The Role of Data Repositories in Named Data Networking

Lixia Zhang
Department of Computer Science
University of California, Los Angeles
Los Angeles, USA
lixia@cs.ucla.edu

Abstract

Persistent in-network storage repository (*repo* for short) has been an integral component in the Named Data Networking (NDN) architecture since the beginning of its design. Up until now, however, little attention has been paid to the design, development, and usage of repo, while various questions often arise with regard to the relations between repos and other storage components in an NDN-enabled system. This position paper aims to highlight the importance of repo, clarify those questions, and identify remaining work to be done to make repos widely usable in various application scenarios.

I. Introduction

As a new Internet architecture, Named Data Networking changes the network protocol architecture's narrow waist from packets carrying source and destination addresses to named data chunks [1]–[3]. That is, every network layer data packet contains a name and the corresponding content. This enables the original data producer to secure its data directly, by using a cryptographic signature to bind the name and content together; the producer may also encrypt the content whenever content confidentiality is required[1].

In today's Internet, by design network layer packets can only be identified by their source and destination addresses at network layer. Consequently, today's practice of network security is by securing the virtual communication channel between two communicating processes (e.g. using Transport Layer Security [4]); the data going into the channel is not verified, and the security protection is gone as soon as the data comes out of the channel. In sharp contrast, each data packet in an NDN network stands on its own with its identity and security, and can be verified by any receiving party, independent from the location the data packet is stored, or being fetched from.

The NDN forwarder design utilizes this unique property of named, secured data packets and develops a content store (CS) component in the forwarding plane to opportunistically cache passing-by data packets, to make them available in meeting future requests for the same data. However opportunistic caching alone is inadequate; as we explain below, what more needed is persistent in-network data storage that can be use by all applications to make their data available.

As a node-centric architecture, IP networking facilitated the development of client-server paradigm in application development, where a client node talks to a server node synchronously, i.e. both parties must be online at the same time. This is achieved by making all the servers available all the time, so that whenever a client reaches out to a server, communication can be established. Clients (user nodes) may go offline whenever they wish, but server nodes must be online all the time to make the client-server paradigm viable.

As a data-centric architecture, we expect NDN to facilitate new generations of truly peer-to-peer applications among end users. Given it is infeasible to assume that all user nodes would stay online all the time, NDN can enable truly peer-to-peer applications via in-network storage by making users data available all the time even when user devices are offline, so that peers can communicate *asynchronously* via in-network data storage.

Meeting this goal requires persistent and managed in-network storage, the role that NDN repos are supposed to fulfill. This role is critical to the overall NDN architecture, and NDN application development efforts have made heavy use of repos, and the application team has persistently requested help to make NDN repos more usable. Unfortunately, up to now there has been little attention from the broad ICN research community to this important research area.

The goal of this paper is to bring up the awareness of the critical role that repos play in the NDN architecture, and to identify several important research issues that must be addressed to make NDN repos widely usable to application development. These issues include

- The trust and security model of repos (Section);
- How to get data into repos (Section IV);
- How to fetch data from repos (Section V);
- How to synchronize data that are replicated across multiple repos (Section VI); and
- How to manage repos (Section VII)

This work is partially supported by the US National Science Foundation under awards CNS-1629922 and CNS-1719403.

[1]The signing and encryption can be done either by the data producing device itself, or by a producer-designated device when it is infeasible for the producer itself to perform these functions, such as the case in an Internet-of-Things scenario when a producer may be limited in its computation resource and power. However in these cases one must ensure the security of the data channel between the producer and the signing device.

Note that the above five issues are the identified ones at this time; by no means the list is exhaustive. Additional issues are likely to be identified when repos start getting widely used by application developments.

II. Repo Usage

As we have stated, repos are persistent in-network storage for data produced by all NDN applications. A repo may be a stand alone device, or a module attached to a device which is used to serve other purposes. For example, a repo can be attached to an application device, so that the application can publish the data packets its produces into the repo directly, which can then serve the data to all interested consumers without the application getting involved further[2]. A repo can also be directly attached to an NDN router, which can then be used to store frequently used data items. For example, routers on the NDN testbed have used their attached repos to store commonly used site certificates, making them widely and readily available to end users, independent from whether a specific site is or is not connected to the testbed at a given moment[3].

The first task of a repo is to get the data into storage. This task can be accomplished in a variety of ways. A few examples include

- **Manual data insertion** Data can be preloaded into a repo.
- **Manual configuration** Instead of loading data, one may load into the system configuration the namespaces of the data to be stored in a repo. For example, a repo can be configured to save network measurement data for a specific router. In this case, the configuration would need to provide the information about the router's name, the names of the measurement data, and the data generation patterns. The repo will start fetching data when it starts running.
- **Passive data collection** If a repo is attached to a router, it can be configured to passively collect data under the configured namespaces from passing traffic.
- **Active data fetching** A repo can be instructed by data insertion commands to actively fetch data into its storage, as we will discuss in Section IV. The command issuer will need to be authenticated as discussed in Section **??**, and then repo policies will be invoked to determine who can insert data into a given repo.

Note that, the above example approaches are complementary to each other (except the first one). For example, manual configurations can provide the namespaces for either passive data collection or active data fetching; active data fetching must adhere to the usage policy of a given repo which can be described in the configuration using approaches similar to schematized trust management [6]; of course the configured policies can also be changed over time.

[2]The issue regarding data access control is discussed in Section VIII-C.

[3]In-network storage is equally useful in developing certificate revocation solutions; one such example solution is described in [5]

III. Repo Trust Model

Since repos are important resources in an NDN network, repo security is critically important, to effectively mitigate malicious attacks that may either aim to disable a repo by overflowing its storage, or to poison its contents. Given repos are yet to get widely used, little attention has been paid to the analysis of its potential attacks.

Although repo's threat model is yet to be investigated, we must establish a basic trust model that can then be used to mount cryptographic defense. At this time, it seems that a variety of trust models may be developed depending on the ownership of the repos. A few example scenarios are described below.

- **Private repos** As mentioned in an earlier example, a network operator may set up repos for network monitoring and measurement data. In this case, there should be a share trust anchor (e.g. the operator) between the data producers (e.g. routers in the network) and the repos. Similarly, an enterprise network may set up its own repos for internal data storage, which can also have shared trust anchor with the data producing hosts.
- **Community repos** One may consider the NDN testbed as a community of users with shared interest of conducting NDN research. As a community the testbed has established a trust anchor which issues certificates to all the users in the community.
- **Public repos** These can be storage services with a usage fee. To be able to charge users, one must have a way to identify users, issue them certificates, which can then be used to authenticate data insertion commands.

We can proceed with developing trust models for the above specific repo use cases. Once a trust model is created, we can then develop implementations for data insertion command authentication; the authorization step follows by check against the configured repo policies. We also need to consider data security policies for various repo deployment scenarios. Again a few examples are given below.

- For private repos, one may demand the repos to perform data authentication for each fetched data packet. This requires the repo to know the data trust models. All data packets that fail the authentications will be dropped and the repo management will be informed.
- For community repos, one may or may not enforce data authentication, or perform authentication on a random sample basis and report any detected problems to repo management.
- For public repos, the decision of data authentication will probably be determined by the repo service model.

End consumers must perform data authentication, independent from whether repos have done so or not.

IV. Repo Data Fetching

The first task of a repo is to get the data into its storage. Based on our current understanding, how to perform this task should be application-specific. The rudimentary data fetch that has been implemented so far assumes a simple case of fetching existing data from a file. Given a file name in the data insertion command, the repo issues interests to fetch data by following a default data naming pattern of simply increasing tje segment numbers in the given data file name.

More general solutions are needed when data to be stored in repos do not pre-exist but will be generated in real time. For example, to use repo to store data generated by a text chatroom application, assuming the chat application uses a sync protocol to keep all the users in the chatroom synchronized in the latest data generation, the repo can simply run the same sync protocol to learn the names of newly generated data, and fetches all the data into its storage.

As another example, an ongoing augmented reality application development at UCLA is using repos to store both raw video data captured by Android phones and the processed results of the raw video data. In this case, the data insertion command needs to inform the repo about both the data naming conventions (so that the repo can automatically generates the name for next data piece to fetch), and the application data generation patterns, so that the repo can issue interest packets with right timing. Solutions are still being developed on exactly how to describe both the naming conventions and the data generation timing patterns. Such input parameters may be long enough that would need to be fetched by the repo instead of being carried in the data insertion command, which can be done by using an approach similar to that described in RICE [7].

In Section III we discuss whether the repo needs to authenticate the fetched data.

V. Fetching Data from Repo

Given a repo may store data from multiple producers, generally speaking it would be infeasible for the repo to make routing announcements for each of the name prefixes of its stored data. To be able to fetch data from repos, one scalable solution is to

- make routing announcements for repos to make them reachable (i.e. showing up in the FIB of all forwarders in the network);
- let each data producer inform its consumers of which repos its data is stored; and
- let consumers fetch data from repos by using forwarding hints to carry the repos names.

We make one statement and one observation here. First, the reachability to data producers must have a solution; it is independent from whether repos are used or not. Second, when an NDN interest packet carries a forwarding hint, this interest carries both an identifier (the data name), and a locator (the repo's name) in parallel, a true separation between identifier and locator. The locator *steers* the interest toward where the named data is located; the identifier identifies the desired data, and the interest does not need to reach the data location if the data is found along the way.

Before leaving this section we would like to make another observation. As edge computing is arising as a promising direction to meet the requirements of new applications such as AR and VR, repos can serve as edge storage as part of the solution package. In such cases, the reachability to repos, or to other edge resources in general, may not necessarily require a full blown routing protocol such as NLSR [8]. Instead, we suggest to explore the development of a secure and light-weight distance-vector style routing announcement protocol that can provide best-effort reachability information dissemination to repos as well as other resources within a local scope.

VI. Repo Synchronizaton

To mitigate single point of failure, all data must be replicated among multiple repos. Data replication leads to the need for data synchronization among repos with replicate data. *This task will be fully described in the final version of the paper, if the paper is accepted.*

VII. Repo Management

As any other devices in a network, a repo must be monitored and managed while in active use. Its resource usage, e.g. how much storage each stored data namespace takes, needs to continuously monitored. Its policy, e.g. which namespace may inject data into the repo, may also change over time, and the updates need to be securely installed into the repo.

Up to now repo management stays as a blank area. New work must get started quickly to make repos usable.

VIII. Discussion

A. Cache Management Research

NDN enables in-network caching as a by-product of its design to name and secure data directly. However one must resist popular misconceptions of (i) in-network caching is, by and large, all that NDN offers, or (ii) cache management strategies are an integral part of the NDN design.

Naming data instead of naming locations brings profound changes to networking that are far beyond in-network catching (a basic security building block at the narrow waist, stateful forwarding and its consequential benefits from this closed-loop control and built-in multicast data delivery, mobility support, to name a few).

Cache management strategies remain an area of future research, however it is not part of the network architecture. The last few years have witnessed a high volume of research papers dedicated to the cache management optimization in NDN networks. Given there is little NDN traffic at this time, these papers used either theoretic models or otherwise measurements from existing IP data traffic to derive their optimality. We speculate that the change of network architecture is likely to lead to fundamental changes to network traffic patterns, just like today's IP traffic patterns are fundamentally different from the telephone network traffic patterns in the old days. Therefore, although cache management strategies in NDN networks remains an area of research, we believe that such research efforts will be most effective once applications running over NDN start spreading widely to provide the input to such efforts.

B. Content Store versus Repo

There has been a frequently asked question regarding the difference, and the relation, between the content store (CS, which is an integral component in NDN forwarding) and NDN repo, one of the questions this paper aims to clarify.

First, the content store at each NDN forwarder performs *opportunistic* caching, in the sense that a CS does not proactively go out to fetch data, but only passively cache (some of) the data it helps forwarding (this includes both data packets requested by other devices and those requested by the local applications). Each CS needs some policy-guided cache management strategy which determines that, when the CS runs out of storage resource[4], which packets to be evicted.

In contrast, a repo is a *managed storage*, in the sense that it will be explicitly instructed regarding what contents to store, and it will go out proactively to fetch those contents as we discussed in Section IV. Note that this does not preclude a repo from passively capture data packets passing by the NDN forwarder that the repo is attached to, however this is likely to be done for only data packets whose names match given name prefixes (the namespaces that the repo is instructed to store data). Cache management determines, among all the passively captured data packets, which ones to evict when required by the resource constraints On the other hand, repo management determines what contents, as identified by their nameprefixes, to load into its own storage; this management can be through either, or both, of pre-configurations and insertion commands in real time.

C. In-Application Memory, Content Store, and Repo

The content store on a device D is a storage component at the network layer, not the application layer. D may be hosting multiple applications, and may also help forward traffic for neighbor devices. Thus the contents of its CS are determined by the combined effect of passing data packets and D's cache strategy and policy.

A running application is likely to need its storage at the application layer specifically used by that application only. When the application A finishes produce each piece of data, A may decides to publish the data into a repo, which can be either attached to the same device or a nearby one, making the data available to interested parties and can be fetched from the repo directly, without interference with the producing application.

Note that the data produced by A may have access policies instead of being accessible to all. In this case, A needs to encrypt the data, so that the access to the data is controlled by the distribution of the decryption key. NDN has developed automated key management solutions as described in [9].

[4]It remains a research topic of whether the CS can be implemented using disk storage in addition to in-node memory (which is the current practice); intuitively using disk storage could bring benefits if the user traffic contains large volumes of static contents, such as movies which can have identifiable popularities. In such cases a large CS can help reduce the delay to users and the traffic to servers, and it will be part of the caching strategy decision on what content to keep in memory and what to move to disk storage

D. Cloud Storage versus Repo

From 1000 feet up, repos in an NDN network seem to play a similar role to that of today's cloud storage. At an abstract level, there is a certain truth to this statement. However fundamental differences exist.

First and formost, we perceive repos providing storage services that come with all the different colors and shapes, owned by different parties, in contrast to cloud storage service controlled by a few cyber giants. We do not see community storage services because there is no easy ways to enable such services with today's protocol architecture. NDN is expected to change the playground.

Second, cloud storage service runs at application layer which is insulated from network layer. This makes it difficult to support user choices, because users have to first decide an IP address to use, rather than asking the question of "which storage service is closest to me". When one gets an IP address from DNS query, the decision has already been made for the user.

Additional elaborations on the difference will be added in the final version of the paper if the paper is accepted.

IX. Summary

The main goal of this paper is to articulate the important role that repos can play in an NDN network, and to attract the attention of the broader ICN research community to the design, development, and use of repos in supporting new generations of edge computing and peer-to-peer applications.

To make data-centric networking truly viable, systematic solutions must be developed to ensure data availability independent from the availability of data producers. This paper articulated several research issues to be addressed, to enable repos to play a center role in the solution development. We hope all interested readers can consider this paper as a call for action to the exploration of this new and important area in NDN design and development.

References

[1] L. Zhang *et al.*, "Named Data Networking (NDN) Project," NDN, Tech. Rep. NDN-0001, 2010.

[2] L. Zhang, A. Afanasyev, J. Burke, K. Claffy, L. Wang, V. Jacobson, P. Crowley, C. Papadopoulos, and B. Zhang, "Named Data Networking," *ACM Computer Communication Review (CCR)*, July 2014.

[3] J. Burke, A. Afanasyev, T. Refaei, and L. Zhang, "NDN Impact on Tactical Application Development," in *IEEE Conference on Military Communications (MILCOM)*, October 2018.

[4] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, August 2018.

[5] K. Yang, J. Sunny, and L. Wang, "Blockchain-based Decentralized Public Key Management for Named Data Networking," in *27th International Conference on Computer Communications and Networks (ICCCN)*, 2018.

[6] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang, "Schematizing trust in Named Data Networking," in *Proceedings of ACM Information Centric Networking*, September 2015.

[7] M. Krol, K. Habak, D. Oran, D. Kutscher, and I. Psaras, "RICE: Remote Method Invocation in ICN," in *Proceedings of ACM Conference on Information-Centric Networking*, 2018.

[8] L. Wang, V. Lehman, A. K. M. M. Hoque, Y. Yu, B. Zhang, and L. Zhang, "A Secure Link State Routing Protocol for NDN," in *IEEE Access*. IEEE, 2018.

[9] Z. Zhang, Y. Yu, S. K. Ramani, A. Afanasyev, and L. Zhang, “NAC: Automating Access Control via Named Data,” in *IEEE Conference on Military Communications (MILCOM)*, October 2018.