Reliable Publish/Subscribe in Content-Centric Networks

Jiachen Chen∗, Mayutan Arumaithurai‡†, Xiaoming Fu∗, K.K. Ramakrishnan∗‡
∗Institute of Computer Science, University of Goettingen, Germany,
Email: {jiachen, arumaithurai, fu}@cs.uni-goettingen.de
†NEC Laboratories Europe, Heidelberg, Germany, Email: arumaithurai@nw.neclab.eu
‡AT&T Labs-Research, Florham Park, NJ, U.S.A., Email: kkrama@research.att.com

ABSTRACT
Managing congestion is a challenge in content-centric networks due to the lack of an end-to-end session context over which a ‘flow’ may be controlled. Flow and congestion control as well as reliable delivery are often considered even more of a challenge in content-centric publish/subscribe systems, where the nature of information dissemination is similar to multicast. These have been long-standing challenges also for IP multicast. With an unknown number of publishers, a content-centric pub/sub environment exacerbates these problems, demanding new solutions. In this paper, we propose a lightweight enhancement to content-centric publish/subscribe systems for flow and congestion control as well as for reliability. R-COPSS allows the publishers to efficiently use the content-centric network by having subscribers generate timely feedback while enabling subscribers to make use of NDN to perform local repair for reliable delivery. Rather than having all subscribers generate feedback (ACK) per packet, we seek to elect particular subscribers in a hierarchy to provide the feedback and the rest of them resort to a periodical summary. Our approach not only reduces the load on the publisher, but also removes the requirement on the publisher to limit its sending rate to the slowest subscriber. Our preliminary results show that R-COPSS performs better in terms of overall throughput and is fair to competing flows.

Categories and Subject Descriptors
C.2.1 [Computer-communication Networks]: Network Architecture and Design

Keywords
CCN, NDN, COPSS, Pub/Sub, Multicast, Congestion Control, Reliability

1 Introduction
Content Centric Networking (CCN) [1–7] seeks to transform content as a first-class entity. Rather than the current location-centric network architecture, CCN presents a networking paradigm that enables content based interaction. It allows the network to provide content from any of the multiple sources where it is available (including the in-network cache), and meets the user’s need to send and receive content, irrespective of where it is or who generated it. Named Data Networking (NDN) [4] is one of the more popular examples of CCN and is based on a query response model. Our proposal for a publish/subscribe model, called the Content-Oriented Publish/Subscribe System (COPSS) [7–9], is built on top of NDN with a new CCN-oriented multicast capability to meet the efficiency and scalability needs of large scale pub/sub systems. We believe the temporal separation of information generation and prior indication of interest is a highly desirable content centric feature. Timely delivery of published information to subscribers is often an important consideration. It has been shown in [7–9] that the COPSS is able to efficiently handle the requirements of applications such as a content-focused Twitter-like pub/sub system [7] and applications with tight timeliness requirements such as online multiplayer gaming and content streaming [8].

In a multicast environment that is carefully configured and managed, it may be easier to configure the rate at which publishers publish, such that the publishers have an assurance of achieving a high probability of delivery. However, in many environments of scale (e.g., a global environment using CCN based information dissemination for popular applications such as Twitter, Gaming or User Generated Content services like Youtube) a careful management of the publication rate is likely to be difficult. Moreover, a large fraction of the participating entities are likely to behave both as a publisher as well as a subscriber, in which case this is even more challenging. For a large number of applications, reliable delivery is an essential requirement, even if it is primarily driven by user expectation rather than a formal requirement. Furthermore, an effective flow-control is required to enhance the reliability of the network. Having an effective flow and congestion control mechanism enables us to ensure that the network is efficiently used and that reliability is not achieved at the cost of lower throughput.

In this paper, we seek to achieve a good balance between the rate a publisher can publish at, while achieving the highest possible rate of progress for the overall system. We seek to exploit the idea of ‘local repair’, so that the publisher does not have to go down to the slowest subscriber, while...
still achieving reliable delivery for all the subscribers. We propose R-COPSS, a lightweight enhancement to COPSS to allow it to provide reliable pub/sub delivery with inbuilt flow and congestion control mechanisms. R-COPSS, ensures that the publisher is able to transfer the data at a rate that is faster than the slowest subscribers in order to support either a rate required by the application (e.g. video conferencing) or to support the majority of the users (e.g., convey emergency notifications). Furthermore, R-COPSS enhances the throughput of the slowest subscribers by making use of the query/response based CCN delivery to support local repair. R-COPSS provides flow and congestion control to enable the publisher to send at a suitable rate by exploiting coarse-grained, periodic feedback from subscribers. These mechanisms not only reduce the load on the publisher, but also remove the requirement that the publisher must limit its sending rate to the slowest subscriber unlike other approaches such as [10]. As shown in Section 4, our preliminary evaluation suggests that our approach performs better than a traditional query-response approach, achieving timely delivery of published information for application scenarios that require “data to be delivered as soon as the publisher produces it, e.g., for real time communication such as live-streaming or disaster management”. R-COPSS benefits from a pub/sub approach to support the majority of the users. We complement this delivery with the typical query/response mechanism to deliver data to the slower subscribers.

The paper is structured as follows. In §2 we list the existing solutions and their shortcomings. §3 presents our mechanism to achieve flow control and reliability while §4 reports the preliminary evaluation results followed by the conclusion in §5.

2 Related Work

A number of approaches have been proposed to provide reliability in multicast. Gopal et al. [11] use per-packet ACK but face the problem of ACK implosion. An optimization to this approach is to employ selective ACK. Works such as [12–14] partially address the ACK implosion problem by grouping multiple ACKs into a single group control. But the publisher still has to align the publication rate to the slowest receiver. Moreover, these optimizations do not allow repair among peers, which results in a burden to the publisher.

Approaches such as Reliable Multicast Transport Protocol (RMTP) [15] and Tree-based Reliable Multicast Protocol (TRAM) [16] address the ACK implosion problem to a certain extent by grouping subscribers in a hierarchical structure and the subscribers ACK to the upper level in the tree so that the ACKs can be aggregated. By introducing the relationship among subscribers, these solutions allow subscribers perform local repair. Moreover, the publisher can align to the speed of the majority of the receivers based on the ACK strategy the subscribers used. Unfortunately, in these proposals, the subscribers have to know each other in order to perform repair as well as send ACKs.

The authors in [17] propose a receiver initiated approach based on NAKs, however since every NAK and retransmission is multicasted to the whole group, multiple NAKs on a same packet sent from different subscribers might cause wasteful traffic and ineffective repair. The authors in [18,19] propose a NAK suppression mechanism in WAN and LAN. Nevertheless, the problems of the solutions include ineffective repair and the publisher has to align the speed to the slowest receiver.

In [10], Luigi proposes a TCP-friendly single rate multicast congestion control mechanism to address the ACK/NAK implosion issue. The solution uses NAKs to elect an ACKer and the ACKer sends the per-packet ACK to the publisher. However, since all the NAKs and repairs are multicasted to the whole group, it results in an overhead. Moreover, the publisher has to align to the slowest receiver.

Another trend for providing reliable multicast is using layered multicast. In [20,21], the publisher creates different multicast groups that transmit different resolutions of the data. The subscribers can select groups according to their link capacity. These solutions are orthogonal to the single-rate reliable multicast. The single-rate reliable multicast provides reliability in a short term and long-term reliability is achieved by subscribers selecting the proper multicast group.

3 Flow and Congestion Control, Reliability

In this section, we present R-COPSS, a light weight flow/congestion control and local repair mechanism to provide flow-controlled, reliable delivery for a content oriented publish/subscribe system that we have proposed (COPSS).

3.1 Problem statement and requirements

As we have shown in earlier papers [7–9], pub/sub is a desirable enhancement to provide immediate delivery of published data in a CCN network. Example applications include gaming, live-streaming of content, real-time communication (conferencing), emergency warning systems, etc. The difficulty in such an environment, as with any multicast-oriented communication, is that the bandwidth and round-trip-time to the various receivers can be quite different; this gets compounded with frequently changing network conditions. Furthermore, application specific needs such as the ones listed below need to be considered:

- Some applications such as a video conferencing tool might require a minimum transfer rate whereas an emergency services application or a file transfer application might want timely delivery to the majority of users.

- support applications that can deliver different rates to cater to the needs and capabilities of different classes of subscribers. E.g., a video conferencing application might be able to support voice functionality at a certain rate to some and video functionality at a higher rate to others. Moreover applications making use of scalable codecs might prefer sending packets with varying importance/quality. The subscribers (or the client applications) can decide there quality according to the available bandwidth dynamically.

- support slower subscribers efficiently

- use the network efficiently

Fairness: When multiple publishers make use of the network, it is essential to achieve a level of fairness, when there is contention. In a multicast environment, fairness is a difficult aspect to specify, achieve, or measure. A publisher
could have multiple subscribers, that could be reached via different paths. The path from a publisher to each of its subscriber could vary in distance, available bandwidth, etc. Moreover, each of the publishers might have a different number of subscribers. Thus, they are not easily comparable to other publishers, for measuring fairness. Furthermore, applications could have varying requirements from the default service offered by COPSS. For example, a real time communication based application might want to ensure that all the content is delivered to all its subscribers, whereas a video-delivery application may be fine with some (albeit limited) lost/missing content. Note that fairness is an issue only when there is contention for a shared resource and as long as there is no congestion, publishers should be able to utilize the resources they require. In this paper, we design R-COPSS to have a flow and congestion control mechanism that adapts to available bandwidth in the network as well as the receiver.

### 3.2 Flow and Congestion Control

The requirement is for the publisher to maintain a sending rate that is appropriate for a group of subscribers (e.g., majority), as recommended by the specific application. Therefore, the publisher may need to publish faster than the slowest set of subscribers. In short, R-COPSS’s congestion control strategy is as follows: R-COPSS aims to align its sending rate to one of the subscribers who is selected as the Ack. The Ack is the pre-selected subscriber that is expected to send an ACK, and the publisher seeks to adjust its sending rate in a TCP like manner based on the ACKs it receives. The Ack is selected for a time-period based on the ranking of the subscribers derived from the loss-rate and throughput that they have reported and the minimum rate that the publisher wants to support for that particular publish-sub tree. All the subscribers send a periodic feedback of the loss-rate and throughput which is used by the publisher to select the Ack for the next time period. Note that unlike other approaches [10], we do not select the Ack as the slowest subscriber. We propose a repair-based reliability mechanism in the next Section that allows the slower subscribers to obtain the packets they missed out via local repair (see Section 4 for the evaluations showing the benefit).

#### 3.2.1 Feedback

For an effective flow and congestion control, some form of feedback to the source (publisher) is essential. Various forms of feedback such as Hop-by-Hop, Intermediate Router, Per-Packet ACK, NACK, etc., have been proposed. Hop-by-Hop feedback, where a router along the tree ACKs the packet upstream, has the disadvantage that the publisher invariably aligns to the slowest subscriber. In an ‘intermediate router’ based solution where the router sends a feedback on dropping a packet, each intermediate router would have to know how to reach the publisher(s). A per-packet based solution where the ACKs are sent to the source would result in ACK implosion at the publisher.

In R-COPSS, all subscribers send periodic feedback of their loss-rate and receive throughput to the sender. The periodicity is managed so that the load on the publisher is manageable. Additionally, the Ack is expected to send an ACK (or a cumulative ACK) for packets that it receives. The rest of the subscribers do not send an ACK and only send a repair-request when it faces loss. The Repair-request is similar to a NACK, but the difference here is that it does not have to reach the publisher (see Section 3.3).

#### 3.2.2 Sending rate

We adopt a window-flow controlled mechanism at the publisher. At the start of a flow, R-COPSS’s window size (w) is set to 1 and sends a packet to every subscriber and all the subscribers are expected to ACK it. On receiving the first ACK from all (or most of) the subscribers, the publisher ranks the subscribers based on their RTT (shorter the RTT, earlier the ACK reception) and elects a subscriber with RankR as the Ack. The RankR selected can depend on the rate the publisher would like to serve all the subscribers. e.g., the publisher would choose Ranklowest for the next time period (t), the publisher ranks the subscribers according to their RTT, loss_rate and perceived throughput and accordingly chooses the new subscriber who is ranked Ranklowest for the new Ack.

#### 3.2.3 Layered multicast support

Based on the needs of an application, it may be desirable to support layered multicast. E.g., certain publishers might require that all its subscribers receive a certain quality of data, i.e., all the subscribers must receive all the packets belonging to a certain subset (corresponding to packets for the lowest quality). In such a scenario, the publisher can set up two groups, one containing the subset that all of them must receive and choose the Ack as the lowest ranked subscriber and have another group for the rest of the packets where he can send at a faster rate by choosing the subscriber with RankR = 75. The subscriber can also use query-response from other subscribers who have received the data, if it makes sense from an application point of view. E.g., in the case of video conferencing, using query-response would allow the subscriber to keep track of discussions, but not actively participate in the conference.

### 3.3 Local Repair: Reliability and Enhanced Throughput

R-COPSS is designed to enhance reliability by performing local repair, i.e., obtain the missing packets from its neighbors. This would allow publishers to support a higher sending rate even in the presence of certain subscribers who are behind a slow or congested link. The request for a missing packet is no different than the query for a piece of data in CCN. As long as the data has a globally unique name, the network can forward the request to all the potential providers (i.e., other subscribers) and deliver the data efficiently. Although such a local repair mechanism is easily facilitated in a CCN environment, the difficulty is in identifying and reaching subscribers who have received the content earlier. Moreover, another issue is that the repair data could traverse the same congested path that the original data arrived on. Here, we describe a repair strategy that is opti-
Content Name: In order to achieve reliability with pub/sub, we require every packet to have a ContentName just as Data packets are named in CCN. But the ContentName does not have to be an explicit field. It can be embedded in the content of a Publish packet. Since every ContentName in Data packet has a segment number as part of the name, the publisher can provide a sequence number for every packet and the subscribers can detect loss using this sequence number. When requesting repair, the subscriber sends an Interest with the ContentName and the content centric network retrieve the packet from any of the subscribers. The naming structure and the design of Interest packet in CCN can also help the subscribers define priorities associated with packets. Every Interest packet contains an “Exclude” field and a “ChildSelector” field. By setting these two fields, the subscriber can easily specify the range he wants. E.g., an audio conference application receives packet n, but missing packets from n − 5 to n − 1. By setting Exclude=“Smaller than n − 6” and using “Smallest” in child selection, the subscriber can request packets within the window.

Applications should be able to determine which packets are of higher priority to be received by repair and which others are less important or unnecessary (timed-out). E.g., for streaming (reliable text transfer, video sharing, etc), a subscriber can only use the content if the whole stream is delivered. So the subscriber needs to recover every packet that is missing. Of course, for delivery to that application the earliest packet has the highest priority. But in applications like audio/video conferences and live streaming, subscribers usually need packets within a window (to optimize QoS). Older packets are timed-out so there is no need to recover those packets.

3.3.2 Local Repair: Forwarding towards Publisher

The COPSS framework provides features for a publisher to be able to authorize, authenticate, and exercise access control to content. Therefore, even though the subscriber does not necessarily know the individual publishers at subscription time, when the content does come from a publisher to a subscriber, there is some identity associated with the data that can enable the subscriber to reach a publisher. In [9], we proposed Rendezvous Point (RP)-based FIB propagation where the path to the RP (going upstream) is part of the path towards the publisher.

In order to perform repair via the publisher, the router will forward the packet according to the FIB using the ContentName and according to the subscription table (ST) using the Content Descriptors (CDs) contained in it. E.g., in Fig 1, when S2 requests a packet, he sends an Interest for the missing packet. On seeing the Interest, R5 will forward it upstream to R2 (towards the publisher) according to FIB. Note that the repair request can also be aggregated. If more than one subscriber in the subtree experienced a packet loss, they will send repair request (nearly) simultaneously. On receiving a request, if there already exists a PIT entry pointing to the RP direction (it is a PIT from ST-based forwarding), the PIT will be consumed.

3.3.3 Local Repair: Forwarding towards other subscribers

We know that in CCN, every data owner can be a potential data provider (by propagating a FIB entry). It is the same in our pub/sub repair situation. The other subscribers that received a packet can also choose to be a data provider for that packet. However, to limit the total FIB size, we do not suggest every subscriber propagate the new ‘willing to serve’ information throughout the network. Moreover, the network already has the information of every subscriber’s interest without the need for new FIB entries. The subscription Table (ST) records the outgoing faces on every router. This information can also help with the repair request dissemination.

Based on Subscription Table entry: In order to perform a repair via the ST entry, the Interest packet would also need to have CDs as forwarding reference. On receiving a packet, the router, based on its ST, will forward the Interest to the subscribers downstream. Additionally, as explained above, based on the FIB, the router can forward the Interest upwards to the publisher. To avoid flooding the request to all the subscribers, we can limit the TTL in ST-based forwarding. If the local subscribers do not have the packet as well, the request will then have to go to the publisher directly. There is a tradeoff between network load and repair latency. The larger TTL in the request, the higher the possibility the Interest (request) can ‘break through’ the congested area and reach a subscriber that has the packet. Thus, the data can potentially be returned earlier than the RTT to the publisher. However, that might also result in higher load in the network. We suggest the application can adjust TTL dynamically according to the network situation. After several probes, the application can find a proper TTL to strive a balance between latency and network load.

Congested Path Avoidance - Local FIB Propagation: A very critical problem facing the local repair is: the data packet has to share the bandwidth in the multicast tree. No matter whether the repair request is sent to a subscriber, a cache in the network or the publisher, the Data packet (repair) returned has to potentially use part of the multicast tree, thus exacerbating congestion. One possible solution to the problem is local FIB propagation. After receiving the Publication from the publisher, the subscriber(s) propagate the FIB for the stream only locally (limiting the range in hops). E.g., a publisher publishes packets with ContentName:

/icn/papers/congestioncontrol.pdf/v_1/s_n,
the subscribers can propagate FIB prefix:

```
/icc/papers/congestioncontrol.pdf/v.1
```

within a certain range. So, when another subscriber requests a packet within that range, the request might have a chance to get routed via another path in the repair phase.

In Fig. 1, the link between R_3 and R_5 is congested so that Subscriber S_2 loses several packets in the “congestioncontrol.pdf” we mentioned before. Independent of whether the network uses a repair request via another subscriber, via the publisher or via an in-network cache, inevitably, the repair packet has to go from R_3 to R_7. But with our proposed solution, S_1 propagates the FIB in a small range temporarily (see the red arrow). The request from S_2 can then have a chance to go directly from R_7 through R_6 to S_1 and thus avoid the congested path.

This solution does not violate the privacy requirement of the subscribers either. Since the network uses the Content-Name as the identify of source and destination, a subscriber who provides repair capability doesn’t reveal his participant in the group by propagating the FIB of a prefix in a data. Just as is shown in the example, S_1 doesn’t reveal his own address or identity.

**Impact of FIB vs probability of receiving the data:**
To reduce the number of FIB entries, one may divide the flow into sub-flows. Subscribers who receive all the packets of a sub-flow make a FIB registration allowing the possibility for other subscribers with lost packet(s) to make a request for them without having to go to the publisher. Here, the number of FIB entries are fewer compared to registering every packet, and the subscribers achieve a data-hit with other subscribers.

### 4 Preliminary Evaluation

In this section, we simulate a R-COPSS implementation in our simulator. To better understand the performance of our solution, we carry out the simulation on a synthetic topology shown in Fig. 2. The publisher is connected to the RP of the 4-level multicast tree with an 8Mbps bandwidth. The bandwidth to the edge routers are varied from 1Mbps to 8Mbps accordingly to emulate different bottleneck scenarios and the subscribers further down that tree are named according to the bottleneck bandwidth ($S_{\text{bandwidth}=a}$). All the edge routers are linked to their neighbors with different bandwidth. We assume that the bandwidth between core routers are high enough (100Mbps) as the real-life situation. We also assume that the CCN routers in the network can perform line-speed forwarding. The payload of the trace is a 120Mb file divided into 30,000 chunks (4kb/chunk) and the real traffic delivered from the publisher is around 131.3Mb including the size of the packet headers. We assume an application scenario where the subscribers must receive all the packets either via multicast or via repair.

#### 4.1 Influence of ACKer Selection:

Here we study the influence of ACKer selection and repair on the overall throughput and the individual throughput achieved by the subscribers. In Fig. 3, we illustrate the effect of different ACKer on the overall throughput the system can achieve. The ACKer is arbitrarily selected at the beginning of the flow to emulate a subscriber who is behind a bottleneck link with a certain bandwidth thereby limiting the rate of publishing by the publisher (since the publisher aligns to the ACKer). By choosing $S_1$ to $S_8$ as the ACKer, the publisher can achieve a publication rate ranging from 1Mbps to 8Mbps. However, as we can observe, a higher publishing rate does not necessarily result in a higher overall throughput. When having a higher publication rate, the slower subscribers see more packet drops and that will result in an increase in the amount of repair. In the case of repair via publisher, the overall throughput increases steadily till $S_{\text{bandwidth}=5}$, and then begins to fall due to the high amount of requests going to the publisher. On the other hand the average throughput in the case of repair via subscriber increases steadily till $S_{\text{bandwidth}=6}$, and then flattens out, thereby positively indicating that the selection of a wrong ACKer does not harm the throughput. Note that the throughput achieved in the case of a pure Query/Response approach is about 1.9Mbps under the same settings.

In Fig. 4, we illustrate the effect of choosing a different ACKer on the throughput experienced by the individual subscribers in repair via subscribers. We can observe that as the ACKer varies, the individual subscribers are able to achieve a rate that is either equal to or above their bottleneck bandwidth, with the upper-bound being the bandwidth of the ACKer.
in the real application (e.g., video transfer, conferencing, etc) and provide better solution in the ICN world. Another direction of the research will be, how to reduce the wasted traffic on the minority so as to save the total network load and provide better service for the other flows.

6 Acknowledgments

The research leading to these results has received funding from the EU-JAPAN initiative by the EC Seventh Framework Programme (FP7/2007-2013) Grant Agreement No. 608518 (GreenICN) and NICT under Contract No. 167. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

7 References


4.2 Fairness:

In Fig. 6, we evaluate the scenario of two publishers having the same ACKer and starting 20s apart. We observe that the subscribers are able to share the bottleneck bandwidth of the ACKer fairly.

We can observe that the subscribers are able to share the bottleneck bandwidth of the ACKer.

5 Summary

R-COPSS enhances a content oriented pub/sub system with flow and congestion control and reliability. We show that the combination of multicast based delivery and query/response based local repair enables R-COPSS to support a rate that is faster than the slowest subscriber’s receive rate. In fact, the average throughput achieved across all the subscribers is greater than or equal to their bottleneck link rate, showing that the congestion control mechanism is effective. R-COPSS is also able to achieve fairness across publishers.

Much future work remains to be done. An analytical model of the ACKer selection will be studied to provide more efficient multicast in terms of both throughput and network load. We will also seek the effectiveness of layered multicast

Figure 5: Repair efficiency (Note that S1 – S8 are not shown, since all of them achieve a bandwidth = 6Mbps)

Figure 6: Fairness of two flows competing 1Mbps bottleneck

In Fig. 5, we investigate further into the individual throughput achieved by the subscribers (the ACKer is $S_{\text{bandwidth}}=6$ Mbps in this case). We can observe that $S_1$ is able to achieve a throughput of about 2Mbps as long as the multicast is on (1Mbps via multicast and 1Mbps via subscriber repair) and able to achieve a throughput of 1Mbps via local repair as soon as the multicast is over (at 23s). Other subscribers observe a similar trend. We can also observe that the subscribers take a while to achieve their max throughput since, at the beginning, very few subscribers have (and serve) the data packets that other subscribers want.

Figure 6: Fairness of two flows competing 1Mbps bottleneck

In Fig. 6, we evaluate the scenario of two publishers having the same ACKer and starting 20s apart. We observe that the publishers are able to share the bottleneck bandwidth of the ACKer.