Liquid Data Networking

John W. Byers
Dept. of Computer Science, Boston University, Boston MA

Michael Luby
ICSI and BitRipple, Inc., Berkeley CA

ABSTRACT

We introduce Liquid Data Networking (LDN), an ICN architecture that is designed to enable the benefits of erasure-code enabled object delivery. A primary contribution of this work is the introduction of SOPIs, a simple and efficient naming mechanism enabling clients to concurrently download encoded data over multiple interfaces for the same object, to optimize caching efficiency, and to enable seamless mobility. LDN offers a clean separation of security into object security and data packet security. An evaluation of the architecture and its use with various types of erasure codes is provided.

CCS CONCEPTS

• **Networks** → **Naming and addressing**; **Network architectures**.

KEYWORDS

Erasure code, ECA, liquid data, LDN, NDN, SOPI

1 INTRODUCTION

Information Centric Networking (ICN), and in particular NDN [31], has many architectural benefits, notably in naming, a transparent approach to security, and in caching content near interest. Several recent proposals have sought to augment NDN with erasure codes, in an effort to realize a range of reliability and performance benefits – we discuss these in more detail in Section 1.2.

This paper introduces *Liquid Data Networking (LDN)*, an ICN architecture designed explicitly to leverage encoded data and erasure codes. Our exploration of the design space highlights the benefits of close integration between naming of objects and naming of encoded data symbols, an aspect not fully explored in prior work. LDN provides provable reliability, performance, and security benefits, is simple, and enables a scalable implementation with low overheads. In this paper we present the basic design of LDN and discuss how the design achieves key evaluation objectives.

1.1 Erasure codes and erasure code architectures

A long stream of research has advocated the use of an *erasure code* approach to reliable object delivery and storage. Example domains include reliable multicast, parallel downloads, network coding, and distributed storage [3], [4], [5], [7], [8], [10], [15], [22], [25]. Briefly, an erasure code takes as input a data object consisting of K source symbols, and produces a set of M repair symbols from the object through an erasure encoding process. The key property of an erasure code is in the decoding process: the object can be recovered from *any* subset of at least K of the $K+M$ symbols (any mix of source and repair symbols), together with their unique identifiers. With erasure codes, the source and repair symbols can be viewed as *liquid data*, by which we mean that symbols are interchangeable; the *number* of distinct symbols at the decoder determines whether or not an object is recoverable, and not the *specific set* of symbols. Our proposed architecture is based on leveraging this fundamental distinction.

As examples of networking benefits achieved with erasure codes in prior work, clients can download objects faster across multiple interfaces [19], packet loss resiliency is provided automatically in transmission protocols [4], [22], simpler protocols for mobile clients are enabled [27], storage of objects is more resilient [8], [10], [15], caching efficiency can be improved [21], and dynamically changing interfaces are simpler to handle. In this work, we seek to obtain all of these benefits in the context of a unified ICN network architecture in which erasure codes are integrated directly – we refer to such a design as an *erasure code architecture* (ECA). Our proposed architecture, *Liquid Data Networking (LDN)*, is framed in the context of NDN and is principled on the desirable properties that any ECA should seek to obtain. In defining these properties and the metrics we propose for evaluation of ECAs, we identify object to network data name-mappings, request-response paradigms, and security considerations as critical components that necessitate careful design and deep integration in the design and realization of an ECA.

1.2 ECAs in the context of prior work

NDN [31] is the starting point of a majority of the previous work on integrating erasure codes into an ICN. In NDN, each object is partitioned into packets, where each packet is assigned a name based on the object name and the position of the packet within the object. *Clients* request and receive objects. Each object can be individually requested by a client via interest messages. To receive an object, a client requests each packet of the object, and can recover the object after all packets are received. Each object is initially provided to one or more *publisher nodes* (PNs). A *node* provides routing and caching functionality: it accepts interest messages for packets of the object and responds with the packets if they are cached locally, and otherwise stores and forwards the interest messages towards a publisher node, and responds with the packets when they arrive.

There have been numerous proposals to augment NDN based on Random Linear Network Coding (RLNC), fountain codes, and other types of erasure codes [1], [11], [12], [13], [17], [18], [20], [23], [24], [28], [30]. All designs incorporate *encoding nodes (ENs)*, nodes augmented with the ability to generate encoded data, to provide some of the reliability and performance benefits described earlier.

The research described in this paper is supported by NSF EAGER Award 1936572 and NSF NeTS Award 1815016.

ICN '20, September 29-October 1, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8040-9/20/09…$15.00
https://doi.org/10.1145/3405656.3418710

Evaluation metrics for ECAs: Metrics relevant to evaluating erasure code architectures, in a variety of use cases, are:

- *Signaling overheads*: The size of the signaling information in interest messages and their responses, and the time complexity of computing this signaling information.
- *Latency overheads*: For real-time objects, the latency between when an object is published and when the object downloaded by a client is available at the client. For on-demand objects, the latency between when a client expresses interest in the object and when the object is available at the client.
- *Security overheads*: The time complexity of object verification and packet verification (See Section 3).
- *Response overheads*: The amount of redundant encoded data received for an object from multiple nodes over multiple interfaces. Redundantly received encoded data is not useful for recovery of an object.
- *Storage overheads*: The amount of storage used for caching encoded data in the network.

While all of these types of overheads are evaluation criteria for ICN architectures broadly, use of erasure codes have potential to introduce new costs, so we specifically consider evaluation of that incremental overhead in ECAs.

Many prior proposals allow *blind encoding*, where *blind* refers to the setting where an encoding node generates encoded data from other encoded data of an object when it does not have the full object. See e.g., [1], [12], [13], [17], [18], [23], [24], [28], [30]. Blind encoding incurs significant signaling overheads, and can incur highly variable response overheads, as for example discussed in [29]. Furthermore, as detailed in Section 3, there are some serious security issues with the usage of blind encoding in an ECA. Thus, we hereafter focus on ECAs that disallow blind encoding.

Request paradigms for ECAs: Another central issue in designing an ECA is the request paradigm a client uses to request data from a node to recover an object. The three request paradigms described in prior work are:

- *Specific data*: The client asks for specific encoded data for the object by name, and the encoding node responds with the specific named data. This generalizes the approach taken by NDN [31] to the case of encoded data.
- *Random data*: The client requests an amount of encoded data generated from the object, and the encoding node responds with randomly generated encoded data. See e.g., [11], [12], [17], [23] and [24].
- *Useful data*: The client specifies data it has already received for the object, and the encoding node calculates and responds with useful encoded data that will help the client recover the object. See e.g., [18] and [28].

There are pros and cons to each of these approaches, but none is ideal. The *specific data* approach ensures that whatever data each client receives is useful and has low signaling overhead, but this uncoordinated approach has poor caching behavior and results in high storage and latency overheads.

The most popular heuristic is the *random data* approach, which also has low signaling overhead, and sometimes has good cache behavior, but has several other drawbacks. Both response and latency overheads are unpredictable, since in general there is no assurance that received encoded data will be useful to the client.

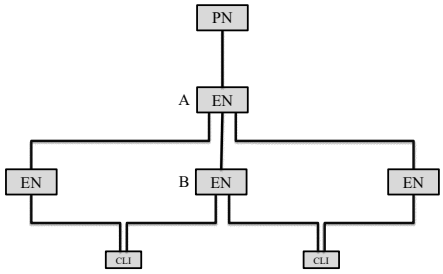

Figure 1: Turkey network topology

To illustrate, consider the turkey-shaped network shown in Figure 1. Suppose that each of the two clients send interest messages for one-half of the encoded data needed to recover an object over each of their two interfaces. If the policy of encoding nodes *A* and *B* is to *aggregate* interest messages received on distinct interfaces (e.g., to improve cache behavior), then encoding node *A* ends up requesting only one-half of the encoded data needed to recover the object. Although both clients will technically receive a total amount of encoded data equal to the object size, it will consist of two duplicate copies of the encoding data requested by *A*, routed on different paths. If instead the policy of encoding nodes *A* and *B* is to keep all of their received interest messages *disaggregated*, then encoding node *A* receives four halves, or 2x the encoded data needed (*B* also receives more than it needs) and thus the response overheads and cache overheads are high. For comparison, see the description in Section 2.4 with respect to Figure 1.

Finally, the *useful data* approach is intuitively appealing but is technically problematic for several reasons described in prior work [24]; moreover, it still has unpredictable latency overheads and additional signaling overheads.

2 LDN DESIGN OVERVIEW

The review of prior work in context highlights the key design challenge for ECAs: How does or can one retain the simplicity and security of the NDN interest-response paradigm while infusing all of the reliability benefits of erasure codes? As the review of request paradigms in prior work illustrates, it is not sufficient to graft codes on top of an ICN. Instead, it turns out to be imperative to integrate codes fundamentally into the architecture itself, with design of appropriate naming and security policies. The key consideration is *coordination*, as the benefits of an ECA described previously (and other benefits we discuss later) are fully realized when certain coordination properties are guaranteed. But before embarking on that, we present basic terminology and concepts of our design.

2.1 LDN terminology

LDN nodes (NNs) have routing and caching functionality, and thus are akin to elements in the spirit of NDN [31]. An encoding node is a NN with the additional ability to generate encoded data from an object, and to recover objects by decoding received encoded data for the object. Each object is initially provided to one or more encoding nodes. An EN is said to be a *publisher node* for an object if the EN has the entire object in its cache. The two ways an EN can become

a publisher node are either by directly ingesting the object, or by downloading enough encoded data to recover the object.

A *client (CLI)* receives requests for objects to download from the application layer, issues requests for encoded data for those objects, and generates the objects from the received encoded data responses. Requests for encoded data of an object percolate from the client towards one or more publisher nodes for the object.

Applications operate on objects, which are sequences of data that are useful to an application, e.g., a video file, a video segment, an audio segment, a photo, etc. LDN is designed to deliver objects to clients, where an object is immutable and has a unique identifier. When creating encoded data from an object, there is an associated *symbol size* that is used to partition the object into source symbols which are used for erasure encoding and erasure decoding. The symbol size is a parameter that can be automatically determined based on factors such as the network packet MTU.

Additional mechanisms are needed to provide object identifiers for objects to interested clients. Encoding nodes also require routing protocols to determine over which interfaces to send requests for encoded data for particular objects, i.e., along interfaces that are typically closer to publisher nodes of objects. While defining these protocols are out of scope for this short paper, we anticipate that methods described for other ICN architectures would be applicable here, for example those described in NDN.

2.2 Coordination in LDN

Maintaining the low overheads described in the introduction is possible only if the following coordination properties are guaranteed *simultaneously*:

- *Cacheability:* Different clients should receive overlapping encoded data for an object from the same node. This ensures caching efficiency and low storage overheads.
- *Additivity:* Any and all encoded data for an object received by a client should be useful to recover the object. This ensures that network resources are used efficiently. For example, a client should receive different and useful encoded data for an object from different nodes. This is necessary to enable efficient reception of encoded data for an object over multiple interfaces and to support mobile clients that download encoded data from different nodes over time.

The LDN design in many ways resembles NDN, but introduces an object to network data name-mapping and a request-response paradigm that harmonizes with erasure codes. In particular, when a client wants to download an object, the client sends interest messages requesting specific amounts of encoded data from specific sets of encoded data generated from the object. This generalizes the specific data approach used by NDN, but (provably) sidesteps the latency and storage overheads of the naive approach. It can also be viewed as a generalization of the random data approach.

The *stream object permutation identifier (SOPI)* is a key to the LDN design of the object to network data name-mapping and request-response paradigm. SOPIs are used to control the specific set of packets containing encoded data that clients will request from specific nodes within the system. This simple mechanism ensures the coordination properties described above. Furthermore, the signaling protocols to distribute and use SOPIs are relatively straightforward,

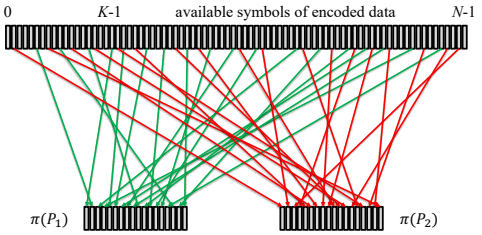

Figure 2: SOPI illustration

and the additional signaling complexity a client incurs is independent of the number of objects downloaded by the client.

2.3 SOPIs and stream objects

Stream objects are fundamental to the design of LDN: they enable a diversity of encoded data to be available for download within the network for each object, while at the same time ensure that different clients request the same encoded data for an object from the same neighboring node. Stream objects can be described in terms of an erasure code with optimal recovery properties and with the property that N symbols can be generated from any object, where N is a large number (and without loss of generality, N is prime).

A stream object for an object logically consists of the N available symbols of encoded data for the object in a specified order. The essential idea is that different stream objects specify completely different orderings of the available encoded data for an object, and thus a client can simply request prefixes of different stream objects for an object to receive different, and thus useful, encoded data. While a stream object is conceptually massive compared to the underlying object, typically only a very small portion of a stream object is present in the network at any one time.

A *stream object permutation identifier (SOPI)* specifies the ordering of encoded data for a stream object. A SOPI P identifies a permutation $\pi(P)$ of the N available symbol identifiers. Thus, a stream object identifier (D, P) is the combination of the identifier D of the object from which it is generated and a SOPI P. We consider SOPIs of the form $P = (A, B)$, where $A \in \{0, 1, 2, \ldots, N-1\}$ and $B \in \{1, 2, 3, \ldots, N-1\}$ define the permutation of symbol identifiers

$$\pi(P) = \{A, A+B, A+2\cdot B, \ldots, A+(N-1)\cdot B\},$$

where each term is taken modulo N.[1] The number of possible stream objects for an object is $N \cdot (N-1)$.

SOPIs are a mechanism to control what encoded data is requested, delivered, and stored in the network. A SOPI is associated with each node. To download an object, clients request a prefix[2] of a stream object associated with the SOPI of an edge node reachable over the interface. SOPIs ensures that same encoded data for an object will be requested from the same edge node by all clients, but that different encoded data will be requested from different edge nodes.

Figure 2 shows the available encoded data for a a given data object at the top, and the caches of two edge NNs with associated SOPIs P_1 and P_2 at the bottom, where the green arrows show the mapping of encoded data to a prefix of stream object P_1 in the left cache, and the red arrows show the mapping the encoded data to a

[1] Note that $\pi(P)$ is a permutation of $\{0, \ldots, N-1\}$ since N is prime.
[2] The length of the prefix can be flexible, depending on network conditions, for example.

prefix of stream object P_2 in the right cache. Clients request a prefix of stream object P_1 from the left NN, and a prefix of stream object P_2 from the right NN, and encoded data received by a client for an object from both NNs is useful to recover the object.

The SOPI and stream object design allows client decisions of which encoded data to request to be simple, robust, and efficient. In particular, SOPIs simultaneously obtain the benefits of the request paradigms for ECA described in Section 1.1, achieve the two coordination properties described in Section 2.2, and incur negligible signaling overhead. Clients request *specific* data from ENs by virtue of SOPIs, and the SOPI coordination mechanism also ensures the data at each EN is cacheable for many clients. When requesting from a given SOPI, the requested encoded data appears *random* with respect to the canonical order of the available encoded data. And finally, the requested data is provably *useful* with high probability, by virtue of the additivity property that SOPIs enable (statement of guarantees appears in Section 2.5).

2.4 LDN request-response paradigm

The client sends SOPI probe messages over an interface when the client joins the interface, and the responses are the SOPIs of the edge nodes reachable over that interface. A client interest message sent over an interface for receiving encoded data for an object includes the object name D, the SOPI P of an edge node reachable over the interface, a starting position S and ending position E within the prefix of stream object (D, P) from which to receive encoded data, and the amount A of encoded data that should be included in the response. The response should be to send an amount A of encoded data within the range S to E of stream object (D, P).

Normally, $S = 0$ and $E >> A$ when a client sends a first interest message for encoded data from a stream object. Allowing $E-S >> A$ allows great flexibility, i.e., nodes can respond with already cached encoded data from the stream object as long as it is in range.

When a client discontinues downloading an object and then continues downloading again at a later point in time from the same stream object, S can be reset to the smallest position in a prefix of the stream object for which the client has not received any encoded data at or beyond that position. This simple mechanism allows a client to seamlessly continue downloading useful encoded data.

The client encoded data request protocol is simple: A client requests enough prefixes of stream objects in aggregate from neighboring nodes so that the amount of encoded data that arrives is at least the object size. If some requested encoded data does not arrive, the client requests more of the prefixes from the same stream objects.

Responses to encoded data interest messages for an object received by a node are provided by the node if the requested encoded data is cached at the node. The same holds for an encoding node, but an encoding node can also respond if it is a publisher node for the object and is therefore able to generate the requested encoded data. Similar to NDN, if neither of these cases hold, a decision is made on whether to forward the full or partial encoded data interest message toward a publisher node, based on whether or not previously forwarded interest messages have already requested overlapping encoded data. Similar to NDN, when responses to forwarded requests are received, they are passed back over the interfaces from which the original interest messages were received. Publisher nodes respond to encoded data interest messages with cached encoded data or with encoded data generated from the object.

The encoded data request protocols ensure:

- *Bandwidth efficiency*: The response overheads are minimal.
- *Cache efficiency*: The storage overheads are minimal.
- *Responsiveness*: The latency overheads are minimal.

To illustrate, consider again the network shown in Figure 1. Suppose that each of the two clients send interest messages for one-half of the encoded data needed to recover an object over each of their two interfaces. Because of the SOPI design, the interest messages from both clients to encoding node B request the same encoded data, and the interest messages sent to the other two bottom encoding nodes request different encoded data. Encoding node A receives three interest message requests for different encoded data, and forwards two of the requests to the publisher node. Encoding node A passes received responses down, recovers the object, and generates and sends the remaining responses down. The bottom encoding nodes pass responses down as they are received from above, and both clients are able to recover the object. Overall, the amount of encoded data received by encoding node A and encoding node B is the object size and one-half the object size, respectively. Thus, the response overheads and cache overheads are low. For comparison, see the description in Section 1.2 with respect to Figure 1.

2.5 SOPI response overhead guarantees

Suppose a perfect erasure code is used to generate encoded data from an object and recover the object from received encoded data, i.e., an object with K source symbols can be recovered from any K encoded symbols among the N available encoded symbols.

The paper [14] proves probabilistic response overhead guarantees when SOPIs are randomly chosen and deterministic response overhead guarantees when SOPIs are generated systematically. As an example of a probabilistic guarantee, when $N = 2^{31} - 1$ and $K \leq 50,000$, an object with K source symbols can be recovered with probability at least 0.999995 from $1.01 \cdot K$ symbols received from any number of stream objects with different SOPIs. Alternatively, when $N = 2^{31} - 1$ and $K \leq 30,000$, a set of 15 billion SOPIs can be generated with the following deterministic guarantee: an object with K source symbols can be recovered with certainty from $1.01 \cdot K + 1$ symbols received from prefixes of any pair of stream objects with different SOPIs. As another example, when $N = 2^{31} - 1$ and $10,000 \leq K \leq 30,000$, a set of 150 million SOPIs can be generated with the following guarantee: an object with K source symbols can be recovered with certainty from $1.015 \cdot K$ symbols received from prefixes of any ten stream objects with different SOPIs.

2.6 Which erasure codes are right for LDN?

Marrying an ideal erasure code to LDN is a key design decision. To minimize overheads, two highly desirable properties of an erasure code for use within LDN are:

- Encoding and decoding are linear time operations, and thus the size of an object that can be efficiently encoded and decoded can be very large and is not constrained.
- $N >> K$, i.e., the number N of available symbols of encoded data for an object is much much larger than the number K of source symbols in the object.

While the desirability of linear-time encoding/decoding is apparent, the rationale for $N >> K$ may be less clear. SOPIs benefit directly from the availability of a large amount of encoded data, because this keeps incidental cross-collisions of encoded data in prefixes of SOPIs to a minimum. Conversely the *cost* of $N >> K$ is negligible under the assumption of an erasure code for which each incremental symbol needed can be generated (or decoded) in constant time.

Many types of erasure codes can be used in LDN, although some are better suited than others. RLNC codes exhibit a significant trade-off between encoding/decoding complexity and response overheads as a function of the source block size, and Reed-Solomon codes exhibit a similar but slightly less severe trade-off.

Our recommended erasure code for LDN is a *fountain code*. The BAT fountain code described in [29] exhibits a reasonable trade-off between encoding/decoding complexity and response overheads as a function of the source block size, and the RaptorQ fountain code described in [16], [26] exhibits a close to optimal trade-off.

3 SECURITY IN AN ECA

Object verification has been identified as a critical aspect of any ICN architecture. An established provenance approach to support object verification is to enable the data source, which we call the *originator*, to digitally sign objects as they are generated.

Another important security aspect is to protect against a denial of service attack. In particular, packets that contain data that are either not useful to recover an object or are bogus can waste valuable network resources and pose a denial of service attack.

We summarize these two ECA security properties as follows:

- *Object verification*: Ensure that objects recovered at clients are identical to objects generated by originators.
- *Packet verification*: Ensure that only useful packets are transported and cached with the network.

3.1 ECA security in context

NDN [31] uses a simple and elegant security design to provide both object verification and packet verification. A publisher node uses a standard digital signature scheme to sign each packet of an object, and trust of the publisher node credentials implies trust that if the packet signature is valid than the packet is valid. This provides packet verification, since a node can verify the signature of a packet, and accepts the packet only if the signature is valid. This also provides object verification, since an NDN object equates to an ordered set of packets, so an object is valid iff all packet signatures are valid.

NDN does not distinguish between originators and publisher nodes, and the most natural assumption is that originators and publisher nodes are synonymous. This is not ideal, as it forces originators to be network-aware, i.e., originators are providing object verification on packets, not objects, and thus originators must partition objects into packets of a size suitable for delivery over the network and sign the individual packets. A cleaner and more flexible architecture would be to limit originator functionality to creating objects and limit publisher node functionality to being the source of objects available within the network.

Many prior ECA approaches allow blind encoding (see Section 1.2), which makes packet verification impossible based on the NDN approach to packet verification using standard digital schemes. As pointed out by [2], even if an encoding node correctly generates and signs a packet with private credentials (as in [12]), it cannot be certain the packet is correct, since it cannot be certain the packets from which the packet is generated are correct (they could be generated by misbehaving encoding nodes). Thus it is impossible to blame an encoding node if a packet it generates is found to be corrupt.

The paper [2] introduces an elegant extension of the NDN security scheme based on the homomorphic signature scheme described in [6] that provides object verification and packet verification when blind encoding is allowed. However, homomorphic signature schemes are orders of magnitude more complex than standard digital signature schemes, so the efficacy of this approach is highly impractical at present. Also, as with NDN, originators must be network-aware.

3.2 LDN security

LDN introduces *originators* to enable end-to-end object verification that is independent of packet structure. LDN also includes packet verification which allows blame to be assigned to publisher nodes that generate corrupt packets. The design cleanly separates object provenance security requirements from network delivery security requirements. Both object verification and packet verification use standard digital signature schemes. Databases of public credentials (e.g., stored in a blockchain) are assumed to be available.

Object verification is simple: The originator of an object signs the object, and a client or publisher node verifies a signed object using the public credentials of the originator.

Packet verification works as follows. Before generating an encoded data packet for any object, a publisher node must first perform object verification on the object. Then it generates encoded data from the object, packetizes the encoded data into a packet, and signs the concatenation of the packet, the object signature, and the originator public credentials.

A node, encoding node or client that receives a signed packet can use the publisher node public credentials to verify the packet. (We assume that public credentials for originators and publisher nodes can be securely obtained through standard mechanisms that are out of scope for this paper). If the signed packet is not verified then it is discarded. If the signed packet is verified then it is accepted and deemed useful for recovering the object from which it was generated.

However, a signed packet that is verified is still untrusted. A misbehaving publisher node can generate and sign a corrupt packet, a corrupt object will be recovered from use of the corrupt packet, and the corrupt object will later be discarded based on its invalid signature. Thus, a misbehaving publisher node can mount a successful denial of service attack. But to stop an attack, it is now possible for any other trustworthy publisher node to produce a short proof that can be used to identify and blackball the misbehaving publisher node. The input to the proof is the signed correct object, the signed corrupt packet, and the originator and misbehaving publisher node public credentials. The proof consists of verifying the correct object and corrupt packet signatures, generating the correct packet from the object, and verifying that the correct and corrupt packets differ.

4 LDN USE CASES

It is useful to study some basic use case examples of LDN to understand and evaluate the design in a variety of network environments.

4.1 Tree

Consider a tree network of encoding nodes with a publisher node at the root, and clients connecting to one or more neighboring encoding nodes at the leaves to download encoded data for an object. It can be verified that with the SOPI approach, the amount of encoded data cached at an encoding node, and the amount of encoded data that flows out of an encoding node is minimized with compared to any other design with the same network topology. Furthermore, the amount of encoded data that flows through or is cached in any encoding node for an object is at most the object size.

4.2 Lossy Multi-Hop Wireless

Consider a multi-hop wireless network with a publisher node storing an object D at one end, a linear series of encoding nodes, and a client at the other end that wants to download D, where the packet loss rate across each hop is known to be 25%.

The client can request a prefix of stream object (D, P) of size around 1.33 times the object size (enough to protect against 25% packet loss). As encoded data generated by the publisher node arrives at its neighboring encoding node it is sent to the next encoding node, and when enough encoded data arrives to recover the object at the neighboring encoding node it generates any additional unsent requested encoded data for the object and sends it to the next encoding node. Each subsequent encoding node acts similarly. The SOPI-based solution provides hop-by-hop protection against packet loss, transmitting the minimum possible encoded data per-hop.

4.3 Roadside Transmitters

Consider a mobile client that travels in and out of the range of transmitters, e.g., a client embedded in an automobile traveling through an area of roadside transmitters equipped with nodes. The client can seamlessly download encoded data for an object, since there is no need to ensure that packets in transit from a transmitter that is no longer in range are redirected to a transmitter that is currently in range. If no client requests more than 20% of the object from any one node, then each node caches encoded data that is at most 20% of the object size. On the other hand, each client recovers the object as soon as it receives encoded data from nodes that is in aggregate the object size. This provides a simple, robust and seamless solution, using concepts similar to those in [9].

4.4 Load Balancing at the Edge

Suppose that a cache server farm is located near a set of clients. In the current Internet, each cache that serves a popular object D to any client stores a full copy of D, and thus each such object is fully replicated in each cache. When a request for D is received from a client at the load balancing switch, the switch picks a cache server to handle the request in a way that spreads out the downloads of D evenly across the caches.

How this can be handled by LDN is quite different: Each node stores a prefix of a different stream object for D that is a small fraction f of the object size, and the client issues requests for $1/f$ prefixes of different stream objects generated from the object. Load balancing of stream object downloads occurs naturally, and a much greater amount of popular content can be cached compared to the current Internet for the same amount of caching storage, using principles also elaborated in [15].

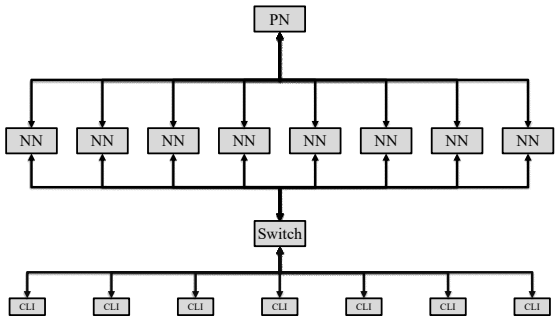

Figure 3: Edge with a load-balancing switch

Figure 3 shows a publisher node for objects, eight nodes that cache encoded data, a load balancing switch, and seven clients download encoded data for objects. The switch has the SOPI of each node, and provides each client with four of the eight SOPIs (chosen either at random, or to achieve a load balance condition). Each client requests a prefix of length slightly more than 1/4 of the object size from each of the four stream objects with the SOPIs it received from the switch, so that the aggregate amount of encoded data requested is slightly more than the object size. The switch sends received requests to the appropriate nodes, which provide the response to the client if the encoded data is already in cache. Otherwise, nodes send requests to the publisher node, and when encoded data responses return, they are sent to the client and are also cached. Since every NN caches a prefix of 1/4 the object size, this example yields a factor of four space savings when compared to current load balancing architectures.

5 SUMMARY AND CHALLENGES

Our work explores the use of erasure codes in an ICN that goes beyond inclusion of encoded data as a feature. We draw motivation from basic use cases where the benefits of erasure codes appear substantial, but are difficult to fully realize absent an approach that provisions for encoded data *within the network architecture*. We identify the architectural components of naming encoded data and establishing appropriate request-response paradigms as key challenges in such a design. With an approach based on novel identifiers (SOPIs), our LDN design specifies an erasure code architecture that builds architectural scaffolding for encoded data within NDN. LDN maintains the underlying integrity and simplicity of the NDN design, including security guarantees, and derives the full benefits of erasure codes with low marginal overheads.

Future research areas include protocols for identifying and blackballing misbehaving publisher nodes, assigning SOPIs to nodes (see [14]), and practical implementation and analysis of LDN.

ACKNOWLEDGEMENT

We are greatly indebted to our shepherd, John Wroclawski, and to the anonymous PC reviewers, who provided a wealth of constructive feedback. Their insights greatly influenced the overall thrust of the paper and helped sharpen our focus.

REFERENCES

[1] M. Bilal and S.G. Kang. "Network-Coding Approach for Information-Centric Networking". In: *IEEE Systems Journal*. Vol. 2. 2018.

[2] R. Boussaha, Y. Challal, M. Bessedik, and A. Bouabdallah. "Towards Authenticated Network Coding for Named Data Networking". In: *Proc. of 24th International Conference on Software, Telecommunications and Computer Networks (SoftCom '17)*. 2017.

[3] J. W. Byers, M. Luby, and M. Mitzenmacher. "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads". In: *Proc. of IEEE Infocom*. Mar. 1999.

[4] J.W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. "A digital fountain approach to reliable distribution of bulk data". In: *Proc. ACM SIGCOMM*. 1998.

[5] B. Calder et al. "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency". In: *Symposium on Operating System Principles*. 2011.

[6] D. Charles, K. Jain, and K. Lauter. "Signatures for network coding". In: *International Journal of Information and Coding Theory*. Vol. 1. 2009.

[7] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan. "Availability in globally distributed storage systems". In: *USENIX Symposium on Operating Systems Designs and Implementation*. 2010, pp. 1–7.

[8] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. "Erasure Coding in Windows Azure Storage". In: *USENIX Annual Technical Conference*. 2012.

[9] M. Al-Khalidi, N. Thomos, M. J. Reed, M. F. AL-Naday, and D. Trossen. *Seamless Handover in IP over ICN Networks: a Coding Approach*. 2017. arXiv: 1702.01963 [cs.NI].

[10] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, and W. Weimer. "Oceanstore: An architecture for global-scale persistent storage". In: *ACM SIGOPS Operating Systems Review* 34.5 (2000), pp. 190–201.

[11] K. Lei, S. Zhong, F. Zhu, K. Xu, and H. Zhang. "A NDN IoT Content Distribution Model with Network Coding Enhanced Forwarding Strategy for 5G". In: *IEEE Transactions on Industrial Informatics*. 2017.

[12] W.X. Liu, S.Z. Yu, G. Tan, and J. Cai. "Information-centric networking with built-in network coding to achieve multisource transmission at network layer". In: *Computer Networks 115*. 2017.

[13] Y. Liu and S.Z. Yu. "Network coding-based multisource content delivery in Content Centric Networking". In: *Journal of Network and Computer Applications 64*. 2016.

[14] M. Luby. "SOPI design and analysis for LDN". In: 2020. arXiv: 2008.13300 [cs.NI].

[15] M. Luby, R. Padovani, T. Richardson, L. Minder, and P. Aggarwal. "Liquid cloud storage". In: *ACM Transactions on Storage*. Vol. 15. 1. Apr. 2019.

[16] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. "RaptorQ Forward Error Correction Scheme for Object Delivery". In: *IETF RFC* 6330 (2011).

[17] K. Matsuzono, H. Asaeda, and T. Turletti. "Low latency low loss streaming using in-network coding and caching". In: *Proc. of IEEE INFOCOM*. 2017.

[18] M.J. Montpetit, C. Westphal, and D. Trossen. "Network coding meets information-centric networking: An architectural case for information dispersion through native network coding". In: *Proc. 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications*. 2012, pp. 3136–3136.

[19] J. J. Nielsen, R. Liu, and P. Popovski. "Ultra-reliable low latency communication using interface diversity". In: *IEEE Transactions on Communications* 66.3 (2017), pp. 1322–1334.

[20] G. Parisis, V. Sourlas, K. Katsaros, W. Chai, G. Pavlou, and I. Wakeman. "Efficient Content Delivery through Fountain Coding in Opportunistic Information-Centric Networks". In: *Computer Communications* 100 (Dec. 2016). DOI: 10.1016/j.comcom.2016.12.005.

[21] KV Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran. "EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 401–417.

[22] L. Rizzo. "Effective erasure codes for reliable computer communication protocols". In: *ACM SIGCOMM computer communication review* 27.2 (1997), pp. 24–36.

[23] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun. "Adaptive Video Streaming With Network Coding-Enabled Named Data Networking". In: *IEEE Transactions on Multimedia*. Vol. 10. 2017.

[24] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun. "CA network coding approach for content-centric networks". In: *Proc. of IEEE INFOCOM*. 2016.

[25] M. Sathiamoorty, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur. "XORing Elephants: Novel Erasure Codes for Big Data". In: *Proceedings of the VLDB Endowment*. Vol. 6. 5. 2013, pp. 325–336.

[26] A. Shokrollahi and M. Luby. *Raptor Codes*. Vol. 6. Foundations and Trends in Communications and Information Theory 3-4. now publishers, 2011, pp. 213–322.

[27] Y. Wang, S. Jain, M. Martonosi, and K. Fall. "Erasure-coding based routing for opportunistic networks". In: *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. 2005, pp. 229–236.

[28] Q. Wu, Z. Li, and G. Xie. "CodingCache: multipath-aware CCN cache with network coding". In: *3rd ACM SIGCOMM Workshop on Information centric Networking*. 2013, pp. 4142–4142.

[29] S. Yang and R. Yeung. "Batched Sparse Codes". In: *IEEE Transactions on Infomation Theory*. Vol. 60. 9. 2014, pp. 5322–5346.

[30] G. Zhang and Z. Xu. "Combing [sic] CCN with network coding: An architectural perspective". In: *Computer Networks 94*. 2016.

[31] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. "Named Data Networking". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 66–73. ISSN: 0146-4833.