ICAN: Information-Centric Context-Aware Ad-Hoc Network

Yu-Ting Yu, Chris Tandiono, Xiao Li, You Lu, M.Y. Sanadidi, and Mario Gerla
University of California
Los Angeles, CA 90095, USA
{yutingyu, cbtandiono, pololee, youlu, medy, gerla}@cs.ucla.edu

Abstract—Recently, Information-Centric Network (ICN) has been attracting much attention with its promising future as next-generation Internet architecture. While ICN is scalable and efficient in Internet, concerns are raised when deploying it in ad-hoc networks. Current ICN proposal strictly follows a receiver-driven transport design. However, many applications in multi-hop ad-hoc network are push-based and require fast communication. ICN’s pull-based transport in such case is costly. Moreover, inefficiency introduced by the default anycasting in ICN may backfire without careful design. In this paper, we introduce ICAN, an efficient, flexible, and adaptive ICN architecture supporting both pull and push transport and context-aware multi-hop/DTN communication all in one system.

Index Terms—MANET, ad-hoc, ICN

I. INTRODUCTION

Today’s mobile networking very much relies on the infrastructure technology. However, the advance of infrastructure deployment cannot always meet the bursting demands of diverse scenarios and applications on the user end. To meet such demands, ad-hoc networking with the advantages of fast deployment and easy device replacement is attracting more and more attention. Ad-hoc communication enables the possibility of communication in rural or emergency scenarios that lacks of infrastructure coverage, and also enables various possibilities for applications that were not usable or reliable in the infrastructure mobile network. For example, autonomous driving cars and crowdsourcing local attraction information.

As the demands of ad-hoc networking increases in the future, it is critical to propose an architectural mechanism rather than point solutions for the ease of rapid on-demand ad-hoc network deployment. To this end, we introduce ICAN, an information-centric context-aware ad-hoc networking architecture that is flexible and adaptive to different application needs and network conditions. While it is difficult to define a one-fits-all architecture, in ICAN, the application needs and network conditions are compiled as context in an extendable format so that it is possible to customize the ad-hoc network in a software controllable manner for quick deployment and management.

The ICAN architecture builds on top of three building blocks: network entity representation, context, and network operation. The network entity representation defines the identifier of communication units flowing in the network such as hosts, data objects, and data chunks. Network entities can be associated with each other. One example is to use the hierarchical namespace broadly used by today’s ICN community, where a file is identified by its file name with a prefix identifying its host ID, and is segmented as chunks each with a chunk ID [1]. Once the network entities are identified, their associated contexts are built automatically. We define the context be either application-related (e.g. content type) or network condition-related (e.g. connectivity). The context is leveraged by the network operation layer to perform decisions on packet forwarding and caching.

Like most architectural proposals, the components of ICAN are inspired by prior work, including ICN, DTN, and opportunistic routing. Our core contribution is extending and integrating them in a new architecture. For example, we adopt the concept of ICN [1], which enforces the receiver-oriented chunk based transport and in-network caching. While in-network caching is beneficial under mobility, many ad-hoc network applications are substantially push-based. Although such applications are also achievable with pull-based transport, the fundamental inefficiency of pull-only designs will degrade or even destroy the performance in MANET due to the limited bandwidth and storage. Therefore, we include in ICAN the push-based transport as a fundamental unit and leverage both push and pull paradigms by application context.

The organization of this paper is as follows. In section II, we discuss the requirements and ICN concept. In section III, we introduce the ICAN system. We describe the design choices in-depth in section IV. Our preliminary results are presented in section V. Finally, we conclude this paper in section VI.

II. REQUIREMENTS AND CONCEPTS

A. Requirements

We first discuss the key requirements of an information-centric ad-hoc networking architecture in the following.

1. Support both push-based and pull-based transport

The current ICN model adopts a pull-based approach in which a receiver-oriented interest is required to transmit a corresponding data. However, many applications that are important in ad-hoc network, such as private messaging and emergency notification, are push-based. The data of these applications must be generated and delivered in real-time. Although it is possible to realize sender-driven applications with the current ICN model [2-3], to adapt to the pull-based model, the interests are registered periodically in advance. There are two drawbacks of this approach. First, the approach requires the receiver to negotiate the data name in advance, which may not be suitable in a dynamically built MANET. Second, the solution backfires in ad-hoc network as it creates large overhead in highly mobile
networks. Therefore, it is necessary to enable push-based transport in which data can be sent directly to destinations.

2. Context-aware operations

It is well-known that in MANET the challenges mainly come from two aspects: mobility and error-prone wireless channel; both lead to intermittent connectivity and unstable routes. The solutions are best made with the knowledge of context. For example, an efficient multi-hop routing is the target solution in a dense urban scenario (say, a sudden power outage in New York City). In contrast, in a sparse rural network, an efficient DTN routing protocol [4-6] is the desired solution.

Application context is especially important with the use of ICN architecture. As mentioned previously, some applications are most efficiently handled with the push-based transport, e.g. an emergent brake alert, while some naturally fit the pull-based transport, e.g. accessing Internet from a shared mobile access point. Moreover, the application context also affects the efficiency of cache storage usage. For example, it is not useful to keep a live video packet in the cache for a day.

3. Extendability

In order to embed context awareness in the networking architecture, it is necessary to define a representation of context. One lesson we learned from the history of Internet is that we cannot predict all demands for the future. Therefore, it is important to preserve extendability in the representation design.

4. Fast deployable

ICAN aims at enabling fast deployment in various scenarios on regular mobile device. While it is possible to obtain hardware support, a MANET by itself must adapt to different context and should be self-configurable in an on-demand manner. Therefore, we assume ICAN will be implemented as an overlay in the form of software applications for easy deployment and upgrades.

B. Information-Centric Networks (ICN)

Our proposed architecture, ICAN, relies heavily on the concept of ICN, or Named Data Network (NDN) [1]. ICN uses the data names instead of host addresses to locate data. It strictly assumes a pull-based, one-interest-one-data transport. Every data chunk has a unique name. To initiate a data transfer, a data consumer must send an Interest to retrieve the corresponding Data. The data chunks are cached along the way by all relays when it traverses the breadcrumb of interest forwarding path from a data provider back to the data consumer.

All ICN nodes are identically built with three data structures: Content Store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB). CS is the cache used to store received data. The use of CS is the core design of ICN and the main reason why we choose to build ICAN as an extended ICN architecture. Caching eases issues caused by unreliable wireless channel. With distributed caches, a data retrieval failure arising from intermittent connectivity can be quickly recovered.

PIT “remembers” all pending interests a node received. The two main purposes of PIT are (1) recording the breadcrumb path, and (2) suppressing redundant interest and data transmission for different data consumers. PIT is good for Internet since breadcrumbs are useful for nearly all multi-hop routing protocols. However, PIT alone is not enough for reliable and efficient data delivery in ad-hoc networks. Under intermittent connectivity, the architecture must be adjusted so that data can be delivered in a carry-and-forward way. On the other hand, interest aggregation is valuable as bandwidth utilization can be largely improved.

FIB is a name-based routing table constructed per name prefix. ICN interests are broadcast on one or more interface recorded in the routing table. The use of broadcast is advantageous due to the fact that more caches can be explored. However, it is well known that the routing table-driven protocols are not suitable under high mobility [7]. In ICAN, we exploit opportunistic routing [8] to utilize wireless broadcast nature for better robustness while preserving the FIB spirit by recording the known providers of a data object.

III. ICAN: INFORMATION-CENTRIC CONTEXT-AWARE AD-HOC NETWORK

A. Network Entity Representation

The context awareness of ICAN depends on the naming of network entities. In ICAN, data, node, and geo-location are all identifiable. The following explains how each is identified:

1. Data: Following the hierarchical naming in [1], data chunks are uniquely identifiable and associated with data object it belongs to. We enforce the following naming format of data chunk: application_id/data_object_id/chunk_id. That is, each application has a globally unique ID. Data object ID is defined by the application and unique in the application’s object namespace. Chunk ID is the sequence number of a data chunk within the data object.

2. Node: each node has a unique node ID. For compatibility, we assume IP or MAC address can be used to identify a node.

3. Geo-location: considering the applications requiring geocast ability, we promote the geo-location also to be a named entity. The naming of geo-location is simply assumed to be the GPS coordinates of a location plus a diameter.

B. Context and metadata

With the universal network entity naming, we are able to identify the context of each chunk, or packet, by mapping its associated name to an application or network condition context in the core of network.

We categorize the context as application-related context and network condition-related context. The application context is represented as the metadata of a data object or an application. For extendability, we assume the metadata format can be configured using XML [9]. Note that we do not piggyback context in XML format. Instead, XML is used for software configuration to define the meaning of piggybacked context.

Applications are responsible for providing part of metadata of its data objects. The metadata format implemented is shown in Table 1. Three attributes are required: content type, effective

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content type</td>
<td>Yes</td>
<td>Offline</td>
<td>Offline</td>
</tr>
<tr>
<td>Effective Time</td>
<td>Yes</td>
<td>[0 ms, ∞]</td>
<td>∞</td>
</tr>
<tr>
<td>Publicity</td>
<td>Yes</td>
<td>Public/Private</td>
<td>Private</td>
</tr>
<tr>
<td>Popularity</td>
<td>No</td>
<td>High/Medium/Low</td>
<td>N/A</td>
</tr>
<tr>
<td>Max delay</td>
<td>No</td>
<td>[0 m/s]</td>
<td>N/A</td>
</tr>
</tbody>
</table>
time, and publicity. The content type indicates whether the application content is generated in real time. Applications also indicate the effective time of its data. For example, a news website’s homepage may have an effective time of 4 hours, as shown in table 2. Another required attribute is publicity. We define a data object as private if it is access-restricted (e.g., encrypted data such as Facebook notification). If the application does not provide metadata, default values are used. Optional metadata fields may be indicated by the application or generated automatically. In our example, the maximum delay is indicated by the application, and the popularity is collected based on the statistics of request frequency.

ICAN also generates the network condition context automatically. Nodes maintain their node metadata including its location, a list of known connected nodes, and a list of out-of-contact nodes. We assume ICAN nodes use GPS to determine their locations; the list of known connected nodes is maintained by overhearing network traffic; the list of out-of-contact nodes is maintained by implicitly detecting the retransmission failure toward known destination.

Besides the metadata generated, nodes also retrieve metadata from processed or overheard data chunks. In this way, ICAN implicitly collects the context necessary for processing and adapts suitable forwarding and caching strategies accordingly.

C. Service

ICAN considers system services such as location service (used to retrieve the metadata of a node) as a special type of application. Unlike the user applications, services use pre-defined service name prefixes concatenated by the node or geo-location identifier to indicate the information that is requested. For example, a node location query of a location service is named location_service/node_ID/

ICAN allows services to be installed at system level, and thus the service requests can be answered by any node that has the specified service installed, leveraging the context and caching functionalities. This means services such as location or content search are distributed, better suited the ad-hoc environment.

D. API

The API provided for user applications are as followed.

1. query(data_name): for the requester to send an interest.
2. put(data_chunk_name, data_chunk_content): for the data provider to publish a data with a certain data name.
3. set(data_object_name, metadata, dissemination_type): for the provider to specify the metadata of a data object. If metadata is not specified, the default value is assumed.
4. push(data_name, data_content, metadata, destination_names): used to push a data object to a certain destination. The destination is either a set of nodes or a geo-location. With this API, any push-based application using unicast, multicast, or geocast can be implemented.

---

### Table 2. Sample metadata

<table>
<thead>
<tr>
<th>Application or Data object</th>
<th>Content type</th>
<th>Effect. time</th>
<th>Publicity</th>
<th>Popularity</th>
<th>Max delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>Offline</td>
<td>1 min</td>
<td>Public</td>
<td>High</td>
<td>N/A</td>
</tr>
<tr>
<td>Car/Airport-alert</td>
<td>Real-time</td>
<td>15 mins</td>
<td>Public</td>
<td>High</td>
<td>N/A</td>
</tr>
<tr>
<td>Map/Westwood-blvd</td>
<td>Offline</td>
<td>2ms</td>
<td>Public</td>
<td>High</td>
<td>N/A</td>
</tr>
<tr>
<td>Traffic/Westwood-blvd</td>
<td>Real-time</td>
<td>4hrs</td>
<td>Public</td>
<td>Medium</td>
<td>N/A</td>
</tr>
<tr>
<td>NYT/moderneidge</td>
<td>Offline</td>
<td>4hrs</td>
<td>Public</td>
<td>Medium</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

E. System Architecture

Figure 1 illustrates a typical ICAN node. ICAN is an all-broadcast system. At the bottom, the broadcast layer ensures reliable hop-by-hop broadcast-based transmission for an ad-hoc ICN. All packets are further processed by ICAN packet operator, which extends the current ICN architecture with context, push transport, and DTN mode support. We keep CS and PIT with extended functionality for the purpose of caching and interest aggregation. The context generator collects and generates metadata locally to support context-aware operations. The collected context is stored in the context resolver, which is a database that provides the context needed by other modules.

The forwarding module is in charge of packet routing. Based on the application and network context, four basic forwarding protocols are supported: push data forwarding, interest forwarding, pull data forwarding, and DTN forwarding. By examining the context, forwarding module decides if a packet is push-oriented or pull-oriented, and apply different routing protocols accordingly. When ICAN judges from the provided context the destination is not reachable, the packet is saved in the DTN packet storage and will be re-broadcast by DTN forwarding. Note that as the context format is extendable, different algorithms can be easily implemented as plug-ins to replace any of the four forwarding protocols.

We summarize the packet processing procedure as follows. Incoming interest packets from application, service, or broadcast layer first enter the context generator. After inspecting the packets, updated metadata is stored in the context knowledge base. Next, ICAN searches CS to find matching data. If nothing is found, the interest goes to service and application manager (SAM), which is responsible to match the name prefix of a packet to applications and installed services. Note that if the prefix matches a service’s, the distributed service may or may not reply to the interest after inspecting it. If the service decides not to answer the query, the packet is returned to ICAN for further processing. At this point, the node searches its PIT to decide if it has previously sent an interest with the same name. Interests from different requesters are integrated to avoid redundant transmissions, which means the relay does not transmit the interest again but only adds the requesting face to the existing PIT entry. Otherwise, the interest to be forwarded is handed to the forwarding module. Based on the context, the
forwarding module decides whether to broadcast the packet right away. If the node is detected partitioned from the destination, the packet enters DTN mode. DTN packets are deferred, compiled and sent periodically or when new contact is detected.

Data packet processing is similar. A data packet along with metadata it carries are inspected by the context generator. The data content is then cached by CS. At this point, CS consults the context resolver to perform caching decisions. Later, if the data packet is not targeted to a local service or application, it is handed to the forwarding module. According to the data chunk context, the forwarding module decides to forward it in push mode, pull mode, or DTN mode.

IV. IMPLEMENTATION

A. Broadcast-based Routing

In ICAN, nodes automatically decide the packets’ routing strategies by looking up the network condition context. If the destination is judged reachable, the packets are forwarded using multi-hop routing. Otherwise, the packets, no matter pull or push, interest or data, are forwarded using DTN routing.

1) Multi-hop Routing

Since the pull data is forwarded via breadcrumb path following the same principle in [1], the multi-hop routing path depends on the interest and push data routing. Fortunately, both can be done using the same name-based routing protocols, as all destinations are named in ICAN. There have been abundant literatures for multi-hop ad-hoc routing. In this paper, we focus on discussing the two different ICN multi-hop routing paradigms: nearest-replica routing and data source routing.

Traditional ICN assumes nearest-replica routing. However, it is fundamentally difficult to realize this paradigm in MANET. Moving provider is a well-known problem for ICN [10]. Some previous work proposed solutions such as extending GHT [11]. However, even GHT may not be scalable for nearest-replica routing as the traffic of location updates for all names overload the network. The other solution is to always flood interest packets to find nearest replica. This approach, while is good for popular public data, still incurs high overhead if applied to all packets. Some may argue that pull data forwarding can be modified so that the data only pursues the shortest path. However, replacing breadcrumb routing will remove ICN’s well-appreciated interest aggregation ability. We explain the reasoning by the example Figure 2. Suppose A and C both want to retrieve the same data located at F. C floods its interest first. The interest traverses two paths: C-B-D-F and C-E-F. If the corresponding data is returned via the shortest path F-E-C, there is no way to consume pending interests at B and D except waiting for their expirations. Suppose A then floods the same interest later, B will aggregate the interest with the pending interest and drop it. The only solution is to give up the interest aggregation feature of ICN. In other words, if a flooding-based nearest replica interest routing is applied, in order to preserve interest aggregation, the data must traverse all interest paths.

Another option is to route the interest packets towards the data origin. This can be easily accomplished by sending a routing request looking for the data source first. However, ICN is naturally a multi-source environment due to its relay caching ability. Thus, a source selection process is needed. We define a location service to select data sources and converting the data chunk name to the source location. If the location service does not have the source location, it performs an “exploration” to search for the data object’s sources. In the exploration phase, the location service floods an interest location_service/object_name, where object_name may be a data object name or a node name. All nodes, while receiving this interest, pass it to their own location services, which then search the cached and application data object list. If a match is found, a location service replies a “data packet” carrying the metadata of the queried content. This data packet is treated as a normal pull data packet, following breadcrumb back to the request sender. If no match is found locally, the request is returned to ICAN and the forwarding process continues and passes the request to others.

Since the location request is flooded, the metadata is propagated along the flood paths, and therefore is available to the flooded area. The advantage of this approach is that all relays between the data source and the data consumer obtain the context and hence have consistent knowledge of how to handle the following packets. Note that even with multiple sources, this approach guarantees the paths to found sources are disjoint. The reason is that ICAN treats the location requests and responses as normal interest and data. Hence the returned data are blocked normally if the location request has been consumed. This is the benefit of promoting the service to user application level.

B. DTN Routing

In DTN mode, the node stores all DTN packets until they expire. The nodes only broadcast DTN packets in two conditions:

1) When new contact is discovered.

New contact may be discovered when receiving packets from neighbors. If a node detects a new contact, it starts broadcasting DTN packets. The packets are prioritized based on a context-aware scheduling algorithm. In our implementation, we assume a fairly simple scheduling algorithm that sends the interests first, push-data later. Pull data is only sent when its corresponding interest is received during the contact.

2) Periodically

Since silent contacts may not be detected, it is necessary to periodically broadcast DTN packets. The packets are scheduled in a round robin fashion for periodic broadcast.

C. Reliable Broadcast

All transmissions in ICAN are done by MAC layer broadcast. The consequence of this design choice is that there is no support from MAC layer retransmission. Therefore, we implement a reliable broadcast layer to guarantee robust transmission.

The reliable broadcast utilizes the packet names a node heard to ensure delivery. A packet is retransmitted, up to a certain limit of times, if a relay node does not detect a progress is made, that is, if one of the three conditions are met:

![Figure 2. Interest aggregation](image-url)
(1) A packet carrying the same nonce is received from neighbors
(2) The node receives a data packet carrying the same name as the sent interest
(3) The node receives an ACK that acknowledges a nonce.
Note that condition (3) implies that when a packet reaches its destination, the destination must send an ACK with the received packet nonce to stop the retransmission from the last hop.

D. Metadata Dissemination

Application metadata can be disseminated periodically or on-demand. We define a default metadata dissemination service for answering on-demand requests. In general, the metadata is retrieved on-demand for most data objects. Only the emergent for answering on-demand requests. In general, the metadata is retrieved on-demand. We define a default metadata dissemination service described in Sec IV. A is used to distribute the node metadata.

V. CASE STUDY

We prototype ICAN as an android application. As a case study, we conduct a small-scale experiment at parking structure 8 at UCLA to demonstrate the multi-hop routing design choice. The system is deployed on six Nexus S Android smartphones, which are separated by obstacles so that the topology is as in Fig 3. A simple FTP application is implemented. Two requesters send interests to retrieve a 75KB file. The chunk size is 512 bytes. Interests are initiated every two seconds. For reliable broadcast, packets are retransmitted up to 2 times every 150ms. Interests are re-initiated by requesters every 1 second if the corresponding data is not received. The experiments are repeated three times.

Table 3 shows the results. The congestion level is measured by the average number of packets sent per node. Nearest-replica routing’s congestion level is about 1.75 times data source routing’s even in our small scenario. Both schemes achieve 100% completion rates since the scenario is static and reliable broadcast ensures the robustness. To get a closer look, we present the cumulative distributions of delay and interest re-initiation in Fig. 4. We observe that nearest-replica routing delivers most of the chunks with lower delay since data source routing spends extra time on location service exploration, but its maximum delay is larger due to more re-initiations needed due to its higher congestion level. We expect the congestion level has more significant impact than the delay does in larger networks.

VI. CONCLUSIONS

We present ICAN, an ICN-based ad-hoc networking architecture. ICAN achieves high efficiency, flexibility and backward-compatibility. Unlike the current ICN proposal, ICAN supports push transport and context-aware forwarding and caching. Our design aims at improving the ad-hoc ICN efficiency by utilizing the context awareness in all aspects. A prototype is implemented as a proof of concept and tested on our Android testbed.

REFERENCES