MCBS: Matrix Computation Based Simulator of NDN

Xiaoke Jiang, Jun Bi, You Wang
Institute for Network Sciences and Cyberspace, Tsinghua University
Department of Computer Science and Technology, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
Email:{jiangxk10, wangyou10}@mails.tsinghua.edu.cn, junbi@tsinghua.edu.cn

***Abstract*—This paper presents a lightweight matrix computation-based simulator of Named Data Networking (NDN). This simulator treats the experiment network as a whole. Matrix is used to describe network states, and matrix operation is used to simulate different network events. The simulator, just like Newtonian mechanics to some extend, once given the system initial state, including routing table, content distribution, and initial data requesting information of NDN network, can figure out all of the subsequent network state based on matrix computation.**

This simulator splits packet process into different events, such as interest generating, interest forwarding, cache or content hitting and transmitting, and turns them into matrix computation on the network scale.

One of advantages of the simulator is convenient and user-friendly compared to CCNx Project and NS-3 based simulator.

And they can provide similar simulation result. This simulatorcan works on different platform, including Linux, mac OSX and Windows.

Computational complexity of the simulator depends on number of involved contents and nodes, which makes analysis between small group of nodes and contents very simple and fast.

***Index Terms*—NDN, CCN, Simulator**

I. Introduction

Named Data Network (NDN) [1] or Content-Centric Network (CCN) [2], as one of most promising future Internet architecture propose, has drawn a lot of attentions among the research community. NDN introduces some new elements into Layer-3 design, such as named content, cacheable router, and stateful data plane. Those new elements change data transmission pattern from sending data to specific destination with end-to-end pipeline, to retrieving specific data within a unified content bucket.

There is lots of progress in the implementation of NDN. The most famous implementations are CCNx Project and ndnSIM. The former is the official prototype of NDN built by PARC and the latter is the NS-3 based simulator developed by UCLA. However, CCNx Project is complex and is not easy for users to configure and run the experiment scenario. It is even harder to modify the C code to implement the users' own innovation on NDN. That's why we propose a the matrix computation based simulator of NDN as a alternative tool apart from ndnSIM.

In this paper, we build a user-friendly and light-weighted NDN simulator based on matrix computation. This simulator can run on multiple platforms, including Linux, Mac OSX, and Windows, since it relies on MatLab to execute the corresponding code. Compared to our previous work[3], this paper simplifies the core algorithm and clarifies the rationalities which stands behind the matrix computation. As a consequence, the new algorithm is easier to understand and finishes the simulation with less time. We also update latest progress on related NDN prototype and simulators.

II. Related Works

NDN [1] named very content chunk, instead of every host on Layer-3. There are two kinds of datagrams in NDN: Interest and Data, both of which contain the name of content chunk. NDN adopts a pull model to retrieve contents and completely abandons the way of the traditional end-to-end pipeline. The application that needs specific data is called consumer, and the application that originally provides specific data is called producer. As the way to expressing its needs, consumer sends an Interest contained corresponding name; On the other side, producer announce the name prefix of the data it provides into the routing system. NDN Routers forwards Interest according to the name until the Interest meets a Data that contain the same name. During the process, applications request and/or provides data by announcing their names; NDN routers forward and/or aggregate Interests by name, Interest-Data is matched by name, and cached Data is indexed by names.

The most famous and official implementation of NDN is CCNx Project [4] lead by Van Jacobson, who is also the author who propose CCN. They also support NDN on the Open Network Lab (ONL), which contains 14 programmable routers over 100 client nodes. Every node and router runs CCNx Project NDN implementation.

Manuscript received January 9, 2014; revised June 2, 2014; accepted July 3, 2014.

Supported by the National High-tech R&D Program ("863" Program) of China (No.2013AA010605), Jun Bi is the corresponding author of this paper.

doi:10.4304/jcp.9.9.2007-2012

Urbani[5] et al. took a completely different approach. They provide support of Direct code Execution (DCE) for CCNx Project NDN implementation inside NS-3 simulator.

Mini-CCNx[6] adopts container-based emulation and resource isolation techniques to enable hundreds of emulated NDN nodes in a commodity laptop. And the best advantage is that the code you run on Mini-CCNx is the same code which you'll use in a real network. This really adds a realistic behavior to the tests.

CCNx Project is the most official and standard implementation of NDN in C. But It's quit complicated and very hard to modify existing code or add new module. Or we can say it's a standard implementation of NDN with specified and official NDN architecture and mechanism, but it's not a good choice as a simulator to verify new mechanism, which should be scalable and easy to modify and add new modules.

Rossi and Rossini[7] implemented ccnSim to evaluate caching performance of NDN, ccnSim is written in C++ under Omnet++ framework. Muscariello and Gallo[8] from Orange Labs also implemented an NDN simulator, called Content Centric Networking Packet level Simulator (CCNPL-Sim). CCNPL-Sim is based on SSim which implements a simple discrete-event simulator. But ccnSim focuses on caching policy and performance, thus other parts, such as Content Store, Forwarding and routing, are implemented with simplest and unsalable way.

ndnSIM[9] is another NS-3 based NDN implementation led by Lixia Zhang. As a simulator, It's quite convenient to add new modules or improve existing modules. It also packed with different cache updating policies and forwarding strategies ndnSIM is now a popular tool among ICN research community. We also implemented a NS-3 based NDN implementation, which is similar with ndnSIM but focus on NDN protocol itself

III. Methodology

Our simulator does not simulate behavior of individual nodes, instead, we treat the network as a whole. Data set and node set are the two primary entities. Then we use matrix to describe the network states, such as the relationship between data and nodes: a specific data is possibly requested, cached or originally provided by a specific node. And matrix operation is used to simulate packet processing, such as forwarding Interest and cached data hit.

A. Entities

Matrix is employed to describe the network states. There are 2 types of matrix: type A and type B. And Matrix F, P and S describe the FIB, PIT and Content Store respectively. Matrix A describes the contents producer by the host applications and Matrix R describes the request status. Among those matrix, F and P are type B, the others are type A. We list those entities in Table I

TABLE I: Entities

symbol	Meaning
m	Content Number
n	Node Number
Type A	2-Dimension Matrix schema, $c \times n$
Type B	3-Dimension Matrix schema, $c \times n \times n$
Matrix F	Type B, to describe FIB
Matrix P	Type B, to describe PIT
Matrix S	Type A, to describe Content Store
Matrix A	Type A, to describe Data provided by Application
Matrix R	Type A, to describe current requests
Matrix Q	Type A, to describe all requests
Matrix H	Type A, to describe Hit with cache/application
Matrix T	Type A, to describe the temporary transmission

We give more explanations here:

• C: content set, for example, {c1, c2} □ C, and m = |C|

• V: host (node) set, for example, {v1, v2, v3, v4} □ V , and n = |V |

□• E: edge set, for example, {e1, e2} □ E.□And physical NDN network can be described with G =<V,E,C>.□

Schema Type A Matrix is shown in Figure 1 and Type B Matrix is shown in Figure 2.

$$A = \begin{array}{c} \\ c_1 \\ c_2 \\ \dots \\ c_m \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & \dots & v_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}$$

Fig. 1: Schema of Type A Matrix

Type B: There are also some user defined matrix operations

• A $\bigotimes$ B, $\bigotimes$ is operation between type A Matrix and type B Matrix. Let T = R $\bigotimes$ F, t_{ij} is element of matrix T at row i column j, then

$$t_{ij} = \sum_{k=1}^{j} r_{ik} \cdot f_{ikj}$$

Or, as result shown in Figure 3.

• A $\odot$ A or A $\odot$ B. let T =H $\odot$ S, then

$$t_{ij} = h_{ij} \cdot \vec{s_{ij}}$$

Or, as shown in Figure 4

$$t_{ij} = q_{ij} \cdot \vec{s_{ij}}$$

When S is type B matrix, the computation rule is the same with the above, but change s_{ij} to be a row vector.

$$T = \begin{pmatrix} q_{11} \cdot s_{11} & q_{12} \cdot s_{12} & \dots & q_{1n} \cdot s_{1n} \\ q_{21} \cdot s_{21} & q_{22} \cdot s_{22} & \dots & q_{2n} \cdot s_{2n} \\ \dots & \dots & \dots & \dots \\ q_{m1} \cdot s_{m1} & q_{m2} \cdot s_{m2} & \dots & q_{mn} \cdot s_{mn} \end{pmatrix}$$

Fig. 4: Schema of Result: $\odot$

• *A* $\bigoplus$ *B Let T = R* $\bigoplus$ *F , then*

$$t_{ijk} = r_{ik} \cdot f_{ikj}$$

B. Algorithm

The algorithm is described here in Algorithm 1.

Note that, operator "|" is matrix xor. Let T = S|A, then

$$t_{ij} = s_{ij} \mid a_{ij}$$

And operator "+" is matrix addition, let T = F + P, then,

$$t_{ijk} = f_{ijk} + p_{ijk}$$

matrix xor and matrix addition are both supported over Mat- Lab platform.

The algorithm is implemented on MatLab platform. What's more, cache_update function is implemented with Least Frequently Used (LFU) and Least Recently Used (LRU), and interest_generate function is implemented with zipf distribution

IV. Simulation

Our simulator is kind of discrete events system, it resolves all the NDN handling process to different events then arranges events step by step. We also implement NDN on NS3 platform and compare the result of NS3 based simulator and matrix computation based simulator. In next section, we will compare the result.

In the simulation, the topology is a real intra-AS network shown in Figure 5. We assume that there are 5 content producers and one content consumer under each router. There are 10000 contents in total, which are distributed equally among the producers. Cache size is 1% of the whole contents amount, which is 100, and cache updating policy here is LRU. We use a real intra-AS network as the topology for simulation, as shown in Figure 5, which includes 21 routers numbered from 1 to 21. There are 5 content producers who connect to the network with router 2, 6, 10, 12 and 16 respectively. The mobile, a moving Producer, is producer connecting with router 12. There is one content consumer connecting each

$$B = \begin{array}{c} \\ c_1 \\ c_2 \\ \dots \\ c_m \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & & & \end{array} \\ \begin{pmatrix} b_{111} & b_{112} & \dots & b_{11n} \\ b_{211} & b_{212} & \dots & b_{21n} \\ \dots & \dots & & \\ b_{m11} & b_{m12} & \dots & b_{m1n} \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} v_2 & & & \end{array} \\ \begin{pmatrix} b_{121} & b_{122} & \dots & b_{12n} \\ b_{221} & b_{222} & \dots & b_{22n} \\ \dots & \dots & & \\ b_{m21} & b_{m22} & \dots & b_{m2n} \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} \dots & & & \end{array} \\ \begin{pmatrix} \dots & & & \\ \dots & & & \\ \dots & & & \\ \dots & & & \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} v_n & & & \end{array} \\ \begin{pmatrix} b_{1n1} & b_{1n2} & \dots & b_{1nn} \\ b_{2n1} & b_{2n2} & \dots & b_{2nn} \\ \dots & \dots & & \\ b_{mn1} & b_{mn2} & \dots & b_{mnn} \end{pmatrix} \end{array}$$

Fig. 2: Schema of Type B Matrix

$$T = \begin{pmatrix} \sum_{k=1}^{n} r_{1k} \cdot f_{1k1} & \sum_{k=1}^{n} r_{1k} \cdot f_{1k2} & \dots & \sum_{k=1}^{n} r_{1k} \cdot f_{1kn} \\ \sum_{k=1}^{n} r_{2k} \cdot f_{2k1} & \sum_{k=1}^{n} r_{2k} \cdot f_{2k2} & \dots & \sum_{k=1}^{n} r_{2k} \cdot f_{2kn} \\ \dots & \dots & \dots & \dots \\ \sum_{k=1}^{n} r_{mk} \cdot f_{mk1} & \sum_{k=1}^{n} r_{mk} \cdot f_{mk2} & \dots & \sum_{k=1}^{n} r_{mk} \cdot f_{mkn} \end{pmatrix}$$

Fig. 3: Schema of Result: $\otimes$

Algorithm 1 NDN Node Model: pseudocode shown here is the skeleton of the algorithm, the implementation in practice needs log information, input data etc.

Require: F, P, A, S, R as input

1: **while** True **do**
2: $H = Q \bigodot (S|A|T)$
3: **if** $H \neq \vec{0}$ **then**
4: $T = H \bigotimes P$
5: $S = cache\_update(S, T)$
6: $P = H \bigodot P$
7: **end if**
8: $P = R \bigoplus F + P$
9: $R = R \bigotimes F$
10: $R = interest\_generate(R)$
11: $Q = Q|R$
12: **end while**

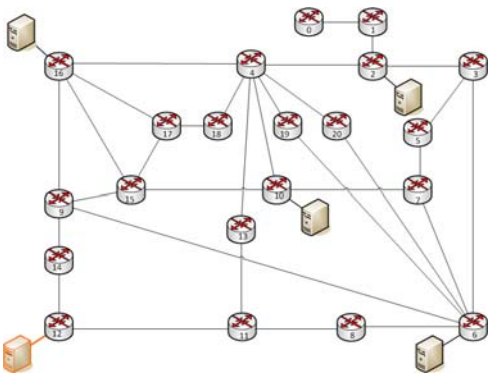


Fig. 5: Topology Used in the simulation, a real intra-AS topology

router, which is representative of all the end hosts connecting to that router. There are 10000 contents numbered from 0 to 9999. Contents are distributed equally among the producers. We expect the ICN traffic model has the same features as those of the web traffic model which prior studies have exhibited. [10]. shows that the request rates of HTTP objects follows the Zipf-like distribution. The probability of each consumer to request the ith content is P(i). P(i) follows Pareto-Zipf distribution :

$$P(i) = \frac{\Omega}{(i+q)^{\alpha}}$$

Ω is the normalization constant, making:

$$\sum_{i=0}^{9999} P(i) = 1$$

Here α and q are 0.7, i□[0,9999].□

Each consumer requests the contents according to the requesting probability above. We treat 1000 continuous request sequences as one *"turn"*. Note that default cache settings are that cache size is 100 contents, which is 1% of all the contents and cache updating policy is Least Recently Used (LRU).

We also define two index to describe performance of NDN network. one is Average Reply Hops, which is average distance from other original content consumer to the Interest meet cor- responding content measured by router hops, and the other one is Cache Reply Percentage, which is percentage of requests who get corresponding contents from cache copy but not the original content producer.

X-axis is the request sent and 20000 requests are sent in total during whole simulation. Y-axis is value of average request hops (Figure 6a) or that of cache reply percentage (Figure 6b).

The exciting news is that the result of this simulator and the NS3 are very similar, especially when the network reaches a stable state, meaning average request hops and cache reply percentage fluctuate in a small range. What's more, this simulator is a little fast for the network to reach stability and even more stable than NS3, which is because of the discrete algorithm employed by this simulator.

We also change the cache size and cache updating policy, and the results of the two simulator is very similar with each other.

This simulator can easily provide more details, such as the average request hops and cache reply percentage of specified nodes and contents. It can even simulate mobility scenario, only needing to change the Matrix A under user defined condition.

Another great advantage is that this simulator is really easy to use. M-file of MatLab is an interpreted language, which is much easier to read, modify and maintain than C++ code of NS3 or C code of CCNx. What's more, M-file is much more shorter than C/C++ source files.

V. DISCUSSION

A. Precision Evaluation

We call interest generating, pit record timeout, data trans- mission error or fail accident events. In actual operation, accident event can't be predicted and only be found after they happened. So in current model, we haven't taken them into consideration. But we can import some accident event which is random or produced by some rules, if need.

We haven't imported features of data transmission error or fail accident events, which provides the simulation an ideal situation. In actual operation, those accident events can't be predicted and can only be found after they happen. This will be involved in our future work. This simulator currently can only provide an approximate prediction, because accidents may happen when our prediction steps are over. The involved nodes and contents are less; the prediction steps are less, the prediction is more accuracy.

The current model doesn't take time of processing, such as table lookup time and data transmission time, into consideration. So the current model only measures data transmission cost by hops. I think measuring by hops is popular currently, because of the dynamic of network, which lead result of measuring by time changes with network status.

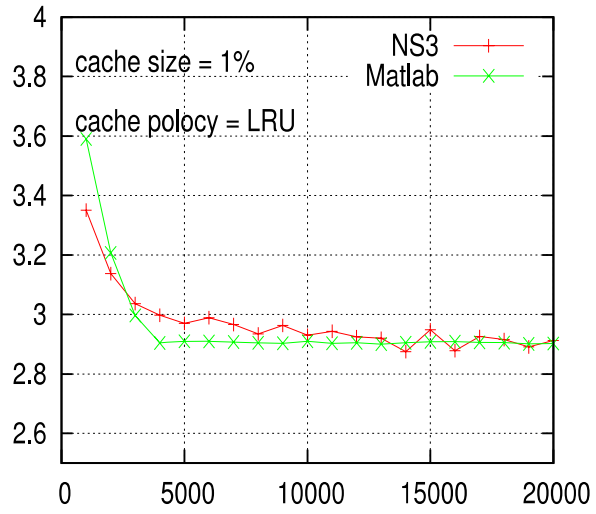
Approximate Prediction: If and only if we know the network status (F, S, R, P, A Matrix) of some specified time. The computation based on our model can only provide a approximate prediction, because accident may happen when our prediction steps is over. The involved nodes and content is less, the prediction steps is less, the prediction is more accuracy.

B. Different with NS-3 based Simulator

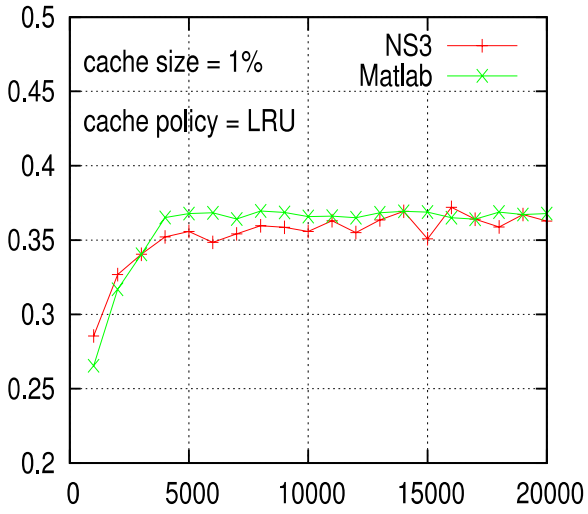
There do be some similarities between our design and NDN simulator. For example, they both should give network topology, Forwarding Table, Content Distribution and Requesting Status first. And they both use a iteration process to drive the networking-processing running. What's more, they both give the result of NDN data communication.

However, our work still has great differences:

1) Matrix Computation vs. Host Simulator: NS-3 simulates every node and its packet processing of the whole network. Our design employs a deeper abstraction by split entities' covering. NS-3 do also need computation to simulate the host and its behavior, otherwise our □design just employs pure computation.
2) Centralized vs. Distributed: Every host of NS-3 finish its □computation so the network runs, we think its distributed control. Our design is single thread computation and the matrixes record needed information, which is centralized control.
3) Formal Description vs. Numerical Solution: NS-3 only gives numerical solution aiming at one specific network assumption, but our design could give a formal description whatever the network assumption is.



(a) Average Request Hops, X-axis donates to number of sent Interests



(b) Cache Reply Percentage

Fig. 6: NS3 vs. MatLab

There are some similarities, such as basic input and output, but the intrinsic mechanisms and coverage area are totally different. Our design is a higher abstraction of NDN.

C. Scalability

There are billions of nodes and contents in the network. If we include all nodes and contents into computation, this would make a disaster. The number of nodes and contents depends on the problem to be solved. In general, we should only take related nodes and contents into consideration, which may greatly decrease complexity and scale of computation. For example, if we only want to know how node n gets content c. First we should get all the nodes involved. If N hops can reach the original source, we only need to get nodes limited in N hops and restrained by F Matrix. If an interest is flooded to all neighbors, the node set may have lots of element. But if not, the node set may be small.

Currently, the speed of our implementation is at 0.5 seconds per step when there are 10000 contents and 21 nodes. It's not very far, but there should be a lot of room for improvement, however most matrixes are sparse and there are a lot of optimization algorithms for sparse matrixes computation.

D. Covering Areas

The current implementation aims at NDN, but this design can describe the current IP network after some changing. What's more, It's even easier, for the one packet can only has one next hop to send. The difference is that NDN data packet is sent back following the P Matrix, but IP packet only routing by F Matrix. So there should some modification.

E. Limitation

Frankly speaking, our work is also on preliminary stage, it couldn't provide a lot of scenario settings, which is neither our goal. Our goal is to provide a very easy to use simulator to users to verify their creative idea with little cost and as quick as they can. But professional simulation and deeply analysis need advance implementation, such as CCNx Project.

VI. Conclusion

We present an alternative NDN simulator based on matrix computation. This simulator treats the experiment network as a whole. Matrix is used to describe network states, and matrix operation is used to simulate different network events. This simulator can figure out all the subsequent network states after given an initial condition. What's more, our simulator employs a centralize control algorithm instead of simulating behavior of individual nodes, and changes node operation processes to pure matrix computation which is quite precise under ideal conditions and friendly to users.

Our work makes the network computable. We can infer how interest and data trace to the network. Base on those computation, we can analysis the whole network performance, the subset network performance and one host connectivity performance. We also can make decision on how to adjust Forwarding Table.

But honestly, there is still a lot of work to be done in order to make the simulator faster and fit other scenarios.

References

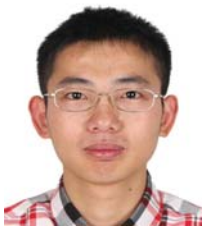
[1] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J.D. Thornton, D.K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al. Named data networking (ndn) project. Technical report, Tech. report ndn-0001, PARC, 2010.

[2] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies. ACM, 2009.

[3] X. Jiang, J. Bi, Y. Wang, P. Lin, and Z. Li. An easy matrix computation based simulator of ndn. In the Third International Conference on Networking and Distributed Computing (ICNDC 2012), Hangzhou. IEEE, 2012.

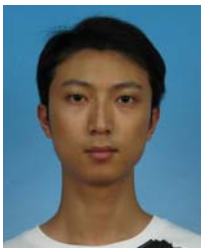
[4] CCNx Project. http://www.ccnx.org.

[5] NS3 DCE CCNx Project. http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/getting-started.html.
[6] Carlos Cabral, Christian Esteve Rothenberg, and Maurício Ferreira Magalhães. Mini-ccnx: fast prototyping for named data networking. In Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking, pages 33–34. ACM, 2013.
[7] Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). Relatório técnico, Telecom ParisTech, 2011.
[8] CCN Packet Level Simulator (CCNPL-Sim). http://code.google.com/p/ccnpl-sim/wiki/Installation.
[9] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim: Ndn simulator for ns-3. http://irl. cs. ucla. edu/ndnSIM. html.
[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Com- munications Societies. Proceedings. IEEE, volume 1, pages 126–134. IEEE, 1999.



Xiaoke Jiang is born in 1987. Now he is a Ph.D. Candidate at Tsinghua University. His research interest is Named Data Networking, especially the transport functionality, such as optimal chunk size, mobility support, and Interest aggregation.

Jun Bi is born in 1972, post-doctoral, and professor at Tsinghua University. His research interests include Next Generation Internet Architecture and Protocols, High Performance Routers and Switches, Source Address Validation, Internet Routing, IPv4/IPv6 Transition, etc. He is the corresponding author of this paper.



You Wang is born in 1988. Now he is a Ph.D. student at Tsinghua University. His research interests include IPv4/IPv6 transition, IP mobility management protocols, Future Internet architectures and Software Defined Networking.