Leveraging Hybrid Information Centric Networking for Broker-Free Publish/Subscribe in IoT

Alvin C. Valera and Duncan E. Cameron
{alvin.valera, duncan.cameron}@ecs.vuw.ac.nz
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand

***Abstract*—In many Internet of Things (IoT) applications, clients are "content-centric" in the sense that they are interested in obtaining content or data regardless of their physical source. This pattern is different from "host-centric" communications wherein clients connect to a particular server to access its resources. The content-centric nature of IoT require a new kind of message delivery support whereby a client can query for data, and obtain the data from any node that has a copy of that data. *Publish/subscribe*, often called pub/sub for short, is a messaging paradigm that can facilitate this new communication pattern. In this paper, we propose Framework for Lightweight Publish-Subscribe (FLIPS), a lightweight topic-based pub/sub for IoT that uses the Hybrid Information-Centric Network (hICN) stack for distributed, broker-free operation. We implemented FLIPS using the open-source hICN transport library and tested its operation through network emulation. Compared with Message Queuing Telemetry Transport (MQTT), FLIPS can provide similar reliability and lower latency especially in scenarios where caching can opportunistically be used.**

I. Introduction

Internet of Things (IoT) applications use the network for retrieving content without regard to the host or physical source of the content [1]. This "content-centric" communication pattern is divergent from the original "host-centric" design of the TCP/IP protocol suite, whereby client applications connect to servers to run applications or submit requests for processing. The incongruity between the design of the Internet and its current use is posing significant difficulties on the underlying TCP/IP architecture [2]. *Publish/subscribe* messaging, commonly known as *pub/sub*, is an emerging paradigm for efficient and scalable data exchange between producers and the consumers [3]. State-of-the-art topic-based pub/sub schemes for the Internet require three distinct entities: (*i*) a *publisher* that posts messages to a certain *topic*; (*ii*) a *subscriber* that indicates topic(s) to subscribe and receives messages posted to its subscribed topic(s); and (*iii*) a *broker* that performs filtering and routing of the messages from publishers to subscribers [4].

The requirement for a dedicated broker is a key drawback of current pub/sub schemes as this can be a single point of failure and source of scalability issues [5]. More importantly, as content-centric applications become more popular and make inroads to home and office environments, it is inconceivable to expect normal home or office users to setup and maintain dedicated broker hosts to support pub/sub. There is therefore a strong imperative to develop scalable broker-free pub/sub schemes to support emerging IoT applications for home, office and industrial settings [5].

In this paper, we introduce Framework for Lightweight Publish-Subscribe (FLIPS), a lightweight publish/subscribe framework which is *distributed* as it does not require the use of dedicated broker hosts for message filtering and routing. Key to the design of FLIPS is the use of the Hybrid Information-Centric Network (hICN) stack [6] to perform filtering, routing and caching of published content. We implemented FLIPS using the open-source hICN library (https://fd.io) and tested its operation through network emulation. Our experiments show that FLIPS can provide similar delivery performance as Message Queuing Telemetry Transport (MQTT) [7] and more importantly, its latency is lower that the latter especially in scenarios where caching can opportunistically be used.

The rest of this paper is organized as follows. In Section II, we present the state-of-the-art in pub/sub over ICN. In Section III, we present the design of the pub/sub framework. In Section IV, we describe how we implemented FLIPS on top of hICN, present the experiments that we have performed and discuss the results that were gathered in the experiments. Finally, we conclude the paper in Section V.

II. Background and Related Work

Hybrid Information-Centric Network [6] is one of the many frameworks that use Information-Centric Networking (ICN) [2]. Its operation is similar in many respects to other ICN stacks such as Named-Data Networking (NDN) [8], [9] and Content-Centric Networking (CCN) [10]. One important difference is that hICN aims to operate alongside IP whereas other ICN stacks espouse a clean slate design or complete replacement of IP with ICN. FLIPS was designed on top of hICN to enable distributed pub/sub over IPv6 which is the network infrastructure of choice for present and future IoT deployments [11], [12].

Applications in ICN explicitly request for a content without regard for the host and at the same time, allowing the use of in-network caches for efficient data dissemination and retrieval [13]. ICN operates using two types of packets, namely *Interest* and *Data* (i.e. content). Every content is given a unique name; a name cannot refer to more than one content. A node that wishes to obtain a content sends an interest packet containing the content name. A node receiving such interest would either (*i*) send back the content if it has the content in its store (or it is the content producer); or (ii) forward the interest using entries in its Forwarding Information Base (FIB). A key principle in ICN, known as *flow balance*, is that an

interest must be satisfied by at most one content reply [14]. Unsatisfied interests are stored in a data structure known as Pending Interest Table (PIT). When an interest is satisfied, it is removed from the PIT. Interests have lifetime: expired unsatisfied interests are likewise expunged from the PIT.

Because content retrieval in ICN is *pull-based*, existing ICN pub/sub schemes employ one or more of following mechanisms to disseminate content from publishers to subscribers: designation of special nodes for storing content [15]–[17]; out-of-band control messages [16]; broker overlay or broker-based [18], [19]; and modification of ICN data structure [15]. Several pub/sub schemes meanwhile use interest packets to disseminate and synchronize published content names [20], [21]. Compared to existing schemes, FLIPS completely adheres to the central tenets of ICN, using only interest and data packets while ensuring distributed broker-free operation.

Many of the ICN pub/sub schemes are broker-free as the ICN layer takes the responsibility of routing and filtering. However, none of them can operate in existing IP-based IoT networks. In IP, multicast has been used to support distributed pub/sub such as in ZeroMQ [22]. A key drawback of using multicast is the potentially high network load as all published content are delivered to all participants regardless of their interests (applications need to perform the filtering). Another IP-based distributed pub/sub is Data Distribution Service (DDS) [23]. One drawback of DDS is the involvement of the application layer in the routing of content to subscribers. DDS is also complex and entails high overhead [24], making it unsuitable for constrained IoT devices [25]. FLIPS's use of hICN makes its operation lightweight, and at the same time, the routing and filtering of content is performed more efficiently at the network layer.

III. FRAMEWORK FOR LIGHTWEIGHT PUBLISH-SUBSCRIBE

The main goal of FLIPS is to enable distributed, broker-free publish/subscribe for IoT applications. We envision FLIPS to be a library which publisher and subscriber applications can use to perform pub/sub functions. The architecture of FLIPS is shown in Fig. 1. To support publishers, FLIPS maintains `publish_table` to store information about published content, `interest_cache` for storing interest packets for future content, and non-volatile `own_content_store` for storing all generated content objects. To support subscribers, FLIPS maintains `subscribe_table` to track subscriptions.

Unlike existing ICN pub/sub schemes, FLIPS is designed to adhere to all ICN principles. This means that all its control and data exchange processes do not violate the principles of ICN through the introduction of special nodes, out-of-band control messages, or special messages that require the modification of the underlying ICN stack.

A. FLIPS *Topic and Content Identifier*

FLIPS is *topic-based*, meaning that every published content must belong to a *topic*, and that subscribers must indicate a topic of interest when initiating a subscription process. A topic

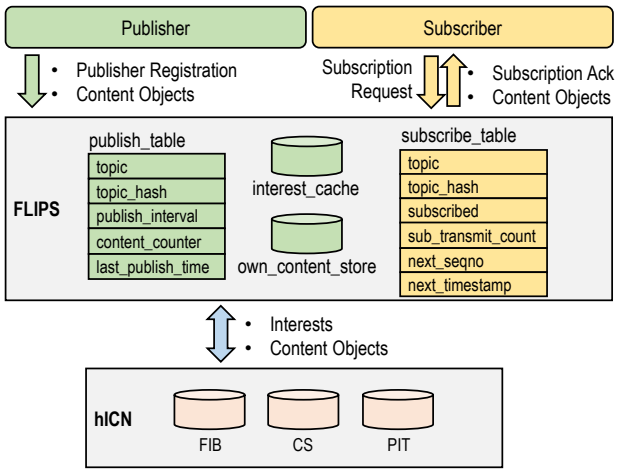


Fig. 1: Framework for Lightweight Publish-Subscribe Architecture.

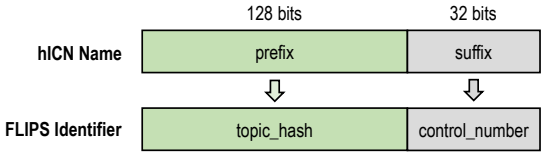


Fig. 2: hICN name and FLIPS content identifier mapping.

is a UTF-8 string of an arbitrary length with a minimum size of one UTF-8 character. Every published content is also uniquely identifiable using a *content identifier* which is a concatenation of *topic hash* and *control number* which we denote by the convention $topic\_hash|control\_number$. We will discuss the details of this convention in the following.

1) Mapping FLIPS *Content Identifier to hICN Name:* To be able to take advantage of hICN routing and caching, we need to map FLIPS content identifier to hICN content name. In hICN over IPv6, every content has a unique name which can be divided into two parts: 128-bit *name prefix* and 32-bit *name suffix*. We employ a simple mapping of prefix to topic hash, and suffix to control number. Fig. 2 illustrates the mapping scheme between FLIPS content identifier and hICN name.

2) Topic Hash: To convert the topic string into a 128-bit number, we use the MD5 hash algorithm. Because the hICN name prefix (hence the topic hash) needs to be a valid routable IPv6 address, we replace the first two bytes of the hash with `FLIPS_CODE` which we arbitrarily set to `0xabcd` to ensure that the hash will not be in the range of IANA's special-purpose IPv6 addresses [26]. Moreover, as hashing can result in different topic strings resolving to the same 128-bit hash value, we employ a topic collision detection process during publisher registration (see Section III-E).

3) Control Number: The 32-bit control number portion of the identifier is used to identify different types of messages. Table I enumerates the usage of the control number. Published content objects are assigned sequence numbers in the range starting from 100 until 1500000000, and which should wrap

back to 100 in case the latter is exceeded.

B. FLIPS Messages

As mentioned in Section II, hICN only supports two types of packets, namely, interest and data. FLIPS uses several message types to accomplish its operational goals, working within the hICN constraints by encapsulating the messages in either an interest or data packet. We use the following conventions to denote messages:

- INTEREST$\{n, [p]\}$: An interest packet requesting for a content named n carrying an optional payload p.
- DATA$\{n, [p]\}$: A data packet for a content named n, optional payload p carries the content, if any.

1) Topic Hash Collision Detection: To determine if a topic hash is already in use, a node sends a **topic hash collision detection request** (THCD_REQ). THCD_REQ is sent as INTEREST$\{topic_hash|0, topic\}$, where $topic\_hash$ is the topic hash to be resolved, and the payload $topic$ is the topic string. This packet tells receivers that the interest is meant for detecting hash collisions since $control\_number$ is 0. A node that detects a hash collision must reply with a **topic hash collision detection reply** (THCD_REP). This is essentially a data packet DATA$\{topic_hash|0\}$.

2) Application Content Objects: To request for content objects generated by publishers, a node sends a **published content request** (CONTENT_REQ). This is sent as INTEREST$\{topic_hash|seqno\}$ where $seqno$ is the content sequence number. This interest is satisfied by a **published content reply** (CONTENT_REP) which is essentially a data packet that matches the name $topic\_hash|seqno$, i.e., DATA$\{topic_hash|seqno, p\}$. Moreover, the payload p is formatted to contains the following fields: (*i*) $next\_seqno$: sequence number of next data to be generated under this topic; (*ii*) $next\_timestamp$: the time at which the data will become available, allowing subscribers to time the transmission of subsequent CONTENT_REQ packets; and (*iii*) $app\_content$: the application generated content.

3) Subscription Initiation: Subscription initiation essentially involves the discovery of the latest content published under the topic, to enable subscribers to follow the content stream. To perform this, a node sends **subscription request** (SUB_REQ). This is sent as INTEREST$\{topic_hash|ts\}$ where $ts \geq 1500000001$ is the current timestamp. When a publisher generating content under $topic\_hash$ receives this packet, it sends back the latest generated content as CONTENT_REP, giving it an alias $topic\_hash|ts$. This results in a data packet DATA$\{topic_hash|ts, p\}$. Since the payload p contains information about the next sequence number, the subscriber will now be able to follow the content stream under the topic.

TABLE I: FLIPS Identifier Control Numbers

Control number value(s)	Usage
0	Topic hash collision detection
1 – 99	Reserved
100 – 1500000000	Published content sequence numbers
1500000001 – $(2^{32} - 1)$	Subscription initiation (Unix timestamp)

C. Data Structures

FLIPS maintains appropriate data structures at the publisher and subscriber nodes. For subscribers, a table named `subscribe_table` is maintained to keep track of subscriptions. Every entry in this table has the following fields: (*i*) $topic$: UTF-8 topic string; (*ii*) $topic\_hash$: 128-bit MD5 hash of $topic$ string; (*iii*) $subscribed$: a flag to indicate whether subscription is active; (*iv*) $sub\_transmit\_count$: number of times subscription interest packet was transmitted; (*v*) $next\_seqno$: full name of next data to be generated under this topic; and (*vi*) $next\_timestamp$: time at which data will become available.

Meanwhile, a table named `publish_table` is maintained to store information regarding publications. Each entry in this table has the following attributes: (*i*) $topic$: UTF-8 topic string; (*ii*) $topic\_hash$: 128-bit MD5 hash of $topic$ string; (*iii*) $publish\_interval$: interval at which content are going to be published; (*iv*) $content\_counter$: counter incremented every time a content is published; and (*v*) $last\_publish\_time$: timestamp of last content publication.

A volatile `interest_cache` is maintained to store interest packets for future content, and non-volatile `own_content_store` to hold payload of all published content. Note that the latter is different from the hICN content store. It is needed to ensure request for any previous content already expunged from network caches can be satisfied.

D. Subscription Process

Algorithm 1 lists the functions used in the subscription process. To initiate subscription, the application calls SUBSCRIBE() with $topic$ as parameter. Note that the function MD5_HASH() computes MD5 hash and replaces the first two bytes with `FLIPS_CODE` to ensure the hash is a valid routable IPv6 address. Now, if the topic already exists in `subscribe_table`, the process immediately terminates because either the topic is already subscribed or there is a subscription process in progress. After creating and initializing an entry in `subscribe_table`, FLIPS composes a SUB_REQ packet, i.e., INTEREST$\{topic_hash|ts\}$, and invokes HICN_SEND() to send it. The packet is handled by the hICN stack accordingly: if it can be satisfied by the local cache, then FLIPS should receive a corresponding CONTENT_REP packet. If no matching content is available in the local and network caches, the packet eventually reaches a publisher node.

When FLIPS at the publisher node receives SUB_REQ, it composes a CONTENT_REP, i.e., DATA$\{topic_hash|ts, p\}$, using the latest published content as the payload p and sends it back as the reply. CONTENT_REP should propagate back to the subscriber node, with intermediate hICN nodes saving a copy of the packet in their respective caches. When subscriber receives the packet, the function ON_DATA() is invoked. The subscriber is now aware of the sequence number and timestamp of the next content, and it uses these information in SCHED_DATA_REQUEST() to schedule the sending of CONTENT_REQ for the next data in the stream T seconds

Algorithm 1: Subscription Process

```
function SUBSCRIBE(topic)
    topic_hash ← MD5_HASH(topic)
    if topic_hash ∉ subscribe_table then
        e ← new subscribe_table entry
        e.topic ← topic
        e.topic_hash ← topic_hash
        e.subscribed ← false
        e.sub_transmit_count ← 0
        ts ← Unix timestamp
        HICN_SEND(INTEREST{topic_hash|ts})
        START_TIMER(SUB_TIMEOUT)
    end if
end function

function ON_DATA(DATA{topic_hash|ts, p})
    CANCEL_TIMER(SUB_TIMEOUT)
    e ← subscribe_table entry with topic_hash
    e.subscribed ← true
    e.next_seqno ← p.next_seqno
    e.next_timestamp ← p.next_timestamp
    SCHED_DATA_REQUEST(topic_hash)
    notify application of received data
end function

function ON_SUB_TIMEOUT(topic_hash)
    e ← subscribe_table entry with topic_hash
    if e.sub_transmit_count < SUB_MAX_TRIES then
        ts ← Unix timestamp
        HICN_SEND(INTEREST{topic_hash|ts})
        INCREMENT(e.sub_transmit_count)
        START_TIMER(SUB_TIMEOUT)
    else
        delete e from subscribe_table
        notify application of subscription failure
    end if
end function

function SCHED_DATA_REQUEST(topic_hash)
    e ← subscribe_table entry with topic_hash
    at time e.next_timestamp − T:
    HICN_SEND(INTEREST{topic_hash|e.next_seqno})
end function
```

prior to the actual generation time of the content. FLIPS also notifies the application about the received content.

If transmission of the SUB_REQ fails, i.e., no corresponding CONTENT_REP is received within the specified timeout period, the function ON_SUB_TIMEOUT() gets invoked. If the number of attempts is not yet exhausted, the subscriber sends a new SUB_REQ and updates `subscribe_table` accordingly.

E. Publication Process

Algorithm 2 lists the functions involved in the publication process. Before an application can begin publishing content under a topic, it must first call REGISTER() with *topic* and *interval* as parameters. Publishers that generate content at unknown time intervals can set the latter to 0. This creates a new entry in `publish_table` if no corresponding entry exists yet. FLIPS then sends a THCD_REQ packet (INTEREST{*topic_hash*|0, *topic*}) to commence the topic hash collision detection.

1) Detecting Hash Collisions: In dealing with topic hash collisions, FLIPS adopts a *lazy* approach whereby it simply notifies the application if a collision is detected. As the probability of hash collisions is extremely low ($1/2^{112}$ since 16 bits are fixed), the cost of performing collision resolution significantly outweighs the benefits. As mentioned, collision detection is initiated in the call to REGISTER() through the sending of THCD_REQ. When a publisher using *topic_hash* receives this packet, ON_THCD_REQ() gets invoked. The topic string in the payload of THCD_REQ is compared with the topic string in `publish_table`. If the strings do not match, then a collision exists, requiring this publisher to send a THCD_REP (DATA{*topic_hash*|0}). When the sender receives THCD_REP, the function ON_THCD_REP() is called. The receipt of THCD_REP essentially means that *topic_hash* is already in use and as already mentioned, FLIPS simply notifies the application of this collision. Note that the absence of any THCD_REP means that no other publisher is using *topic_hash*. Thus when the `THCD_TIMEOUT` timer expires, FLIPS deems the registration successful and proceeds to call HICN_REGISTER(*topic_hash*) to instruct hICN to notify FLIPS when it receives any interest for *topic_hash*.

2) Content Publication: Content publication is straightforward but slightly nuanced. hICN does not allow active pushing of data without a corresponding interest. As such, FLIPS simply stores published content, i.e., data passed using the PUBLISH() call, in `own_content_store`.

When FLIPS receives an interest packet from hICN, it searches the store for the corresponding content. If found, the content is passed to hICN for eventual transmission to the subscriber. Note that once the content is cached by hICN, the publisher should stop receiving interests for the same content. However, for reliability, the publisher keeps a copy of the content in its `own_content_store`.

IV. IMPLEMENTATION AND EXPERIMENTS

We developed a reference implementation of FLIPS, as well as publisher and subscriber applications, using the hICN C++ transport library (version 20.09), an open-source library available in the Linux Foundation Fast Data project (https://fd.io).

A. Experiment Settings

We conducted experiments using Common Open Research Emulator (CORE) [27] (version 8.2.0) with Extendable Mobile Ad-hoc Network Emulator (EMANE version 1.3.3). CORE is a network emulator developed by the U.S. Naval Research Laboratory and Boeing supporting real-time emulation of virtual networks for protocol research, demonstration, application and platform testing [28]. We compared the performance of our FLIPS reference implementation against MQTT version 3.1 (using the Eclipse Mosquitto implementation), a popular lightweight topic-based pub/sub protocol for IoT.

1) Topology and Configuration: We used an IEEE 802.11g wireless network which consisted consisted of 21 hosts (20 subscribers and 1 publisher). For MQTT experiments, we added an additional host to act as a dedicated broker. The hICN

Algorithm 2: Publication Process

function REGISTER(*topic*, *interval*)
  topic_hash ← MD5_HASH(*topic*)
  if *topic_hash* ∉ `publish_table` **then**
    e ← **new** `publish_table` **entry**
    e.topic ← *topic*
    e.topic_hash ← *topic_hash*
    e.publish_interval ← *interval*
    e.content_counter ← 0
    HICN_SEND(INTEREST{*topic_hash*|0, *topic*})
    START_TIMER(`THCD_TIMEOUT`)
  end if
end function

function ON_THCD_REQ(INTEREST{*topic_hash*|0, *topic*})
  e ← `publish_table` **entry with** *topic_hash*
  if *e.topic* ≠ *topic* **then**
    HICN_SEND(DATA{*topic_hash*|0})
  end if
end function

function ON_THCD_REP(DATA{*topic_hash*|0})
  CANCEL_TIMER(`THCD_TIMEOUT`)
  e ← `publish_table` entry with *topic_hash*
  delete e from `subscribe_table`
  notify application of topic hash collision
end function

function ON_THCD_TIMEOUT(*topic_hash*)
  HICN_REGISTER(*topic_hash*)
  notify application of registration success
end function

function PUBLISH(*topic*, *content*)
  e ← `publish_table` entry with *topic*
  if *e* does not exist **then**
    notify application of publication failure
    return
  end if
  p ← **new** payload object
  t ← *e.topic_hash*
  s ← 100 + (*e.content_counter* mod 1500000000)
  ts ← Unix timestamp
  p.next_seqno ← *s* + 1
  p.next_timestamp ← *ts* + *e.publish_interval*
  p.app_content ← *content*
  INCREMENT(*e.content_counter*)
  store DATA{*t*|*s*, *p*} **in** `own_content_store`
  if INTEREST{*t*|*s*} ∈ `interest_cache` **then**
    HICN_SEND(DATA{*t*|*s*, *p*})
  end if
end function

function ON_CONTENT_REQ(INTEREST{*n*})
  if *n* ∈ `own_content_store` **then**
    d ← Data named *n* **from** `own_content_store`
    HICN_SEND(*d*)
  else
    store INTEREST{*n*} **in** `interest_cache`
  end if
end function

light daemon (version 20.09), a lightweight implementation of hICN also available through the Fast Data project, was used to provide hICN functionality. The daemon was executed in each of the hosts using statically configured routes, along with either the publisher or subscriber program implementation.

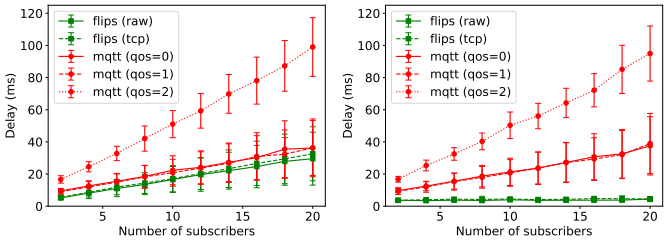


Fig. 3: Data delivery latency as a function of number of subscribers at publish interval of 5 s: (a) subscriber applications running at different hosts; and (b) subscriber applications running on a single host.

Table II lists the emulation parameters used in the experiments.

2) Performance Metrics: The key performance metrics of interest are *data delivery rate*, *data delivery latency*, and *normalized control overhead*. Data delivery rate is the fraction of data received, while data delivery latency is measured from the data generation time until its delivery to a subscriber. Finally, normalized control overhead is the number of control messages sent for every data. To obtain these metrics, we performed experiments wherein a number of subscribers would simultaneously subscribe to a topic and wait for the arrival of 100 data packets.

B. Data Delivery Rate

In all experiments, both FLIPS (using raw and TCP transports) and MQTT (all QoS levels) attained 100% data delivery regardless of the number of subscribers and publish intervals. This result is expected, given that our pub/sub implementation performs retransmission of timed out interests. We however observed the impact of simultaneous subscriptions in terms of higher data delivery latency.

C. Data Delivery Latency

Latency is an important metric is many applications [29]. For mission-critical applications (e.g. intrusion detection, fire monitoring), data must reach the subscribers as fast as possible to avoid undesirable consequences. Fig. 3 shows the data delivery latency as a function of the number of subscribers at publish interval of 5 s. In Fig. 3a, the subscriber applications executed at different hosts (1 subscriber per host) whereas in Fig. 3b, the subscriber applications executed at a single host.

TABLE II: Emulation parameters

Parameter	Value
Publisher payload size	1,350 bytes
Publisher content generation intervals	1–10 seconds
Number of subscribers	{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
MQTT QoS	0, 1, and 2
MQTT Broker	mosquitto
FLIPS transport	Raw and TCP
hICN Interest lifetime and timeout	2,000 ms
hICN Interest packet size	64 bytes
hICN Content object lifetime	50,000 ms
IEEE 802.11g data rate	54 Mbps

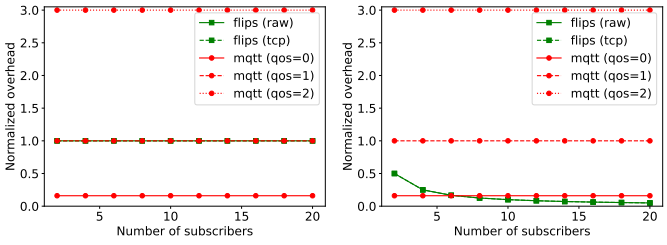
Let us first focus on the scenario where subscriber applications executed at different hosts. Here, we observed that the latency (and standard deviation as indicated by the error bars) of all the schemes linearly increased as the number of subscribers increased due to the effect of higher contention at higher number of subscribers. FLIPS (using raw and TCP transports) showed the lowest latency in all cases, as it edged MQTT (at QoS level 0 and 1) by around 5 ms. Note that at at low number of subscribers, this gain is significant. The lower latency of FLIPS is due to the lower number of links that content needs to traverse from publisher to subscribers. In MQTT, content needs to traverse links from publisher to broker, then from broker to subscribers whereas in FLIPS, content only needs to traverse links from publisher to subscribers. The significantly higher latency of MQTT at QoS level 2 is due to its more sophisticated handshaking for transferring content. Note that our MQTT results are consistent with earlier results [30], [31].

One key advantage of FLIPS is that the underlying hICN layer performs caching. Thus, content gets distributed to caches closer to subscribers, which should reduce latency as subsequent interests could be satisfied by nearby caches. To demonstrate this, we performed experiments where the subscriber applications executed at a single host. This scenario also mimics cases where publishers and subscribers are located in different networks and that the latency between the networks is much more significant than the latency within the local network. The results, as shown in Fig. 3b clearly show the dramatic impact of caching. While MQTT at all QoS levels showed the same latency as in the first scenario, FLIPS showed consistent and substantially lower latency. This is because in FLIPS, only the first copy of the request and content traverses the network. Subsequent requests use the cached copy, consequently reducing latency and network congestion.

Before discussing the rest of the results, we note that in our experiments, we did not observe any significant difference between the use of raw and TCP transports in FLIPS. This was due to the insignificant packet loss in all the test cases. We observed that when packet loss occurred, the latency of raw increased significantly to a multiple of the interest timeout value. This was because when using raw, FLIPS handled packet loss detection which took one interest timeout duration at the very least. This was not the case with TCP as it detected packet loss quickly and successfully completed retransmissions in a shorter amount of time.

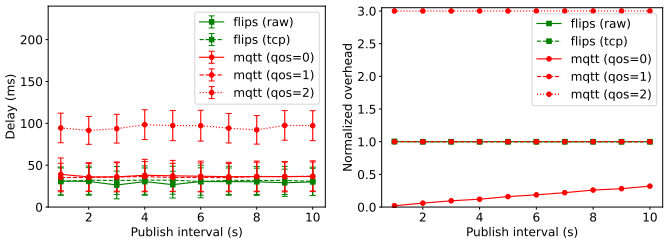
D. Normalized Control Overhead

Control overhead is an important metric as it indicates the additional load incurred by the pub/sub scheme on the network and in ideal situations, we want overhead to be as low as possible. Fig. 4 shows the normalized control overhead as a function of number of subscribers at publish interval of 5 s. In the first scenario where subscriber applications executed at different hosts (see Fig. 3a), we observed that the overhead did not change with respect to the number of subscribers for all schemes. The best performance was obtained by MQTT at



(a) (b)

Fig. 4: Normalized control overhead as a function of number of subscribers at publish interval of 5 s: (a) subscriber applications running at different hosts; and (b) subscriber applications running on a single host.



(a) Latency (b) Control overhead

Fig. 5: Latency and control overhead as a function of publish intervals.

QoS level 0 as it generated less than 0.2 control packets per content. This was mainly due to the infrequent transmission of `PINGREQ` and `PINGRESP` packets for keeping the session between subscribers and broker alive. Meanwhile both FLIPS and MQTT at Qos level 1 generated around 1 control packet per content. For FLIPS, this was due to the transmission of an interest packet to request for every content, whereas for MQTT, this was due to `PUBACK` packet sent for every content received by subscribers. Finally, MQTT at QoS level 2 generated the most overhead at 3 control packets for every content as at this level, the protocol uses `PUBREC`, `PUBREL` and `PUBCOMP` for every content.

In the second scenario where the subscriber applications executed at a single host, we observed that MQTT at all QoS levels showed the same flat overhead as the number of subscribers increased. Whereas, FLIPS showed a decreasing overhead as the number of subscribers increased. This was due to the fact that only the first copy incurs control overhead over the network.

E. Effect of Publish Intervals

To investigate the effect of publish intervals, we performed experiments where we varied the publish interval from 1–10 s and fixed the number of subscribers to 20. Fig. 5 shows the latency and control overhead as a function of publish interval. We did not observe any significant dependence between latency and publish interval, as all schemes showed the same delay in all cases. This is because even in the most stressful case of 1 s publish interval, the amount of traffic transferred in the network was still below the network capacity (at 1 s publish interval, around $1{,}350 \times 8 \times 20 = 216{,}000$ bps).

Indeed, we will need to substantially decrease the interval to unrealistic values in order to saturate the network and observe any effect. In terms of the control overhead, we observed that all the schemes except for MQTT at QoS level 0 showed the same performance. MQTT at QoS level 0 showed a linear increase as the interval increased. This was expected since as the interval increased, fewer content were generated, resulting in the normalized overhead to increase.

V. Conclusion and Future Work

Publish-subscribe messaging is gaining popularity in the Internet of Things. In this paper, we proposed FLIPS, a broker-free, lightweight and distributed pub/sub scheme using the Hybrid Information-Centric Network stack. Unlike existing pub/sub schemes that have been proposed for ICN, FLIPS completely adheres to the central tenets of ICN which is to use interest and data packets alone. We implemented FLIPS on top of hICN transport library and performed experiments in CORE to evaluate and compare its performance. We found that FLIPS can provide reliable data delivery. Compared with MQTT, FLIPS showed lower latency especially in scenarios where caches were used. In terms of overhead, FLIPS has slightly higher overhead than MQTT at QoS level 0 as it needs to send an interest packet for every data.

This research has opened up several areas for future investigation. We have identified several key research problems that need to be examined including: (*i*) the performance of FLIPS in pure ICN settings such as NDN, (*ii*) algorithm design for reliable low-latency subscriber for event-driven publishers and (*iii*) design of optimal adaptive interest timeout in high loss or congestion networks.

References

[1] O. Waltari and J. Kangasharju, "Content-centric networking in the internet of things," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016, pp. 73–78.

[2] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[3] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable iot communications," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 48–54, 2015.

[4] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7.

[5] P. Bellavista, A. Corradi, and A. Reale, "Quality of service in wide scale publish—subscribe systems," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1591–1616, 2014.

[6] G. Carofiglio, L. Muscariello, J. Augé, M. Papalini, M. Sardara, and A. Compagno, "Enabling icn in the internet protocol: analysis and evaluation of the hybrid-icn architecture," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, 2019, pp. 55–66.

[7] "MQTT: The standard for iot messaging," https://mqtt.org/, accessed: 1 November 2022.

[8] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[9] X. Tan, Z. Zhao, Y. Cheng, and J. Su, "Flow-based ndn architecture," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[10] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves, "Content-centric networking," *Whitepaper, Palo Alto Research Center*, pp. 2–4, 2007.

[11] S. Ziegler, C. Crettaz, L. Ladid, S. Krco, B. Pokric, A. F. Skarmeta, A. Jara, W. Kastner, and M. Jung, "Iot6–moving to an ipv6-based future iot," in *The Future Internet Assembly*. Springer, 2013, pp. 161–172.

[12] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[13] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[14] B. Wissingh, C. A. Wood, A. Afanasyev, L. Zhang, D. Oran, and C. F. Tschudin, "Information-centric networking (icn): Content-centric networking (ccnx) and named data networking (ndn) terminology." *RFC*, vol. 8793, pp. 1–17, 2020.

[15] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. Ramakrishnan, "Copss: An efficient content oriented publish/subscribe system," in *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. IEEE, 2011, pp. 99–110.

[16] C. Gündoğan, P. Kietzmann, T. C. Schmidt, and M. Wählisch, "Hopp: Robust and resilient publish-subscribe for an information-centric internet of things," in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. IEEE, 2018, pp. 331–334.

[17] H. Kim and N. Ko, "A scalable pub/sub system for ndn," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2020, pp. 1067–1069.

[18] S. Han and H. Woo, "Ndn-based pub/sub system for scalable iot cloud," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016, pp. 488–491.

[19] J. Heeyoung, C. Kangil, K. Haksuh, and K. Sunme, "A networking scheme for large-scale pub/sub service over ndn," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2019, pp. 1195–1200.

[20] M. Zhang, V. Lehman, and L. Wang, "Scalable name-based data synchronization for named data networking," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[21] P. Moll, V. Patil, L. Zhang, and D. Pesavento, "Resilient brokerless publish-subscribe over ndn," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 438–444.

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

[23] "Data distribution service," https://www.omg.org/omg-dds-portal/, accessed: 1 November 2022.

[24] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "Opc ua versus ros, dds, and mqtt: performance evaluation of industry 4.0 protocols," in *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2019, pp. 955–962.

[25] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, "Investigating messaging protocols for the internet of things (iot)," *IEEE Access*, vol. 8, pp. 94 880–94 911, 2020.

[26] R. Bonica, M. Cotton, B. Haberman, and L. Vegoda. Updates to the special-purpose ip address registries, rfc 8190. [Online]. Available: https://www.rfc-editor.org/info/rfc8190

[27] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *MILCOM 2008-2008 IEEE Military Communications Conference*. IEEE, 2008, pp. 1–7.

[28] S. Herrnleben, R. Ailabouni, J. Grohmann, T. Prantl, C. Krupitzer, and S. Kounev, "An iot network emulator for analyzing the influence of varying network quality," in *International Conference on Simulation Tools and Techniques*. Springer, 2020, pp. 580–599.

[29] N. A. Mohammed, A. M. Mansoor, and R. B. Ahmad, "Mission-critical machine-type communication: An overview and perspectives towards 5g," *IEEE Access*, vol. 7, pp. 127 198–127 216, 2019.

[30] S. Lee, H. Kim, D.-k. Hong, and H. Ju, "Correlation analysis of mqtt loss and delay according to qos level," in *The International Conference on Information Networking 2013 (ICOIN)*. IEEE, 2013, pp. 714–717.

[31] D. Borsatti, W. Cerroni, F. Tonini, and C. Raffaelli, "From iot to cloud: applications and performance of the mqtt protocol," in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2020, pp. 1–4.