NDNSD: Service Publishing and Discovery in NDN

Saurab Dulal
University of Memphis
sdulal@memphis.edu

Lan Wang
University of Memphis
lanwang@memphis.edu

***Abstract*—Service discovery is a crucial component in today's massively distributed applications. In this paper, we propose NDNSD – a fully distributed and general-purpose service discovery protocol for Named Data Networking (NDN). By leveraging NDN's data synchronization capability, NDNSD offers a high-level API for service publishing and discovery. We present NDNSD's main design features including hierarchical naming, service information specification, and service accessibility. We also implemented two other discovery schemes, one reactive and one proactive, and compared them with NDNSD. Our evaluation shows that NDNSD achieves (a) lower latency, lower overhead, and same reliability compared to the reactive scheme, and (b) comparable latency, lower overhead at larger scale, and higher reliability compared to the proactive scheme.**

***Index Terms*—Information Centric Networking, Named Data Networking, Service Discovery, NDN Synchronization**

I. Introduction

Modern applications rely on many services, e.g., cloud computing, file storage, and databases, to function properly. These services are provided by not only stationary servers, but also mobile devices offering computing and data service. ***Service discovery*** is the process of finding desired service(s) in a distributed environment.

TCP/IP has a plethora of protocols ([1]–[5]) at different layers to facilitate service discovery. These protocols have a major limitation stemming from the inability of the network layer to recognize names used by the application layer ([6], [7]). For example, a printer application cannot identify itself as */edu/abc/library/printer1* to the network, but instead has to rely on its IP address and port number. This creates a semantic mismatch between the two layers and thus requires either a dedicated server (e.g., the resource directory in CoAP) or a name resolution service such as DNS. This dependency is cumbersome for decentralized applications or edge applications. Popular edge applications such as AWS IoT [8], Google Chromecast [9], and Azure Sphere [10] mostly depend on the cloud, not a local device, for service registration and discovery. This design incurs extra delay for edge devices to discover services residing in close vicinity. More seriously, a disruption in the connectivity to the cloud will disrupt the whole application.

Named Data Networking (NDN) is a new data-centric Internet architecture [11]. It uses application-level names directly in the network layer, so there is no need to resolve the application names into addresses and ports (and there are no addresses in the network layer). Moreover, NDN's distributed dataset synchronization protocols [12], i.e., *Sync*, can make the discovery process fully distributed by directly synchronizing the service information among the participating nodes, thereby eliminating the requirement of a centralized entity to facilitate the service discovery process. Several service discovery solutions ([13]–[17]) have been proposed for NDN (and its predecessor CCN). However, we found various limitations in these solutions (Section II-C) such as high overhead, being limited to a specific environment, or requiring separate servers to store service information.

In this paper, we propose NDNSD, a fully distributed, general-purpose, and scalable service discovery protocol for NDN. NDNSD offers a sync-based high-level API for publishing and discovering services, obtaining measurement information, and controlling access to service information. We have implemented NDNSD and evaluated it by comparing it to two other discovery schemes, one reactive and one proactive. Our evaluation shows that NDNSD achieves (a) lower latency, lower overhead, and same reliability compared to the reactive scheme, and (b) comparable latency, lower overhead at a larger scale, and higher reliability compared to the proactive scheme.

II. Background and Related Work

A. NDN

Named Data Networking (NDN) is an evolving data-centric Internet architecture. Every piece of content in NDN is named (e.g. /edu/abc/servers/cygnux/info), carried in one or more *data packets*. The content is fetched using *Interests* whose name matches the data name. NDN changes IP's host-centric communication by decoupling data packets from their producers. The decoupling is feasible in NDN because data is signed by its producer at the time of creation and thus its authenticity can be verified by other nodes. Once decoupled, data can be served by any node that stores a copy of it.

An NDN forwarder, e.g., the NDN Forwarding Daemon (NFD) [18], implements the network-layer protocols needed for name-based communication. The forwarder consists of three core components: Pending Interest Table (PIT), Content Store (CS), and Forwarding Information Base (FIB). The PIT records incoming Interests, not yet satisfied, and also aggregates them if the same Interest has already been received. The CS caches previously received data packets to satisfy incoming Interests. If an Interest is not satisfied by the CS or matches the PIT, it is forwarded via one or more interfaces with the help of the FIB and a forwarding strategy. Once a data packet is received, it is forwarded to all the incoming interfaces of the matching Interests recorded in the PIT, thus multicast is supported natively in NDN. In addition, since each data packet follows the reverse path of the matching Interests, routers can measure path performance (e.g., RTT), which enables adaptive forwarding decisions [19].

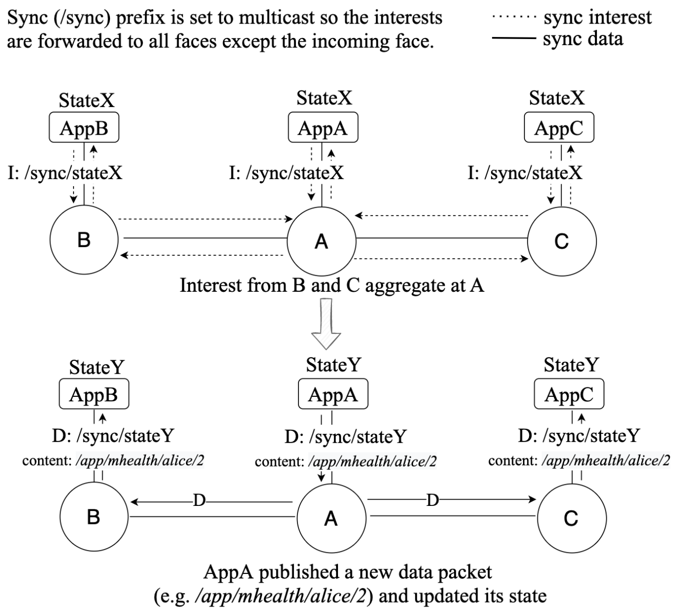

Sync (/sync) prefix is set to multicast so the interests are forwarded to all faces except the incoming face.
sync interest
sync data
StateX
AppB
AppA
AppC
I: /sync/stateX
B
A
C
Interest from B and C aggregate at A
StateY
D: /sync/stateY
content: /app/mhealth/alice/2
D
AppA published a new data packet (e.g. /app/mhealth/alice/2) and updated its state

Fig. 1: Example of NDN Synchronization Process ("I" and "D" represent Sync Interest and Sync Data, respectively.)

B. Data Synchronization in NDN (Sync)

NDN synchronization, or *Sync* in short, provides a powerful abstraction above the Interest-Data exchange to facilitate multi-party communication [12]. Sync ensures that every participating node has up-to-date information of *their shared distributed dataset* by encoding the set of data names in a compact form (i.e., "state") and exchanging the state among the nodes using Sync Interests (Figure 1). Once a new data packet is published, Sync updates its local state and sends the new state and new data name in a Sync Data packet to all other nodes (Figure 1), which can now fetch the data using the new data name.

C. Related Work

Mark Mosko suggested using Sync for service discovery [13]. Devices use a well-defined namespace such as *"/parc/printers/"* to advertise the manifest of service records. However, Mosko's design lacks protocol details, API specifications, and access control. Moreover, it was designed for a different ICN architecture, i.e. CCNx 1.0, as opposed to NDN. Ravindran et.al. [14] proposed two different service discovery protocols: neighbor discovery protocol (NDP) for locally reachable CCNx nodes neighbors, and Service Publish and Discovery (SPDP) for discovering remote services. SPDP uses a recursive query that propagates hop-by-hop among the reachable adjacencies running SPDP instances. Data containing the service list is aggregated by the respective instances and is sent back to the original requestor. This approach searches for services one hop at a time, so it may take a long time if services are multiple hops away.

NDNe [16] uses an expanded ring search technique along with the broadcast for service discovery in edge environments. The request is first broadcasted to a 1-hop neighbor (TTL is used for hop count). If no reply is received within the pre-defined timeout, the request is sent to 2-hop neighbors. This process repeats until either the service is found, or the consumer gives up. The expanded ring search can be expensive and may not scale if there is a large number of consumers.

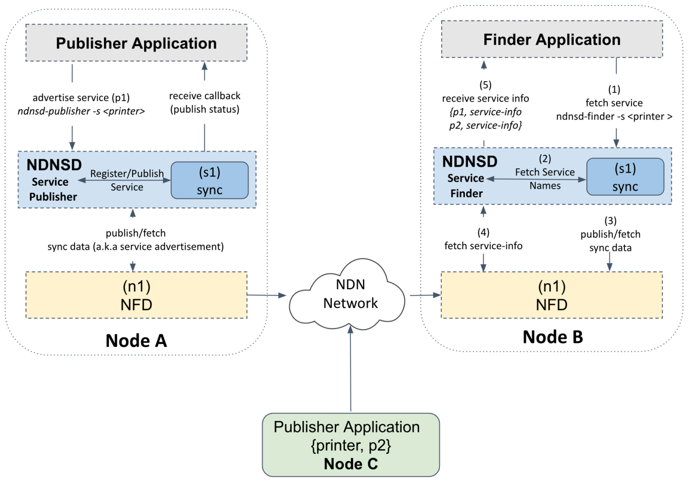


Fig. 2: Application workflow showing all layers involved in service publishing and discovery process

Similar to our work, Mastorakis et. al. use Sync among multiple edge computing servers (ECS) to facilitate service discovery [17]. ECS are special nodes in the network that maintain service information available from service providers. A discovery Interest (e.g. */discovery/ecs/*) from an application is matched with a suitable service by the closest ECS. However, maintaining ECS is an extra infrastructure requirement for service discovery – this may be feasible for a particular edge computing application, but not in the general case.

III. Design

We designed NDNSD to be a fully distributed and general-purpose service discovery protocol for NDN that works in a wide range of environments, e.g., LAN, WAN, and IoT. It provides an API for applications to advertise and discover services. Internally, it uses NDN Sync [12] for service announcement and discovery.

We view service discovery as a pub-sub problem, i.e. publishing service information and subscribing to such information. Participants in this pub-sub system can be one of three types: i) those advertising services via publishing service information, e.g., an NDN repository providing persistent storage service to other devices, ii) those discovering services via subscribing to service information, e.g., a data collection application on a sensor that needs to use a remote persistent storage service, and iii) those doing both, e.g., a group of computers that dynamically share their spare computation resources with each other. Since Sync provides transport service to NDN applications, it can be used to realize a pub-sub system, as shown by Nichols [20]. **Unlike other pub-sub systems designed for TCP/IP or ICN, Sync-based pub-sub systems do not require central servers, changes in the network layer, or a name resolution system**. Instead,

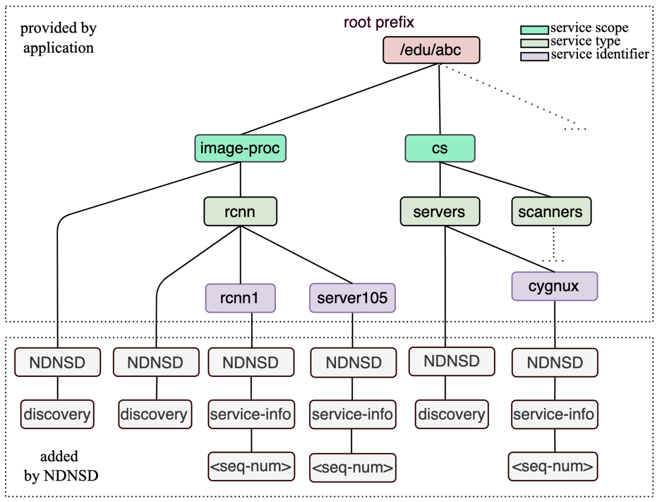


Fig. 3: NDNSD Namespace Design

publishers and subscribers simply agree on a common name for the Sync group, and the subscribers will be notified by the Sync protocol of the new data names whenever a publisher publishes new data.

Given Sync's ability to support pub-sub models, we use it for distributed service discovery. Service publishers and finders can use a semantically meaningful name for their sync group. e.g., *"/edu/abc/library/printers/NDNSD/discovery"* for publishing and discovering the services of all the printers in a university library. Similarly, *"/edu/abc/cs/ndnrepo/NDNSD/discovery"* can be used to discover NDN repositories in a computer science department. These semantic names from the application layer are directly used in the NDN network layer for rendezvous, thus there is no need for name resolution or central servers. Additionally, we use name-based access control (NAC-ABE) to control the accessibility of sensitive services by an unauthorized user. The focus of our work is therefore threefold: (a) design the naming scheme and structure of service information (Section III-B), (b) publish and discover the service information, and (c) Service-info accessibility. Figure 2 shows the workflow of NDNSD among several components of the system. In the following sections, we detail the NDNSD protocol design.

A. Hierarchical Namespace

NDNSD uses two separate namespaces for discovery and application service information, and their design is illustrated in Figure 3

1) Discovery Namespace: Service publishers and finders use a Sync group to rendezvous with each other and we call the name prefix of their Sync group *Discovery Name Prefix*. As shown in Figure 3, the discovery name prefix starts with a routable **root** component, which can be an organization, e.g., */edu/abc*, or an application, e.g., */app/mhealth*. Depending on the specific application scenario, there can be more name components to further constrain the **service scope**, such as physical location (e.g., university library), organization entity (e.g., computer science department), and target users (e.g., computer science students). The next component is **service type**. For example, *image-proc* represents all the image processing services, while *image-proc/rcnn* includes only those running the RCNN algorithm. The above name components are provided by the application to NDNSD through its API (see Section III-C). Finally, the last two name components are added by NDNSD to yield the discovery name prefix, e.g., */edu/abc/image-proc/rcnn/NDNSD/discovery*. "**NDNSD**" differentiates NDNSD's data from other protocols' data, and "**discovery**" differentiates NDNSD's service discovery Sync data from its service-info data.

The above design does not impose a strict limit on the depth of the namespace hierarchy. Moreover, applications have the flexibility to use semantically meaningful names. Note, however, that service publishers and finders in one application need to agree on a service type and avoid collision with other applications. There may be well-known service types emerging as NDNSD is gradually adopted.

2) Service Information Namespace: Each service provider publishes detailed information about its service in the relevant Sync group (see the previous section) so that service finders can choose the most suitable provider. Figure 3 illustrates the design of our service information namespace. It has the same first three name components, i.e., root, service scope (optional), and service type, as the discovery namespace. However, it requires a **service identifier** after service type, e.g., "rcnn1" after */edu/abc/image-proc/rcnn*, to identify a specific service provider, which is supplied by the application. The next two name components, "NDNSD" and "service-info", are added by NDNSD. For example, */edu/abc/cs/servers/cygnux/NDN/service-info* is the name prefix of the service information about the server *cygnux* in the computer science department. The last component is a sequence number for the service information – whenever the server's information changes, the sequence number is increased. The service provider will publish the new name through Sync, which will notify the service finders of the new name so they can fetch the new service information.

Note that service publishers should have a valid certificate for the name they want to use to advertise their service, as NDN requires every data packet to be signed by the public key of the data publisher.

B. Service Information

Service information is a collection of information used by a service provider to advertise its service. Since NDNSD is a generic service discovery protocol, the service information design should support a wide variety of applications. As shown in Figure 4, it is composed of three blocks: service info namespace, required service detail, and optional service detail. The items in the namespace block are used to construct discovery and service information name prefixes (Section III-A) to advertise the service. Using the two service detail blocks, providers can list details of their service using as many key-value pairs as needed. The required block contains the *service name*, i.e., how to reach the service, and *lifetime*, i.e., how long

```
; namespace for service info
service-info-namespace:
  root /edu/abc
  scope computing
  type image-proc/rcnn
  identifier rcnn1

; service specific details
required-service-detail:
  name /edu/abc/computing/image-proc/rcnn/rcnn1
  lifetime 50 ;in seconds

optional-service-details:
  description faster rcnn
  release inception_resnet_v2/1
```

Fig. 4: Sample Service Publisher Configuration File
(The above sample file is written in Boost Info Format [21])

this service information is valid. The optional block contains more application-specific information. For example, an image-processing service can list details about the model used to process the image and release version. IoT applications can provide details about sensor type, location, available memory, sleep time, processing capabilities, etc.

C. API and Protocol Interactions

NDNSD provides the following API for applications to publish and discover service information:

Service Publisher receives and stores the service information from the application. It uses the root, service scope, and service type information in the namespace block to form a discovery prefix (e.g., */edu/abc/library/printers/NDNSD/discovery*) and joins the corresponding sync group. The publisher also uses the root, service scope, service type, and service identifier to construct the service-info data name, e.g., */edu/abc/printers/printer1/NDNSD/service-info/1*, creates a data packet with this name, and stores information from the service-detail blocks in the content of the packet.

Service Finder accepts service discovery requests from an application. Each request contains root, service scope, and service type which are combined to form a discovery prefix. Next, the finder joins the sync group identified by the discovery prefix, fetches the service information from all the service providers, and sends it back to the application. In addition, the finder can provide measurement information to the application on demand (Section III-D).

D. Measurement Information

Measurement information is crucial for applications to determine the Quality of Service and perform load balancing. In order to help service finders make better decisions in selecting service providers, NDNSD computes statistics such as round-trip time (RTT), retransmission count, and timeouts, while fetching the service information or probing a service provider using its service name. The measurements are exposed to the application via the Service Finder's API.

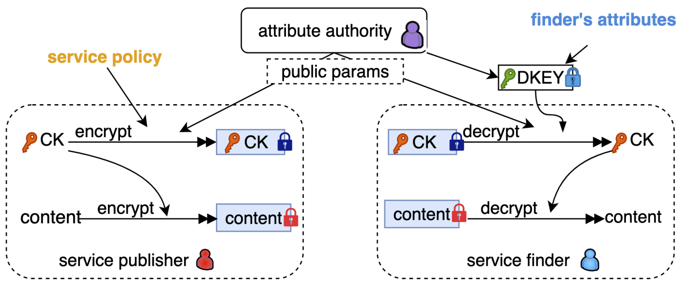


Fig. 5: Use of NAC-ABE in Our System. Note: *content* in the context of NDNSD is service information

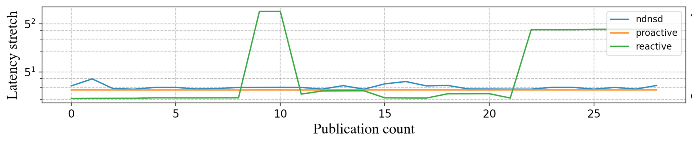
E. Service-info Accessibility

Sensitive services may require some form of access control such that only authorized users can access their service information. For example, a printer in the CS department chair's office may want to restrict public use. We use Name-Based Access Control with Attribute-based Encryption (NAC-ABE) [22] to control the visibility of service information. As shown in Figure 5, NAC encrypts the service information with policies composed of attributes, and generates decryption keys for only those with appropriate access rights. For example, the printer in the Chair's office can advertise a printer service and encrypt the service information with the policy ("CS Chair" or "CS Admin Assistant"). Now to decrypt this service information, the finder needs to have a decryption key with either the "CS Chair" or the "CS Admin Assistant" attribute.

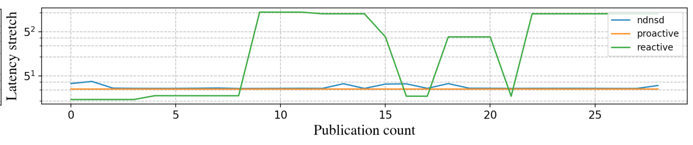
Note that NAC is not used for all the advertised services by default. It is up to each service provider to decide whether to use NAC, based on the sensitivity of the services it offers. For example, public printers do not need access control on their service information.

IV. EVALUATION

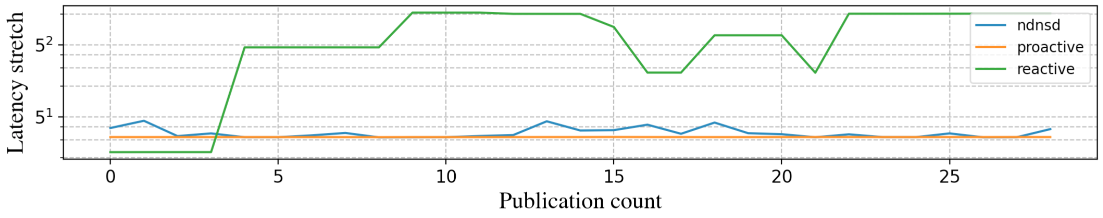
We evaluate NDNSD by comparing it with two simple service discovery schemes, one is **proactive** and the other is **reactive**. In the *Proactive* scheme, the service provider periodically multicasts its service to the entire network using a notification Interest. The Interest consists of the discovery prefix (e.g. */edu/abc/cs/servers*), the application service name (e.g. */edu/abc/cs/servers/cygnux/service-info*), and a sequence number. The sequence number is increased whenever the service status is updated. An application interested in the service listens to the multicast, uses the application service name to construct a service-info Interest, and fetches the corresponding service-info using unicast. The sequence number tells the finder if the multicast was already received and, if so, the finder avoids fetching the same service-info. In this scheme, multicast Interests from multiple service providers (of the same service) are not aggregated due to the presence of the application service name. This can significantly increase packet overhead with an increasing number of providers. In the *Reactive* scheme, the finder multicasts service discovery Interests to the network. Since there can be more than one service provider, the finder iteratively sends multicast Interests,



(a)



(b)



(c)

Fig. 6: Figure (a), (b), and (c) shows the median, 75th, and 90th-percentile service discovery latency stretch, respectively, for each successive publication

with known providers carried in the application parameter field. This process continues until the Interest times out, which means the service-info from all the providers have been fetched. Each provider, after receiving the multicast Interest, checks if its name is present in the application parameter. If so, it ignores the Interest. Otherwise, it sends back its service-info. This iterative process is also periodic, which helps the finder fetch all the updates from the providers. For a fair comparison, we set the frequency of periodic Interest multicasts for all the schemes to once every second in our experiments.

Setup For our evaluation, we emulate multiple service providers offering *Computation Service*. Each provider updates its service-info, such as available GPUs, TPUs, Disk Storage, and Server-load, every 10 seconds for a total of 30 updates. We use Mini-NDN [23], a lightweight network emulator tool, and the NDN testbed topology[1] consisting of 37 nodes and 97 links for all experiments.

A. Performance Metrics

We use the following metrics in our evaluation:

a) Service Discovery Latency Stretch is the ratio between the actual service-info latency (i.e., time to discover some service-info data) and the expected minimum of the same.

b) Normalized Packet Overhead (per node) is the ratio between the overhead per node and the expected minimum number of packets per node. For example, suppose there are 5 service finders and 2 service providers each publishing 30 times at an interval of 10 seconds, the expected minimum number of packets = number of providers $\times$ number of links in both directions $\times$ number of publications = 2 $\times 194$ $\times$ 30 = 11640. For NDNSD, the overhead consists of (i) Sync Interest and Data packets used by the service providers and finders for synchronization, and (ii) the duplicate NACKs sent by the nodes to notify the downstream of receiving the same Interest twice. For the Proactive scheme, the overhead consists of periodic service multicast Interests from the service providers and the duplicate NACKs. For the Reactive scheme, the overhead consists of periodic multicasts to find service from the finders and the duplicate NACKs. Additionally, for all the schemes, extra service-info Interests and data packets are also counted as an overhead. Ideally, there should only be one Interest and one Data packet for each publication.

c) Satisfaction Ratio for a specific service is defined as the number of service providers learned by a finder divided by the total number of service providers offering the same service.

[1]NDN testbed topology: http://ndndemo.arl.wustl.edu/

B. Emulation Results

In Figure 6, we present the median, 75th, and 90th-percentile of the *Service Discovery Latency Stretch*. The results show that both NDNSD and the Proactive scheme have similar low stretches. This is because in both schemes, the service provider advertises its service to the entire network after it is published or updated. The advertisement helps the finders to fetch the service-info immediately after receiving the multicast. Figure 6 also shows that the Reactive scheme performed worst among the three, because in this scheme the publisher needs to wait for a multicast Interest from the finder to send back the service-info. Thus, delayed arrival of the Interests will increase the service-info latency, which explains its high stretch. Occasionally, in the Reactive scheme, Interest arrival and service advertisement can happen at the same time, resulting in a short service-info latency equal to the delay between the service provider and the finder. This can be seen in Figure 6(a) & (b) where the Reactive stretch is lower than the other two schemes and close to 1.

In Figure 7, we present the *Normalized Packet Overhead* of the different schemes. The overhead in the Proactive scheme remains constant as the number of providers increases because the expected and actual number of packets increased by the same factor due to the lack of Interest aggregation in the Proactive scheme. It can be seen that for the other two schemes, NDNSD and Reactive, the overhead starts decreasing as the number of providers increases. This is because of the Interest aggregation in both schemes. In addition, NDNSD performed much better than the Reactive scheme. This is because: (a) NDNSD fetches service-info using unicast whereas

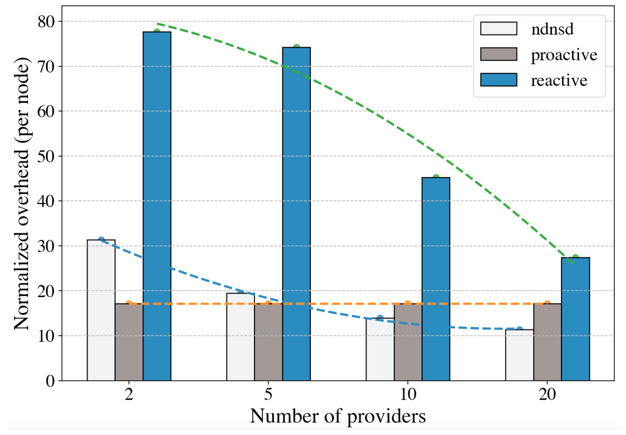


Fig. 7: Normalized Packet Overhead

the Reactive scheme uses multicast, (b) The Reactive scheme always uses one extra final multicast Interest, which times out, to make sure service-info from all the providers are fetched, and (c) a greater number of multicast packets in the Reactive scheme leads to more duplicate NACKs. Thus, we can expect that NDNSD will perform better than the Reactive scheme, even if the number of publishers grows much higher.

Finally, we ran a set of experiments introducing 1, 2, and 4 percent link loss per link and compared the *Satisfaction Ratios*. We observed that both NDNSD and the Reactive scheme were able to receive all the service-info data regardless of the link losses, thereby maintaining a 100% satisfaction ratio. For NDNSD, Sync actively synchronizes the publications using the states among all participating nodes. Thus, advertisements from all the service providers reach the finder. Once an advertisement is received, NDNSD fetches the service-info and does multiple retransmissions if needed. In the Reactive case, the finder's periodic discovery Interest is able to retrieve/recover all service-info data. In contrast, in the Proactive case, there is no finder retransmission, so lost service-info packets cannot be recovered. Hence, we observed 99%, 97.5%, and 97% satisfaction ratios for 1, 2, and 4 percent link loss, respectively.

V. Conclusion and Future Works

We have presented NDNSD, a fully distributed, general-purpose protocol for service publishing and discovery in NDN. We use a hierarchical namespace for NDNSD, which provides applications with fine-grained control over the advertised names of their service(s). Moreover, we leverage NDN Sync to make service discovery independent of any external infrastructure, and designed the service information to support a wide variety of applications. NDNSD provides measurement information to applications to allow better decision-making when selecting service providers, and is able to control the accessibility of service information using NAC. Finally, we have shown that NDNSD outperforms two other baseline service discovery solutions.

We plan to do the following in our next steps: (a) evaluate NDNSD's performance in applications such as building management systems and improve our design and implementation accordingly, (b) refine the collection and representation of measurement information, and (c) conduct evaluation over NDN testbed and release NDNSD to the public.

Acknowledgment

This work was supported by the National Science Foundation awards 1629769 and 2019085. We thank the anonymous reviewers for their insightful feedback.

References

[1] M. Jeronimo and J. Weast, *UPnP design by example: a software developer's guide to universal plug and play*. Intel Press, 2003.
[2] M. Boucadair, R. Penno, and D. Wing, "Universal plug and play (upnp) internet gateway device-port control protocol interworking function (igd-pcp iwf)," *RFC 6970*, 2013.
[3] "IEEE standard for local and metropolitan area networks – station and media access control connectivity discovery," *IEEE Std 802.1AB-2005*, pp. 1–176, 2005.
[4] E. Guttman, "Service location protocol: Automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71–80, 1999.
[5] L. Smith, C. Roe, and K. S. Knudsen, "A jini/sup tm/ lookup service for resource-constrained devices," in *Proceedings 2002 IEEE 4th International Workshop on Networked Appliances*, 2002, pp. 135–144.
[6] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in IoT networking via TCP/IP architecture," *NDN Technical Report 0038*, 2016.
[7] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named Data Networking of Things," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2016, pp. 117–128.
[8] Amazon Inc., "AWS IoT," https://aws.amazon.com/iot/solutions/connected-home/., (Accessed on 07/05/2022).
[9] S. Weber, *Chromecast Users Manual: Stream Video, Music, and Everything Else You Love to Your TV*. USA: Weber Systems Inc., 2014.
[10] Mircosoft Inc., "Azure iot edge, 2020," https://azure.microsoft.com/en-us/services/iot-edge/., (Accessed on 07/05/2022).
[11] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.
[12] P. Moll, W. Shang, Y. Yu, A. Afanasyev, and L. Zhang, "A survey of distributed dataset synchronization in named data networking," NDN, Technical Report NDN-0053, Revision 2, 2021.
[13] M. Mosko, "CCNx 1.0 collection synchronization," in *PARC Technical Report*. Palo Alto Research Center, Inc., 2014.
[14] R. Ravindran, T. Biswas, X. Zhang, A. Chakraborti, and G. Wang, "Information-centric networking based homenet," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 1102–1108.
[15] ndn lite, "Ndn-lite service discovery," 2018, accessed: 2020-04-22. [Online]. Available: https://github.com/named-data-iot/ndn-lite/wiki/Service-Discovery
[16] M. Amadeo, C. Campolo, and A. Molinaro, "NDNe: Enhancing Named Data Networking to support cloudification at the edge," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2264–2267, 2016.
[17] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, "ICedge: When edge computing meets information-centric networking," *IEEE Internet of Things Journal*, 2020.
[18] NDN Project Team, "Named Data Networking Forwarding Daemon," https://github.com/named-data/NFD, (Accessed on 07/05/2022).
[19] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013.
[20] K. Nichols, "Lessons learned building a secure network measurement framework using basic ndn," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, 2019, pp. 112–122.
[21] M. Kalicinski and S. Redl, "Boost. propertytree," *Online: http://www.boost. org/doc/libs/1*, vol. 42, no. 0, 2008.
[22] Z. Zhang, Y. Yu, S. K. Ramani, A. Afanasyev, and L. Zhang, "NAC: Automating access control via Named Data," in *IEEE Military Communications Conference (MILCOM)*, 2018, pp. 626–633.
[23] NDN Project Team, "Mini-NDN GitHub," https://github.com/named-data/mini-ndn, (Accessed on 07/05/2022).