Resilient Brokerless Publish-Subscribe over NDN

Philipp Moll, Varun Patil, Lixia Zhang
UCLA, Computer Science
{phmoll,varunpatil,lixia}@cs.ucla.edu

Davide Pesavento
Associate, NIST
davide.pesavento@nist.gov

MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM) | 978-1-6654-3956-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/MILCOM52596.2021.9652885

***Abstract*—Publish-subscribe (pub/sub) is a popular API used by today's distributed multiparty applications. TCP/IP, however, does not directly support multiparty communication, therefore realizing pub/sub requires complex logic at the application layer. In this paper, we introduce SVS-PS, a brokerless pub/sub protocol running over NDN. SVS-PS utilizes NDN for data-centric security and many-to-many communication. Compared to IP-based implementations, SVS-PS enables publishers and subscribers to rendezvous "in the air", thereby reduces the complexity of the application layer and lowers network traffic load. Our open-source implementation of SVS-PS makes NDN's networking primitives transparent to applications, allowing developers to work with a familiar pub/sub API while benefiting from NDN's secure and resilient multiparty communication support.**

***Index Terms*—Pub/Sub, Named Data Networking, NDN Transport, Distributed Dataset Synchronization**

I. Introduction

Publish-subscribe (pub/sub) is a popular application programming interface (API) that has been widely used in the development of distributed applications among multiple parties. Today's popular pub/sub implementations run over TCP/IP. Since TCP/IP only supports point-to-point connections, the existing pub/sub deployments either use a central message broker as the rendezvous point that all parties connect to, or set up $N \times N$ TCP connections between all publishers and subscribers. Although each of these two approaches is associated with its own drawbacks (see Section II), they share two commonalities. First, they bring *all* the published data to the application layer, either at the central broker or at each of all the nodes, in order to sort out which piece of published content goes to which subscriber. Second, they secure the TCP connections (e.g., by using Transport Layer Security, TLS) as a substitutive means to authentication of published contents.

Unfortunately, the above practice is unlikely to work well in disaster recovery or battlefield scenarios. Mobility and network partitions can lead to intermittent connectivity, making reliance on TCP connections to deliver all data infeasible. The use of IP multicast alleviates some of the issues, but brings up other ones as we discuss in the next section. Consequently additional protocols, such as the *Bundle protocol* (BP) [1], are introduced to support asynchronous communication. Yet again, not only BP brings into the system its own dependencies such as the Bundle Protocol security support (BPSec) [2], it also does not fit the needs of the wireless edge, which desires to take advantage of wireless broadcast connectivity in support of pub/sub applications. Regrettably, BPSec is another point-to-point security solution as TLS and DTLS.

We believe that all the above issues are due to the same root cause: the conflict between IP networks' node-centric delivery and security, and pub/sub applications' data-centric distribution needs. Today's pub/sub implementations resolve this conflict by bringing *all* published data to application layer to handle through secured, synchronous node-centric connections, a luxury that is not available in scenarios with intermittent connectivity. We postulate that the most promising way to resolve this conflict is to develop pub/sub support over a data-centric network architecture.

In this paper, we describe the design of *State Vector Sync Pub/Sub (SVS-PS)*, a brokerless pub/sub protocol running over NDN. In an NDN-enabled network, data consumers request data by sending Interest packets carrying the names of desired Data, and data publishers name and sign each generated Data packet to cryptographically bind its name to the content. This way, data security stays with each data packet, independent from delivery channels. Interest packets are forwarded based on their names towards corresponding data, and any node with matching Data packets can reply, letting the Data packet reverse the Interests' paths to get back to the consumers. SVS-PS leverages NDN's data-centric design that enables multicast data delivery and data muling, and NDN's built-in security primitives [3] to secure published content directly.

NDN's use of application-layer names to fetch data at the network layer enables publishers and subscribers to rendezvous "in the air". However, it is difficult to build pub/sub services directly on top of NDN's network-layer Interest-Data exchanges. First, NDN Interests are forwarded as datagrams, while pub/sub needs *reliable* many-to-many delivery. Second, one must provide many-to-many transport support which, as we explain in Section IV, requires *transport identifiers* and provides 1:1 mapping between transport identifiers and published contents. In addition, many-to-many data delivery should accommodate heterogeneity of individual parties, especially when those parties are mobile devices which may be hampered by intermittent connectivity or resource shortages.

The contributions of this paper are i) the design of the brokerless pub-sub protocol *SVS-PS*, ii) an open-source library implementation of the protocol, and iii) an initial evaluation of the performance of SVS-PS, together with a comparison with an IP-based pub/sub service that uses epidemic routing as its DTN support. In the rest of the paper, Section II summarizes existing TCP/IP-based and NDN-based pub/sub solutions; Section III describes SVS, on which SVS-PS is built; Sections IV and V elaborate on the design of SVS-PS and its pub/sub

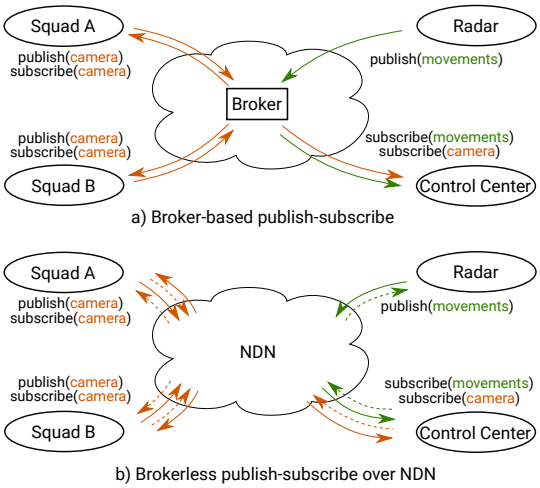

Fig. 1: Broker-based and brokerless pub/sub

API, followed by the evaluation results (Section VI), additional discussions (Section VII), and the conclusion (Section VIII).

II. Background and Related Work

In this section, we briefly discuss the existing IP-based pub/sub frameworks, NDN's basic operations, and the existing NDN-based pub/sub solutions.

A. IP-based Pub/Sub Protocols and Frameworks

IP-based pub/sub implementations can be roughly sorted into three approaches: use of a central message broker, as represented by MQTT, brokerless message queueing, and the use of IP multicast.

MQTT. The Message Queuing Telemetry Transport protocol (MQTT) [4] is among the most popular pub/sub protocols, especially in IoT systems. In MQTT, publishers publish messages classified by topics. Subscribers express their interest in specific topics. A central message broker manages all subscriptions (Fig. 1a). Publishers and subscribers connect to the broker using a pre-configured IP address or hostname. Publishers send all their messages to the broker, which dispatches the messages to subscribers according to their interests.

MQTT's central broker represents a single point of failure, and placing the broker can be challenging, especially when publishers or subscribers may change over time, or when operating in mobile ad-hoc (MANET) environments. MQTT has no security protection but relies on security being added at different layers [5]: running MQTT over VPNs to add network-layer security and/or over TLS channels to add transport-layer security. One could further add application-layer security by providing additional means for participant authentication, access control, payload encryption, and integrity protection. However, there is no coherent framework that coordinates security protections across layers, in particular there is no clear picture on how trust relations are managed to bootstrap all the security protection.

Brokerless Message Queuing. Alternative frameworks, such as ZeroMQ [6], provide pub/sub functionality by setting up $N \times N$ connections instead of using a central broker. In such frameworks, a publisher node creates individual connections to all potential subscribers and selectively sends messages to them based on their interests. Obvious drawbacks of this approach include additional steps needed to discover all subscribers, the overhead from sending the same message multiple times to all interested subscribers, and, similar to broker-based approaches, the reliance on synchronous end-to-end connectivity in the use of TLS for security protection.

IP Multicast based Pub/Sub. IP multicast [7] aims to provide efficient group communication. However, the deployment of IP multicast is hampered by its complexity and usable reliable multicast solutions are yet to be developed. The NORM protocol [8] is one attempt to provide reliable multicast delivery over IP. NORM multicasts UDP packets and lets receivers unicast negative acknowledgments (NACKs) to the packet sender for retransmission. In addition, NORM handles congestion control using an RTT-based approach; the sender rate is set to the rate of the limiting receiver to prevent congestion. Similarly, the Pragmatic Multicast Protocol (PGM) [9] also multicasts UDP packets and achieves reliability using NACKs. However PGM NACKs are multicast to the group to trigger retransmissions from closer hosts. Experimental implementations of NORM and PGM in the ZeroMQ framework support pub/sub on top of these multicast protocols.

B. NDN and Existing NDN Pub/Sub Support

The basic design of NDN consists of three simple ideas: using application layer semantically meaningful names to fetch data at network layer, setting Interest forwarding state and enabling in-network caching, and securing data directly. Not only every data packet is signed by its producer to ensure the authenticity, but also the packet payload can be encrypted so that only authorized parties can access the carried information [10].

One main difference between IP networks and NDN networks is that IP names destination nodes, not packet content. Therefore an IP network cannot make topic-based forwarding decisions; instead applications use IP to forward all requests for data to dedicated message brokers which run *at application layer* to manage message distribution. In NDN, Interest packets carry names that are used for topic-based forwarding decisions *at network layer*, removing the need for message brokers (Fig. 1b).

Existing NDN Pub/Sub Designs. The *NDN-Lite pub/sub* framework provides a pub/sub API with the security support embedded within [11]. However, the NDN-Lite design is specifically tailored to support smart homes and assumes all communications are over broadcast media. Thus, its pub/sub support does not work over multihop network topologies.

PSync is the first protocol designed to support pub/sub in wide-area networks [12]. It names each published data blob B under its publisher' name, concatenated with a sequence number (seq#) to uniquely identify B. PSync uses such blob names as its *transport identifier* to reliably synchronize the dataset names: it encodes the publisher names together with their latest seq#s into an Invertible Bloom Filter (IBF)

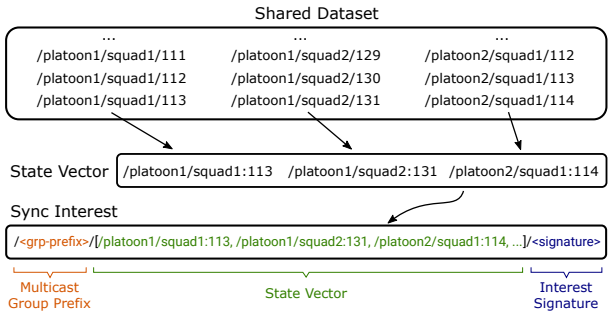

Fig. 2: The relation between sequentially named datasets, State Vectors, and Sync Interests

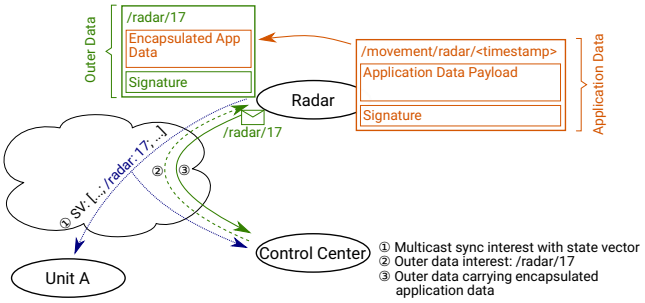

Fig. 3: Encapsulation of Application Data

and distributes the IBF via NDN Interest packets to inform subscribers about data productions; the IBF is used as a means to compress the encoding of latest data productions.

syncps [13] removes PSync's restriction on data naming. It allows published data blobs using arbitrary names. Similar to PSync, syncps also encodes the names of published data blobs into an IBF to distribute to subscribers. Since the IBF must fit into an NDN Interest packet, this limits the size of IBF, which in turn limits the number of data names that can be encoded into an IBF. syncps sets a limit on the time period that each data name will be included in the IBF. Given the dataset covered by the IBF changes over time and syncps does not have a transport identifier for each published data blob, syncps does not guarantee reliable data delivery.

Unlike IP, NDN does not push data from one node to another, but lets consumers fetch data as appropriate. To fetch data, consumers first need to learn what new data names have been produced. PSync solves this problem by naming data using the synchronized sequence numbers, and syncps automatically fetches all published data for all participants in the same group. In our design, this need is fulfilled by the State Vector Sync protocol, which we describe next.

III. State Vector Sync

As a data-centric network architecture, NDN's transport function provides dataset synchronization, dubbed *Sync*, among multiple parties sharing the same dataset, where any of them may produce data at any given time, and multiple may produce simultaneously. NDN Sync protocol designs have evolved over time [14], and the latest design is *State Vector Sync* (SVS) [15].

SVS assumes that every entity has a publishing prefix and publishes data with increasing seq#s under that prefix (similar to PSync). Knowing an entity's publishing prefix and its latest seq# allows enumeration of all its publications; and the ⟨publishing prefix, seq#⟩ tuples of all the entities in a distributed application represent the entire dataset state. This state can be encoded into a data structure called *State Vector* (SV), essentially a list of all ⟨publishing prefix, seq#⟩ tuples as shown in Fig. 2. In order to operate effectively and resiliently in both infrastructure-based and infrastructure-free environments, SVS encodes the SV in the name of an NDN Interest packet, called *Sync Interests*, which are the only message type in SVS. Each Sync Interest is multicast to all the entities sharing the same dataset. Since receiving a single Sync Interest informs one of the entire dataset state, SVS is resilient against losses of individual Sync Interests. SVS signs all Sync Interests to prevent malicious actors from injecting false information into the dataset state.

Sync Interests are sent in two cases: i) event-driven, to notify dataset updates, e.g., when a new publication is produced; and ii) periodically, to mitigate losses of event-driven messages to ensure a consistent view of the dataset state among all the entities. An incoming Sync Interest may also trigger the transmission of a Sync Interest, e.g., if the incoming SV is outdated (at least one sequence number of the incoming SV is lower than the local SV), which means that some entity missed the latest update, likely caused by packet losses or network partitions, thus sending a Sync Interest helps bring other entities state up to date. Infrastructure-free environments may also benefit from latest SV being relayed to nodes that may not have heard previous transmissions due to limited radio range or mobility. We refer interested readers to [15] for a complete description of SVS's operations.

IV. Building Publish-Subscribe over SVS

Although one can use SVS to inform subscribers of latest publications, the existing SVS design can only identify published data items by publishers prefixes and seq#s, therefore a gap exists between transport data identifiers and applications' semantic data names needed by subscribers. This gap is due to different functions provided by different protocol layers. As a transport protocol, SVS uses sequence numbers as loss-resilient data identifiers to achieve reliable delivery. Applications, on the other hand, need semantic names for data production and consumption.

SVS-PS resolves this gap by adopting the approach used in syncps [13], which encapsulates published contents, named and secured by applications, in transport data packets. However, syncps uses an ephemeral IBF value as transport identifier to be carried in its Sync Interests, and sends the encapsulated publications as responses to these Sync Interests. Since the IBF value changes over time and does not offer 1:1 with data publications, syncps cannot guarantee reliable delivery of all publications to all interested subscribers, especially under adverse conditions with significant packet losses or long delays. Large size publications also cause complications when they cannot fit into a single data reply to a Sync Interest.

Based on the lessons learned from syncps and previous

NDN Sync protocols, SVS uses its Sync Interests for publication notifications only. SVS-PS uses the seq# to uniquely identify each published content, and encapsulates the content, named and secured by applications, in data packets named by SVS seq#s (*outer data packets*, one data blob may be segmented to multiple outer packets as needed). Fig. 3 outlines this encapsulation approach. A radar unit with the publishing prefix `/radar` generates movement data with the name `/movement/radar/<timestamp>`. SVS-PS encapsulates this application data in an outer data packet, and multicasts a Sync Interest carrying the new seq# in the SV (see ①). After learning about the new seq#, other nodes may choose to retrieve the outer data packet using standard Interest-Data exchanges (see ② and ③). The subscribers find the original application name by extracting the encapsulated data. Both the encapsulated and the outer data packets are signed by the appropriate keys provided by the application and transport, respectively. Double signing allows executing different security policies of the encapsulated application data and the transport protocol (due to space limit we omit the discussion on SVS-PS security support).

Letting SVS carry encapsulated application data allows publishers to produce data with arbitrary application names. However, subscribers may want to selectively retrieve published content, e.g., matching specific sub-topics, or within certain size limit, especially when they have constrained connectivity or local resources. This requires subscribers to know the published content names before retrieving the outer data packets named by sequence numbers. Our design provides the 1:1 mapping from sequence numbers to application data names, and two complementary ways to retrieve this mapping.

i) Sync Interest Extension: Each data publication triggers a new Sync Interest. This data-triggered Sync Interest can piggyback the mapping between the seq# of the newly produced data and the application data name. This immediately informs subscribers of the application data name, enabling them to decide whether to retrieve the new publication.

ii) Mapping Information Retrieval: Subscribers that join late or do not receive a data-triggered Sync Interest due to packet loss need a way to learn the mapping information. SVS-PS meets this need by allowing subscribers to fetch the mapping names for a given range of seq#s. We note that a round of Interest-Data exchange for retrieving mapping information adds overhead in terms of bandwidth usage and latency.

We also note that, once a subscriber learns the mapping between a seq# and the corresponding data name, it may choose to fetch the published data by using either the seq# or the application data name, assuming the data name can be directly used to fetch the data (i.e., the name/its prefix is in router FIBs, or the Interest packet carries Forwarding Hints).

V. The Pub/Sub API

One of our goals is to make the use of our publish-subscribe protocol as easy as possible, even for NDN beginners. Our API

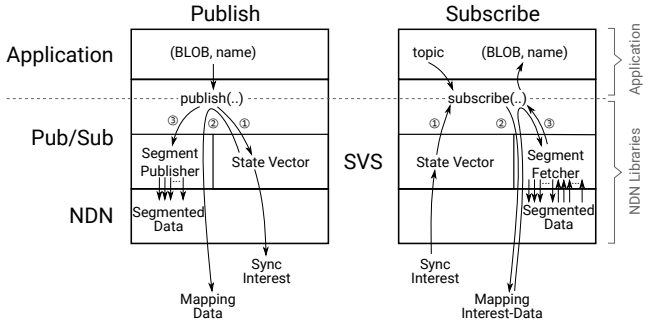

Fig. 4: Interplay of the proposed publish-subscribe protocol with applications and lower-level libraries.

design [16] and implementation[1], as outlined in Fig. 4, hides the use of NDN from application developers while providing the benefits of data-centric communication. Note that the figure and further discussion exclude the security aspects of the protocol for the sake of simplicity and brevity.

Our API provides two interfaces to applications: one to publish arbitrary data as a BLOB under a hierarchical name; the second one allows the application to subscribe to a topic and triggers the API to callback the application whenever corresponding data are received. In the background, entirely transparent to the application, the pub-sub API leverages SVS for the notification of new publications and handles naming, packetization, and congestion control. When creating an instance of the publish-subscribe API, an application-provided multicast group prefix specifies the SVS Sync group (cf. Section III). The pub/sub library joins the Sync group by initializing the underlying SVS module, which allows the application to take part in the group communication.

On the publishing side, SVS broadcasts the new publication's name to all communicating entities (①). At the same time, a mapping from the SVS sequence number to the application name is made available (②). If the actual application data is too large to fit in a single data packet, a segmenter utility packages the BLOB into up to multiple named NDN data packets each within the network MTU limit (③).

On the subscriber side, SVS notifies of a new data item in the dataset and possibly already receives the application data name piggybacked onto the Sync Interest (①). If not already received, the original application data name can be retrieved using a dedicated mapping Interest (②). The application name is matched against existing subscriptions, and only if subscribed, the actual data is retrieved. In that case, a fetcher utility retrieves all parts of the segmented application data, merges them into the original BLOB, and callbacks the application (③). The mapping information can also optionally carry extra information about the publication, such as the size of the data item, which the consumer can use to decide whether to fetch the data.

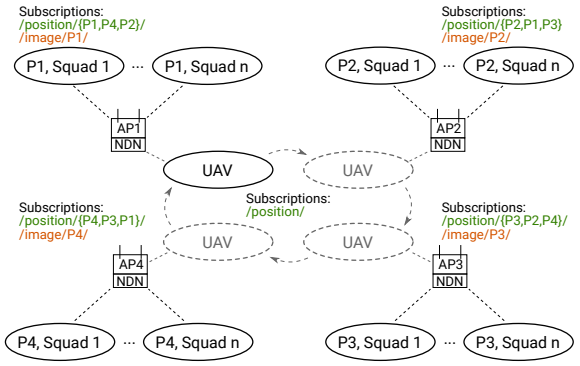

Fig. 5: Evaluation scenario with UAV's acting as data mules to bridge network partitions between platoons in a battlefield.

VI. Evaluation

This section describes the evaluation of our resilient pub-sub protocol, including a comparison to an IP-based DTN protocol. In the evaluation, we test SVS-PS under adverse conditions and evaluate the protocol's reliability in DTN scenarios. Hence, one evaluation metric is the service reliability, measured by the aggregate percentage of messages that are successfully delivered to subscribers interested in those messages. We also evaluate the protocol's bandwidth overhead.

Fig. 5 depicts the evaluation scenario. We assume a battlefield environment with four infantry platoons. Each platoon consists of five squads, each squad is equipped with one communication device. All squads in each platoon are connected to an AP[2]. We assume there is no packet loss between the AP and the squads due to physical proximity, but the bandwidth of each link is capped to 5 Mbps to emulate a resource constrained environment. The communication device at each squad periodically publishes position data and an image from an attached camera. We assume that position information is bundled with other metadata, yielding a total payload of 1 kB, and is shared approximately every 5 seconds by every squad. Image data is only disseminated within each platoon and published at random intervals from 10 to 60 seconds, with the payload size between 30 and 50 kBytes.

The four platoons are isolated from each other and communicate asynchronously via a UAV, which circles between platoons as illustrated in Fig. 5 and connects to each platoon's AP for 30 seconds before moving to the next. The UAV behaves as a data mule carrying messages to subscribers. When using SVS-PS, the UAV subscribes to position data and caches the mapping data of all squads. Squads subscribe to position data of their own platoon and adjacent ones, and to image data of their platoon only. Interests sent to the Sync group prefix are forwarded using the multicast strategy [17].

We use MiniNDN [18] to emulate the described environment. We first evaluate the pub/sub functionality of SVS-PS, with squads publishing data under different topics. We then compare SVS-PS with an implementation[3] of the Bundle Protocol v7 [1] running epidemic routing (ER) over IP. The scripts used for the evaluation are available on GitHub[4].

[1]https://github.com/named-data/ndn-svs

[2]We emulate APs as wired routers connected to all squads in the platoon.

[3]https://github.com/dtn7/dtn7-go, accessed: 2021-08-06

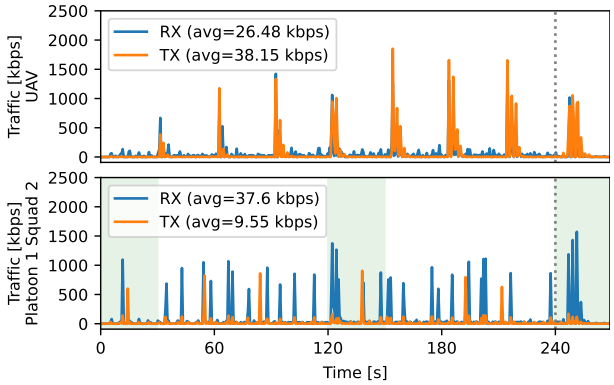

Fig. 6: Bandwidth utilization of the UAV and one representative squad, as observed during the evaluation.

A. Showcasing Delay-Tolerant Publish-Subscribe

In this scenario, we configure a 10% packet loss for the link between each platoon's AP and the UAV (when connected). The UAV mules position data among all platoons by subscribing to the `/position` topic and serving Interests requesting position data. After 4 minutes, all squads stop publishing data but continue reconciliation of all previously published data.

Our evaluation results show 100% reliability for SVS-PS[5]. The theoretical average bandwidth utilization at the UAV to deliver the position data messages to all squads (considering Interest-Data exchanges for application data retrieval only) can be calculated to $\approx$ 32 kbps taking into account interest aggregation at the APs. The measured utilization is $\approx$ 38 kbps (upper chart in Fig. 6), which includes all packets sent by the UAV, and thus also considers the protocol overhead for Sync Interests and mapping exchanges.

Fig. 6 visualizes the bandwidth utilization of the UAV and one of the squads. Shaded areas denote the squad's platoon being connected to the UAV. As soon as the UAV moves to a new platoon, we observe traffic spikes on both the UAV and the squad, caused by the squads and the UAV synchronizing position data[6]. While the UAV is connected, we continue seeing smaller spikes at the squad but not at the UAV. These spikes denote image transmissions within the platoon that are not carried to other platoons by the UAV. The lack of these spikes at the UAV confirms that SVS-PS's name-based subscriptions are handled by the network layer, and image data is not delivered to the UAV, as it is not subscribed to the same.

B. Comparison to Epidemic Routing

While ER entities broadcast data to all other nodes, SVS-PS allows entities to subscribe to a subset of data. To ensure a fair comparison between the two protocols, we drop subscriptions

[4]https://github.com/phylib/svs-pubsub-eval

[5]Since squads keep publishing after the UAV moves away, the UAV requires up to two rounds to carry all data to all platoons after publishing stops.

[6]The spikes may not appear immediately after the UAV moving. Receiving a Sync Interest (which is triggered by squads publishing data) indicates out-of-sync datasets and triggers data exchange.

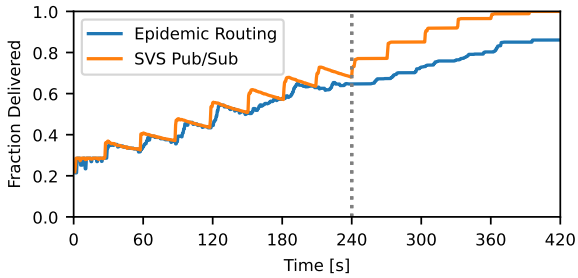

Fig. 7: Progression of successfully delivered data items over time, when comparing ER and SVS-PS.

of image data and only broadcast position data in this scenario. We also reduce the packet loss rate to 5%.

As visualized in Fig. 7, SVS-PS achieves reliable delivery of all data items. Every time the UAV moves to a different platoon, the percentage of delivered data increases. ER, on the other hand, loses messages continuously after some time, resulting in a delivery rate of about 85%. This loss of data is observed because the ER UAV does not have enough time to synchronize all the data with the squads over TCP due to losses and limited channel bandwidth. SVS-PS does not suffer from this issue since the Data packets are multicast to the squads after being received by the AP from the UAV, due to Interest aggregation. Consequently, the average outgoing bandwidth usage at the UAV for ER is $\approx$ 582 kbps, while it is much lower for SVS-PS at $\approx$ 38 kbps.

The superior performance of SVS-PS can also be explained by the use of the underlying SVS protocol. SVS communicates the entire dataset state with a single State Vector (cf. Section III). Receiving the State Vector informs a squad about all missed publications. Publish-subscribe then pulls individual publications either from the UAV or directly from another squad (achieved by multicast Interest forwarding at the network layer). ER, on the other hand, employs a catalog containing the list of publications. This catalog is not sequentially structured, requiring the dissemination to all nodes of the entire list of publications to identify new ones. This causes excessive traffic overhead, as visible in Fig. 8. With SVS-PS, the UAV's bandwidth consumption is low while staying at a platoon, and only increases after moving to another (lower chart). With ER, the UAV's bandwidth utilization is continuously high, with larger spikes after moving (upper chart), indicating the higher overhead due to catalog exchanges.

VII. Discussion

A. Enabling Network Layer to Identify Data

Comparing our NDN-based publish-subscribe protocol to IP-based implementations shows one major difference—the protocol's position in the protocol stack. An NDN name that is used as message topic is a data packet's identifier throughout the entire NDN protocol stack. In contrast, IP packets are identified by addresses, and message topics in IP-based protocols are opaque application-layer payloads. TCP/IP's packet headers include endpoint addresses, not topic names, which does not allow realizing one-to-many message distribution as

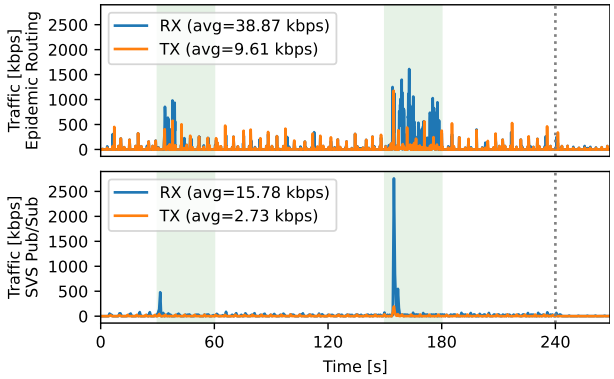

Fig. 8: Network utilization of ER and SVS-PS

required by pub/sub. Hence, topic-based message distribution in TCP/IP has to be implemented at the application layer.

Transport functionality in the the application layer increases the application's complexity, but also limits efficiency considerations. For instance, realizing multicast in the application layer requires redundant unicast of packets to multiple recipients (unless running application-layer code on every hop).

Having the name as packet identifier available on the network level moves the implementation of publish-subscribe to lower layers. For instance, the multicasting of data is an inherent result of multiple subscribers requesting the data using Interest packets forwarded hop-by-hop. NDN Data packets include a signature field and allow payload encryption on a per-packet basis, which allows to secure data transparent to the application layer. Providing these features on lower layers of the stack removes responsibility and complexity from applications, allowing developers to build simpler applications that are inherently more efficient and secure.

B. Comparison with Pub/Sub over IP Multicast

Protocols such as NORM and PGM intend to provide reliable data multicast over IP networks. Pub/sub implementations such as ZeroMQ can run over these protocols in lieu of establishing $N \times N$ TCP connections. In such an implementation, all participants of the pub/sub group must connect to a multicast group and receive all packets sent to the group IP address. Using such a pattern is in direct conflict with publish-subscribe. Since every publication is sent to the group address, IP delivers the publication to all participants regardless of whether they are subscribed to the corresponding topic, and the filtering is done by the receiver at the application layer. As a result, the network may potentially deliver large amounts of data that is not required by the application. In contrast, SVS-PS delivers data only to the consumers that are subscribed to the topic. This is enabled by NDN's naming of individual data packets instead of end hosts, which allows data to be multicast to only the consumers that requested it.

Other existing solutions such as DDS [19] provide data-centric pub/sub over IP by exchanging data between nodes and selectively delivering to subscribed parties. DDS can also use IP multicast to reduce the overhead of such data delivery. Such a design, however, relies on intermediate hosts to selectively

relay data for others, which must happen at the application layer. This intermediate host effectively acts as an ad-hoc broker in the system and creates a central point of failure. NDN performs equivalent tasks at the network layer, thus naturally allowing fault tolerance through the use of multiple forwarding paths without complex application-layer logic. Furthermore, any IP multicast solution still presents the challenges described earlier, and these can only be partially circumvented depending on the network topology and data generation pattern.

C. Data Centric Security

IP-based pub/sub brokers for protocols such as MQTT secure communication between the end hosts and the centralized broker by securing the transport channel. This is typically achieved using point-to-point protocols such as TLS, or CurveZMQ [20], the security protocol used by the distributed pub/sub framework ZeroMQ. However, the point-to-point nature of these protocols prevent them from being used when data is multicast to the receiving entities, requiring applications using IP multicast to implement complex application layer security logic.

NDN provides built-in security primitives for data centric applications. Since every Data packet in NDN carries a signature, the authenticity of received data can be established regardless of the transport channel or path it was received from. This enables SVS-PS to securely muticast data to all subscribers in the group without any application layer logic. Security frameworks for NDN such as the Data-Centric Toolkit [21] also provide easy-to-use abstractions for encryption of data using a secret shared among all participants. Data-centric security also allows for the definition of trust policies and access control rules for various members in the group based on data names [3].

VIII. Conclusion

This paper describes SVS-PS, a brokerless pub/sub protocol over NDN. Unlike IP-based pub/sub implementations that operate on the application layer, NDN supports pub/sub message distribution at the network layer. SVS, one of the NDN Sync protocols, uses seq#s to provide resilient publication notifications to all subscribers. SVS-PS adds application semantics on top of SVS's seq#s to enable subscription-based many-to-many message distribution at network layer. Our evaluation demonstrates SVS-PS's resilience against packet losses and its ability to work asynchronously in DTN scenarios. Our open-source SVS-PS API provides a familiar pub/sub interface that allows exploiting the advantages of data-centric networking while sheltering developers from the low-level NDN networking primitives. We hope that SVS-PS' easy-to-use API with resilient pub/sub support can help foster the use of NDN in tactical scenarios, as well as broader uses in other application areas.

References

[1] S. Burleigh, K. Fall, and E. Birrane, "Bundle Protocol Version 7," Internet Requests for Comments, Delay-Tolerant Networking Working Group, Internet Draft, Jan 2021.

[2] E. J. Birrane and K. McKeever, "Bundle Protocol Security Specification," Internet Engineering Task Force, Internet-Draft draft-ietf-dtn-bpsec-27, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpsec-27

[3] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, "An Overview of Security Support in Named Data Networking," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, November 2018.

[4] mqtt.org, "MQTT: The Standard for IoT Messaging," 2020, accessed: 2021-07-19. [Online]. Available: https://mqtt.org/

[5] HiveMQ GmbH, "MQTT Security Fundamentals," 2021, accessed: 2021-07-19. [Online]. Available: https://www.hivemq.com/mqtt-security-fundamentals/

[6] P. Hintjens, *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc., 2013.

[7] S. Deering, "Host Extensions for IP Multicasting," Internet Standard, RFC 1654, August 1989.

[8] B. Adamson, C. Bormann, M. Handley, and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol," Proposed Standard, RFC 5740, November 2009.

[9] T. Speakman, J. Crowcroft, J. Gemell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano, "PGM Reliable Transport Protocol Specification," Experimental, RFC 3208, December 2001.

[10] Z. Zhang, Y. Yu, R. Kaushik, A. Afanasyev, and L. Zhang, "NAC: Automating Access Control via Named Data," in *2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018.

[11] T. Yu, Z. Zhang, X. Ma, P. Moll, and L. Zhang, "A Pub/Sub API for NDN-Lite with Built-in Security." Named Data Networking, Tech. Rep. NDN-0071, Revision 1, Jan 2021.

[12] W. Shang, A. Gawande, M. Zhang, A. Afanasyev, J. Burke, L. Wang, and L. Zhang, "Publish-subscribe communication in building management systems over named data networking," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019.

[13] K. Nichols, "Lessons Learned Building a Secure Network Measurement Framework Using Basic NDN," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*. ACM, 2019, p. 112–122.

[14] P. Moll, W. Shang, Y. Yu, A. Afanasyev, and L. Zhang, "A Survey of Distributed Dataset Synchronization in Named Data Networking," Named Data Networking, Tech. Rep. NDN-0053, Revision 2, May 2021.

[15] P. Moll, V. Patil, N. Sabharwal, and L. Zhang, "A Brief Introduction to State Vector Sync," Named Data Networking, Tech. Rep. NDN-0073, Revision 2, July 2021.

[16] V. Patil, P. Moll, and L. Zhang, "Supporting Pub/Sub over NDN Sync," in *Proceedings of the 8th ACM Conference on Information-Centric Networking*, ser. ICN'21. ACM, 2021.

[17] A. Afanasyev, J. Shi, B. Zhang, L. Zhang *et al.*, "NDN-0021: NFD Developer's Guide," Named Data Networking, Tech. Rep., 2021, rev. 11.

[18] Mini-NDN Authors, "Mini-NDN: A Mininet-based NDN emulator," 2021, accessed: 2021-07-21. [Online]. Available: minindn.memphis.edu/

[19] omg.org, "Data Distribution Service," 2015, accessed: 2021-10-25. [Online]. Available: https://www.omg.org/spec/DDS/

[20] curvezmq.org, "CurveZMQ - Security for ZeroMQ," accessed: 2021-11-1. [Online]. Available: http://curvezmq.org/

[21] K. Nichols, "Trust Schemas and ICN: Key to Secure IoT," in *Proceedings of the 8th ACM Conference on Information-Centric Networking*. ACM, 2021.