NDN, Technical Report NDN-0020, 2014. http://named-data.net/techreports.html
Revision 1: June 3, 2014

Kite: A Mobility Support Scheme for NDN

Yu Zhang Hongli Zhang
Harbin Institute of Technology
{yuzhang, zhanghongli}@hit.edu.cn

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Named Data Networking (NDN) natively supports the mobility of data consumers through its data-centric design and stateful forwarding plane. However, the mobility support for data producers remains open in the original proposal. In this paper, we introduce Kite, a design of mobility support for NDN. Kite leverages the state of the Pending Interest Table (PIT) at each router to reach mobile nodes to make the location of a mobile node transparent to other parties in communication with it. We describe how Kite can support typical scenarios including group communication among mobiles. Our preliminary evaluation shows that Kite outperforms the mapping-based mobility solutions in terms of path stretch and mobile delay, with similar signaling overhead in the case of high-frequent movement. We also discuss security considerations and architectural implications on NDN.

1. INTRODUCTION

Lacking native support of mobility is a marked pain point of IP architecture confronted with the tremendous explosion of mobile devices in the Internet. Therefore, a plethora of endeavors have been made to affix mobility support onto the Internet, to answer the same question: where is (how to reach) a moving destination? To solve that problem as well as other issues with IP in a unified way, Information-Centric Networking (ICN) highlights that the destinations should be data names rather than machine addresses, so the question for ICN becomes where is (how to retrieve) a piece of data?

Named Data Networking (NDN, aka CCN)[7, 19] has mainly two features beneficial to mobility: the *data-centric* nature and the *stateful* forwarding plane. First, the data-centric communications are built directly upon data names which are not necessarily bound to topological positions. Since each data is self-identified by its unique name, there is no concept of TCP-like bitstream to count, nor host-connection to maintain in NDN. Second, the states of data request, namely Interest (packets) in Pending Interest Tables (PITs), in the forwarding plane enable the reverse path forwarding of Data, which makes both the location and identity of data consumers transparent to the routing plane and data producers[1]. Thus, generally the mobility has no impact on forwarding Data back to the consumers.

However, in the seminal paper of NDN[7], the mobility support of producers was a missing piece. Essentially, for NDN, the reachability of data is implied by data names either directly in the routing plane or indirectly via an intermedia mapping, such as CBIS[6]. When the data producer is moving, some updating messages or operations on packets will be necessary in response to such changes. Moreover, in some scenarios, a producer may need to actively notify the mobile consumer that new data is available before the consumer fetches data from the producer. Furthermore, in the case of distributed data production by multiple mobile producers, it is even difficult to find out all of producers at the first place. Therefore, the mobility support in NDN calls for further investigations.

In this paper we present *Kite*, a mobility support scheme for NDN. Inspired by NDN's Data forwarding, the key idea of Kite is to fully exploit NDN's forwarding states to keep track of moving nodes. Specifically, an application can send an Interest to a routable *anchor* to create the PIT entries as breadcrumbs (a hop-by-hop *trace*) leading back to itself on demand. The mechanism of Data forwarding is extended to Interest forwarding, so the mobile node (MN) can be reached via the anchor by Interests from correspondent nodes (CNs). This can be described by analogy with flying a kite: a kite (like an application) will be reachable along a string (like a trace) from hands (like anchors).

Kite contributes two new features to the mobility support of NDN. 1) *Locator-free*: There is no explicit locator for MN which is implicitly addressed by hop-by-hop states instead. Consequently, the topological locations of MNs are transparent to routers, CNs, and even MNs themselves[2]. 2) *Scenario-aware*: As traces are generated and utilized directly by application protocols, protocol developers are partially empowered to devise their

[1]This anonymity holds unless the consumer is the direct neighbor of a router or the producer.

[2]The access point of MN may learn the location of MN, but it is not necessary for Kite.

own designs of mobility support tailored to their scenarios, rather than only counting on some middlewares or the network layer.

We stress that Kite does not provide a one-size-fits-all solution to all mobility issues. First, Kite only aims to mobility over a stable network infrastructure, not for wireless mobile ad-hoc networking, because trace setup and maintenance depends on stable underlying routing. Second, we consider Kite an alternative to, not a replacement of, those mapping-based schemes with explicit locators. Particularly, Kite is not suitable for the long-term relocation of data producers.

The rest of paper is organized as follows: Section 2 describes problems and challenges. The Kite scheme is articulated in Section 3. Section 4 presents the mobile application protocols powered by Kite. Section 5 shows the evaluation. The related work is briefly reviewed in Section 6. We make discussion in Section 7 and conclude the paper in Section 8.

2. PROBLEMS AND CHALLENGES

2.1 Mobile Application Scenarios

This paper focuses on data-centric mobile applications which are expected to deliver data seamlessly when the node is moving topologically in the Internet. All NDN applications share the same receiver-driven communication pattern: a consumer fetches a named Data (packet) from producers or in-network caches by expressing an Interest (packet) with the data name. Thus no Data will be sent out without a request of Interest.

The mobile application scenarios can be generally categorized according to the role of MN. 1) *Is the MN a consumer, a producer or both?* The CN which communicates with the MN will play the opposite role. 2) *Is the MN the one who sends the first Interest to initialize an application?* Usually, the purpose of the *active* producer to express the first Interest is to notify a *passive* consumer that the data is ready to fetch, so the producer-issued Interest will trigger the consumer-issued Interest. Note that a producer-initialized application is still receiver-driven, as Data is still to be pulled by the consumer.

By combining the answers, we have five categories of scenarios, as listed in Table 1. If an MN, e.g., a smart phone, is active, it may either 1) *Upload*: a photo to a server; or 2) *Download*: a photo from a server. If the immobile CN, e.g., a server, is active, it may either 3) *Pull*: data from a phone; or 4) *Push*: data to a phone. In the case of group communications among mobiles, the MNs can 5) *Share* data, such as chat via phones directly without a centric server.

2.2 Forwarding in NDN

Interests and Data are forwarded by NDN's stateful forwarding plane. In each NDN node, packets are processed by an NDN forwarder which maintains three data structures: Content Store (CS) to cache Data, Pending Interest Table (PIT) to record Interests associated with incoming interfaces, and Forwarding Information Base (FIB) to store name prefixes associated with outgoing interfaces to the next hop.

Table 1: Categories of application scenarios

Mobile	**Initialized by**	
Role	mobile node	correspondent node
producer	*Upload*	*Pull*
consumer	*Download*	*Push*
both	*Share* among mobiles	

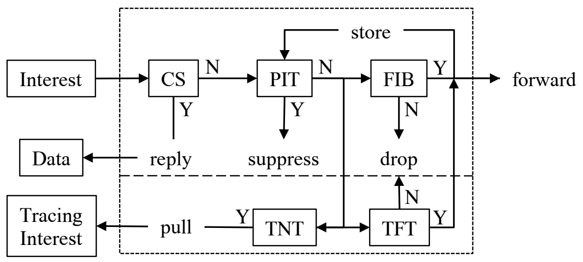

Figure 1: Interest forwarding process inside a forwarder. 'Y' denotes a match, 'N' no match.

Inside forwarders, Interests and Data are forwarded in two different ways: Interests are forwarded along routes in FIBs set up by the routing plane; Data are forwarded along traces in PITs set up by the Interests. The process of forwarding an Interest is illustrated in the upper half part of Figure 1. Upon receipt of an Interest, the forwarder searches its CS, PIT and FIB in turn for the name of Interest in an if-then-else manner. If there is no match in CS and PIT, but a match in the FIB, the Interest will be stored into the PIT and be forwarded to the next hop. When a Data comes back and satisfies an Interest in the PIT, the Data will be cached and sent back to the incoming interfaces of satisfied Interests, and the corresponding PIT entry will be removed. So an Interest and its corresponding Data travel along the symmetric paths.

2.3 Opportunities and Challenges

Thanks to NDN's data-centric nature and stateful forwarding plane, the mobility has no effect on Data forwarding, so the Download scenario is natively supported in NDN without any additional mechanism. No matter how the active mobile consumer is moving, the Interest can reach the immobile producer along the routes in FIBs. Afterwards, the Data will return along the trace of the Interest. In the case of *move-before-get*, i.e., the receiver moves before the return of packet, the Download application on the consumer will detect the timeout

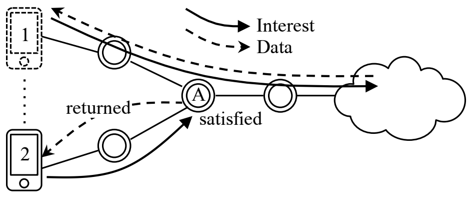

Figure 2: Fetching Data if move-before-get

of Interest request, retransmit the Interest, and get the Data from the CS of *junctional router* at the junction of old and new paths towards the producer. As shown in Figure 2, after relocation from Location 1 to Location 2, the MN will fetch the Data from Router A by expressing a new Interest. Therefore, no one is actually aware of the movement of an active consumer during the communication.

However, when the MN is either a producer in the Upload and Pull scenarios, or a passive consumer waiting for the notification from the producer in the Push scenario, some Interests have to be sent to the MN. Because Interests are forwarded according to FIBs fed by the routing plane, which is subject to the scalability issue with the frequent updates, some additional mechanisms have to adopted to cope with the relocation of MNs. Moreover, in the Share scenario, the participants in a chatroom may not even know each other beforehand, so some new schemes are desirable to let the MNs share data mutually.

In summary, the mobility of active consumers is transparent to networks and applications in the current NDN design, but the mobility of producers and passive consumers is not.[3] And for the group communication, the MNs need to find each other in the first place. Essentially, all those challenges point to the same question: *how to forward Interests to mobile applications?*

3. THE KITE SCHEME

3.1 Framework of Kite

Inspired by Data forwarding process, Kite utilize the traces of Interests left in the NDN's stateful forwarding plane. Specifically, the PIT entries created by an Interest from an MN serves as a trace which leads back to the MN. In other words, an Interest can be transmitted along the trace left by another Interest, just like a Data, which allows Interests and Data to share the same advantages of NDN on mobility support. Thus Interests can be forwarded according to two types of information: routes in FIBs fed by the routing plane, and traces in PITs fed by applications.

[3]The in-network caches in CSes or replicas can make the location of original producer obscure to the routing plane and consumers probably, but not absolutely.

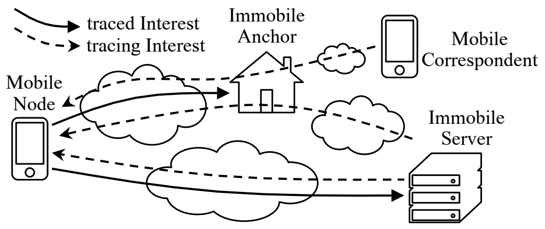

Figure 3: Framework of Kite

Figure 3 shows the framework of Kite. The Interest which leaves the trace back to a *trace source*, i.e., MN, is called *traced Interest*, and the latter Interest traveling along the trace of previous Interest is called *tracing Interest*. The trace source issues traced Interests, of which the name prefix is called *trace prefix*, to a *trace anchor*, which is reachable by announcing the trace prefix into the routing plane. Note that both traced and tracing Interests are still Interests in a sense that they are the requests for Data.

Kite supports the mobility in two ways. In *direct* Kite, the MN as the trace source can set up a complete trace by issuing a traced Interest to an immobile CN, in turn which can send an Interest back along the trace. In *indirect* Kite, the MN keeps sending the Interests to an immobile trace anchor which is application-specific. The CN, either immobile or mobile, can send tracing Interests towards the MN, optionally first along the route in FIBs to the anchor, and then along the trace in PITs to the MN. Therefore, the routing plane ensures the reachability of anchor, while the forwarding plane takes care of the reachability of MN.

3.2 Interest Trace Forwarding

To enable an Interest to travel along the trace, we introduce a new forwarding mechanism, *Interest trace forwarding*, into NDN's forwarding plane by extending the Interest packet format and the Interest forwarding process. Three new optional fields are added into Interests:

- `TraceName` field indicates which Interest is to be traced. The trace name in a tracing Interest should be the exactly same with the name of traced one.
- `TraceOnly` flag indicates how to forward the Interest. If the flag is unset by default, the Interest will be delivered along both routes in FIBs and traces in PITs in parallel. In the case that the flag is set, the Interest will only follow the traces if there is a match, otherwise follows the routes.
- `Traceable` flag indicates whether the Interest is allowed to be traced by the others. If the flag is set, it is allowed, otherwise disallowed. This flag serves as an access control point.

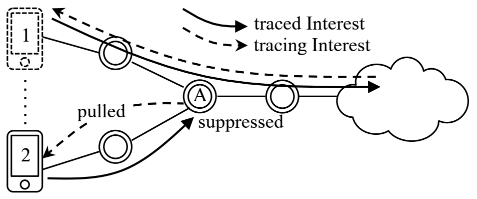

Figure 4: Pulling Interests if move-before-get

Two new tables are added into (and also implemented inside) the PIT: *Trace Forwarding Table (TFT)* contains all traced Interests with the `Traceable` flag. *Trace Name Table (TNT)* is composed of all trace names of tracing Interests associated with the corresponding PIT entries. Tracing Interests will be forwarded according to the TFT, while traced Interests will pull the corresponding tracing Interests in the TNT.

The process of Interest trace forwarding is illustrated in the lower part of Figure 1. For a tracing Interest, if there is a match of its trace name in the TFT, it will be sent via the incoming interfaces of matched TFT entry. In the case of multiple incoming interfaces, the tracing Interest will fan out to all. By default, the trace forwarding and the route forwarding are processed separately in parallel. If the `TraceOnly` flag is set and there is a match in the TFT, the forwarding process via the FIB will be skipped.

For a traced Interest, if there is a match of its data name in the TNT, the matched tracing Interests will be sent, like being pulled, to the incoming interface of the traced Interest. For example in Figure 4, after issuing a traced Interest, and before receiving the corresponding tracing Interest, an MN moves from Location 1 to Location 2. At Location 2, the MN sends another traced Interest, which will pull the tracing Interest from the junctional router A back to the phone, and also may be suppressed at Router A.

3.3 Trace Setup and Maintenance

Essentially, from the perspective of network layer, there is no difference between a traced Interest in Kite and an Interest for Data. The network layer, where Kite lies, does not guarantee the reliable transmission of traced Interests, while applications are responsible for the setup and maintenance of traces, because there is no separate transport layer in NDN. A simple and general method to set up and maintain traces is to send a traced Interest periodically as if the Interest has not yet been satisfied. The *retransmission timer*, T, serves as a knob to make a tradeoff between signaling overhead and reliability.

A trace is soft-state. The time for which an Interest will stay in the PIT is indicated by the `Lifetime` field in the packet. To prolong the lifetime, more traced Interests may be resent. Assume the remaining lifetime of an Interest in the PIT is t_r. Let the lifetime of a new incoming Interest with the same name be t_n. If $t_n > t_r$, t_r will be prolonged to t_n, and the new Interest will forwarded to the next hop, and so on. Otherwise $t_n \leq t_r$, the Interest will be suppressed.

Thanks to the locator-freeness of Kite, the movement of MN is transparent to applications. So the new lifetime t_n will be set constantly the retransmission timer T. To set up the new trace upon relocation of MN, the forwarder may (or may not) automatically re-express unsatisfied (traced) Interests with their remaining lifetime T_r. There are two other topics related to the optimization to be handled by applications.

Dynamic timer T: After the new trace is set up and before the old trace expires, the tracing Interests will fan out along both traces as if there are two trace sources at both old and new locations. To keep the current trace alive and to clean up old traces quickly, the lifetime of new Interest should be dynamically adjusted to the expected period for which the node will stay before moving, e.g., the average period of past several stays.

Feedback packet: A traced Interest may be dropped due to congestion, and the existing trace may be broken due to the overload of PITs or the change of underlying topology. Besides periodical retransmission, the application may need a feedback (acknowledgement) packet, which can be a tracing Interest from the CN or the anchor. In the case that a PIT entry is removed due to PIT overload, the forwarder may or may not return a NACK message to notify the trace source.

3.4 Path Optimization

The path may not be optimal when the messages between two nodes go through the anchor rather than along the shortest direct path. However, we consider such non-optimal stretch the expected expense of transparency to the routing plane and locator-freeness. Therefore, the design of Kite does not aim to produce optimal paths. One can add some optimization methods transparent to Kite, such as to relocate the anchor towards the MN, or to deploy multiple topologically-distributed anchors. But Kite can reduce stretches by itself in two ways as follows:.

Path shortcut is a path to the MN not through the anchor, which will occur if the MN and the CN are on the same branch of shortest-path tree sourced from the anchor. The tracing Interest will run into the traced Interest at the lowest common ancestor on the tree. By setting the `TraceOnly` flag, the tracing Interest will only go via the shortcut. Therefore, by path shortcut, Kite remit one of the worst cases that the path between two nearby nodes is stretched to the anchor. In other words, if the MN and the CN are in the same domain, path shortcut keeps the communication local even if the anchor is outside of the domain.

Path migration is to migrate the communication from a longer path into a shorter path. In general, a communication in NDN is composed of two phases: 1) to learn the data name, and 2) to transmit the data by Interest-Data exchange. In Section 4, we will show that in the case of immobile CN, Phase 1 is via the indirect trace, while Phase 2 can be along the shortest path.

4. SCENARIO-AWARE PROTOCOL DESIGN

With Kite in hand, we demonstrate the scenario-aware design of application protocols for the scenarios in Section 2.1. For the sake of simplicity, for each scenario we select a typical instance of application which is to deliver only a single Data. Those protocols satisfying the basic functional requirement of applications are only for conceptual demonstration, not for practical implementation.

4.1 Upload

The Upload scenario can be simply supported by direct Kite. For instance, Alice wants to upload her selfie from her phone to a Facebook server. The trace anchor is the immobile consumer, i.e., a Facebook server, and the trace prefix is the name prefix of the consumer, i.e., `/Facebook`. The trace source is the mobile producer, i.e., Alice's phone.

The messages are exchanged as shown in Figure 5 (a). First, the phone sends an Interest 1 with the `Tracable` flag to the server to set up the direct trace inbetween and to notify the server of the name of data to be uploaded. Then the server can fetch Data with a tracing Interest 2, of which the name is derived from Interest 1. The `TraceName` is the same name of Interest 1, and the `TraceOnly` flag is set. When the phone is moving with Alice, Interest 2 may not reach the phone in the case of move-before-get. When the application issues another traced Interest 1, Interest 2 will be pulled back to the phone, just as shown in Figure 4. After the upload is completed, there is an additional step, which is skipped in Figure 5 (a), that the server can rely to Interest 1 with a Data, which functions as both an acknowledgement to the MN and the eraser to clean the trace.

4.2 Pull

The Pull scenario can be supported by indirect Kite. For instance, with Alice's permission, a Facebook server wants to pull Alice's geolocation data from Alice's phone. The trace anchor will be Alice's home anchor (HA) where the traced Interests from Alice meets the tracing Interests from Facebook. The trace prefix is the name prefix of the mobile producer, i.e., `/Alice`, which should be announced into the routing plane by the HA. Note that the HA only serves as a fixed anchor, and will not take part in the communication.

The messages are exchanged as shown in Figure 5 (b). To maintain the reachability, the phone periodically sends Interest 1 back to the HA. To fetch Data from the phone, the server will issue Interest 2 following Interest 1. Interest 2 will first follow the route to the HA by the data name, and later travel along the trace to the phone by the trace name. Because the `TraceOnly` flag is set, after arriving at the anchor, Interest 2 will only be forwarded along the trace. Upon receipt of Interest 2, the phone will reply with a Data.

The path stretch can simply be reduced by path migration. The method is to append the immobile consumer's name `Facebook` on Interest 2 whose new name becomes `/Alice/geoloc.data/Facebook`. And then the data is delivered with the Upload protocol along the shortest path.

4.3 Push

The Push scenario can be supported by indirect Kite with a push service anchor. For instance, a Facebook server wants to push a new message to Alice's phone. The trace anchor can be provided by a third-party alert service provider, and the trace prefix `/Alert` is announced by the anchor.

The messages are exchanged as shown in Figure 5 (c). The phone periodically sends Interest 1 to register in the alert service and to set up the trace. To notify the phone of new message, the server will issue a tracing Interest 2 of which the name contains the name of server, i.e., `/Facebook/`. Upon receipt of Interest 2, the phone will directly download the Data from the server by expressing Interest 3 with a data name derived from Interest 2.

4.4 Share

In the Share scenario, a group of MNs have the common interest to share data with each other. Every node is both producer and consumer. Kite supports this scenario by building a *bidirectional (shared) tree* rooted at a trace anchor. For simplicity, we first consider the case of two nodes. For instance, Alice and Bob chat with each other via their phones in a virtual chatroom. The trace anchor will be the chatroom's rendezvous anchor (RA), and the trace prefix is `/Chatroom` announced by the RA. Note that the RA like the HA will not take part in the communications.

The message exchanges are shown in Figure 5 (d). First, every member in the chatroom should issue the same join Interest to the RA. In the join Interest, the trace name is the same with the data name. When the join Interests 1 and 2 from Alice and Bob meet at the RA, Interest 2 will trace Interest 1 to Alice, and also Interest 1 will be pulled towards Bob. As a result, a bidirectional trace between Alice and Bob is set up. With this trace, Alice and Bob can request for the next messages from each other by expressing Interests. For

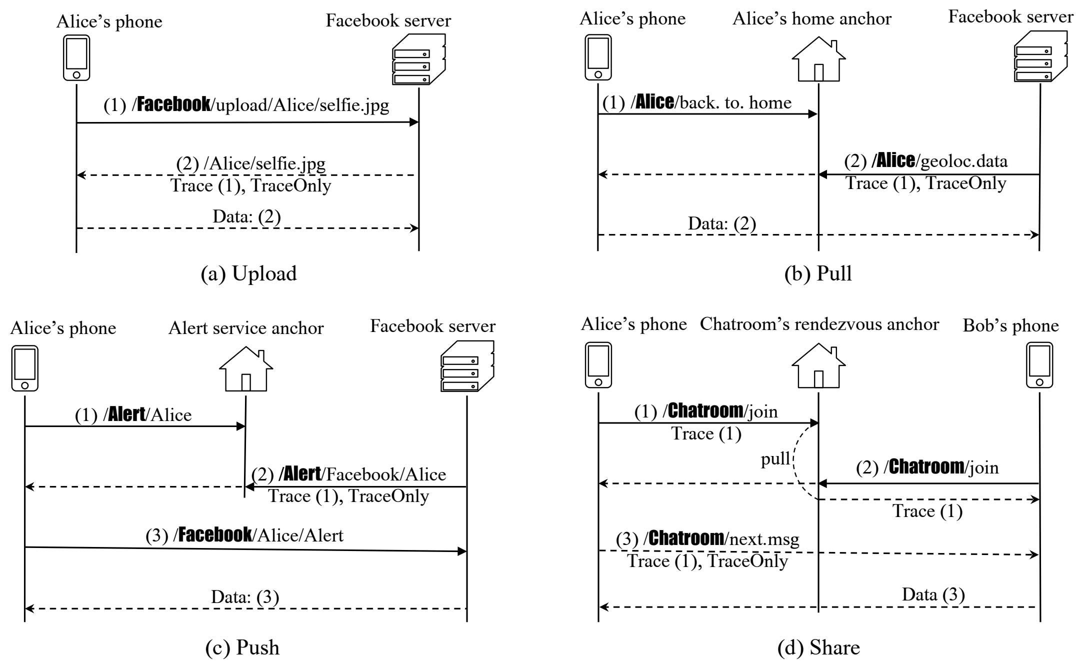

Figure 5: Message exchanges in mobile application protocols. Bold prefixes are in FIBs.

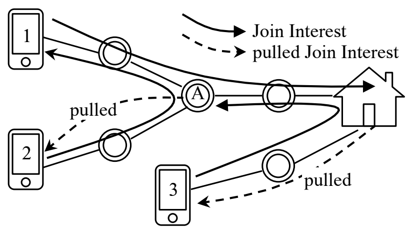

Figure 6: Bidirectional tree for sharing

example, Alice sends the Interest 3 to Bob, and then Bob can send his chat message as a Data to Alice. For details on how the chat protocol works, refer to the real-world protocol for serverless chatroom, ChronoSync[21].

There are three paths between two MNs via either the RA, or the two HAs of MNs. Phase 1 may be over the path via the RA, while Phase 2 can migrate to the shortest among the three paths, and the data will be delivered with the Pull or Push protocols.

In the case of more than two members in the group, the join Interests from each member will construct a bidirectional tree which is rooted at the RP and connects all MNs as the leaves. Figure 6 illustrates how a tree spanning a group of three nodes is built up. Three phones will issue join Interests one by one. First, the join Interest 1 from Phone 1 leaves a trace between Phone 1 and the RP. Then the join Interest 2 from Phone 2 meets Interest 1 at Router A. Because the `TraceOnly` flag is unset, the join Interests will be forwarded along both the routes to the RA and the traces to the Phones. Therefore, Interest 2 is delivered to Phone 1, and Interest 1 is pulled to Phone 2. At last, Interest 3 issued by Phone 3 will be forwarded to the RA, then follow the trace of Interest 1, and finally be suppressed at Router A. At the same time, Interest 1 at the RA will be pulled back to Phone 3. Once this tree has been built, there exists a bidirectional trace between any pair of MNs, so every Interest tracing the join Interests from any MN will be multicasted to all of others.

5. EVALUATION

We evaluated Kite in two ways: first, we benchmarked Kite against a conceptual scheme; second, we implemented Kite and protocols as a proof of concept.

5.1 Benchmarking

5.1.1 Methodology

From the previous works (see the related work in Section 6), we abstract a conceptual scheme, Mapping-based Mobility Support (MMS), as the benchmark scheme. MMS uses explicit locators and the name-to-locator mapping. In MMS, an MN learns and reports its current routable location to a stationary anchor by sending a location update message. According to the role of anchor, there are two versions of MMS. 1) MMS-PX: the anchor serves as a proxy to redirect the Interests from the CN to the MN; 2) MMS-NS: the anchor serves as a name resolution server to tell the CN the location of MN. In both versions, the location information of MN in the anchor is hard-state.

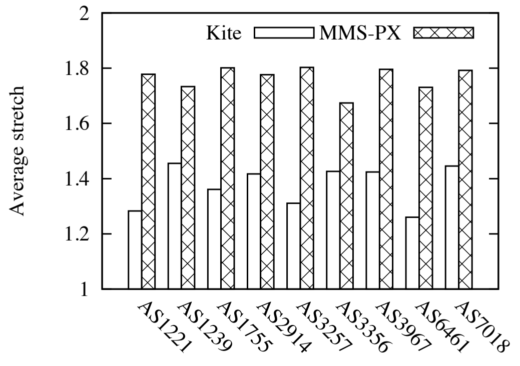

Figure 7: Comparison on average path stretch

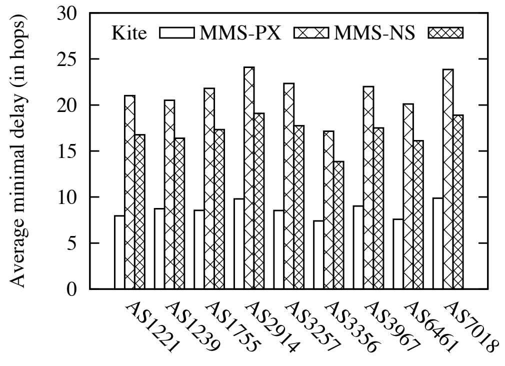

Figure 8: Comparison on average minimal delay

The router-level network topologies are built from the measurement results of Rocketfuel[16]. For each topology, we run $0.1 \times N^2$ experiments, where N is the number of nodes. In each run, the access routers of MN, CN and anchor are chosen independently and uniformly at random.

5.1.2 Path Stretch

Path stretch is the length ratio of the actual path to the shortest path between the MN and the CN. We only compare the stretch of indirect Kite with that of MMS-PX, as the stretch in both direct Kite and MMS-NS is obviously 1. For the topologies of nine ISPs, the average lengths of shortest paths are 5.28∼7.03. As shown in Figure 7, the average stretches of indirect Kite are 1.26∼1.46, about 20% shorter than those of MMS-PX around 1.67∼1.80. This result is expected, because indirect Kite can take advantage of path shortcut, while MMS-PX cannot.

In addition, the results also show that path shortcut can remit the worst cases that the path between two nearby nodes is stretched to the anchor. For example, for 86∼98% of cases with stretch greater than 3 in MMS-PX, path shortcut can help to reduce the average stretch from 3.72∼4.04 to about half (1.48∼2.17). Moreover, indirect Kite has more chances to obtain the shortest path than MMS-PX. For example, in AS1239, the number of cases with the shortest path in indirect Kite is 10 times of that in MMS-PX.

5.1.3 Mobile Delay

Mobile delay is the amount of time it takes for the MN to receive the missed Interest due to move-before-get after the MN gets network access at the new location. In Kite, the missed traced Interests can be pulled before expiring, as shown in Figure 4, while in MMS, the missed cannot be retrieved, and should be resent. For MMS, in the best case, upon receipt of update message from the MN, the anchor can immediately notify the CN to resend the missed Interest.

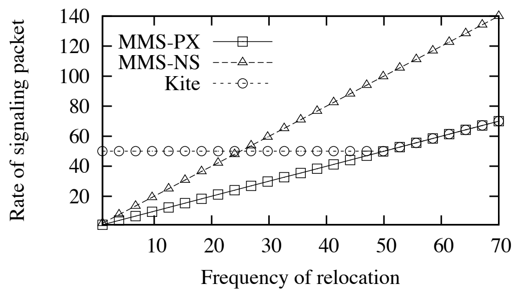

Figure 9: Comparison on signaling overhead

So when the anchor and the CN are fixed, the minimal delay in Kite depends on how far the MN move (the position of junctional router depends on both the old and new locations), while the minimal delay in MMS depends on where the MN move to (the new location of MN). We assume that the MN moves to a random new location independent from the old location, which is the worst case for Kite and has no impact on MMS.

Figure 8 shows that Kite has the smallest average minimal delay. Specifically, the average minimal delay in Kite is 7.58∼9.87 times of link latency, which is 38∼43% of that in MMS-NS and 47∼53% of that in MMS-PX. Note that those results are the worst case of Kite and the best case of MMS.

5.1.4 Signaling Overhead

Signaling overhead comprises the packets to signal the location of MN. Ideally, every time the MN relocates during communication, one packet is necessarily sent to set up a new trace in Kite, or to update the location binding at the anchor in MMS. MMS-NS needs at least one more packet to notify the CN of the new locator. In Kite, when staying in the same location, the MN still keeps sending Interests to prolong the trace.

Figure 9 shows the rate of signaling packets as the function of the frequency of relocations, where the lifetime of traced Interest is constantly 1/50 time unit. When the MN is relatively stable, i.e., the lifetime is

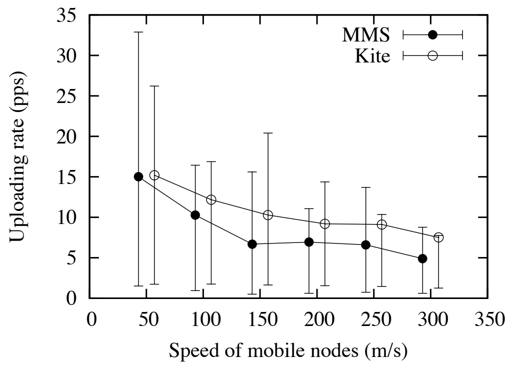

Figure 10: Uploading rate vs. Speed

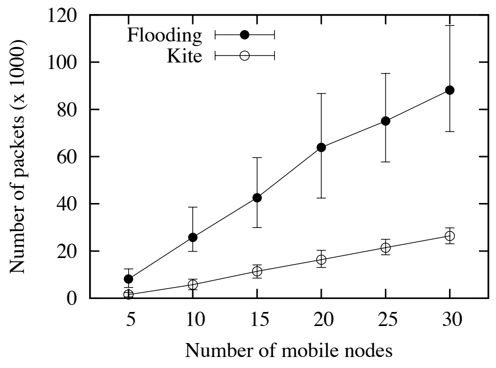

Figure 11: # packets vs. # MNs

much shorter than the duration of stay at the same location, the overhead of Kite is higher than that of MMS. If the MN moves so fast, or the lifetime is so long, that no overhead is needed to prolong the trace, then Kite costs the same overhead as MMS-PX, and less than MMS-NS. This result reflects a typical difference between soft state and hard state. We prefers the simplicity of soft state to the performance of hard state in the case of slowly moving MNs.

5.2 The Proof-of-Concept Implementation

To proof the concept, we implemented Kite and two instances of Upload and Share scenarios in ndnSIM[2]. The source code of simulation is available at [1].

5.2.1 Upload

We simulated the Upload scenario wherein an MN continually uploads data to a fixed CN, i.e., the server. We compared Kite and MMS in terms of uploading rate when the MN moves at different constant speeds. Due to space limitation, refer to [1] for the simulation setup.

Figure 10 shows the average uploading rate with the maximum and the minimum. As expected, the uploading rate decreases when the MN speeds up. Kite can help the MN to upload data faster than MMS, and the average uploading rate of Kite is up to 1.54 times of that of MMS. This is because Kite has smaller delay than MMS by pulling the missed Interests from junctional routers instead of the CN, which has been demonstrated in Section 5.1.3. Particularly, if the MN moves faster, the chance of move-before-get will be greater, and the advantage of Kite will be more obvious.

5.2.2 Share

We applied the Share protocol to ChronoSync [21], a real-world NDN protocol for decentralized data synchronization. Kite's bidirectional tree solution can improve the scalability of ChronoSync by *multicasting* Interests within mobile group members instead of *flooding* Interests over the whole network. To evaluate how much volume of traffic Kite can save compared to flooding, we measured the total number of Interest/Data transmitted by routers and MNs. Due to space limitation, refer to [1] for the simulation setup.

Figure 11 shows the average number of packets with the maximum and the minimum. Compared to flooding, Kite can help ChronoSync to reduce the total number of packets to 17∼31% according to the averages. As the number of MNs increases, the ratio of reduction decreases, while the size of reduction increases. For the same number of MNs, the total number of packets generated by flooding ranges more widely than that in Kite, which means the ratio of reduction also depends on some factors other than the number of MNs. We leave this to our future work.

6. RELATED WORK

Prior work is organized into three domains in a specific-to-general order.

1) *NDN/CCN mobility*: CBIS[6] introduced custodian entities as intermediary between name prefixes and communication endpoints. The mobility of custodian is supported directly by updating Custodian-to-Endpoint table (CET), or indirectly by underlying mechanisms, such as SIP[15] for NDN-over-IP. Kite and CBIS are beneficial mutually: Steady custodians in CBIS can serve as trace anchors in Kite, and Kite can reduce the CET updates for mobile custodians.

In both [5] and [11], each MN obtains an explicit locator and updates its location binding at its home agent, which will redirect the Interests to the MN by either appending the locator in front of the original data name [11] or inserting the locator as a separate field into Interests [5]. On the contrary, Kite is locator-free and needs neither any separate locator nor any touch on the packets.

In [9], an MN issues a special Interest to its previous location (or an equivalent of home agent) to set up an on-demand reverse path by creating the FIB entries pointing back to the MN. Instead of making change of

FIBs which are fed by the routing plane, Kite utilizes the states in PITs fed by applications, and separates traces in the forwarding plane from routes in the routing plane by using explicit trace names.

2) *ICN mobility*: [17] has surveyed mobility support in various ICN designs. We stress that in most designs for future Internet architecture (not limited to ICN), the data consumers are addressed by explicit locators or equivalents (e.g., source routing in LIPSIN[8]), while in NDN the consumers are anonymously addressed by the trace of hop-by-hop states. Consequently, in NDN the mobility of active consumers is transparent to producers, routers and even consumers themselves. Kite just extents that locator-free feature further to the mobility support of producers and passive consumers in a scenario-aware manner.

3) *Internet mobility*: The long and fruitful progress of mobility support research has been reviewed in [10, 20]. As indirection points, trace anchors in Kite are similar to home agents in Mobile IP[13], but trace anchors do not change packets, nor participate in communications, and even may be bypassed in the case of path shortcut.

Cellular IP[18], HAWAII[14] and TIMIP[3] share the similar idea with Kite to set up hop-by-hop reverse paths back to MNs, but they were designed for IP and are not scenario-aware. Moreover, as they are not architecturally built-in, they only support the micro-mobility within a local domain. By contraries, Kite leverages the native feature of NDN to provide a unified solution to both micro- and macro-mobility.

MSM-IP[12] pointed out that supporting multicast is kin to supporting mobility, so it is not surprising that Kite's soft relocation is almost identical to multicast, and Kite's bidirectional-tree-based Share protocol looks like BIDIR-PIM[4] [4]. Both BIDIR-PIM and the Share protocol are independent from any specific underlying unicast routing and scale well with a single shared tree in the case of many sources, while they are oriented to different architectures. Furthermore, since NDN's stateful forwarding can naturally ensure loop-free forwarding, it is not necessary for Kite to do designated forwarder election or check reverse path forwarding like BIDRI-PIM.

In summary, the combination of data-centricity, stateful forwarding, locator-freeness and scenario-awareness distinguishes Kite from those explicitly making use of separate locators, encap/decap tunnels, or global namespace mapping.

7. DISCUSSION

7.1 Security Considerations

The main security concern with Kite is *bogus traced Interests*. We briefly discuss some potential threats to investigate in the future work.

As a traced Interest indicates where the data is, attackers can pretend to be the victim, i.e., data producer, by expressing a bogus traced Interest to anchors. Then the attacker can poison caches and eavesdrop Interests to the victim. A countermeasure is *Interest signature*: The sender should sign traced Interests, and unless Interests are verified, tracing Interest will not be sent. And the signature scheme should also be secure under replay attack. Moreover, the node which detects the bogus Interest 1 may send a counterattack Data to remove the trace of Interest 1.

The verification of Interest signatures may be bypassed by attackers. For example, after sending a signed Interest, the mobile victim may move away, but the attacker may move in and disguise as the victim. And in the case of path shortcut, tracing Interests will bypass the anchor and follow the bogus trace towards the attacker. If the attacker behaves as a black hole, the routers on the trace will learn that there is no data from the interface towards the attacker, and may try other interfaces. If the attacker replies to consumers, fake data will be detected by consumers as all data are supposed to be authenticated, which is the bottom line of security in NDN.

A collusion attack can conduct cache poisoning by bypassing Interest authentication. For example, in the Upload scenario, the attacker can send a normal Interest 1 with the name prefix of a colluder node, and in turn the colluder returns Interest 2 with the name of victim to the attacker along the trace of Interest 1. Then the fake data can be sent out to poison the caches on the trace. One countermeasure can be to publicly specify *the policy of binding between data name and trace name of tracing Interest*, just like the policy of binding between data name and KeyLocator name. With a simple policy, such as that data name and trace name should have the common prefix, routers can determine that Interest 2 is not legitimate and should be dropped.

7.2 Architectural Implications

Besides mobility support, Kite has mainly two implications on the NDN architecture. First, Kite can help to relieve the burden of routing plane by enhancing the forwarding plane. In Kite, the routing plane only looks after steady anchors, while the forwarding plane keeps track of MNs. The locator-free feature can make the locations of active nodes completely transparent to the routing plane, which indirectly improves the routing scalability. For example, in the Upload scenario mobile producers which initialize application only appear on-demand on the paths rather than on the whole routing system. Generally speaking, it is better to let

[4]Actually, the idea of Kite emerged when we tried to design an Interest multicast protocol.

the routing plane take care of long-term changes and leave temporary high-frequent changes to the forwarding plane.

Second, Kite can help to further reduce the gap between application layer and network layer by empowering applications to directly guide Interests. The rest gap can be filled by the scenario-aware design of application protocols. For example, in the Share scenario, applications build up the multicast tree by themselves directly over network layer without a separate multicast protocol in the middle. More importantly, rather than a black box opaque to applications, Kite serves as an open platform for the innovation of applications, which is also one of motivations of NDN.

8. CONCLUSION

To the question how to forward Interests to an MN, Kite's answer is *to send Interests towards where the trace of MN can be found.* Inspired by Data forwarding, Kite utilizes PITs as traces to reach MNs, and ensures the reachability of MNs with routable anchors. Kite provides the mobility support of NDN with two new features: locator-freeness and scenario-awareness. The former makes the movement of MN transparent to all parties in communications, while the latter allows us to design application protocols for various scenarios. Compared to the mapping-based schemes, Kite has shorter stretch, smaller delay, and similar overhead when the MN moves frequently. Moreover, the simulations shown that Kite improves the uploading rate in the Upload scenario and reduces the traffic volume in the Share scenario. In addition to supporting mobility, Kite opens a new door to the design of NDN application protocols. Hence we believe that Kite will be a promising building block of NDN to further unleash the strength of stateful forwarding plane.

9. ACKNOWLEDGMENTS

This work was done during the first author's visit at UCLA supported by National NSF of China (No. 61202457) and State Scholarship Fund from China Scholarship Council (No. 201206125041).

10. REFERENCES

[1] Kite simulation with ndnSIM. *https://github.com/YuZhang/ndn-kite*, 2014.

[2] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. *Tech. Rep. NDN-0005*, 2012.

[3] A. Grilo, P. Estrela, and M. Nunes. Terminal independent mobility for IP (TIMIP). *IEEE Commun. Magazine*, 39(12):34–41, 2001.

[4] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. Bidirectional protocol independent multicast (BIDIR-PIM). *RFC 5015*, 2007.

[5] F. Hermans, E. Ngai, and P. Gunningberg. Global source mobility in the content-centric networking architecture. In *NoM '12*, 2012.

[6] V. Jacobson et al. Custodian-based information sharing. *IEEE Communications Magazine*, 50(7):38–43, 2012.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT '09*, 2009.

[8] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. Lipsin: Line speed publish/subscribe inter-networking. In *SIGCOMM '09*, 2009.

[9] D.-h. Kim, J.-h. Kim, Y.-s. Kim, H.-s. Yoon, and I. Yeom. Mobility support in content centric networks. In *ICN '12*, 2012.

[10] D. Le, X. Fu, and D. Hogrefe. A review of mobility support paradigms for the internet. *IEEE Commun. Surveys Tutorials*, 8(1):38–51, 2006.

[11] J. Lee, S. Cho, and D. Kim. Device mobility management in content-centric networking. *IEEE Commun. Magazine*, 50(12):28–34, 2012.

[12] J. Mysore and V. Bharghavan. A new multicasting-based architecture for internet host mobility. In *MobiCom '97*, 1997.

[13] C. Perkins. IP mobility support for IPv4, revised. *RFC 5944*, 2010.

[14] R. Ramjee, K. Varadhan, L. Salgarelli, S. Thuel, S.-Y. Wang, and T. La Porta. Hawaii: a domain-based approach for supporting mobility in wide-area wireless networks. *IEEE/ACM Trans. on Networking*, 10(3):396–410, 2002.

[15] H. Schulzrinne and E. Wedlund. Application-layer mobility using SIP. *SIGMOBILE Mob. Comput. Commun. Rev.*, 4(3):47–57, 2000.

[16] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *SIGCOMM '02*, 2002.

[17] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe. A survey of mobility in information-centric networks. *Commun. ACM*, 56(12):90–98, 2013.

[18] A. G. Valkó. Cellular IP: A new approach to internet host mobility. *SIGCOMM Comput. Commun. Rev.*, 29(1):50–65, 1999.

[19] L. Zhang et al. Named data networking. *Tech. Rep. NDN-0019*, 2014.

[20] L. Zhang, R. Wakikawa, and Z. Zhu. Support mobility in the global internet. In *MICNET '09*, 2009.

[21] Z. Zhu and A. Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *ICNP '13*, 2013.