A Solution for IP Mobility Support in Software Defined Networks

You Wang and Jun Bi
Institute for Network Sciences and Cyberspace
Department of Computer Science
Tsinghua National Laboratory for Information Science and Technology (TNList)
Tsinghua University, Beijing 100084, China
wangyou10@mails.tsinghua.edu.cn, junbi@tsinghua.edu.cn

***Abstract*—A large number of solutions have been proposed to support mobility in IP networks including original Mobile IP, its derivatives and several newly proposed protocols. However, these solutions have drawbacks in different aspects including triangle routing, handover inefficiency, heavy signaling overhead, etc. In this paper, we argue that these problems can be addressed based on Software Defined Networks (SDN). We discuss why SDN helps to solve the problems in current IP mobility protocols and give our algorithms to demonstrate how the problems are solved. We also present an Openflow-based protocol design to realize our idea. We show performance benefits of our protocol comparing with existing IP mobility protocols through implementation and experiments on Mininet.**

Keywords—mobility management; mobile ip; home agent; software defined networks; openflow

I. INTRODUCTION

A large number of proposals have been proposed to bring mobility function into IP networks. Amongst all related proposals, Mobile IP [1][2] is one of the earliest and most well-known protocol. Mobile IP is an Internet Engineering Task Force (IETF) standardized protocol which allows Mobile Nodes (MN) to keep session survivability while roaming around and changing IP addresses. To achieve this goal, Mobile IP uses Home Address (HoA) to identify a MN and Care-of Address (CoA) to locate a MN. Mobile IP introduces an indirection agent called Home Agent (HA) to store *binding cache* of each MN in the form of a mapping between the MN's HoA and its current CoA. Having this binding cache, HA is responsible for tunneling packets to and from the MN.

The employment of a fixed HA brings the triangle routing problem when the MN is away from HA. Triangle routing results in data path stretch because the entity that stores the MN's binding cache is no longer on the shortest path between MN and Correspondent Nodes (CN), and all the packets from CN to MN have to take a detour to pass the HA. Triangle routing also leads to large handover signaling cost, as well as heavy load on HA due to the same reason. Different kinds of approaches have been proposed to address the triangle routing problem in Mobile IP. One class of such solutions deploys multiple HAs over the network [3]-[6] and distributes the MN's binding cache among the HAs. In this way, triangle routing can be alleviated. However, as discussed by Zhang et al. [7], it may require complex protocols to realize related functions including HA discovering, HA switching, binding cache synchronization, etc., and it is still an open issue to achieve these functions in a scalable way in large networks.

There are some recently proposed IP mobility protocols that do not belong to Mobile IP derivatives, such as HIP [8] and ILNP [9]. These protocols share similar ideas: when the MN moves to a new network, it uses an end-to-end way to inform all the CNs of its new IP address, thus data packets can flow directly between MN and CN and triangle routing never exists in such protocols. However, according to RFC 4830 [10], these protocols belong to global mobility protocols and have drawbacks in supporting micro mobility scenarios: since binding cache of MNs are always sent to CNs, it may result in large signaling overhead (especially on wireless links) and binding update latency during each network-layer handover.

Therefore, though we have various ways to support mobility in IP networks, they have drawbacks in different aspects, making it remain an unsolved issue. In this paper, we try to address the problem using SDN and Openflow. Software Defined Networking is an emerging network architectural approach while Openflow is one of the most well-known instantiations of SDN [11]. In SDN, network structures, functions and performance can be defined in a simpler way which is usually achieved by providing programmable devices and a centralized control logic. As we will show in this paper, network functions or services required to support IP mobility can also be realized in a software-defined way.

We argue that SDN helps to solve problems in IP mobility protocols: firstly, programmable network devices in SDN can alleviate or even eliminate triangle routing. It is because the binding cache of a MN can be flexibly placed on the shortest path between the MN and CN instead of on some fixed HAs. Secondly, centralized control in SDN helps to reduce protocol complexity. The discussion above shows that it is not an easy task to distribute a MN's binding cache within the network to avoid triangle routing without introducing extra cost, and as we will present in Section IV, we can achieve this goal using a light-weight centralized algorithm. Thirdly, IP mobility in SDN architecture requires less host involvement. Most mobility functions can be realized on the network side as we will show in Section III, and this implies faster handover without IP re-configuration, less signaling overhead especially on wireless links, as well as higher security and privacy assurance.

978-1-4799-3572-7/14/$31.00 ©2014 IEEE

This paper has the following contributions:

- As far as we know, this is the first paper that gives a general discussion on whether and why SDN helps to solve problems in current IP mobility protocols, as well as how to seek for the best solution.
- We present our protocol design to enable mobility support in SDN (see Section III) and address an NP-hard problem, which serves as a key component of the protocol design, to demonstrate why our protocol avoids weaknesses of current IP mobility protocols (see Section IV).
- We implemented our protocol based on Mininet and did some experiments to compare our solution with two representative IP mobility protocols. Experiment results demonstrate performance benefits of our solution in terms of both data path stretch and handover efficiency (see Section V).

II. RELATED WORK

In this section we give an overview of related IP mobility protocols. First we review Mobile IP and its derivatives as well as protocols that share similar idea to store and distribute MN's binding cache among mobility anchors deployed in the network, and we call them *network-based protocols*. Then we review *host-based protocols* which fully rely on end hosts to maintain binding cache. Finally we mention some research on providing IP mobility support in SDN architecture.

A. Network-based Mobility Protocols

As explained in Section I, Mobile IP [1][2] centralizes both mobility signaling and data forwarding functions into a single HA, which increases signaling cost and data path stretch when MN is not within the home network. To address the problem, some Mobile IP extensions have been proposed. HMIPv6 [12] deploys Mobility Anchor Points (MAP) in the network and uses them to localize mobility signaling when the MN is away from HA. Specifically, MN attaches to a nearby MAP which is located using a Regional CoA (RCoA), and then the MAP is responsible for keeping the bindings between the MN's HoA and a Local CoA (LCoA), which is exactly the MN's current location, and tunneling packets to the MN. When attaching to a new MAP, to keep its reachability, MN informs HA of the new MAP's RCoA. Proxy Mobile IPv6 (PMIPv6) [13] is a similar solution, and it frees MNs from mobility signaling and employs Mobile Access Gateways (MAG) to perform mobility management functions on behalf of MNs. However, both HMIPv6 and PMIPv6 cannot avoid triangle routing when MN is not located within home network, because data packets toward the MN still need to be redirected by the HA.

Other Mobile IP derivatives address triangle routing by making more modifications to the original protocol. Home Agent Migration protocol [3] assigns anycast addresses to all HAs distributed in the Internet. MNs and CNs also utilize anycast to find and attach to the nearest HA. To maintain reachability of a MN, its binding cache is synchronized among all the deployed HAs in the network. Distributed IP Mobility Approach (DIMA) [4] also distributes central HA functionality onto several new inter-working entities called Mobility Agents (MA), and then MAs form a Distributed Hash Table (DHT) to store the binding cache of MNs. Peer-to-Peer HA Network (P2PHAN) [5] has the similar idea while it uses a P2P network to discover a close HA for MNs. DHARMA protocol [6] organizes distributed HAs as an overlay network and proposes both measurement and heuristic based algorithms to locate a nearby HA. LISP Mobile Node [14] and TTR-Mobility [15] are not Mobile IP derivatives but extensions of core-edge separation protocols which usually use distributed tunnel routers to maintain mappings between addresses used in core networks and that in edge networks. Therefore, these tunnel routers and mapping mechanisms are naturally employed to store and distribute binding cache for mobility support. All the protocols above use a distributed way to store, propagate and update binding cache of MNs and normally work in global scope. Therefore, as mentioned above, such approaches can be quite complex and costly facing the rapid increasing of mobile users in the Internet.

B. Host-based Mobility Protocols

Host-based mobility protocols address triangle routing by fully placing mobility functions on end hosts. MIPv6 has offered such a solution called Route Optimization (RO) mode [2]. In RO mode, MN sends its new binding cache directly to CN after moving to another network. The main drawback of RO mode is its complex validation procedure between MN and CN for security reasons, as both sides have no pre-knowledge to validate each other's identity. Host Identity Protocol (HIP) [8] addresses the security issue by introducing an identity layer into host protocol stack, which makes identity authentication an inherent feature. HIP provides mobility support by keeping host identity to IP address mappings in a similar end-to-end way. Identifier-Locator Network Protocol (ILNP) [9] is a recently standardized protocol with similar goals. It offers light-weight security checking and avoids the complexity from encryption and decryption in HIP. These analogous protocols share the same problem: even if the MN is moving within a small area, the movement events must be propagated to the CN side, which can be quite inefficient in many cases.

C. SDN-related Mobility Protocols

Researchers have begun studying on how to offer better mobility support under SDN architecture. Yap et al. [16][17] proposed OpenRoads to improve robustness during mobility handover using multicast in Openflow networks. They showed how this is achieved by demonstration and also described their testbed deployment. In the following paper [18], they further abstract their idea as separating wireless services from infrastructures and rename OpenRoads to Openflow Wireless which serves as a blueprint for an open wireless network. Our focus in this paper differs from the research above: we focus on improving basic IP mobility functions commonly adopted by existing protocols, while they paid more attention to adding new features, such as multicast, to basic mobility functions.

Pupatwibul et al [19] proposed to enhance Mobile IP networks using Openflow, which share similar goals to us. However, as we will show, they only proposed one possible way to solve the problem, which may not be optimal in many scenarios, while in this paper, we will abstract the problem and give a general discussion to seek for the best solution.

III. PROTOCOL DESIGN

In this section we describe our protocol design. Note that though our design in this paper is based on Openflow, we believe that it can also be implemented in a similar way using other techniques that realize the idea of Software Defined Networking.

A. Protocol Description

Like all the other mobility protocols, we assign a stable identifier to each MN. We also call the identifier Home Address (HoA), but HoA here is different from that in Mobile IP, as HoA in our solution is non-routable and should belong to a specific address block. In this way, we can use a MN's HoA to lookup its up-to-date location, other than to reach its home network which leads to potential triangle routing. A MN's location is represented by Care-of Address (CoA), which is not owned by MN, but its first-hop openflow switch. It means MNs never require to re-configure IP addresses when attaching to new networks, but the network side helps to accomplish the work, which is similar to PMIPv6 [13]. CoAs are routable addresses and thus are used to reach MNs when they are moving around.

Openflow controller is responsible for maintaining binding cache which maps a MN's HoA to CoA. For each MN, a subset of openflow switches in the network serve as indirection agents for the MN. They store replica of the MN's binding cache in the form of flow table which is downloaded from the controller, and redirect packets toward the MN according to the flow table.

To describe details of the protocol, we illustrate how CN reaches MN in Fig.1, in both communication initiation (the left figure) and handover (the right figure) procedures. We assume that HoA of MN and CN are IP_M and IP_C respectively. When switch S3 detects the attachment of MN, it learns the MN's HoA, assigns a CoA IP_S3 to the MN, and then sends a *Binding Update* message which contains a (IP_M, IP_S3) tuple to its controller. The controller stores the binding locally and immediately downloads a flow table entry to S3 which indicates "for all packets with destination address IP_S3, rewrite their destination addresses to IP_M".

First we describe the communication initiation process. We let CN appear and begin communication with MN. Since CN only knows HoA of the MN, the destination address in the packets it sends to MN is IP_M. When CN's first-hop switch S1 receives such a packet, it learns IP_M is non-routable, and there are no local flow tables that match the address. Thus it forwards the packet to its controller via *Packet-in*. The controller (for simplicity, here we assume the controller of S1 and S3 are the same one, and the case with multiple controllers will be discussed in the following subsection) looks IP_M up in local binding cache table and gets the corresponding CoA. Then the controller forwards the packet to S3 via *Packet-out* and at the same time places the binding cache to S1 by downloading a flow table entry indicates "for all packets with destination address IP_M, rewrite their destination addresses to IP_S3". Then the following packets can flow directly from CN to MN: first they are redirected from S1 to S3, and then from S3 to the MN. Note that all the operations above are realized by the network side and are transparent to the end hosts.

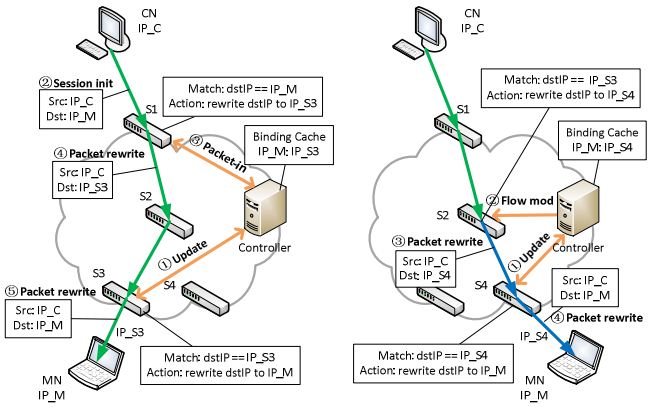

Fig. 1. Protocol overview: how to initiate communication between CN and MN (left figure) and how to handle movement of MN (right figure)

Next we describe the handover process. We assume the MN leaves S3 and attaches to S4. Similarly, detecting the attachment, S4 assigns IP_S4 to the MN and sends Binding Update to the controller. Controller receives the update and learns that the MN has just moved, thus it is responsible for accomplishing the handover by modifying existing CN-to-MN flow path towards MN's new location. Take the scenario in Fig.1 as an example: since the controller knows how the flow goes from CN to MN, it places the new binding cache to S2 by downloading a flow table entry to S2 that indicates "for all packets with destination address IP_S3, rewrite their destination addresses to IP_S4". Then the new CN-to-MN flow will go through three redirections: S1 to S2, S2 to S4 and S4 to MN. For more protocol details, we refer the readers to Section V where the protocol implementation is described.

Of course, Fig.1 only shows one possibility to place the new binding cache to accomplish the handover, and there may exist various ways to place the binding cache in practice, especially in more complicated scenarios. We regard the binding cache placement algorithm an important component of our protocol design since it is closely related to the protocol performance. We will further discuss this issue in Section IV.

B. Discussion

Considering that the scenario shown in Fig.1 is quite simple, in this subsection we will discuss cases that are more complicated to complement our protocol design.

1) Multiple controllers

If CN and MN are located far apart from each other, or located in different domains, it is possible that the controllers of their first-hop switches are different. If the two controllers belong to the same administrative domain, the problem may become simpler since intra-domain communication between controllers is more common. If the two controllers belong to different administrative domains, inter-domain interactions between controllers are required, which may bring larger cost comparing with the intra-domain case. Specifically, inter-domain communication initiation between MN and CN is less costly than inter-domain handover of MN, because the former only requires a query and response of binding cache and is easier to handle, but the later requires the controller to know the CN-to-MN flow path, which is not a pre-knowledge of the controller in inter-domain case.

To deal with inter-domain handover, we propose the following solution. First we argue that inter-domain handover is not common in practice. There are two possible ways to trigger an inter-domain handover: in one case the MN moves for a long distance and then leaves one domain and enters another, which can be quite infrequent; in the other case the MN switches between different providers (e.g. different Wi-Fi or 3G networks) without long-distance movement. For the second case, we can further reduce its occurrence probability by extending one SDN domain so that it contains heterogeneous local networks. In such an extended SDN domain, MN's switching among different access networks is converted to an intra-domain handover.

However, though infrequent, inter-domain handover is unavoidable. To improve inter-domain handover efficiency, we allow the protocol to temporarily fall back to triangle routing which only requires one flow table downloading to the MN's previously attached switch. After MN-CN communication is restored, further operations can be performed to optimize the path between MN and CN.

2) Dual mobility

In the previous example, CN stays immobile all the time. But in practice both communicating sides can be mobile. When CN is also moving, MN can use exactly the same way to reach CN. Actually both sides are treated equally in our protocol, and we distinguish MN and CN in the previous description only for convenience. Even though both communicating sides move simultaneously, the communication can be restored because the handover algorithm runs in a centralized way.

IV. Binding Cache Placement

In this section we further research into the binding cache placement during MN's handover procedure. Theoretically any switch on the CN-to-MN flow path before MN's movement (e.g. S1, S2 and S3 in Fig.1) can serve as a candidate switch, which we call Target Switch (TS). However, choosing some TS may lead to serious performance drawbacks. For example, it is a straightforward idea to choose MN's first-hop switch before movement (e.g. S3 in Fig.1) as TS, but this method will result in triangle routing in most cases. Another idea is to choose CN's first-hop switch (e.g. S1 in Fig.1) as TS, but this method may result in a large number of flow table downloading and high handover latency, which is analogous to the end-to-end binding update method adopted by host-based mobility protocols. Therefore, we regard it worth studying on the binding cache placement problem. In the following of this section we formalize the problem and give our solutions.

A. Binding Cache Placement Problem

To formalize the problem, first we need to make clear goal of the binding cache placement algorithm. Considering that we have various performance metrics to evaluate a handover process, we give the following three goals:

Goal 1: keep optimal forwarding path. This goal ensures the shortest forwarding data path between MN and CN and avoids triangle routing.

Goal 2: minimize the distance between MN and TS. The purpose of this goal is to localize the signaling caused by MN's mobility events. It is reasonable to infer that longer MN-TS distance also implies that the controller of MN's first-hop switch may be located farther away from TS, making handover latency larger. Besides, to download flow table to a distant switch also increases the possibility to trigger inter-controller communications, which further reduces handover efficiency. On the contrary, small MN-TS distance is helpful to confine mobility signaling within a limited area and ensure an efficient handover.

Goal 3: minimize flow entry downloading per movement. This goal can help to both limit the mobility-related flow table maintained on switches and reduce the signaling overhead introduced by flow table downloading.

Given these goals, we define a general *Binding Cache Placement Problem (BCPP)* as an optimization problem: Given a set of TS to place the binding cache for a MN, BCPP problem is to find a subset of the switches which optimizes some goals.

However, the proposed goals conflict with each other in many cases, e.g. selecting MN's first-hop switch before movement as TS will always satisfy Goal 3 but has a large possibility to conflict with Goal 1. Thus we further specify BCPP into the following two problems:

BCPP-1: BCPP that takes Goal 2 as optimization objective and Goal 1 as constraint.

BCPP-2: BCPP that takes Goal 3 as optimization objective and Goal 1 as constraint.

As we will show in the following, BCPP-1 is easier to solve while BCPP-2 is more difficult. However, under certain circumstance, solutions to both problems are identical, which means Goal 2 and 3 can be optimized at the same time. Note that Goal 1 serves as the constraint and thus is always satisfied.

B. Problem Formalization and Solution

1) BCPP-1

Before formalizing BCPP-1, we first assume that during a handover procedure, MN moves from switch s_n to $s_{n'}$ and CN stays attaching to switch s_1 as shown in Fig.2. Then we give these definitions:

Definition 1:

$\boldsymbol{path_{prev}}$ is defined as a set of switches on the MN-CN path before movement of MN, e.g. $\{s_1, s_2, ..., s_i, ..., s_n\}$ in Fig.2.

$\boldsymbol{path_{current}}$ is defined as a set of switches on the MN-CN path after movement of MN, e.g. $\{s_1, s_2, ..., s_i, ..., s_{n'}\}$ in Fig.2.

Path Pair is a $(path_{prev}, path_{current})$ tuple.

Switch s satisfies path pair p means after placing the binding cache (of the MN) on s, the new MN-CN path $path_{new}$ equals to $path_{current}$. This ensures Goal 1, i.e. optimality of the forwarding path (no triangle routing).

Then we formalize BCPP-1 as:

Problem 1: Given each path pair p, find a switch s which satisfies p and at the same time minimize its distance to the MN.

The solution to Problem 1 is relatively simple. First we give another group of definitions:

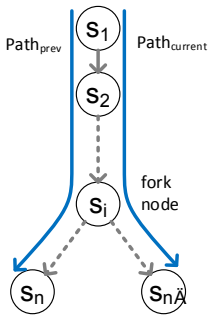

Fig. 2. This figure helps to demonstrate Definition 1, 2 and 3: $path_{prev}$ consists of nodes from s_1 to s_n, while $path_{current}$ consists of nodes from s_1 to $s_{n'}$; and s_i is the fork node.

Definition 2:

Satisfactory Switch Set C_p for path pair p is defined as $\forall\, s \in C_p$, s satisfies p, e.g. $\{s_1, s_2, \ldots, s_i\}$ is a C_p for path pair $(path_{prev}, path_{current})$ in Fig.2.

Fork Node of path pair p is defined as the node where two paths of the path pair "forks", e.g. s_i is the fork node of path pair $(path_{prev}, path_{current})$ in Fig.2.

Then we give our solution:

Algorithm 1: given a path pair p, find its fork node s.

The complexity of the algorithm is $O(d \cdot n)$ where n represents number of path pairs and d represents length of the path. Given a path pair p, switch s generated by Algorithm 1 both satisfies p and minimizes MN-TS distance. Related proof is omitted here due to the page limit and can be found in our technical report [24].

2) BCPP-2

We give more definitions and then formalize BCPP-2:

Definition 3:

Switch s satisfies a set of path pairs P means $\forall\, p \in P$, s satisfies p.

A set of switches S satisfies a set of path pairs P means $\forall\, p \in P, \exists\, s \in S$ s.t. s satisfies p.

Problem 2: Given a set of n path pairs P, find the smallest set of switches S which satisfies P.

We describe solution to Problem 2 using two steps:

Step 1: for each path pair p, find the largest satisfactory switch set C_p;

Step 2: find the smallest set S s.t. for each C_p, $S \cap C_p \neq \emptyset$.

The complexity of Step 1 is $O(d \cdot n)$. Step 2 can be reduced by *Set Covering Problem* which is NP-hard, thus Problem 2 is an NP-hard problem. We proved NP-hardness of Problem 2 in our technical report [24].

We try to solve Problem 2. Obviously, we can use exhaustive search to get the optimal result, but its complexity is $O(d^n)$ and is unacceptable. Therefore we need to find heuristic algorithms. We find that under certain circumstance, Problem 2 can be solved using a simple algorithm. We describe the circumstance as:

Assumption 1: two paths to the same destination share identical "suffix" after they "meet".

Assumption 1 is satisfied as long as packet forwarding between MN and CN only relies on destination IP address, which is a common case in current intra-domain scenarios. With this assumption, we propose our solution to Problem 2:

Algorithm 2: Find the fork node s_i of each path pair $p \in P$, $S = \cup\{s_i\}$.

Actually Algorithm 2 is the same as Algorithm 1 except that Algorithm 2 works on a set of path pairs. Algorithm 2 takes $O(d \cdot n)$ to get the **optimal** result. We proved optimality of Algorithm 2 in our technical report [24].

Note that Algorithm 2 also optimizes Problem 1. Thus if Assumption 1 is satisfied, Algorithm 2 can generate optimal results for both Problem 1 and 2, which means we can simultaneously achieve all three goals.

When Assumption 1 cannot be satisfied, we give another algorithm to solve Problem 2:

Algorithm 3: Greedy Set Covering

Step 1: let $X = P$, $S = \emptyset$.

Step 2: repeat the following process until $X = \emptyset$: find i s.t. S_i contains the largest number of elements in X, then let $S = S \cup \{s_i\}$, $X = X \backslash S_i$.

According to existing research [20], Greedy Set Covering algorithm takes $O(d \cdot n^2)$ to get a result with approximation ratio $\ln n + 1$.

Note that Algorithm 3 also generates optimal result when Assumption 1 is satisfied. The proof is omitted in this paper.

C. Evaluation

In this subsection we make an evaluation of the previously proposed algorithms to see how they perform in real network topologies. Since it is difficult to get real intra-domain routing data which conflicts with Assumption 1, we make our evaluation under Assumption 1 and use real intra-domain topology with shortest path routing to evaluate Algorithm 2. We compare Algorithm 2 with two additional algorithms:

Algorithm-random: for each path pair p, this algorithm randomly selects a switch s which satisfies p as TS.

Algorithm-CN: for each path pair p, this algorithm selects the first-hop switch of CN as TS.

All three algorithms satisfy Goal 1, thus we compare them using metrics from the other two goals: one metric is MN-TS distance, and the other metric is the number of binding cache downloaded per MN per movement.

Our evaluation topology and routing data are calculated using intra-domain topologies from Rocketfuel [21] including AS1221, AS1755, AS6461, AS3257, AS3967 and AS1239. As evaluations based on different topologies show similar results, we choose three of them to demonstrate, and they are AS1221 with 208 nodes, AS3257 with 322 nodes and AS6461 with 276 nodes. To study the performance of our algorithm based on various topologies, we generated another two topologies to add

differentiation: one is a 200-nodes hierarchical topology with a densely inter-connected core network and several tree-like edge networks, and the other is a 200-nodes flat topology in which nodes randomly connect to each other with an average degree.

For each one of the topologies above, we ran the evaluation for 100 turns. In each turn we select a different node in the topology as the MN and 10 randomly located nodes as CNs. The MN performs 10 movements per turn using a modified Markov chain based random walk model: during each movement, the MN randomly attaches to a new node which is one-hop away from its previous location.

Evaluation results are demonstrated in Fig.3. The y-axis represents normalized average MN-TS distance in Fig.3(a), and average number of binding cache downloading per each CN in Fig.3(b). The x-axis in both figures represents the evaluation topology including three intra-domain ones, the hierarchical one ("Hier") and the flat one ("Flat").

We can see that Algorithm 2 has the lowest value in both figures. Fig.3(a) shows that MN-TS distance of Algorithm 2 only takes 10%~20% of the network diameter, which means TS is located about 2 hops away from MN on average, and this offers a good guarantee on the handover efficiency. Algorithm-CN has the largest MN-TS value since it always pushes binding cache to the CN side. The value of Algorithm-random stays between Algorithm-CN and Algorithm 2. We also observe that when evaluation topology becomes more flat, MN-TS values of three algorithms approximate to each other. It is because with the "flatten" of topology, average distance between nodes also drops.

Fig.3(b) shows that, on average value, Algorithm 2 only generates about 0.3~0.5 flow table downloading per each CN in three intra-domain topologies, while the other two algorithms always require downloading one flow table for each CN. Thus Algorithm 2 is quite helpful in reducing the signaling overhead caused by flow table downloading in our protocol, as well as decreasing the number of flow table entries maintained on switches. Again, we find that in flat topologies, Algorithm 2 requires more flow table downloading. Just as the fact that hierarchical topology helps to reduce routing table size, it also helps to reduce binding cache maintenance, so we can say that our protocol performs better in topologies that are more hierarchical.

V. Implementation and Experiment

A. Implementation

1) Implementation environment

We implemented our protocol based on Mininet 2.1.0 [22]. Since we need to simulate host mobility by making hosts change their attachment to different switches, while Mininet does not provide such support, we modified the code to make Mininet allow a host's switching between different switches.

Pox [23] is chosen as the controller in our implementation. We wrote about 300 lines of Python code to implement all mobility-related functions into the controller. Though our implementation is not perfect, it still proves that providing basic mobility support in SDN architecture is not a difficult task.

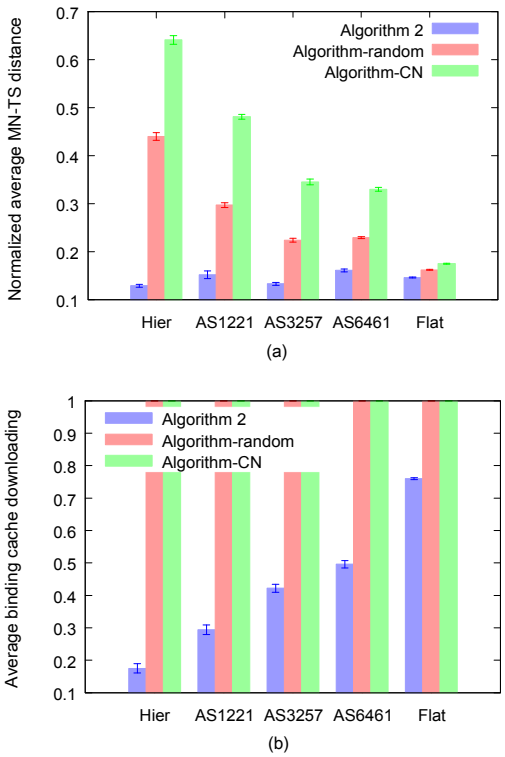

Fig. 3. The two figures show results of comparison among three proposed algorithms based on five different topologies. (a) shows normalized average MN-TS distance, and (b) shows average number of binding cache placed per CN. Algorithm 2 outperforms the other two in all scenarios

2) Protocol flow

We give more details of our implementation by describing protocol flows of both communication initiation and handover procedure, which also complements the description in Section III. Note that Mininet currently does not support layer-3 routing, but our protocol is a network-layer protocol, thus before running the protocol we realize layer-3 routing by pre-downloading static routing tables to all the switches.

We use Fig.4 to demonstrate protocol flow. When switch S3 detects MN's attachment, it assigns a CoA to the MN and registers (HoA, CoA) tuple on the controller using *port-status* message. Receiving registration from S3, controller stores the binding cache locally and downloads a "rewrite" flow entry (the same as that in Section III) to S3 using *flow-mod* message.

When switch S1 receives a packet towards an unknown host, it sends the packet to controller using *packet-in* message. Upon receiving packet-in, controller rewrites its destination IP address and resends it out using *packet-out* message. At the same time controller downloads MN's binding cache to S1 using flow-mod message. In order to generate the path pairs used in the binding cache placement algorithm, controller needs to keep a record of all the switches that have stored the MN's binding cache, which we call *Path Pair Record (PPR)*. For example, in this case, when controller downloads MN's binding cache to S1, it adds one entry into PPR indicating "S1 has stored MN's binding cache". When the binding cache on S1 expires, S1 will acknowledge controller, and then controller will delete the related entry in PPR.

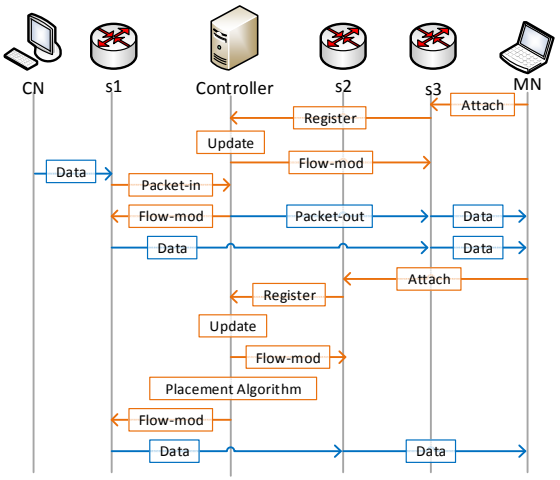

Fig. 4. Protocol flow of both communication initiation and handover procedure in our implementation

After MN moves to S2, another similar procedure handles related registration and flow table downloading procedures. To deal with the handover, controller runs the binding cache placement algorithm discussed previously. It looks up in PPR to find all path pairs related to the MN (only one path pair in this case) and obtains a set of switches that require updating. Then controller downloads the MN's up-to-date binding cache to these switches using flow-mod messages. Note that controller also needs to update PPR after this turn of flow table downloading.

B. *Experiment*

1) Methodology

We make several experiments based on our implementation to compare our protocol with another two IP mobility protocols: PMIPv6 and ILNP. We choose these two protocols because they serve as good representatives of the solutions we reviewed in Section II: a network-based protocol and a host-based protocol. To make comparisons, we also implemented another two controllers to realize the basic mobility functions of PMIPv6 and ILNP respectively. PMIPv6 is easier to implement based on Mininet since it is a network-based protocol. But ILNP is more difficult, thus we simulate the protocol in an approximate way: we move the mobility functions from hosts to their first-hop switches.

We use ubuntu-13.04 as our experiment environment. The experiment topology is shown in Fig.5. As we can see, the topology consists of one controller, two hosts, and eight switches. We regard the topology as three inter-connected subdomains: (S7, S2, S3), (S8, S4, S5) and (S6, S1). Delays of inter-subdomain links, intra-subdomain links and "wireless links" (between H2 and attached switches) are 20ms, 2ms and 10ms respectively. Bandwidths of the above three types of links are 100Mbps, 100Mbps and 10Mbps respectively. Since in the current version of Mininet, in-band control between switches and controller is not supported, thus control traffic is out-of-band in our experiment. We make H2 serve as MN and move back and forth between switch S2 and S5, and H1 serve as CN and keep immobile. We ran *Iperf*, which is a commonly used network testing tool, between the two hosts and collect

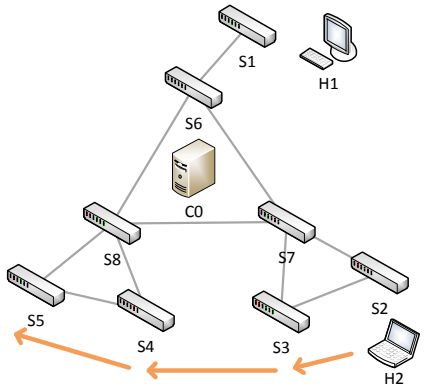

Fig. 5. Experiment topology: H1 keeps immobile, H2 moves back and forth between switch S2 and S5. We use Iperf to test end-to-end performance between H1 and H2 during the movement

end-to-end performance including Round Trip Time (RTT), packet loss rate as well as throughput.

When simulating PMIPv6 in this topology, we use S7 as Home Agent, S2, S3, S4 and S5 as Mobile Access Gateway (MAG), S7 and S8 as Local Mobility Anchor (LMA). H2's moving from S3 to S4 indicates that it leaves its home network and needs to rely on S7 and S8 for packet indirection. When simulating ILNP, each time H2 moves, its first-hop switch will send Binding Update to S1 on behalf of H2, and then S1 handles the update on behalf of H1. Note that our experiment actually favors PMIPv6 and ILNP for two reasons: firstly, IP re-configuration is ignored in the handover process of the two protocols (origin PMIPv6 also needs IP re-configuration when moving between different PMIPv6 domains). Secondly, Binding Update process is simplified and only takes one-way delay: MN-to-HA delay in PMIPv6 case and MN-to-CN delay in ILNP case. Both simplifications help to improve handover efficiency of the two protocols.

2) Results

In the first experiment, we ran Iperf between H1 and H2 for 10 seconds during which period H2 moves from S2 to S5 and performs three handovers. Fig.6 shows collected TCP sequence of PMIPv6, ILNP and our solution within the simulation time, from which we can infer that our solution generates smoother handover than the other two: TCP based on ILNP experiences timeout and slow start during each handover, which makes ILNP performs the worst in the experiment. It is because ILNP needs to send Binding Update from MN side towards CN side, and this may seriously degrade handover efficiency especially when both sides are located away from each other. TCP based on PMIPv6 experiences only one timeout during the second handover, as the other two handovers can be handled locally by LMA, while the second handover is an inter-subdomain handover and requires interactions with HA. In contrast, TCP based on our solution can always recover from packet loss during handover using fast retransmit.

Fig.7 shows RTT of three protocols collected in the same experiment scenario, where we observe that RTT value of all three protocols temporarily raises to a higher value during handover process. Besides, RTT of PMIPv6 stays at a higher value after the second handover. It is because when H2 leaves

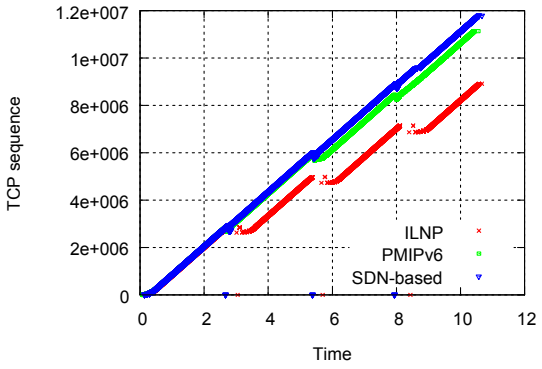

Fig. 6. TCP sequence of three simulated protocols during 3 handover events in 10 seconds, and we can infer that SDN-based protocol generates smoother handover than the other two protocols

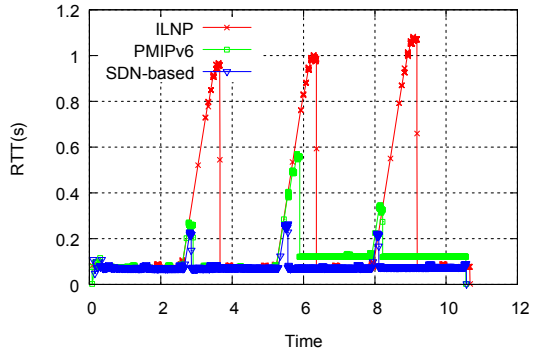

Fig. 7. Round Trip Time (RTT) of three simulated protocols during 3 handover events in 10 seconds, and we observe that RTT of PMIPv6 stays at a higher value after the second handover

home network (after moving from S3 to S4), all packets to H2 are relayed by Home Agent S7 which results in triangle routing. Our solution avoids triangle routing as the binding cache placement algorithm ensures optimal forwarding path, and in this scenario it is achieved by downloading binding cache to S6.

We made another experiment to test average throughput and packet loss rate with different mobility frequencies. Results show that our proposal keeps a high average throughput and low packet loss rate with the growing of mobility frequency, and advantage of our proposal is more obvious when mobility frequency is higher. However, due to the page limit, we refer readers to our technical report [24] for details.

VI. Conclusion and Future Work

In this paper we address mobility in IP network under SDN architecture. We argue that SDN has advantages in handling problems in current mobility protocols because of its programmable devices, centralized control as well as other features. We designed an Openflow-based protocol to realize our idea and implemented it based on Mininet. We compared our proposal with both PMIPv6 and ILNP and results prove advantages of SDN-based mobility support. In future work, we are planning prototyping of our proposal based on real SDN devices and network environment.

Acknowledgment

Supported by the National High-tech R&D Program ("863" Program) of China (No.2013AA010605) and National Science & Technology Pillar Program of China (No.2012BAH01B01). Jun Bi is the corresponding author.

References

[1] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944, 2010.

[2] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275, 2011.

[3] R. Wakikawa, G. Valadon, and J. Murai. Migrating Home Agents Towards Internet-scale Mobility Deployment. ACM CoNEXT, 2006.

[4] M. Fisher, F.U. Anderson, A. Kopsel, G. Schafer, and M. Schlager. A Distributed IP Mobility Approach for 3G SAE. 19th International Symposium on Personal, Indoor and Mobile Radio Communications, (PIMRC 2008).

[5] R. Cuevas, C. Guerrero, A. Cuevas, M. Caldern and C.J. Bernardos. P2P Based Architecture for Global Home Agent Dynamic Discovery in IP Mobility. 65th IEEE Vehicular Technology Conference, 2007.

[6] Y. Mao, B. Knutsson, H. Lu, and J. Smith. DHARMA: Distributed Home Agent for Robust Mobile Access. in Proc of the IEEE Infocom 2005 Conference, March 2005.

[7] L. Zhang, R. Wakikawa, Z. Zhu. Support mobility in the global Internet. In Proc of the 1st ACM workshop on Mobile Internet through Cellular Networks, 2009.

[8] R. Moskowitz, and P. Nikander. Host Identity Protocol (HIP) Architecture, RFC 4423. May 2006.

[9] R. Atkinson, and S. Bhatti. Identifier-Locator Network Protocol (ILNP) Architectural Description, RFC 6740. Nov 2012.

[10] J. Kempf, Ed. Problem Statement for Network-Based Localized Mobility Management (NETLMM), RFC 4830. April 2007.

[11] N. McKeown, et al., OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69-74, 2008.

[12] H. Soliman, C. Castelluccia, K. El Malki and L. Bellier. Hierarchical Mobile IPv6 Mobility Management (HMIPv6). RFC 4140, 2005.

[13] S. Gundavelli, K. Leung, V. Devarapalli, K. CHowdhury and B. Patil. Proxy Mobile IPv6. RFC 5213, 2008.

[14] D. Farinacci, D. Lewis, D. Meyer and C. White. LISP Mobile Node. draft-meyer-lisp-mn-09, Jul. 2013.

[15] R. Whittle and S. Russert. TTR Mobility Extensions for Core-Edge Separation Solutions to the Internet's Routing Scale Problem. Technical report, Rosanna, Vic, Australia, Aug. 2008. [Online]. Available: http://www.firstpr.com.au/ip/ivip/TTR-Mobility.pdf

[16] K. Yap, et.al. Lossless Handover with n-casting between WiFi-WiMAX on OpenRoads. In ACM Mobicom (Demo), 2009.

[17] K. Yap, et.al. The stanford openroads deployment. In ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH), September 2009.

[18] K. Yap, et al. Blueprint for Introducing Innovation into Wireless Mobile Networks. In Workshop on Virtualized Infrastructure Systems and Architectures, pp. 25–32, 2010.

[19] P. Pupatwibul, A. Banjar, AAL Sabbagh and R. Braun. Developing an Application Based on OpenFlow to Enhance Mobile IP Networks. Local Computer Networks (LCN) 2013 Workshop on Wireless Local Networks, 2013.

[20] V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. Mathematics of Operations Research, 1979; 4(3): 233-235.

[21] Rocketfuel: An ISP topology mapping engine. http://www.cs.washington.edu/research/networking/rocketfuel/

[22] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/

[23] POX Controller. http://www.noxrepo.org/pox/about-pox/

[24] Y. Wang and J. Bi. Technical Report. http://netarchlab.tsinghua.edu.cn/~shock/THU-NetArchLab-Mobility-TR-SDN-20140303.pd