UNIVERSITY OF AMSTERDAM

MSc Security and Network Engineering
Research Project I

Elastic Named Data Network (NDN) for data centric application in cloud environments

by
Sean Liao
February 9, 2020

Supervisor:
Dr. Zhiming Zhao

Abstract

Research projects such as ENVRI-FAIR both generate and consume large amounts of data. The use of persistent identifiers along the vision of a Global Digital Object Cloud in naming makes managing and locating the data much more palatable. These structured and persistent names along with the large amounts of data they represent aligns closely with the data model of Named Data Networking (NDN), a Future Internet architecture that makes data objects first class citizens of the network, enabling among other things, widespread use of caching to reduce both bandwidth and latency. While the design of NDN enables content distribution in a more efficient manner, operating such a network is still largely a manual process, especially in regards to scaling cope with varying demand levels. This research explores the possible ways for NDN networks adapt to elastic demand and proposes a load balanced configuration to efficiently scale NDN nodes.

Introduction

The internet as we know it is used for many things, but as early as 2006 it was recognized that a significant portion of traffic is used in distributing content [11]. There are many technologies dedicated to making content distribution more efficient, from BitTorrent to Content Distribution Networks (CDN). Along with the problem of moving data is the problem of identifying a specific piece of data to be retrieved. URLs may appear to solve the problem but the content they point to is often mutable. Link rot is also a serious problem. As such, projects such as the Internet Archive use a combined URL and timestamp to uniquely identify data.

Within research communities such as ENVironmental Research Infrastructures (ENVRI) [3], the data distribution problem is exacerbated by the large amounts of data generated by various projects and the need to combine data from multiple sources in interdisciplinary research. Identifying the data is also a problem, especially with the expectation that names can always be resolved to a unique instance of data to aid in the reproducibility of research. Within the more tightly knit research community (compared to the wider internet) it is possible to more or less enforce the use of Persistent Identifiers (PID), globally unique names per instance of data, to solve the naming problem. To achieve the vision of a Global Digital Object Cloud (DOC) [9], requires both the adoption of a shared naming scheme and a way to resolve the names back to data. Current implementations use centralized resolvers to redirect users to the various institutions responsible for hosting and distributing the data.

Information Centric Networking (ICN) [1] is the broader concept of making information (data) first class citizens of the network. The idea is to abstract away the end hosts and route directly to data. Of the active projects under the ICN umbrella, Named Data Networking (NDN) [16] is one implementation. It utilizes persistent, hierarchical names and cryptographically signed data to enable efficient forwarding and in network caching in all of its routers. With a matching data model, NDN is an ideal candidate in implementing a decentralised version of the DOC, removing the need for centralized resolvers and redirects by routing requests directly across connected networks and returning the data directly, all while taking advantage of the in network cache.

Trial deployments of the various ICN technologies [10], including NDN [7], have identified several concerns[15] [17]. Some, such as dynamic naming are less relevant for the research use case due to the use of PIDs. Others, such as routing and cache placement and sharing are more relevant. Overall, scaling has been identified as an issue as testbeds have been limited to less than 1000 users and typically with a fixed infrastructure.

ENVRI-FAIR is the project to connect the ENVRI community to the wider European Open Science Cloud (EOSC) [2]. In using cloud technologies, resources are typically dynamically allocated, this brings with it advantages such as more efficient use of resources but also challenges as the resources are no longer static. While manually managing such resources is certainly possible, it is both tedious and error prone, and even more challenging if coordination is required between multiple operators as in ENVRI-FAIR. Of the many problems users may face in operating a NDN network in a situation such as ENVRI-FAIR, this research will specifically look at:

- Scaling the caching capacity of an individual node
- Scaling the number of nodes within a network
- Establishing connections to the network
- Routing over the established network

within cloud environments in a semi automatic manner. The remainder of this report will consist of related work, a look into NDN design and implementation, the proposed load balancing solution, a proof of concept implementation, validation tests, and a discussion on both the efficiency and problems of the proposed solution.

Related Work

The ICN Research Group (ICNRG) has an Internet Draft on deployment guidelines [10], which lists the different possible configurations and known trial deployments. Amongst the common concerns were scalability [15] [17], as trials have been limited to less than 1000 users. Based on NDN development progress, the reality check [8] on wide scale NDN deployment still appears to hold true,

Multiple routing strategies have been proposed, OSPFN [13] is a modification of OSPF to distribute NDN names over IP networks, NLSR [5] is a NDN native implementation of link state routing, DCR [4] implements distance based routing, and CRoS-NDN [12] uses a centralised controller to distribute routes based on global state. While they implement different routing protocols, they all require some form of existing topology.

Named Data Networking

As the name suggests, names play a big part in NDN. They are hierarchical and human selected, as opposed to the flat, hash based naming schemes used in other content addressed protocols. Each request (Interest) returns a single piece of data. The data used to satisfy the request is authenticated through the use of a hierarchical public key infrastructure and namespace delegations, being valid if it is signed by an appropriate key. The other major design decision made by the NDN project is the use of a stateful forwarding plane. All requests are held in memory until they are resolved or time out.

NDN Forwarding Daemon (NFD) is the NDN project's reference router. At the time of writing this report, it was the only well documented, running router that conformed to the latest version of the NDN specification. Written in C++14, it uses a single-threaded event-driven architecture.The stateful forwarding plane is internally represented as a NameTree, a combined hashmap

and tree data structure. It only has an in-memory implementation of its cache.Based on preliminary testing, it had issues scaling beyond 50000 concurrent in-flight requests.No further tests were made to determine the cause of the slowdown, though other research shows it may have to do with decoding names [14] or decoding packets [6]. While NFD supports using Ethernet, TCP, UDP, and WebSockets as it's underlying transport, only TCP and UDP are practical in the cloud environment this research is interested in.

Proposed Solution

The problems considered in this research can be split into scaling a single node and scaling the entire network. Scaling the network can itself be split into establishing the connections over which the NDN protocol will run and propagating the available routes across the network.

Single Node Scaling

This problem has two major drivers, first is the fact that NFD's performance degrades with a large amount of cache entries. The second is the fact that NFD's cache is in-memory only, which means it is almost impossible to grow beyond the initially configured available RAM space. While it is possible to configure the host operating system to use hard disk space as swap, thereby extending the available memory for the cache to grow, it is clearly a suboptimal solution as it severely impacts both cache performance and the operation of other applications, such as controller or monitoring systems.

There are several potential solutions to this problem. The first is to simply create a new node of the appropriate size and destroy the old node. This is problematic since it both disrupts client connections and loses the contents of the cache. The first problem can be solved through the use of a traffic redirector that allows for the graceful shutdown of the old node but it doesn't solve the cache contents problem. To solve the cache problem, the new node could be placed in front of the old node, copying cache contents as they are requested by clients. This is however still limited by NFD's performance, effectively placing a low upper limit on cache size. Another idea is to place new nodes behind the existing node(s), forming a chain, though this will suffer from inefficient caches as content is duplicated across nodes while also increasing latency as requests have to traverse multiple nodes. Instead the proposed solution is to place new nodes behind the existing node side by side, all directly connected to the existing node. The existing node will act as a load balancer distributing requests between the different nodes. This requires a stable partitioning of requests between the nodes to ensure no duplicated content, to which one solution is to partition the routes.

Establishing Connections

This is largely an unsolved problem for NDN running in overlay mode over IP networks. For physical deployments of IP, it is largely a non problem as neighbouring nodes are connected to the same broadcast domain (Ethernet). For protocols running over IP, the options are to configure IP subnets to allow broadcast to continue to work, use an environment where multicast is supported, use a bootstrap list and allow nodes to gossip about each other, or use a central discovery server (which can be seen as a special case of the bootstrap list).

Using a bootstrap list with some form of gossiping is what most fully developed decentralised protocols use (examples: BitTorrent and IPFS). However, such algorithms are more difficult to write and get right, compounded by the fact that they are almost always only eventually consistent. The development of a (or modification of an existing) protocol is left for future work, instead, the proposed solution will use a central discovery server, limiting the information shared to the presence of a node to make a future migration to a decentralised protocol easier. Nodes will directly (or through their cache servers) connect to all other nodes within the network, essentially forming a fully connected mesh. This has the advantage that all requests will pass through at most 3 caches (local load balance, local cache server, remote load balancer) before reaching a data host, this provides a balance between the number of caches visited and the latency introduced by multiple hops.

Route Propagation

NFD's modular design does not enshrine any particular routing protocol, instead the intention is for it to be able to take advantage of the latest developments in routing technology. While in theory any routing protocol could be used here, the operation of of multiple NFD routers together as a load balancing group presents its own challenges. First is the need to enforce a one way data flow between load balancers, caches, and their upstream servers. Violation of this data flow severely decrease the efficiency of the caching servers. The other major challenge is the need to represent the load balancing group as a single node during network wide routing but have the available route be partitioned and applied across several caching servers.

Due to time constraints, the proposed solution is have caching servers connect directly to their upstreams to request the available routes. These routes are not further propogated back to the local load balancer, instead relying on NFD's Access Router strategy (Access) to probe the caches and learn the optimal routes. Access is designed to both learn optimal routes but also adapt to producer mobility, which is important when routes are reassigned to different caching servers due to the addition or removal of caches. The use of Access also reduces the need for route updates on the load balancer. For the connections between caches and their upstreams, simplistic fully connected topology means there is no simple way of assigning route costs. Instead, caching servers use Adaptive Smoothed RTT-based Forwarding Strategy (ASF) to forward to their upstreams. Instead of selecting by route cost, all upstreams are periodically probed for reachability and response times and the fastest s selected.

Prototype

Architecture

To implement the proposed solution above requires 3 components: a discovery server, a load balancer, and a caching server. The discovery server will be a standalone piece of software. The load balancer and caching server will be implemented as controller software to be run in conjunction with NFD, they will communicate with other components to gather the required information to send control signals to NFD. Figure 1 shows the flow of information between the different components. Data takes a different path, only flowing through the different instances of NFD as shown in Figure 2.

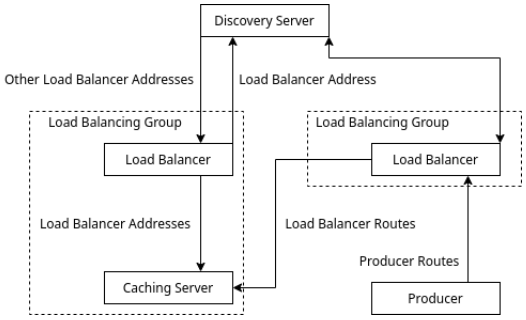

Figure 1: Control Flow

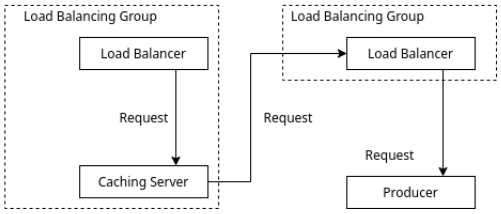

Figure 2: Data Flow, responses/data flows along the reverse path of requests

Technical Choices

The prototype to implement the proposed solution and the architecture outlined above was developed against NFD version 0.7.0, the latest release at the time of writing this report. An attempt was made to have the controller components communicate over the NDN protocol, though this was dropped after discovering that only the C++ client libraries were either updated to use the latest specification or implemented the nonstandard name component format used by NFD. Instead the controller components were implemented in Go, due to the language's general suitability to writing networked applications and simple deployment model, as well as the author's familiarity with the language, enabling a fast development cycle. Communication between controller components happen over gRPC, selected for its well defined message format and support for bidirectional streaming. Controller components interact with NFD through NFD's provided command line interface (CLI), mainly out of necessity.

Final results are packaged as Docker containers, which enjoy wide cross platform support including native deployment options in the cloud, but also due to the need of ensuring both NFD and the controller run together. The controller takes a single configuration option through an environment variable (`WATCHER` for load balancers to specify the discovery server, and `PRIMARY` for cache servers to specify the load balancer to connect to) and NFD configuration can be overriden by mounting to the `/usr/local/etc/ndn` directory.

Validation Testing

As validation, a test recording request latencies during cache scaling events was run. 4 load balancing groups with attached data sources were placed in separate locations (Google Cloud Platform

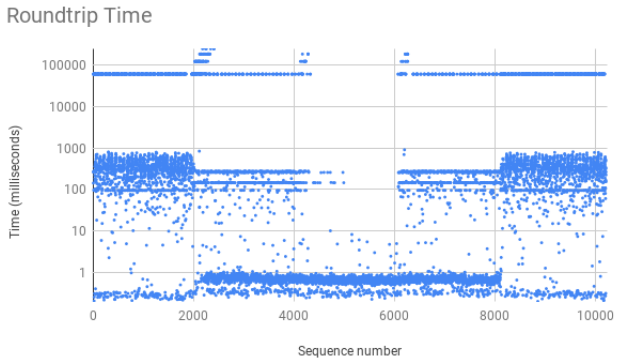

Figure 3: Roundtrip time during cache scaling events

datacenter locations: `us-east1-b`, `us-west1-b`, `asia-east1-b`, `australia-southeast1-b`). A fifth load balancing group with a data consumer was placed in `europe-west4-a`. The consumer made requests to all 4 sources with an even distribution. Starting with 0 caching servers, after every 2000 requests a scaling event was performed: (2000, 4000: add a caching server, 6000, 8000: remove a caching server). The load balancers were configured with a minimal cache capacity and the caching servers each had a capacity equal to half of all unique requests.

From Figure 3 we can observe the consumer load balancing cache as the line just above 0. The effects of the caching server are clearly visible as line between 2000 and 8000 requests at just under a millisecond. All other requests are hitting their respective origins. This shows that the partitioning of requests between the caching servers works, as only a non overlapping partition would allow the combined caches to cover all requests.

Discussion

In the load balancing configuration shown above, the partitioning is not particularly efficient, dividing entire upstream load balancing groups between the caching servers. This leads to the possibility of hot caches, where all requests go to a single cache leaving the others unused. Finer grained partitioning is hindered by the lack of observability on per object cache statistics, currently to obtain that information would require implementing a caching strategy or tailing NFD's debug log.

The proposed solution to discovering and connecting new nodes relies on a central discovery server. This was an unfortunate but practical choice. The design is intentionally simple to reduce the possibility of failure but it is still a single point of failure, though an existing network should continue to operate, its just that new nodes cannot join.

In automating the discovery and distribution of routes, the implementation sacrifices configurability of the network architecture, opting to simply connect everything together. A minor modification of the current implementation should allow load balancers to talk to multiple discover servers, effectively forming multiple zones that can be joined together. While the goal is to allow direct communications between any 2 nodes, should a connections fail, requests could still in theory be forwarded through other nodes if an appropriate prefix is installed. One way would be to install the root prefix `/` on all nodes and lean on the ASF strategy to probe and find the fastest path.

The proposed solution is introduces multiple components and therefore fault points. The discovery server could fail or become unreachable, but the design is simple enough that a restart should allow all nodes to reconnect and continue functioning. A failure of a load balancer would effectively equal removing a node from the network, and be handled accordingly. A failure of a caching server would also be seen as a regular removal, triggering a rebalancing of routes between the remaining caches. With both the load balancer and the caching server there is the possibility of the internal states between NFD and the controller component going out of sync, in this case a restart is probably the simplest solution.

There were several issues in implementing the proof of concept. NDN is a research project used to further refine the protocol specification. This results in updates to the protocol and the C++ reference implementation, but not necessarily the client libraries. Additionally, despite standardisation efforts, different libraries still have different implementations of the same concept such as data segmentation, leading to incompatible libraries.

Conclusion

As the above has shown, running multiple instances of NFD in a load balanced configuration can provide users with greater control over scaling while working round some of the limitations of NFD's implementation. While the centralised solution for discovering and connecting network nodes certainly works, the fact that it is a centralised component is definitely a weak point. The NDN project is primarily research focused and evolves rapidly, its implementations are modular but not very efficient, a lot of work would be necessary to either modify or reimplement it to be production grade and ready for widespread use in research clouds.

Future Work

Designing or adapting a decentralised gossip protocol for discovering potential neighbours and establishing the connections between them is left as future work. So are the application of advanced routing protocols and smarter partitioning of routes between caches. Finally, it would be interesting to see coordinated caching without an explicit load balancer, doing so would likely require deeper integration with NFD's caching and forwarding policies.

References

[1] B. Ahlgren et al. "A survey of information-centric networking". In: *IEEE Communications Magazine* 50.7 (July 2012), pp. 26–36. ISSN: 1558-1896. DOI: 10.1109/MCOM.2012.6231276.

[2] Leonardo Candela, Donatella Castelli, and Franco Zoppi. *Final EOSC Service Architecture*. Apr. 2019. DOI: 10.5281/zenodo.3258798. URL: https://doi.org/10.5281/zenodo.3258798.

[3] *ENVRI-FAIR - Interoperable environmental FAIR data and services for society, innovation and research*. Zenodo, Sept. 2019. DOI: 10.1109/eScience.2019.00038. URL: https://doi.org/10.1109/eScience.2019.00038.

[4] J.J. Garcia-Luna-Aceves. "Name-based content routing in information centric networks using distance information". In: Sept. 2014, pp. 7–16. DOI: 10.1145/2660129.2660141.

[5] A Hoque et al. "NLSR: Named-data link state routing protocol". In: Aug. 2013, pp. 15–20. DOI: 10.1145/2491224.2491231.

[6] Jong Kees de and Anas Younis. "Planning and Scaling a Named Data Network with Persistent Identifier Interoperability". PhD thesis. University of Amsterdam, Sept. 2019. DOI: 10.5281/zenodo.3521549. URL: https://doi.org/10.5281/zenodo.3521549.

[7] H. Lim et al. "NDN Construction for Big Science: Lessons Learned from Establishing a Testbed". In: *IEEE Network* 32.6 (Nov. 2018), pp. 124–136. ISSN: 1558-156X. DOI: 10.1109/MNET.2018.1800088.

[8] Diego Perino and Matteo Varvello. "A Reality Check for Content Centric Networking". In: *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking*. ICN '11. Toronto, Ontario, Canada: Association for Computing Machinery, 2011, pp. 44–49. ISBN: 9781450308014. DOI: 10.1145/2018584.2018596. URL: https://doi.org/10.1145/2018584.2018596.

[9] Larry Lannom ad Peter Wittenburg. *Global Digital Object Cloud (DOC) - A Guiding Vision*. Sept. 2016. URL: http://hdl.handle.net/11304/a8877a1a-9010-428f-b2ce-5863cec4aff3.

[10] Akbar Rahman et al. *Deployment Considerations for Information-Centric Networking (ICN)*. Internet-Draft draft-irtf-icnrg-deployment-guidelines-07. http://www.ietf.org/internet-drafts/draft-irtf-icnrg-deployment-guidelines-07.txt. IETF Secretariat, Sept. 2019. URL: http://www.ietf.org/internet-drafts/draft-irtf-icnrg-deployment-guidelines-07.txt.

[11] Pavlos Sermpezis and Thrasyvoulos Spyropoulos. "Effects of Content Popularity on the Performance of Content-Centric Opportunistic Networking: An Analytical Approach and Applications". In: *CoRR* abs/1601.05266 (2016). arXiv: 1601.05266. URL: http://arxiv.org/abs/1601.05266.

[12] João Vitor Torres et al. "Evaluating CRoS-NDN: a comparative performance analysis of a controller-based routing scheme for named-data networking". In: *Journal of Internet Services and Applications* 10.1 (Nov. 2019), p. 20. ISSN: 1869-0238. DOI: 10.1186/s13174-019-0119-6. URL: https://doi.org/10.1186/s13174-019-0119-6.

[13] Lan Wang et al. "OSPFN: an OSPF based routing protocol for named data networking". In: (Jan. 2012).

[14] H. Yuan and P. Crowley. "Reliably scalable name prefix lookup". In: *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. May 2015, pp. 111–121. DOI: 10.1109/ANCS.2015.7110125.

[15] Haowei Yuan, Tian Song, and Patrick Crowley. "Scalable NDN Forwarding: Concepts, Issues and Principles". In: (July 2012). DOI: 10.1109/ICCCN.2012.6289305.

[16] Lixia Zhang et al. "Named Data Networking". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 66–73. ISSN: 0146-4833. DOI: 10.1145/2656877.2656887. URL: https://doi.org/10.1145/2656877.2656887.

[17] Yu Zhang et al. "A Note on Routing Scalability in Named Data Networking". In: May 2019, pp. 1–6. DOI: 10.1109/ICCW.2019.8756677.