Supporting Delay Tolerant Networking: A Comparative Study of Epidemic Routing and NDN

Tianxiang Li
UCLA
tianxiang@cs.ucla.edu

Zhaoning Kong
UCLA
jonnykong@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

***Abstract*—Delay Tolerant Networking (DTN) is characterized by its dynamic and intermittent connectivity, resulting in the absence of end-to-end communication paths in general. Many proposed solutions have been developed over the years to enhance TCP/IP protocol stack for DTN environment; Epidemic Routing (ER) is among the earliest and most well-known designs. Recent years have seen both renewed interests and investigations into Epidemic Routing for vehicular and satellite communications, and the development of a new Internet architecture Named Data Networking (NDN) which, due to its data-centric design, can support DTN communications natively. In this paper, we identify the basic functionality requirements for DTN support, compare and contrast ER and NDN to show the commonalities and differences in their designs. We use simulation results to illustrate how the design differences lead to different functionalities and protocol performance: although ER enhances IP nodes with data-centric features to enable packet delivery in DTN environments, compared to NDN's native data-centric design with built-in security, such "patch-on" suffers from not only lower performance with higher overhead, but more importantly the lack of systematic security support.**

***Index Terms*—Delay-Tolerant Network (DTN), Named Data Networking (NDN), Epidemic Routing, Distributed Dataset Synchronization**

I. Introduction

Recent growth in vehicular networking, satellite communication, as well as terrestrial wireless networks drives the need for secure and effective solutions for Delay/Disruption Tolerant Networking (DTN). DTN is generally characterized by a high degree of dynamic and intermittent connectivity among communicating nodes, defeating the conventional address-based, infrastructure-dependent data delivery. Securing communication also becomes challenging due to the lack of stable end-to-end connectivity and stable access to the public key infrastructure (PKI).

The DTN communication framework runs on top of the host-centric IP architecture, and delivers bundles [1] of data identified by the destination end point. Many IP-based DTN routing solutions [2] have been developed to support opportunistic data delivery, which generally use *store-carry-forward* function to let randomly encountered nodes pass packets to each other, so that packets can eventually reach their destinations. Among the existing solutions Epidemic Routing (ER) [3] is a well-known one and introduces some of the fundamental design principles for data delivery under intermittent connectivity. ER enhances IP's point-to-point, *store-forward* packet delivery with data-centric features, mainly assigning a unique identifier to each packet in addition to its IP destination address[1], so that an IP node can recognize each stored packet to perform opportunistic packet forwarding with random encounters, enabling *store-***carry***-forward*.

Named Data Networking (NDN) [4], a data-centric network architecture, has been under design and development since 2010. As NDN matures over time, there have been an increasing number of efforts that apply NDN to support DTN. Due to its data-centric design, NDN can provide effective and efficient solutions to data distribution with end-to-end security, in the absence of end-to-end connectivity. A question naturally arises: given the ER design enhances IP nodes with data-centric features, are there any fundamental differences between IP/ER and NDN's ability in supporting DTN, which can lead to significant differences in the functionalities and performance each provides?

This paper aims to answer the above questions. We first identify the basic requirements for secure delivery of data over intermittent connectivity, explain how each protocol meets those requirements. We then compare in detail the similarities and differences between the designs of IP/ER and NDN, analyze how these design variations lead to differences in protocol performance. Finally, we summarize the important design components to build a secure, efficient, and resilient communication protocol in DTN.

II. Background

To deliver packets over intermittent connectivity, a DTN-capable node must be able to identify and carry received data packets when disconnected, until it can pass them to the next encounter. To support communication security, whoever interested in using the received data must be able to authenticate it. Achieving the above goals require the following supports: (i) unique identifiers for each packet, so a node can tell which ones itself, or its encountered neighbor nodes, may be missing, in order to fetch/pass along; (ii) memory space at each node to hold packets when disconnected; and (iii) the ability to authenticate packets without requiring additional communication with some centralized servers.

To provide the audience necessary information for a in-depth comparison, in this section we first describe the basic design of ER, then introduce the NDN architecture and *NDN*

[1]IP addresses alone do not provide enough information for packet identification, because multiple packets can have identical source and destination addresses.

978-1-7281-7440-2/20/$31.00 ©2020 IEEE

Sync, NDN's transport support by using the DDSN protocol as a specific example.

A. Epidemic Routing

Epidemic Routing (ER) is a proposed solution for message delivery in DTN environments. ER's data propagation design is based on the "Epidemic Algorithms" [5] designed to synchronize updates between distributed databases. Each site with new updates exchanges database information with a randomly selected neighbor to resolve conflict. Through this pair-wise exchange of updates, all the sites will eventually reach consistency within a bounded time range. ER utilizes this idea for data dissemination over intermittent connectivity. Each ER node periodically broadcasts Beacon messages to detect neighbors Opportunistically encountered ER nodes exchange packet set information, and fetch missing packets from each other. Through this process, packets are passed through intermittent connectivity with a high probability of eventually reaching their intended destinations.

ER uses IP unicast for packet delivery. To support ER, each IP node is enhanced with a data buffer to carry packets. When a node comes into contact with another node, the two synchronize their packet sets. To do so, ER assumes that each packet is associated with a unique identifier (UID), which can be the concatenation of the packet's originating node's identifier and a locally generated number. The node's identifier can be generated using decentralized or centralized mechanisms similar to network address assignment in Mobile Ad-Hoc networks [7]; the exact approach is outside the scope of the ER design [3]. Each ER node maintains a *Summary Vector* which enumerates the UIDs of all the packets it keeps in its memory. When two nodes A and B encounter each other, they first exchange Summary Vectors to determine which packets node A has but B doesn't, and vise versa. Then each node either requests the missing packets from the other, or pushes packets that the other misses. This process is repeated in a pairwise, point-to-point fashion among all the encountered nodes. When a packet reaches its destination, the destination may send an optional ACK message to inform other nodes to remove the already delivered packet from their local buffer to stop further dissemination. As one of the early protocol designs in DTN, ER did not consider communication security. This is also because the traditional channel based security mechanisms such as TLS or DTLS do not apply to DTN environments, due to the absence of E2E connections and the lack of access to the PKI infrastructure.

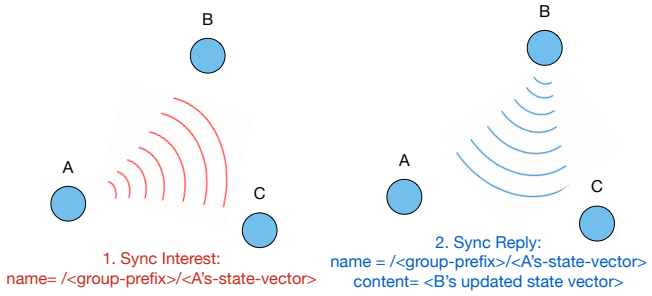
B. NDN

From 10,000 feet, one might view the basic idea of NDN as shifting HTTPs request (for a named data object) – and –response (containing the object) semantics from the application layer to network layer delivery. Applications generally identify data by hierarchically structured names, which consists of a number of semantically meaningful components. NDN uses these application-generated names to fetch data at network layer. This design brings two benefits: (i) all data pieces already have unique names that are meaningful to applications and independent from network layer connectivity (which changes dynamically in a DTN environment); and (ii) one can secure data directly by having data *producer* generate a crypto signature to bind the name and content in each NDN data packet.

In NDN, A data *consumer* sends *Interest* containing the name of the requested *Data*. To fetch Data effectively and efficiently, each NDN node keeps three main components: (i) Content Store (CS) to buffer received Data packets; (ii) Pending Interest Table (PIT) to buffer each Interest that has been forwarded but not replied, together with its incoming and outgoing interfaces; and (iii) Forwarding Information Base (FIB) to keep information regarding where to forward Interests. The buffering capability for Interests (PIT) and Data packets (CS) makes NDN nodes resilient to network disconnects, thus by design an NDN network can fetch individual Data packets over intermittent connectivity. However, effective DTN support also requires that NDN nodes have means to learn what data a node can fetch from random encounters. This task is performed by NDN Sync [8], NDN's transport layer abstraction. One may view TCP, a transport protocol in use, as synchronizing the data reception state between two *connected* communicating ends, so that the sender can retransmit any lost data. NDN Sync enables dataset state synchronization among *multiple* communicating parties in the same application instance (we call these parties *members* of a *sync group*). In addition to multi-party support, NDN Sync also differs from TCP in another fundamental ways: NDN lets receivers, instead of senders, be responsible for data delivery reliability, thus NDN Sync only informs all members in a sync group of new data generations, leaving data fetching as a separate step by individual members. Furthermore, NDN Sync can function resiliently over intermittent connectivity, thanks to PIT and CS at each NDN node.

Each member in a sync group keeps a local view of the dataset, called the members *sync state*, and the process of synchronizing the sync state of members is called *state synchronization*. A number of NDN Sync protocols had been designed [9]–[13], with Distributed Dataset Synchronization in Disruptive Networks (DDSN) [13] being one of them whose design specifically takes into account network scenarios with mobility and intermittent connectivity.

DDSN uses a shared name prefix to name a group's dataset, and adopts a sequential naming convention for application generated data (e.g. "/[group-prefix]/[member-prefix]/[data-sequence-number]"). Each producer in a group names its newly generated data with monotonically increasing sequence number, and DDSN encodes the dataset of a group in the form of a *State Vector*, i.e. a list of (p_i, seq_i) pairs, where p_i is the name of ith producer and seq_i the sequence number of p_i's latest data piece. The members in the same sync group exchange the State Vector through the transmission of Sync Interest and Sync Reply, as shown in Figure 1. Sync Interest contains the State Vector in its name, for example in a chat group of three users A, B, and C, the Sync Interest



(a) DDSN Sync Interest (b) DDSN Sync Reply

Fig. 1. A sends a Sync Interest (a) which is received by B and C, then B returns a Sync Reply (b), which is received by A and C.

carries a name "`/ucla_chat/group5/sync/[A:3,B:1,C;6]`". Sync Interests is generated both periodically, and event-driven: whenever a node's dataset state is updated, it will send out a Sync Interest with its updated State Vector through wireless broadcast channel, reaching all other members within the communication range. A receiver of the Sync Interest I_s updates its *sync state* if I_s contains new information, and sends a Sync Reply reporting its latest State Vector. If missing data is identified, a node sends data Interest to fetch. Because state Vector represents the raw state information, its processing has no dependency on the receiver's state. This property is particularly suited to DTN environment, where node connectivities are short-lived, and randomly encountered nodes may have potentially large differences between their dataset states.

III. Protocol Comparison

In this section, we identify the commonality and differences between ER and DDSN. To support DTN, the two protocols share a few common steps. First, mobile nodes need to discover the existence of neighbors, which requires a node to proactively send packets without knowing whether any other node may be around. Once neighbors are discovered, nodes exchange dataset information to identify one's missing data, i.e. performing *state synchronization*, and then fetch missing data from each other. All the above steps require security protection, which is especially critical for communicating with ad hoc encounters. Below we start with a security comparison before proceeding to comparisons of the other steps.

A. Security

We noted earlier that the ER design has no security consideration. One good reason is that today's security solutions require end-to-end connectivity and access to centralized certificate authorities, which do not exist in DTN environments. NDN provides end-to-end secure support in DTN environments through the following means: i) establishing trust relations among communicating entities *a prior* by installing trust anchors, certificates, and security policies into each node [14] [2]; and ii) securing data directly, which allows consumers to verify received data using the established security policies, independent from where the packets come from. There has also been a growing trend in the IP-based DTN protocols for securing data directly, [15] is an example of the different efforts in this direction.

[2]These security primitives are a part of the NDN protocol architecture; similar concepts exist at the application layer in the TCP/IP architecture, thus their utilization depends on infrastructure connectivity.

B. Data Naming and State Encoding

Although both ER's Summary Vector and DDSN's State Vector encode a set of data identifiers to exchange dataset state between neighbor nodes, the two differ significantly. As we explained in §II-A, ER is designed to run over IP which names nodes, and develops its own unique data packet identifiers (UIDs) to enable neighbor nodes identify missing packets from each other through Summary Vector exchanges; the packet UIDs are added between application and IP, and used for packet dissemination only. Each UID is a datagram identifier and has no semantic relation to any other UIDs. Consequently a Summary Vector must enumerates the UIDs of all the packets carried by a node, which results in a large size. Alternative encoding mechanisms, such as Bloom Filter [16], have been proposed to reduce the size of Summary Vectors, by increasing the overall ER complexity.

A more effective size reduction in state encoding is to utilize structured data namespace as NDN Sync does. As described in §II-B, DDSN uses monotonically increasing sequence numbers to name data, and encodes the dataset of a group in the form of a State Vector, i.e. a list of (node, seq#) pairs. Thus the size of a State Vector is upper-bounded by the number of producers in a sync group.

C. Neighbor Discovery

Both ER and NDN detect neighbors through periodic broadcasts. ER uses small size Beacon messages, which contain only the sender's identifier, for neighbor discovery, and exchanges the dataset information after neighbor detection. When a node, say B, receives a Beacon sent by node A, B first checks to see if it has talked to A recently; if not, it sends its Summary Vector as a response, thus A and B discover each other.

A DDSN node A broadcasts Sync Interest packets periodically to serve two purposes: neighbor discovery and dataset state difference detection (if/when any other node receives the Sync Interest). Upon receiving a Sync Interest, a node B sends back a Sync Reply reporting its latest dataset state, B's reply indicates a neighbor's existence, and if the reply reports any new data that A is unaware of, A sends data Interest to fetch. The use of Sync Interest for both neighbor discovery and dataset difference detection saves one packet transmission if neighbor node(s) exists, but pays the cost of a bigger Interest packet size as it carries the state vector.

D. State Exchange

Besides the difference in the dataset state encoding between ER and NDN, there exist two other differences, both result from ER being a patch on node-centric IP communication versus NDN being a native data-centric design.

ER runs over the point-to-point IP packet delivery. If we assume four ER nodes encounter each other, six pairwise

Summary Vector exchanges will be needed among the four nodes, despite the fact that those exchanges are over wireless communication which is broadcast by nature. NDN nodes use names to fetch data, thus when a node broadcasts its Sync Interest, all other nodes within the wireless range can receive it and process accordingly. One might argue that ER could also broadcast UDP packets encapsulated in IP to exchange Summary Vectors among multiple nodes. However doing so leads to the observation that the IP addresses carried in those packets are no longer needed. Instead, the communication utilizes *data* identifiers, the packets' UIDs.

Another difference between ER and NDN is the propagation of dataset state. When an ER node A detects a new neighbor B that A has not communicated with recently, A exchanges Summary Vector with B. Assuming A detects node C soon after and fetches new packets from C, if B cannot hear from C, A will not convey the information about C's new packets to B. That is, dataset state exchange is triggered by new *node encounters* only. Sine DDSN takes a data-centric approach, whenever node A detects new dataset state from a newly received Sync Interest, A further propagates the new information. Thus Sync Interests carrying new dataset state information can quickly propagate through multiple hops within connected node clusters. Furthermore, an ER node must receive all the missing data the current neighbor can offer before it starts Summary Vector exchange with another neighbor, while DDSN decouples state propagation and data fetching, as we explain next.

E. Data Fetching

In ER, data fetching is tied with the state exchange between two specific nodes. That is, an ER node A does not know which piece of data it may be missing, A depends on the Summary Vector exchange with neighbor B to determine what data it can get from B, and vice versa. After A and B exchange Summary Vectors, each can obtain identified missing packets from the other.

DDSN separates the state exchange and data fetching into two independent processes. Through state exchange, each DDSN node maintains an up-to-date state about the newest data names available, and then fetches the missing data from any encountered nodes. A requester does not know, and does not care, which neighbor may have which piece of data. It simply sends data Interests to request missing data periodically when it is connected to some neighbors, or whenever it discovers new neighbors.

NDN's opportunistic caching enables a node to cache Data packets it overhears, independent of whether those Data packet have been requested by the local process or not. The node can also reply to a received data Interest when the data with matching name is found from its local CS. The local CS of each NDN node makes it an effective data mule to carry Data packets around, and fetching data by names makes effective use of NDN nodes' data muling function.

IV. Performance Evaluation

In this section, we evaluate the performance difference between ER and DDSN based on the same experiment set up used in our previous work [13]. We also reuse some of our previous results, but provide a more detailed and in-depth exploration on the major factors that caused the performance differences between ER and NDN/DDSN.

A. Experimental Setup

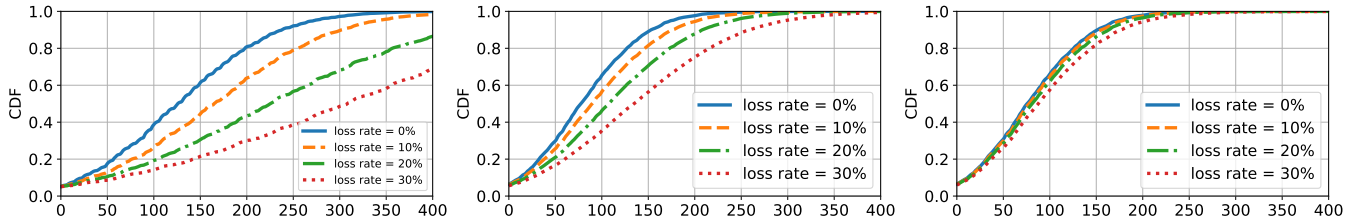
We implemented DDSN in C++ and ported into the ndnSIM [17], an NDN network simulator based on NS-3. For comparison, we used an open-source implementation of ER [6], and conducted simulation in NS-3. Our simulation topology is a 800m$\times$800m grid, consisting of 30 mobile nodes. Among the 30 nodes, 20 run DDSN or ER protocol, the other 10 are pure forwarders that forward any received packets at 50% probability. The mobile nodes follow a random-walk mobility model: every 20sec, a node randomly changes its direction and speed (1m/s to 20m/s). Each node generates data (100-1024 bytes random text) for the first 800s of the simulation, following a Poisson process at rate $\lambda = 1/40$ per second. The results are averages of 10 experiment runs. We set the interval for periodic Beacon transmission of ER and periodic Sync Interest transmission of DDSN to both be 8sec.

For comparison, we mainly consider two evaluation metrics: (i) *Data Retrieval Delay*: the time needed for one newly generated data to be disseminated to members in the group; and (ii) *Overhead*: the total size of all the packets transmitted (bytes) for all the members to receive all the generated data.

B. Data Retrieval Delay

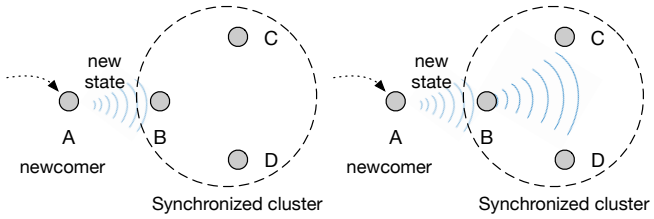
Figure 2 presents the CDF of the Data Retrieval Delay for ER and DDSN under varying packet loss rates. ER nodes take approximately 230s and 570s to retrieve 90% of the generated data, under the loss rates of 0% and 30%, respectively (Figure 2(a)). In comparison, DDSN nodes retrieve 90% of the generated data in less than 150s and 170s of delay, under the loss rates of 0% and 30%, respectively (Figure 2(c)). In summary, DDSN shows 34-70% lower data retrieval delays than ER under loss rates from 0% to 30%.

1) host-centric vs data-centric state exchange: A ER node A initiates state exchange *only when* it encounters a new neighbor which it had not recently talked to. However, keeping track of recently encountered nodes is not equivalent to keeping track of their dataset state; some of them may have learned new data *after* the Summary Vector change with A. Ideally, nodes should exchange state when any of them can offer new state, to do so requires a data-centric communication model. DDSN nodes detect state inconsistency through carrying the State Vector by Sync Interest packets, which can trigger a chain of state exchange when new dataset state is detected anywhere within a connected node cluster, minimizing the delay in state update. For example, in figure 3, node A carries new state and comes into contact with node B. Node B is part of a connected cluster, where B, C, D are synchronized with each other. In figure 3(a), ER nodes A and B exchange states,



(a) ER (b) DDSN (default) (c) DDSN (with fast retransmit)

Fig. 2. Data Retrieval Delay



(a) ER State Propagation (b) DDSN State Propagation

Fig. 3. State Propagation Example

but B will not propagate new states to C and D immediately, since C and D are existing neighbors of B. This is because two ER nodes do not exchange data more than once within a specified period to prevent excessive traffic. In figure 3(b), the state exchange between A and B will trigger B to immediately propagate the new dataset state to C and D. C and D can then further propagate states into other connected neighbors.

2) point-to-point vs multi-party communication: Generally speaking, wireless network is broadcast by nature. However ER is designed on top of the point-to-point communication model of the IP architecture, where each ER node exchanges state and data with one individual node at a time, sending unicast packets over a broadcast wireless medium. In comparison, DDSN utilizes NDN's data-centric communication model by using application layer data names for network layer data fetching. An NDN packet does not target a specific destination, and can be received by any node within one-hope wireless range. For example in Figure 3(b), a packet transmitted by node B can be received by both nodes C and D. The State Vector in the Sync Interest name can be directly interpreted by C and D to update their state. The name of a data Interest can be understood by all the receivers, and any node which has the matching Data packet can reply. The Data reply packet can then be cached by any node receiving it.

3) Resilience to Loss: As shown in Figure 2, DDSN has a much more stable performance under varying loss rates compared to ER. This is mainly due to two reasons. First, DDSN's event-driven state propagation enales any lost updates to be retransmitted whenever state difference is detected in the network. The periodic transmission of Sync Interest provides a periodic detection of any state mismatch in the neighborhood. A successful transmission of a Sync Interest with new state will trigger the dissemination of the updated state throughout the network. This chain of events continue until all nodes in the connected cluster reach a consistent state.

Another reason is that each DDSN node maintains a local state of what data it is missing, and this allows each node to fetch data persistently until all missing data are retrieved. In ER, however, each packet is identified individually by its UID, which makes it impossible for each ER node to learn the existence of all the packets that have been produced in order to know which packets it may be missing. Instead, an ER node can only learn which packets it can fetch from an encounter. Since data synchronization depends on successful exchange of Summary Vectors, and then successful exchange of data messages, any packet loss in this chain of exchanges will cause the data fetching process to break. Consequently, the data retrieval delay in ER worsens as loss rate increases.

4) Retransmission Strategy: By decoupling state and data synchronization, DDSN allows nodes to adopt different retransmission strategies. For DDSN under 0% packet loss, 85% of data was successfully fetched on the first request, meaning that the sender of new dataset state is highly likely to carry the new data as well. Thus, in addition to the default strategy of periodically retransmitting Interests for data at a fixed interval (Figure 2(b)), we adopted a strategy (Figure 2(c)) to prioritize the retransmission of newly learned data names, by retransmitting it a few times (5 in our case) with short delay, to overcome packet loss. If no Data packet is received, the data Interest will be retransmitted at a lower frequency, to opportunistically request data from other encountered nodes as the requester moves in the network. From Figure 2(c), we can see that DDSN's data retrieval delay remains fairly stable under varying loss rates, with a slight increase in average delay as loss rates increase. The difference in data retrieval delay is much less compared to DDSN with simple periodic retransmission (Figure 2(b)). The total number of Interests also lowers as data Interests have a higher chance of fetching Data packets back.

C. Overhead

In terms of transmission overhead, DDSN generates 1.44 to 1.49 bytes$\times 10^7$ of transmission overhead under loss rates from 0% to 30%. Comparatively, ER generates 2.48 to 2.67 bytes$\times 10^7$ of overhead under loss rates from 0% to 30%. In summary, DDSN generates 40-45% lower overheads under 0-30% loss rates compared to ER.

1) Encoding: The major contributing factor to the overhead difference is state encoding. As mentioned in Section III-B, DDSN's structured namespace allows the entire group dataset

to be represented by enumerating the (producer, newest sequence number) pairs, so the State Vector's size depends on the number of producers in the sync group. ER's Summary Vector enumerates the UIDs of all the packets a node carries, whose size scales with the number of packets generated. In our simulation, the State Vector size remained to be 150-200 bytes, while the Summary Vector size quickly increased to 1600 bytes, reaching its upper limit. When the number of generated packets exceeds a certain threshold, some data identifiers will be evicted from the summary vector. This could cause some data to never be received under a high loss environment.

2) Avoiding Redundant Transmissions: Both protocols adopted mechanisms to avoid redundant transmissions. ER nodes maintain a list of recently communicated nodes, and only exchange messages with newly encountered neighbors. DDSN nodes suppress Sync Interest propagation based on whether or not the received State Vector contains new states. Although event-driven Sync Interest propagation triggers the flooding of Sync Interest in the network, no duplicate Sync Interests will be transmitted due to state based suppression. However, the difference in the total number of packets between ER and DDSN is much less significant compared to the size difference of the two protocols' state exchange messages.

V. Lessons Learned

In this section we summarize the lessons learned from conducting this comparative study of IP/ER and NDN/DDSN on how to design an effective protocol in support of DTN. Although NDN Sync, and DDSN in particular, is still under active development, we believe the following basic lessons should remain true in the future solution development.

1) What to name: IP's model of delivering packets to destinations identified by IP addresses over a *connected infrastructure* cannot serve DTN environments with mobile nodes under intermittent connectivity. ER recognized the need to assign unique packet identifiers, but its design is constrained by the underlying IP's point-to-point datagram delivery model. By design NDN names and secures data directly, thus its Interest and Data exchange (data requests and replies), both carrying data names, can fully utilize wireless broadcast. A fundamental gain comes from the structured namespace of NDN, which allows nodes to identify what data is missing, support security and trust policies, as well as efficiently utilize caching. It is also an ongoing work in the wider area of Information-Centric Networking (ICN) [18] to identify the benefits of using structured namespace in DTN, and the challenges to address.

2) Security: Security support is essential when communicating with random encounters. The unfortunate fact that many existing DTN solutions neglected security might be partly due to overlooking its importance, and partly due to the high challenge as the conventional channel-based solutions do not work. There are ongoing efforts in the DTN research area to move towards data-centric security [15]. In addition to making communication security as a property of the data itself, instead of data containers or communication channels, NDN also recognizes the fact that the trust relations exist among all communicating entities, independent from the existence of, or lack of the channels between them. Thus NDN nodes are equipped with pre-established trust relationships between communication endpoints that can be used for data verification.

3) State and Data Exchange: Dataset dissemination involves two steps: state synchronization and data synchronization. In ER and other IP based solutions, because data names are *invisible* at network layer, ER has to bundle the two steps together. DDSN's decouples the two steps, minimizes state propagation delay, and its naming conventions allow nodes to easily identify missing data.

4) In-network storage: DTN communications critically depend on *store*-**carry**-*forward* mechanism, which in turn requires data be identified on its own, independent from its destinations, to facilitate data dissemination over intermittent connectivity. Together with named, secured data, NDN's built-in per node caching can offer high data availability despite heavy network losses.

References

[1] S. Burleigh, K. Fall, and E. Birrane, "Bundle protocol version 7," IETF Secretariat, Internet-Draft draft-ietf-dtn-bpbis-24, 2020.

[2] C. Sobin, V. Raychoudhury, G. Marfia, and A. Singla, "A survey of routing and data dissemination in delay tolerant networks," *Journal of Network and Computer Applications*, vol. 67, pp. 128–146, 2016.

[3] A. Vahdat, D. Becker *et al.*, "Epidemic routing for partially connected ad hoc networks," 2000.

[4] L. Zhang *et al.*, "Named Data Networking," *ACM Computer Communication Review*, July 2014.

[5] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987.

[6] M. Alenazi *et al.*, "Epidemic routing protocol implementation in ns-3," in *Proceedings of the 2015 Workshop on ns-3*, 2015, pp. 83–90.

[7] M. Al-Shurman, M. F. Al-Mistarihi, and A. Qudaimat, "Network address assignment in mobile ad-hoc networks," in *International Congress on Ultra Modern Telecommunications and Control Systems*. IEEE, 2010.

[8] T. Li, W. Shang, A. Afanasyev, L. Wang, and L. Zhang, "A brief introduction to ndn dataset synchronization (ndn sync)," in *2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018.

[9] Z. Zhu and A. Afanasyev, "Let's chronosync: Decentralized dataset state synchronization in named data networking," in *21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2013, pp. 1–10.

[10] W. Fu, H. Ben Abraham, and P. Crowley, "isync: a high performance and scalable data synchronization protocol for named data networking," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM, 2014, pp. 181–182.

[11] M. Zhang *et al.*, "Scalable Name-based Data Synchronization for Named Data Networking," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, May 2017.

[12] W. Shang, A. Afanasyev, and L. Zhang, "Vectorsync: distributed dataset synchronization over named data networking," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017.

[13] T. Li, Z. Kong, and S. Mastorakis, "Distributed dataset synchronization in disruptive networks," in *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2019.

[14] Z. Zhang *et al.*, "An overview of security support in named data networking," *IEEE Communications Magazine*, vol. 56, pp. 62–68.

[15] G. Selander *et al.*, "Object security for constrained restful environments (oscore)," Internet Requests for Comments, RFC 8613, July 2019.

[16] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[17] S. Mastorakis *et al.*, "On the evolution of ndnsim: An open-source simulator for ndn experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, 2017.

[18] J. Seedorf *et al.*, "Research directions for using icn in disaster scenarios," IETF Secretariat, Internet-Draft draft-irtf-icnrg-disaster-10, 2020.