Achieving Resilient Data Availability in Wireless Sensor Networks

Xin Xu, Haitao Zhang, Tianxiang Li and Lixia Zhang
Computer Science Department, University of California, Los Angeles
Email: xinxu129, haitao, tianxiang, lixia@cs.ucla.edu

***Abstract*—Currently, most Wireless Sensor Networks (WSNs) utilize sleeping mechanisms to conserve energy, which introduces the problem of data availability. To address this problem, we designed DSSN, a Dataset Synchronization protocol for WSN with Sleeping sensors over Named Data Networking (NDN). DSSN divides sensors into groups, each has a shared dataset; through dataset synchronization within each group, DSSN ensures that the group's latest dataset is always accessible from its active sensors. DSSN utilizes *State Vector* to enable reliable dataset synchronization, and utilizes NDN's *Interest* aggregation and *Data* caching to optimize energy consumption. We evaluated DSSN through extensive simulation experimentation, and our results show that DSSN ensures data availability close to 100% under various network conditions with negligible overhead.**

I. Introduction

Energy consumption is one of the most important issues in wireless sensor networks (WSNs) with battery-powered sensors. WSNs usually adopt sleeping mechanisms to conserve energy and extend network lifetime. Therefore each sensor can be in one of the two states at any time: active or sleep. In active state, a sensor communicates with other sensors to make its data available; in sleeping state, a sensor turns off its radio and stops communication, making its data unavailable.

One way to solve this problem is to let a sensor replicate its *Data* with other active sensors before it goes to sleep. Named Data Networking (NDN) [1], [2], a proposed Internet architecture, can greatly facilitate data replication in WSN. NDN replaces TCP/IP's host-oriented communication model with a data-centric communication model, enabling retrieval of named *Data* regardless of its location. Motivated by NDN's attractive feature, NDN+CoCa [3] developed a distributed cooperative *Data* caching solution over NDN to improve WSN data availability. However, [3] performs best-effort caching rather than reliable data replication, thus it ensures data availability only to certain extent.

In this paper, we present DSSN, a dataset synchronization protocol for WSN with sleeping sensors over NDN, which enables reliable data replication in WSN. The design of DSSN is inspired by VectorSync [4], where each producer names its *Data* by a unique prefix and a monotonically increasing sequence number; such naming convention enables the use of a *sequence number vector* to represent one's knowledge about the shared dataset in a compact way. However different from VectorSync which assumes that all nodes would be online, DSSN must operate reliably even when not all the sensors are active at any given time. Therefore, DSSN removes the group management mechanism of VectorSync, which requires consensus from all the group members; instead DSSN performs dataset state synchronization by explicitly enumerating sensor names together with their data sequence numbers, which we call *State Vector*. Compared to Summary Vector in [5], which has an unique identifier associated with each message, DSSN State Vector uses a sensor ID plus data sequence number pair to represent all the data produced by one sensor. DSSN also makes use of NDN's *Interest* aggregation and *Data* caching to reduce packet transmissions, and adopts mechanisms for collision avoidance, to lower its own energy consumption in a wireless environment.

The rest of this paper is organized as follows. Section II gives an overview of NDN and gives assumptions we make in WSN. Section III describes the DSSN protocol design in detail. Section IV presents the simulation results and analysis on the performance of DSSN. Section V discusses our future work. Section VI concludes the paper.

II. Background

This section introduces the NDN architecture, a summary of distributed dataset synchronization protocols in NDN, and a WSN example used in this paper.

A. Named Data Networking

Named Data Networking (NDN) architecture utilizes the fetching model to communicate. Consumers request *Data* by sending an *Interest* which carries the name, or the prefix, of the desired *Data*. The *Data* naming is unique, hierarchical structured, and semantically meaningful. An *Interest* is forwarded based on names. NDN Forwarding Daemon (NFD) [6] running on every NDN node implements NDN's forwarding logic. When forwarding an *Interest*, NFD first check its Content Store (CS) for previously cached *Data* matching the *Interest* name. If a match is found, NFD returns the cached *Data* and stops forwarding process. If no match is found, NFD looks up its Pending Interest Table (PIT), which lists all the *Interests* received recently but not yet satisfied, and the interfaces where it was received from. If a matching *Interest* is found in the PIT, NFD aggregates the *Interests*. Otherwise NFD adds the newly received *Interest* in the PIT. Then it looks up the forwarding table (FIB) containing next hop information and applies a forwarding strategy to forward the *Interest*. The *Data* is returned to the consumer by following reversely along the path used to forward the *Interest*. The returned *Data* is also cached by the routers in its CS along the transmission path.

B. Distributed Dataset Synchronization in NDN

Dataset sync is an important concept in the NDN architecture. [7] summarizes three key design aspects of existing sync protocols, which is the basis for the DSSN protocol design.

1) Group Communication Namespace: A shared group communication namespace is required for members to exchange messages for data synchronization. Messages sent to this namespace should be received by all other members in the same group, in the absence of packet loss.

2) Dataset State Representation: Sync is conducted in group. Each member in a sync group keeps a local set of dataset shared with other members. Each member also keeps a local dataset state, which represents its local knowledge of the shared dataset namespace, i.e., what *Data* is in the shared dataset. Thus the node can determine the previously unaware data by comparing dataset states with others.

3) Dataset Synchronization Mechanism: Each member in the sync group can generate new *Data* in its shared dataset at any given time, causing a difference in the dataset state among group members. A sync protocol needs to notify other members of the dataset state update and provide a mechanism for them to synchronize their dataset states to achieve an overall agreed dataset state. After dataset state synchronization, a member could then fetch some or all of the dataset update based on its own desire.

C. A WSN Example

To facilitate explanations, we use an example scenario throughout this paper. Assuming in each room of UCLA Boelter Hall, there are a number of wireless temperature sensors powered by capacity-limited batteries. To conserve energy, sensors switch between active modes and sleeping modes by applying sleeping strategies. We also assume that all sensors in the same room and adjacent rooms can directly communicate with each other via WiFi channel, and there is no sensor failure problem. Figure 1 shows details of the example.

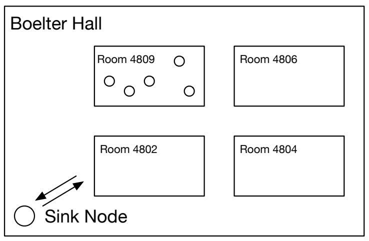


Fig. 1. Boelter Hall Example. Due to space limitation, it only shows 4 rooms and 5 sensors in room 4809. The sensors in Room 4809 can directly talk to all sensors in Room 4809, 4806 and 4802. Sensors generate temperature data every 1 - 8 seconds. There is a sink node which is responsible for collecting sensing data from all sensors in the building every 5 minutes. Data can be transmitted from sensors to the sink node hop by hop.

III. DSSN Design

In this section, we design DSSN for the example to maintain availability of all temperature data at any time, including those generated by sensors in sleeping states. We first generally illustrate the DSSN synchronization process, and then elaborate DSSN's key features for optimization in WSN scenarios.

A. Sensor Grouping

In the example, we split sensors into groups based on their room numbers, for example, sensors in 4809 are in the same group. For two considerations, DSSN only realizes dataset synchronization inside groups, but not in the entire WSN. First, it largely reduces the amount of data a sensor needs to replicate, thus lowering energy consumption by decreasing packet transmissions. Second, per the assumption, sensors in the same group can directly communicate with each other, thus DSSN design does not need to consider network partition problem, or sensors forward others' DSSN packets.

B. Naming and Security

Figure 2 shows group, sensor, and data naming in the example, note that it only shows room 4809 and its 5 sensors. NDN adopts a hierarchical structured and semantically meaningful naming convention. Boelter Hall is named "`/edu/ucla/boelter`", reflecting its affiliation. To uniquely identify each group and ease intra-group communication, each group is named "building name + room number", for example, room/group 4809 is named "`/edu/ucla/bolter/4809`". Same as VectorSync, each sensor names its *Data* with its unique name and a monotonically increasing sequence number, i.e., "sensor name + sequence number". To reflect group affiliation and make name unique, each sensor is manually assigned a unique ID (e.g. A, B, C, D, E), and named "group name + sensor ID". For example, sensor A is named "`/edu/ucla/boelter/4809/A`", and the latest *Data* generated by it is "`/edu/ucla/boelter/4809/A/128`".

Our design model utilizes standard NDN security features. All *Data* packets are signed by sensors at their generation time. All sensors and the sink node have trust relationships established with each other.

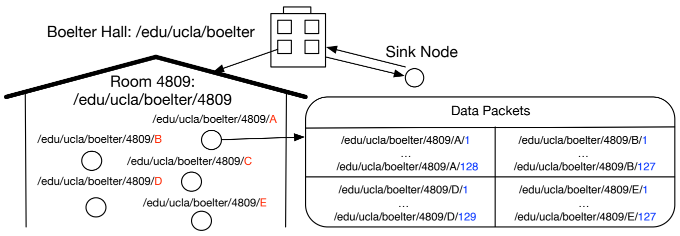

Boelter Hall: /edu/ucla/boelter
Sink Node
Room 4809:
/edu/ucla/boelter/4809
/edu/ucla/boelter/4809/A
/edu/ucla/boelter/4809/B
/edu/ucla/boelter/4809/C
/edu/ucla/boelter/4809/D
/edu/ucla/boelter/4809/E
Data Packets
/edu/ucla/boelter/4809/A/1 ... /edu/ucla/boelter/4809/A/128
/edu/ucla/boelter/4809/B/1 ... /edu/ucla/boelter/4809/B/127
/edu/ucla/boelter/4809/D/1 ... /edu/ucla/boelter/4809/D/129
/edu/ucla/boelter/4809/E/1 ... /edu/ucla/boelter/4809/E/127

Fig. 2. Group, Sensor, and Data Naming in Boelter Hall Example

C. Time to Sync Data

In theory, any active sensor could request for dataset synchronization at any time. However, to minimize packet transmissions and lower energy consumption, an active sensor should do so only before it goes to sleep. First, this guarantees that there is no *Data* stored only in sleeping sensors. Second, as the union of datasets stored on all active sensors covers the entire shared dataset, i.e., data availability is 100%, there is no synchronization needed at other time points.

D. Dataset Synchronization

This section illustrates DSSN from three key design aspects mentioned in Section II-B, and takes sensor sleeping into account. We also show an dataset synchronization example.

1) Group Communication Namespace: In the example, each group has already got a communication namespace "`/edu/ucla/boelter/<room number>`". To ease group communication, every sensor sends and receives *Interests* and *Data* under its group namespace through one-hop multicast, so other sensors in the same group can receive all its packets and it can receive the packets of all the other sensors, in the absence of packet loss. The packet is sent via multicast face, layer 2 send frames using broadcast address, layer 3 multicasts using group name to do packet filtering.

2) Dataset State Representation: Since the sequence number of each sensor's *Data* increments monotonically, we could represent a sensor's knowledge about its group's dataset using a *State Vector*. A *State Vector* is a vector of [sensor ID : sequence number] pairs, where sensor ID represents a sensor in the group, and sequence number is the latest sequence number the *State Vector* owner knows of *Data* generated by that sensor. The [sensor ID : sequence number] pair allows the State Vector to explicitly list group members and distinguish each member in the list, thus avoiding the need to have a separate group membership management mechanism as required in VectorSync [4]. Note that, a sensor has the knowledge about a dataset, i.e., a *State Vector*, does not mean it has fetched all *Data* packets in the dataset, whether and when to fetch a specific *Data* packet is a separate problem. Figure 3 shows an example of the *State Vector* [A:128, B: 127, D: 129, E: 127], which is owned by sensor A in room 4809.

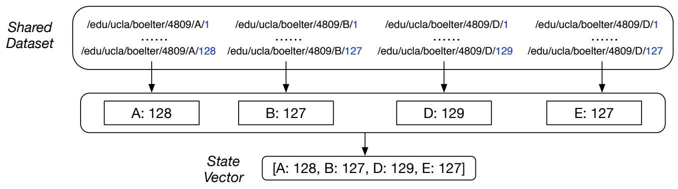

Shared Dataset
/edu/ucla/boelter/4809/A/1
......
/edu/ucla/boelter/4809/A/128
/edu/ucla/boelter/4809/B/1
......
/edu/ucla/boelter/4809/B/127
/edu/ucla/boelter/4809/D/1
......
/edu/ucla/boelter/4809/D/129
/edu/ucla/boelter/4809/D/1
......
/edu/ucla/boelter/4809/D/127
A: 128
B: 127
D: 129
E: 127
State Vector
[A: 128, B: 127, D: 129, E: 127]

Fig. 3. Sensor A's State Vector. Sensor A has the knowledge about *Data* of B, D, E and itself. The latest sequence number sensor A knows of *Data* published by itself is 128, and 127 for B, 129 for D, 127 for E.

3) Dataset Synchronization Mechanism: A sensor requesting to start the synchronization process is called a sync-requester, other active sensors in the same group are sync-responders. When knowledge about the group's dataset is not synced up among all sensors in the group, different sensors may have different *State Vectors*. Motivated by this, the dataset synchronization consists of two steps. First, the sync-requester sends a SYNC *Interest*, which is named "`/[group-name]/sync/[sync-requester-id]/[encoded-sync-requester-state-vector]`". The last component is used to announce the sensor's latest State Vector, so that other active sensors in the group can learn the new *Data* by *deducting* its own State Vectors with the sync-requester's, and update its State Vectors by *merging* the two.[1] See the following example for how to perform *deducting* and *merging*. Second, each sync-responders constructs pending *Interest* list for new *Data* and sends them to fetch *Data*.

4) Dataset Replication Success Notification: As discussed in section III-C, a sync-requester only starts the dataset synchronization process before going to sleep. A sync-requester can only go to sleep after it confirms that its dataset has been replicated successfully. This is achieved when a sync-responder, which has fetched all the missing *Data* from the sync-requester, sends SYNCACK *Interest* "`/[group-name]/syncACK/[sync-requester-id]/[encoded-sync-requester-state-vector]/[sync-responder-id]`"to the sync-requester, and the sync-requester receives the SYNCACK *Interest*.

A sync-requester can go to sleep after receiving the first SYNCACK, ensuring that one sync-responder has fetched all its *Data*, or go to sleep later, ensuring more than one sync-responders have fetched all its *Data*. Those different choices do not affect data availability, because each active sensor does not necessarily contain all *Data* published by the group; as long as the union of active sensors' local dataset covers *Data* of the entire group, including *Data* generated by sleeping sensors, the sink node can always retrieve all *Data* successfully[2]. Based on when, or say, after how many SYNCACK *Interests* are received, the sync-requester goes to sleep, DSSN could achieve different replication levels: replicate its *Data* on one or more active sensors. More replications will bring higher data reliability, but at the same time increase average data replication time. The tradeoff between the two is discussed in Section IV. Figure 4 illustrates the dataset synchronization process.

E. Handling Packet Loss

Packet loss happens frequently in WSN. In our scenario, it is mainly caused by collision[3] and intermittent connectivity.

1) Collision Avoidance: We make use of two types of timers, *delay timer* (DT for short) and *waiting timer* (WT for short), aiming to maintain only one *Interest-Data* exchange round at any given time. DT is a random timer used to delay packet transmission in order to reduce collision rate. WT is set for the sensor to wait for replied *Data* before sending consecutive messages and retransmitting the same *Interest*.

First, each sensor has a DT for each *Interest* or *Data* to be sent out; if the DT times out, it sends out the *Interest* or *Data*. Second, after sending out an Interest, a sensor sets a WT for receiving the corresponding *Data*; if a sensor's WT times out, which means either the *Interest* or the *Data* was lost, it sets a

[1]Per the assumptions, all sync-responders can receive the SYNC *Interest*.

[2]Assume network connectivity can be guaranteed.

[3]Since all packets are transmitted through multicast channel, lower layers cannot handle collision perfectly.

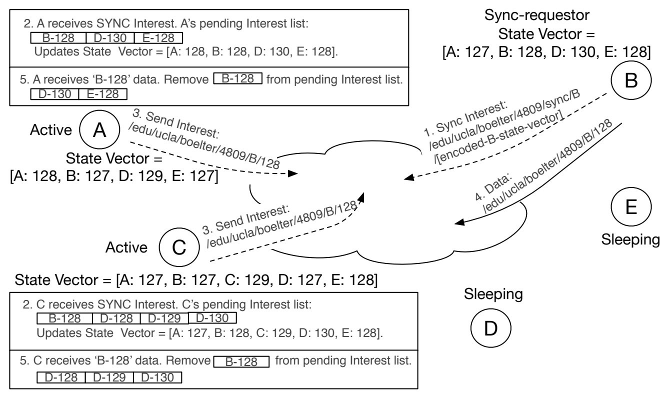

2. A receives SYNC Interest. A's pending Interest list:
B-128 | D-130 | E-128
Updates State Vector = [A: 128, B: 128, D: 130, E: 128].
5. A receives 'B-128' data. Remove B-128 from pending Interest list.
D-130 | E-128
Sync-requestor
State Vector =
[A: 127, B: 128, D: 130, E: 128]
B
Active
A
State Vector =
[A: 128, B: 127, D: 129, E: 127]
3. Send Interest:
/edu/ucla/boelter/4809/B/128
1. Sync Interest:
/edu/ucla/boelter/4809/sync/B
/[encoded-B-state-vector]
4. Data:
/edu/ucla/boelter/4809/B/128
E
Sleeping
Active
C
3. Send Interest:
/edu/ucla/boelter/4809/B/128
State Vector = [A: 127, B: 127, C: 129, D: 127, E: 128]
2. C receives SYNC Interest. C's pending Interest list:
B-128 | D-128 | D-129 | D-130
Updates State Vector = [A: 127, B: 128, C: 129, D: 130, E: 128].
5. C receives 'B-128' data. Remove B-128 from pending Interest list.
D-128 | D-129 | D-130
Sleeping
D

Fig. 4. Data Synchronization Process. There are 5 sensors in room 4809. D and E are sleeping, A, B, and C are active. B is the sync-requester, A and C are sync-responders. Their *State Vectors* are shown in the figure. B first sends SYNC *Interest* named "/edu/ucla/boelter/4809/sync/B/[encoded-B-state-vector]". Upon receiving B's SYNC *Interest*, A and C generate pending *Interest* lists representing the missing *Data* needed to be fetched and then update their *State Vectors*. Next A and C start to send the *Interest* for 'B-128'. After they receive data sent back from B, they remove *Interest* for 'B-128' from pending *Interest* lists and start to fetch the next missing *Data*. When either A (or C)'s pending *Interest* list is empty, it sends a SYNCACK *Interest*: "/edu/ucla/boelter/4809/syncACK/B/[encoded-B-state-vector]/A", to inform B that its *Data* has been synced up successfully. Then B goes to sleep.

DT for its own next packet (either *Interest* retrasmission, see Section III-E2, or a new packet) to be sent out.

Before a sensor's DT or WT times out, if it receives an *Interest* or a *Data*, it should act accordingly, as shown in Figure 5. (1) If the sensor receives an *Interest* from other sensors before its DT or WT times out, which means another sensor initialized a new *Interest-Data* exchange round, it cancels the current DT or WT. If the sensor has the *Data* requested by the *Interest*, it sets a new DT for sending out the requested *Data*. If the sensor does not have the *Data* requested by the *Interest*, it adds the *Interest* in its PIT, and sets a WT for receiving the corresponding *Data*. (By doing this, sensors interested in the same *Data* avoids sending the same *Interest* to fetch the *Data*. We call this *Interest Suppression*). (2) If the sensor receives a *Data* from other sensors before its DT or WT times out, which means another *Interest-Data* exchange round has just finished, it cancels its DT or WT, and sets a new DT for the next *Interest*, entering the next *Interest-Data* exchange round.

Packet Received \ Before Timeout	Delay Timer (DT)	Wait Timer (WT)
Interest	(1) Cancel Current DT or WT (2) Check if it has the Data requested by Interest? Yes: set a new DT for sending out the requested Data No: add Interest in PIT, set a WT for receiving the corresponding Data	
Data	(1) Cancel Current DT or WT (2) Sets a new DT for the next Interest, entering the Interest-Data exchange round.	

Fig. 5. Handling Different Messages before Timer Timeout

2) Recovering from Packet Loss: Collision avoidance mechanisms reduces the probability of collision but not eliminates it; meanwhile, there are other factors leading to packet loss like intermittent connectivity, which may be the result of obstacles located between two sensors, and radio interference from other electronic devices such as microwave oven.

DSSN utilizes *Interest* retransmission mechanism to resolve this problem. NDN adopts the communication model of fetching *Data* by names; it is the consumers' responsibility to ensure that they have received the requested *Data* packets successfully. Therefore, in DSSN design, if a sensor doesn't receive the *Data* it requested for, it needs to retransmit the corresponding *Interest*.

IV. Performance Evaluation

This section uses a prototype implementation to evaluate the performance of DSSN.

A. Implementation and Simulations

The prototype is implemented on ndnSIM (version 2.3) [8]. To implement collision prevention and retransmission mechanisms, we changed NFD packet processing logic.

DSSN is tested with a single sensor group, where the number of sensors ranges from 5 to 15. All sensors communicate with each other directly using IEEE 802.15.4 2.4GHz radio transceivers. Each active sensor publishes *Data* periodically, between 1s to 8s. The average *Data* packet size is 570 bytes and the average *Interest* packet is 274 bytes. To guarantee that sensors generate enough *Data* packets and have enough dataset synchronization rounds, each single simulation lasts 20 minutes. To simulate packet loss, we configure sensors to randomly drop received packets at a pre-configured error rate.

A simple sleeping strategy is adopted. Every 4 seconds, a sensor wakes up and another sensor goes to sleep; the number of active sensors is either 4 (3 sync-responders and 1 sync-requester) or 3 (if the sync-requester goes to sleep). A sync-requester goes to sleep when it receives the number of SYCACK *Interest* it needs, or when it is time for another sensor to wake up. To maintain balanced per-sensor accumulative sleeping time, sensors take turns to wake up and go to sleep, periodically, according to pre-defined orders. With this sleeping strategy, each time a sensor wakes up, it works for 12∼16s (12s for normal working, the reset for replicating data before going to sleep), then goes back to sleep again.

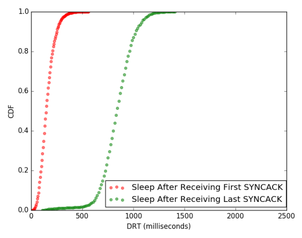
B. Evaluations

1) Data Availability and Data Replication Time Overhead: We define *data availability* (DA) to be the ratio of the number of *Data* packets available in active sensors to the total number of *Data* packets published by the group; *data replication time* (DRT) to be the time a sensor spends on replicating its data before going to sleep, or say, the time between a sensor sends the first SYNC *Interest* and it receives a certain number of SYNCACK *Interests* and then goes to sleep; *data replication time overhead* (DRTO) to be the ratio of a sensor's DRT to its total working time. We explore how DA and DRTO are affected by packet loss rate, group size, and DT threshold[4].

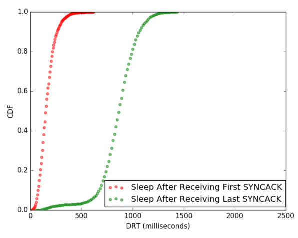
Figure 6 shows the cumulative distribution function (CDF) of DRT under different packet loss rates. The red lines

[4]*DT* threshold: *DT* is determined by a random timed-out number between 0ms to threshold. WT is always set to DT threshold + 3.

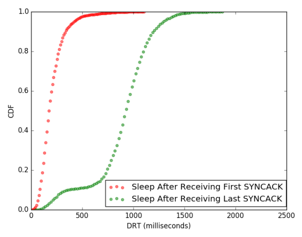
represent DRT when sync-requesters go to sleep after receiving the first SYNCACK, and the green lines represent DRT when sync-requesters go to sleep after receiving the last SYNCACK. They show that, DRT (thus DRTO, as DRTO = DRT / 12 $\sim$ 16) after receiving the last SYNCACK is approximately 5 times of that of after receiving the first SYNCACK. As DRTO is around 1.28% and 6.2% after receiving the first and the last SYNACK respectively, DA is close to 100% in both situations, and there is no sensor failure (per our assumption), in order to lower DRTO by not sacrificing DA (thus saving more energy), sync-requesters should go to sleep after receiving the first SYNCACK. Therefore, in following experiments, sync-requesters go to sleep after receiving the first SYNCACK.



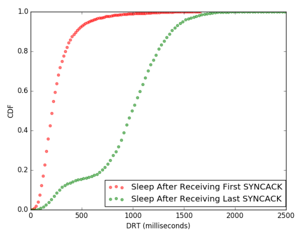
(a) 0% Packet Loss



(b) 1% Packet Loss



(c) 5% Packet Loss



(d) 10% Packet Loss

Fig. 6. SYNCACK Delay under Different Packet Loss Rates (group size = 10, DT threshold = 50ms, WT = 53ms)

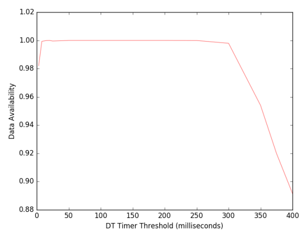
Figure 7 shows DA under different packet loss rates. Even if packet loss rate is 10%, DSSN still achieves nearly 100% DA, meaning that DSSN can handle packet loss well.

Packet Loss	0%	1%	5%	10%
Data Availability	100%	100%	99.9627%	99.9213%

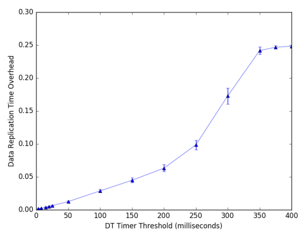
Fig. 7. Data Availability under Packet Loss Rates (group size = 10, DT threshold = 50ms, WT = 53ms, sync-requesters go to sleep after receiving the first SYNCACK)

With packet loss = 0%, DT threshold = 50ms, WT = 53ms, and sync-requesters go to sleep after receiving the first SYNCACK, DA and DRTO are calculated for group size ranging from 5 to 15. Results show that DA remains around 99.94% to 100% for different group sizes. DRTO keeps almost unchanged at 1.28%, because in the sleeping strategy, if a sync-responder is not the newly wake-up sensor, the size of its pending *Interest* list is small and not influenced much by group size, thus it can retrieve all missing data within a relatively stable time period, leading to a stable and small DRTO.

Figure 8 shows DA and DRTO under different DT thresholds. According to Figure 8(a), when DT threshold is smaller than 10ms, DA cannot reach 100%, because packet collision happens a lot, replication of a *Data* packet will fail if maximum retransmission time is reached; when DT threshold is larger than 10ms and smaller than 300ms, DA is nearly 100% as when DT threshold is larger, collision possibility is lower; however, when DT threshold continues to increase, DA decreases again, since average *Interest/Data* exchange round becomes longer, sync-responders can not retrieve all missing data before the sync-requester go to sleep (caused by a new sensor wakes up). Figure 8(b) shows that DRTO increases, starting from lower than 1%, with the DT threshold and finally approaches 25% (25% is the limit of DRTO; if a sync-requester goes to sleep when a new sensor wakes up, then DRTO = 4 / (4 + 4 * 3) = 25%). Based on those analysis, to guarantee DA (close to 100%) while have small DRTO (lower than 3.5%), DT threshold should be 10 $\sim$ 100ms.



(a) Data Availability



(b) Data Replication Time Overhead

Fig. 8. DT Threshold's Influence (group size = 10, WT = DT threshold + 3ms, sync-requesters go to sleep after receiving the first SYNCACK)

Last, we explored the trade-off between DA and DRTO. To this end, we set a *Sync Timer* (ST) for each sync-requester: it requests to sync data, and goes to sleep after its ST times out instead of receiving SYNCACKs. Figure 9 shows when ST's value increases from 50ms to 450ms (DRTO increases by 800%, from 0.415% to 3.61%), DA raises by approximately 3.5%. Therefore, if 100% DA is not required, we can lower energy consumption by not sacrificing much DA.

2) Collision Prevention and Interest Suppression: With group size = 10, packet loss = 0%, and sync-requesters go to sleep after receiving the first SYNCACK, we evaluate our collision prevention and *Interest* suppression mechanisms. To evaluate the performance of DSSN's collision prevention mechanisms, we calculated the retry rate - the ratio of retransmissions to the total packet transmissions. We found that as the *DT* threshold increases from 3ms to 60ms, the retry rate decreases from 30.5% to 2.17%. This shows that a proper DT threshold can greatly reduce packet collision. To evaluate DSSN's *Interest* suppression mechanisms, we calculated the suppression rate - the ratio of the number of suppressed *Interests* to the total of sent-out *Interests* - with DT threshold = 50ms and WT = 53 ms. Results show that *Interest* suppression rate is 24.38%, indicating that DSSN saves many redundant and needless *Interest* transmissions.

C. Summary

Simulation results show that, by adopting collision prevention and *Interest* suppression mechanisms, if sensors go to sleep after receiving the first SYNCACK, DSSN could achieve DA close to 100% with negligible overhead, under different packet loss rates (0%∼10%), group sizes (5∼10), DT thresholds (10ms∼100ms).

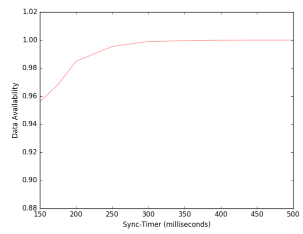


Fig. 9. ST's Influence (group size = 10, DT threshold = 50ms, WT = 53ms, packet loss rate = 0%, sync-requesters go to sleep after STs time out)

V. Future Work

Although the initial design of DSSN has achieved the expected performance, there are still several tasks need to be completed to make DSSN a success. We discuss each issue below, offering a brief description of each challenge, together with proposed ideas to address them.

1) Avoid More Communication Collision: DSSN has collision avoidance mechanisms for its own transmissions, which is proven to work well through experiments. However, DSSN traffic may still collide with other communications, such as DSSN traffic of adjacent groups. We should extend DSSN's design and implementations to include more collision prevention mechanisms for handling those collision problems.

2) Relax Radio Coverage Limitation: For simplicity, DSSN assumes that each sensor is able to directly communicate with all other sensors in the same group, and sensors communicates through one-hop multicast. However, if this assumption doesn't hold true, the existence of sleeping sensors may break network connectivity within groups. One solution is to design a sleeping strategy to make sensors go sleep intellectually without disturbing intra-group network connectivity.

3) Experiment DSSN with Different Sleeping Strategies: As described in Section IV, we have tested DSSN under a simple sleeping strategy. We believe that DSSN should be insensitive to sleeping strategies; that is, it can ensure data availability with negligible overhead under different sleeping strategies. But this needs to be verified through further experimentations.

4) Ensure Higher-level Security: DSSN ensures packet security by signing them at generation time, so receivers can verify them based on application trust models. To protect DSSN from other attacks, further work is needed to enumerate all potential security threats and examine how well NDN's built-in security mechanisms can fence of such attacks.

5) Address Scalability Concern of State Vector: Our experience suggests that *State Vector* can keep dataset synchronized among a group of sensors even under adverse conditions. However, because *State Vector* explicitly lists all data sources in a group, one must address its scalability concern when groups grow large in size. Organizing sensors into a hierarchy of groups seems a simple and promising direction that we plan to explore next.

6) Making Use of Wireless MAC Layer Functions: Previous work [9] have studied the potentials of making use of Layer 2 built-in functions through the dynamic mapping of MAC address to NDN faces, and the benefits of using Layer 2 unicast and error handling. In this paper, our goal is for multiple nodes, each acting as consumer and producer to sync data, thus we make use of Layer 2 broadcast transmission within each group to provide high redundancy. Given wireless channel is broadcast by nature, future Layer 2 designs should also consider better support for broadcast transmission.

VI. Conclusion

This paper presents DSSN, a dataset synchronization protocol over NDN, which ensures data availability for WSN with sleeping sensors. DSSN provides reliable dataset synchronization in face of intermittent sensor availability due to their sleeping cycles. DSSN utilizes a naming convention of numbering each sensor's data by a monotonically increasing sequence number, simplifying the dataset state representation as a compact *State Vector*, which could be used to efficiently synchronize the dataset of a sensor group.

Our simulation evaluation results show that DSSN is able to ensure data availability close to 100% without introducing much network traffic and energy consumption. This work provides insight for NDN Sync protocol design in environments with intermittent connectivity. Because the *State Vector* used in DSSN explicitly lists data sources and their current state, it enables fast and effective dataset state synchronization despite of packet losses or state inconsistency between sensors, a unique advantage over the previous NDN Sync protocol designs. We believe that our experience with DSSN provides a stepping stone into resilient dataset synchronization protocol designs for environments with intermittent connectivity, including ad hoc networking or delay/disruption tolerant scenarios.

Acknowledgment

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

References

[1] L. Zhang, *et al.*, "Named Data Networking (NDN) project," NDN, Technical Report NDN-0001, 2010.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Communication Review*, 2014.

[3] O. Hahm, E. Baccelli, T. Schmidt, M. Wählisch, C. Adjih, and L. Massoulié, "Low-power internet of things with ndn & cooperative caching," in *ACM 4th ACM Conference on Information-Centric Networking*, 2017.

[4] W. Shang, A. Afanasyev, and L. Zhang, "VectorSync: Distributed dataset synchronization over Named Data Networking," NDN, Technical Report NDN-00, 2018.

[5] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," 2000.

[6] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto *et al.*, "NFD Developer's Guide," *TR NDN-0021, University of California, Los Angeles*, 2014.

[7] W. Shang, Y. Yu, L. Wang, A. Afanasyev, and L. Zhang, "A Survey of Distributed Dataset Synchronization in Named Data Networking," NDN, Technical Report NDN-0053, 2017.

[8] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: An open-source simulator for NDN experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 19–33, 2017.

[9] P. Kietzmann, C. Gündoğan, T. C. Schmidt, O. Hahm, and M. Wählisch, "The need for a name to mac address mapping in ndn: Towards quantifying the resource gain," in *ACM 4th Conference on Information-Centric Networking*, 2017.