable.

* When a provider has more than one device (e.g., laptop and smartphone) that can publish contents under a same name /GROUP_ID/PROVIDER_ID, the solution would have the same synchronization issue as Long-lived Interest v.1. If the solution mandates each device to have a separate provider ID, it will end up with even more PIT entries (states) in the network, and the solution becomes less "information-centric".

Polling v.2: To reduce the states and the control overhead in Long-lived Interest v.2, the solution Polling v.2 allows the provider process the requests in the application layer. Periodically, the consumer would query each provider "if there is any update after Nx" (Nx is name of the last content the consumer has received). The query would be in the format: /GROUP_ID/PROVIDER_ID/Nx/NONCE.

The provider would reply aggregated results in one response (with different segments, but under the same name), and an indication of "no update" if there is no publication after Nx. Since a same query for /GROUP_ID/PROVIDER_ID/Nx can get different responses ("no update", or aggregated publications), a NONCE has to be added in the name to prevent possible cache hits in the network. This solution can be effective in games since the publication rate (actions of the provider in the game) is much higher than the polling rate (refresh rate on the consumer). However, it still has some issues (inefficiencies):

* There is a tradeoff between timeliness vs. in-network traffic when choosing the polling frequency. The solution can be inefficient when the polling is too frequent: most of the polling will get "no update" responses. This can consume a large amount of traffic in the network and extra computation on both the providers and the consumers. The timeliness can be impaired when the polling is infrequent since the publication can only reach the consumer when the consumer queries. The average delivery time of a publication in such solution is half of the polling period.

* In-network cache cannot be used since the response to a same query (without nonce) can be different according to the time (and maybe the consumer).

* Consumers still have to know all the potential providers similar to Long-lived Interest v.2.

Polling with A Server: To relieve the consumers from knowing all potential providers in Polling v.2, solution Polling with A Server introduces a server (or broker) as the delegate of all the providers. The providers would publish data into the server and the consumers would poll for the updates from the server (similar to Twitter and Facebook in IP network). In this solution, the consumers do not have to poll each provider for the updates, which reduces the overhead in the network. With the aggregated response on the server, the network traffic is further reduced. However, it still has several issues:

* Similar to all the server-based solutions like Facebook and Twitter, the server has to deal with all the polls. This can cause single point of failure.

* It is not easy for the providers "publish contents to the server". This becomes another notification problem and has to be solved by the other solutions mentioned in this section.

* Cache is not used in this solution similar to Polling v.2.

* This solution is not really "information-centric" as the consumers have to get the location of the content rather than the content itself.

Interest Overloading: Since all the aforementioned query/response solutions have issues with efficiency, scalability and/or timeliness, Interest Overloading tries to modify the communication pattern by using Interest packets to deliver publications directly. The consumers in this solution propagate FIB entry of /GROUP_ID to all potential providers (or simply flood the network). When a provider sends a publication, he would send an Interest with name /GROUP_ID/NONCE/<Payload> and the lifetime set to zero. Since the traditional Interest packets do not have payload, the solution has to embed (e.g., URL encode [1]) the payload in the name of the Interest. NONCE is used to prevent PIT aggregation since providers may publish contents with same payload (e.g., sensor readings). This solution can address the timeliness and scalability issues with the Polling and Long-lived Interest solutions, yet there are still some issues:

* This solution creates ambiguity in the meaning of Interest packets (and the corresponding forwarding behaviors on the routers). For a normal Interest packet, the forwarding engines should perform an anycast (send it to only one of the providers) according to FIB. However, in this solution, the forwarding engines should use multicast logic for prefix /GROUP_ID (and avoid PIT storage). Solution in [8] specifies some multicast prefixes so that the forwarding engines can distinguish the publications from the normal requests. Yet, this places higher overhead on both the forwarding engines and the network management. It also prevents providers to create contents under the /GROUP_ID prefix (since the query will be forwarded using multicast, and not kept in the PIT).

* The routing is also a concern in this solution. When the consumers propagate FIB, it should reach all potential providers (in most of the time it will flood the network since all the users can be potential providers). Naturally, in a multi-provider, multi-consumer scenario, the FIB entries would form a mesh in the network. It is less scalable compared to the tree-based routing in IP multicast (PIM-SM). The network has to specify another routing policy specifically for these prefixes, which places even higher overhead on network management.

  * As is mentioned in [9], it is not efficient to embed large amount of data into the name of the Interest packets. It adds more computation and storage overhead in the forwarding engines (PITs).

Interest Trigger: Similar to Interest Overloading, Interest Trigger uses an Interest packet as notification. To eliminate the overhead of embedding the content in the Interest, this solution places the name of the publication in the name of the notification (Interest) packet. On receiving the notification, the consumers can extract the content name and send another query (Interest) for the real content. While this solution reduces the overhead of embedding the payload, it still has the ambiguity and routing issues similar to Interest Overloading solution. It also incurs additional round trip delay before the produced data arrives at the listening consumer.

To summarize CCN and NDN operates on PULL primitive optimized for content distribution applications. Emulating PUSH operation over PULL has the following issues:

o It is a mismatch between an application’s intent to PUSH data and the PULL APIs currently available.

o Unless Interests are marked distinctly, overloading Interests with notification data will undergo PIT/CS processing and are also subjected to similar routing and forwarding policies as regular Interests which is inefficient.

o Another concern in treating PUSH as PULL is with respect to the effect of local strategy layer routing policies, where the intent to experiment with multiple faces to fetch content is not required for notification messages.

This motivates the need for treating notifications as a separate class of traffic which would allow a forwarder to apply the appropriate routing and forwarding processing in the network.

4. Proposed Notification Primitive in CCN

Notification is a new type of packet hence can be subjected to different processing logic by a forwarder. By definition, a notification message is a PUSH primitive, hence is not subjected to PIT/CS processing. This primitive can also be used by any other transactional or content distribution application towards service authentication or exchanging contextual information between end points and the service.

5. Notification Message Encoding

The wire packet format for a Notification is shown in Fig. 1 and Fig. 2. Fig. 1 shows the Notification fixed header considering the CCNx1.0 encoding, and Fig. 2 shows the format for the CCN Notification message, which is used to transport the notification data. We next discuss these two packet segments of the Notification message.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|    Version    | PacketType=   |         PacketLength         |
|               | Notification  |                              |
+---------------+---------------+---------------+--------------+
|   HopLimit    |   Reserved    |     Flags     | HeaderLength |
+---------------+---------------+---------------+--------------+
/               Optional Hop-by-hop header TLVs                /
+---------------+---------------+---------------+--------------+
/           Content Object as Notification Message             /
+---------------+---------------+---------------+--------------+
```

Figure 1: CCN Notification fixed header

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
| MessageType = Content Object  |         MessageLength        |
+---------------+---------------+---------------+--------------+
|                           Name TLV                           |
+---------------+---------------+---------------+--------------+
|                    Optional MetaData TLVs                    |
+---------------+---------------+---------------+--------------+
|    Message Payload Type       |      Message Type Length     |
+---------------+---------------+---------------+--------------+
|               Payload or Optional Content Object             |
+---------------+---------------+---------------+--------------+
/            Optional CCNx ValidationAlgorithm TLV             /
+---------------+---------------+---------------+--------------+
/ Optional CCNx ValidationPayload TLV (ValidationAlg required) /
+---------------+---------------+---------------+--------------+
```

Figure 2: CCN Notification Message

Notification Fixed Header: The fields in the fixed header that have new meaning in the context of notifications are discussed next, while the other fields follow the definition in [1].

- o Packet Type: This new type code identifies that the packet is of type Notification [TBD].

- o Optional Hop-by-hop header TLVs : Encodes any new hop-by-hop headers relevant to notifications [TBD].

CCN Notification message: The CCN Notification message is a Content Object as in [1]. Notifications are always routed on the top level Content Object (outer CO) name. Notification itself can be encoded in two forms depending on the application requirement:

- o Notification with single name: In this case the notification contains a single content object. Here the producer generates notification using the same name used by consumers on which they listen on.

- o Notification with two names: In this case the notification contains a top level Content Object (outer CO), that encapsulates another Content Object (inner CO). With an encapsulated Content Object, the meaning is that notification producers and consumers operate on different name-spaces requiring separate name-data security binding. A good application of the encapsulation format is a PUB/SUB service, where the consumer learns about the notification service name offline, and the producer who is decoupled from the consumer generates a new Content Object using its own name and pushes the notification to the consumer.

The interpretation of the fields shown in Fig. 2 are as follows:

- o MessageType : The CCN message type is of type Content Object.

- o Name TLV : Name TLV in the Content Object is used to route the Notification.

- o Optional Metadata TLV: These TLVs carry metadata used to describe the Notification payload.

- o Message Payload Type: This is of type T_PAYLOADTYPE defined in CCNx.1.0 or a new encapsulation type (T_ENCAP) that indicates the presence of another encapsulated Content Object [TBD].

- o Optional Encapsulated Content Object: This is an optional encapsulated Content Object newly defined for the Notification primitive. The name in the encapsulated Content Object corresponds to the producer’s name-space, or anything else based on the application logic. The rational for an encapsulated Content Object was discussed earlier.

- Optional Security Validation data: The Content Object optionally carries security validation payload as per CCNx1.0.

6. Notification Processing

The following steps are followed by a CCN forwarder to process the Notification packet.

- Notification packet type is identified in the fixed header of a CCN packet with a new type code. The Notification carries a Content Object, whose name is used for routing. This name is matched against the FIB entries to determine the next hop(s). Novel strategy layer routing techniques catering to the notification traffic can be applied here.

- CCN forwarder also processes the optional metadata associated with the Notification meant for the network to help with the forwarding strategy, for e.g., mission critical notifications can be given priority over all other traffic.

- As mentioned earlier, CCN forwarder MUST NOT cache the Content Objects in the notifications.

7. Security Considerations

The proposed processing logic of Notifications that bypass the processing of PIT/CS has the following security implications:

Flow Balance : PIT state maintains the per-hop flow balance over all the available faces by enforcing a simple rule, that is, one Content Object is send over a face for a single Interest. Bypassing PIT processing compromises this flow balancing property. For scenarios where the notification traffic volume is not high such as for IoT applications, the impact may not be significant. However, this may not be the case considering the plethora of social networking and emerging IoT applications in a general Internet scenario. This flow balance tradeoff has to be understood considering an application’s intent to PUSH data and the latency introduced by processing such traffic if a PULL primitive is used. Also PIT offers a natural defense mechanism by throttling traffic at the network edge, considering the provisioned PIT size, and bypassing it could exacerbate DDOS attacks on producing end points.

Cache Poisoning: This draft doesn’t recommend the caching of the Content Object in the Notification payload, though doing so might help in increasing the availability of notification information in the network. A possible exception would be if the inner CO is a nameless object [10]. as those can only be fetched from CS by hash We leave this possibility of applying policy-based caching of Notification Content Objects for future exploration. The recommendation for not caching these Content objects is that, in a regular Interest/Content Object exchange, content arrives at the forwarder and is cached as a result of per-hop active Interest expression. Unsolicited Content Objects, as in the case of the Notification, violates this rule, which could be exploited by malicious producers to generate DDOS attack against the cache resource of a CCN infrastructure.

8. Annex

8.1. Flow and Congestion Control

8.1.1. Issues with Basic Notifications

As mentioned in the previous sections, one of the main issues with notification is the flow and congestion control. One naive way to solve this issue is the routers drop the packets from aggressive flows. Flow-based fair queueing (and its variation stochastic fairness queueing) maintain queues for flows (or the hash of flows) and try to give a fair share to each flow (or a hash). Flows can be classified by the prefixes in the ICN case. However, according to [11], the overall network throughput will be affected when there are multiple bottlenecks in the network. Therefore, [11] promotes an end-to-end solution for congestion control. Flow balance is a key requirement to an end-to-end (or end-driven) flow and congestion control. In the case of CCN query/response, flow balance entails that an Interest pulls at most one Data object from upstream. The data consumer can therefore control the amount of traffic coming from the data source(s) either it is a data provider or a cache in the network. However, the basic notification does not follow the rule of flow balance (each Subscription can result in more than one Notifications disseminated in the network). In the absence of a proper feedback mechanism to notify the data sender or the network the available bandwidth and local resource the consumer has, the sender can easily congest the bottleneck link of the receivers (causing congestion collapse) and/or overflow the buffer on the receiver side. In the later sections, we will describe the possible congestion control mechanisms in ICN and how to deal with packet loss when both congestion control and reliability are required.

However, the basic notification does not follow the rule of flow balance (each Subscription can result in more than one Notifications disseminated in the network). There is no way a receiver can notify the data sender or the network the available bandwidth and local resource it has. As a result, the sender can easily congest the bottleneck link of the receivers (causing congestion collapse) and/or overflow the buffer on the receiver side.

8.1.2. Flow and Congestion Control Mechanims

Here we discuss broad approaches towards achieving flow and congestion control in CCN as applied to Notification traffic. Since the forwarding logic of the Notification packets are quite similar to that of IP multicast, existing multicast congestion control solutions can be candidates to solve the flow/congestion control issue with Notification. In addition we also summarize recent ICN research to address this issue.

8.1.2.1. End-to-End Approaches

In the multicast communication, it is not scalable to have direct receiver-to-sender feedback loop similar to TCP since this would result in each receiver sending ACKs (or NACKs) to the data sender and cause ACK (NACK) implosion. To address the ACK implosion issue, two types of solutions have been proposed in multicast congestion control, namely, sender-driven approaches and receiver-driven approaches.

8.1.2.1.1. Sender-driven Multicast

In the first category, the sender controls the sending rate and to ensure the network friendliness, the sender usually align the sending rate to the slowest receiver.

To avoid the ACK implosion issue, TCP-Friendly Multicast Congestion Control (TFMCC [12]) uses rate based solution. This solution uses TCP-Friendly Rate Control (TFRC) to get a proper sending rate based on the RTT between sender and each receiver. The sender only needs to collect the RTTs periodically instead of per-packet ACKs. Similarly, in ICN, the sender can create another channel (namespace) to collect the RTT measurement from the receivers. However, due to the dynamics on each path, it is difficult to calculate the proper sending rate.

To address the rate calculation issue, pgmcc [13], a window-based solution is proposed. It uses NACKs to detect the slowest receiver (the ACKer). The ACKer sends an ACK back to the sender on receiving each multicast packet. A feedback loop similar to TCP is formed between the sender and the ACKer to control the sending rate. Since the ACKer is the slowest receiver, the sender adapts its sending rate to the available bandwidth of the slowest receiver, the solution can therefore ensure the network friendliness. In the ICN case, the receivers can send NACKs in the form of Notification packets through another namespace, and the ACKer can also use the same mechanism to send ACKs.

However, since the sender is always aligning the sending rate to the slowest receiver to ensure the network friendliness, the performance of the solutions can be dramatically affected by a very slow receiver.

8.1.2.1.2. Receiver-driven Multicast

Unlike the sender-driven solutions, the receiver-driven solutions [14] choose to use layered-multicast to satisfy heterogeneous receivers. The sender first initiates several multicast groups (namespaces in the case of ICN) with different sending rates. Each receiver would choose to join a multicast group with the highest sending rate that it can afford. The sender can also adapt the sending rate of each multicast group according to the receiver status.

These solutions can support applications like video streaming (with layered codecs) efficiently. However, they also have some issues: 1) they complicate the sender and receiver logic, especially for simple applications like file transfer; and 2) the receivers are limited by the sending rates initiated by the provider and would therefore under-utilize the available bandwidth.

8.1.2.2. Hybrid Approaches

In this approach, flow balance of Notification is achieved by the receivers notifying the network (rather than the sender or other receivers) about the capacity it can receive. Here, we take advantage of operating the Notification service through a receiver-driven approach and get support from the network.

A solution based on this approach is proposed in [15], which we summarize next.

To retain flow balance, the consumers in this solution send out one subscription for only one next Notification instead of the original logic (that receives all the Notifications). Similar to the flow and congestion control in query/response, the receivers can now maintain a congestion window to control the amount of traffic coming from upstream.

Here, instead of maintaining a (name, outgoing face) pair in FIB (or subscription table), the routers now adds a third field -- accumulated count -- for each entry. The accumulated count is increased by 1 on receiving such a subscription and decreased by 1 on sending a Notification to that face. The routers should also propagate the maximum accumulated count upstream till the 1st hop router of the provider (or the rendezvous point in the network). The subscribers sends a subscription for every successfully received notification. Here we also assume that, the subscribers operate based on the AIMD scheme.

If the dissemination of Notification follows a tree topology in the network, we define the branching point of a receiver R (BP_R) as the router closest to R which has another outgoing face that can receive data faster than R. For receivers that has bandwidth/resources to receive all the data from the provider, BP_R is the 1st hop router of the provider (or the rendezvous point).

In this solution, we can prove that there is a feedback loop between each receiver and its branching point. Therefore, when a receiver maintains its congestion window size using AIMD, the traffic between the branching point and the receiver is similar to TCP. It can get a fair share at the bottleneck on the path, even if the bottleneck is not directly under the branching point. In the multicast tree, the solution can ensure the fairness with other (TCP-like) flows on each branch.

The solution can thus allow the sender to send at an application-efficient rate rather than being affected by the slowest receiver like pgmcc [13].

It is true that the solution requires more packets and more states in the network compared to the basic notification solution, but the cost is similar to (and smaller than) that of query/response. Since we are using one notification per subscription pattern, the amount of traffic overhead is the same as query/response. As for the states stored in the router, the solution only requires 1 entry per prefix per face, which is smaller than the query/response which requires 1 entry per packet per face. Therefore, the overhead of the solution is acceptable in CCN.

8.1.2.2.1. Other Challenges

- Sender Rate Control: The sender in the solution does not have to limit the sending rate to the slowest receiver to maintain network friendliness. Therefore, the choice of sending rate is a tradeoff between network traffic and session completion time. In the case where the application does not require a certain sending rate (like file transfer), the sender can align the sending rate to the slowest receiver (similar to pgmcc) to minimize the repair traffic, but at the cost of longer session completion time. He can also send at the rate of the fastest receiver and try to get peer repair in the network. This allows faster receivers finish the session earlier but causing higher network traffic due to the repair. An ACKer-based solution similar to pgmcc can be adopted to allow the sender align the rate at a proportion of users (e.g., top 30%). The sender can collect feedback (throughput, latency, etc.) from all the receivers periodically and pick an ACKer according to the proportion it desires. On receiving a Notification packet, the ACKer would send an ACK just like TCP. The sender can maintain a congestion window also like TCP. The feedback loop between the sender and the ACKer can align the sending rate at the ACKers’s available bandwidth.

- Receiver Window Control: Slightly different from one-sender one-receiver window control in TCP, the sending rate in the hybrid approach is not controlled by any of the receivers. Receiving intermittent packets can indicate both congestion (similar to TCP) and not enough window size (since the sending rate is higher). In the first case, the receiver should reduce the window size while in the second case, the receiver should increase the window size. An indication of congestion (e.g., Random Early Detection, RED) should be provided directly from the network.The receivers with available bandwidth higher than the sending rate would have too large window size since it does not see any packet loss. Please refer to [15] for a detailed solution on this issue.

8.1.3. Receiver Reliability

The receiver would miss packets when the available bandwidth/resource of the receiver is lower than the sending rate of the Notification provider. Some applications (like gaming and video conferencing) can tolerant such kind of packet loss while the others (like file transfer) cannot. Therefore, another module that ensures the reliability is needed. However, reliability should be separated from the flow and congestion control since it is not a universal requirement.

With the solution described in the receiver-driver or the hybrid approach, the slower consumers would receive intermittent packets since the sending rate can be faster than their fair share. The applications that require reliable transfer can query the missing packets similar to the normal query/response. This also requires that each content in the Notifications should have a unique Content Name (or hash in the nameless scenario). The clients should also be able to detect the missing packets either based on the sequence number or based on a pre-acquired meta-file. Caching in CCN can be leveraged to achieve availability and reliability.

The network can forward the requests (Interests) of the missing packets towards the data provider, the other consumers and/or the in-network cache to optimize the overall throughput of the consumers. This solution is similar to Scalable Reliable Multicast (SRM [16]). However, as mentioned in [17], solutions like SRM requires the consumers communicate directly with each other and therefore lose the privacy and trust. CCN can ensure the privacy since the providers cannot get the information of the identity of the consumers. Trust (data integrity) is also maintained with the signature in the Data packets.

8.2. Routing Notifications

Appropriate routing policies should be employed to ensure reliable forwarding of a notification to its one or many intended receivers. The name in the notification identifies a host or a multicast service being listened to by the multiple intended receivers. Two types of routing strategies can be adopted to handle notifications, depending on whether or not an explicit pub/sub state is maintained in the forwarder.

o Stateless forwarding: In this case the notification only relies on the CCN FIB state to route the notification. The FIB entries are populated through a routing control plane, which distinguishes the FIB states for the notification service from the content fetching FIB entries. Through this logical separation, Notifications can be routed by matching its name with the matching FIB policy in the CCN forwarder, hence processed as notification multicast.

o Stateful forwarding: In this case, specific subscription state is managed in the forwarder to aid notification delivery. This is required to scale notifications at the same time apply notification policies, such as filter notifications or to improve notification reliability and efficiency to subscribing users [18].

8.3. Notification reliability

This proposal doesn’t provide any form of reliability. Reliability can be realized by the specific application using the proposed notification primitive, for instance using the following potential approaches:

Caching: This proposal doesn’t propose any form of caching. But caching feature can be explored to improve notification reliability, and this is a subject of future study. For instance, consumers, which expect notifications and use external means (such as periodic updates or by receiving manifests) to track notifications, can recover the lost notifications using the PULL feature of CCN.

Notification Acknowledgment: If the producer maintains per-receiver state, then the consumer can send back notification ACK or NACK to the producer of having received or not received them.

8.4. Use Case Scenarios

Here we provide the discussions related to the use of Notification in different scenarios.

8.4.1. Realizing PUB/SUB System

A PUB/SUB system provides a service infrastructure for subscribers to request update on a set of topics of interest, and with multicast publishers publishing content on those topics. A PUB/SUB system maps the subscribers' interests to published contents and pushes them as Notifications to the subscribers. A PUB/SUB system has many requirements as discussed in [19] which include low latency, reliability, fast recovery, scalability, security, minimizing false (positive/negative) notifications.

Current IP based PUB/SUB systems suffer from interoperability challenges because of application-defined naming approach and lack of support of multicast in the data plane. The proposed Notification primitive can be used to realize large scale PUB/SUB system, as it unifies naming in the network layer and support for name-based multicasting.

Depending on the routing strategy discussed earlier, two kind of PUB/SUB approaches can be realized : 1) Rendezvous style approach ; 2) Distributed approach. Each of these approaches can use the Notification primitive to implement their PUSH service.

In the Rendezvous style approach, a logically centralized service maps subscriber's topic interest with the publisher's content and pushes it as notifications. If stateless forwarding is used, the routing entries contain specific application-ID's requesting a given notification, to handle scalability, a group of these application can share a multicast-ID reducing the state in the FIB.

In the Distributed approach, the CCN/NDN protocol is further enhanced with new subscription primitive for the subscription interested consumers. When a consumer explicitly susbcribes to a multicast topic, its subscription request is forwarded to the upstream forwarder which manages this state mapping between subscription names to the downstream faces which has expressed interest for Notifications being pushed under that prefix. An example of the network layer based approach is the COPSS notification proposal [19]. Here a PUB/SUB multi-cast state state, called the subscribers interest table, is managed in the forwarders. When a Notification arrives at a forwarder, the content descriptor in the notification is matched to the PUB/SUB state in the forwarder to decide the faces over which the Notification has to be forwarded.

9. Informative References

[1] CCN Wire format, CCNX1., "http://www.ietf.org/id/draft-mosko-icnrg-ccnxmessages-00.txt.", 2013.

[2] Osseiran, A., "Scenarios for 5G Mobile and Wireless Communications: The Vision of the METIS Project.", IEEE Communication Magazine , 2014.

[3] NSF FIA project, MobilityFirst., "http://www.nets-fia.net/", 2010.

[4] NSF FIA project, XIA., "https://www.cs.cmu.edu/˜xia/", 2010.

[5] Observing Resources in CoAp, observe., "https://tools.ietf.org/html/draft-ietf-core-observe-16.", 2015.

[6] Amadeo, M., Campolo, C., and A. Molinaro, "Internet of Things via Named Data Networking: The Support of Push Traffic", Network of the Future (NOF), 2014 International Conference and Workshop on the , 2014.

[7] Shang, W., Bannis, A., Liang, T., and Z. Wang, "Named Data Networking of Things.", IEEE IoTDI 2016, 2016.

[8] Zhu, Z. and A. Afanasyev, "Let’s chronosync: Decentralized dataset state synchronization in named data networking", The 21st IEEE International Conference on Network Protocols ICNP, 2013.

[9] Moiseenko, I. and O. Oran, "TCP/ICN: Carrying TCP over Content Centric and Named Data Networks", Proceedings of the 3rd ACM Conference on Information-Centric Networking ICN, 2016.

[10] Mosko, M., "Nameless Objects.", IETF/ICNRG, Paris Interim 2016, 2016.

[11] Floyd, S. and F. Kevin, "Promoting The Use of End-to-End Congestion Control in The Internet.", IEEE ToN vol. 7(4), pp. 458-472, 1999.

[12] Widmer, J. and M. Handley, "TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification.", IETF RFC 4654, 2006.

[13] Rizzo, L., "pgmcc: A TCP-Friendly Single-Rate Multicast Congestion Control Scheme.", SIGCOMM CCR vol. 30.4, pp. 17-28, 2000, 2000.

[14] McCanne, S., Jacobson, V., and M. Vetterli, "Receiver-driven Layered Multicast.", SIGCOMM CCR pp. 117-130, 1996.

[15] Chen, J., Arumaithurai, M., Fu, X., and KK. Ramakrishnan, "SAID: A Control Protocol for Scalable and Adaptive Information Dissemination in ICN.", arXiv vol. 1510.08530, 2015.

[16] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing.", IEEE TON vol. 5(6), pp. 784-803, 1997.

[17] Floyd, N., Grossglauser, M., and KK. Ramakrishnan, "Distrust and Privacy: Axioms for Multicast Congestion Control.", Distrust and Privacy: Axioms for Multicast Congestion Control NOSSDAV, 1999.

[18] Francois et al, J., "CCN Traffic Optimization for IoT", Proc. of NoF , 2013.

[19] Chen, J., Arumaithurai, M., Jiao, L., Fu, X., and K. Ramakrishnan, "COPSS: An Efficient Content Oriented Publish/Subscribe System.", ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2011) , 2011.

[20] DNS Security Introduction and Requirements, DNS-SEC., "http://www.ietf.org/rfc/rfc4033.txt.", 2005.

[21] Cisco System Inc., CISCO., "Cisco visual networking index: Global mobile data traffic forecast update.", 2009-2014.

[22] CCNx Label Forwarding, CCNLF., "http://www.ccnx.org/pubs/ccnx-mosko-labelforwarding-01.txt.", 2013.

Authors' Addresses

Ravishankar Ravindran
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ravi.ravindran@huawei.com

Asit Chakraborti
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: asit.chakraborti@huawei.com

Syed Obaid Amin
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: obaid.amin@huawei.com

Jiachen Chen
Winlab, Rutgers University
671, U.S 1
North Brunswick, NJ 08902
USA

Email: jiachen@winlab.rutgers.edu