2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD) | 979-8-3503-4918-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSCWD61410.2024.10580443

A Unified Routing Framework for Resource-Constrained Mobile Ad Hoc Networks

1st Yupeng Zhang
College of Intelligence and Computing
Tianjin University
Tianjin, China
zyp_20211208@tju.edu.cn

2nd Shunkang Hu
College of Intelligence and Computing
Tianjin University
Tianjin, China
hsk6706@tju.edu.cn

3rd Zenghua Zhao*
College of Intelligence and Computing
Tianjin University
Tianjin, China
zenghua@tju.edu.cn

Abstract—**In resource-constrained mobile ad hoc networks(MANETs), routing protocols are usually implemented inside mobile nodes for resource saving and fast response to topology changes. However, existing routing protocols are implemented in either user space or inside the kernel of operating systems, which suffers from sophisticated system-level programming and inefficient routing management. In this paper, we design a unified routing framework integrating implementation and management for both proactive and reactive routing protocols in MANETs. Under the framework, we provide a series of application interfaces in user space for implementation of routing protocols, hiding complicated programming details in the kernel. In this way, both types of routing protocols are able to be implemented just in the user space thus significantly reducing implementation efforts. Moreover, through interactive terminal multiple routing protocol routines can be managed and setup in a unified pattern. To demonstrate the feasibility of the unifed routing framework, we implement its components on Linux 5.4 kernel. Furthermore, a well-known routing protocol AODV is implemented under the framework as an example. The source codes are publicly available for further research in the community.**

Index Terms—**Mobile Ad Hoc networks, routing architecture, proactive routing protocol, reactive routing protocol**

I. Introduction

Mobile ad hoc networks (MANETs) have been applied in many fields of Internet of Things, such as unmanned autonomous vehicles [1], drone swarms [2], and ocean monitoring [3], [4]. However, many MANETs are resource-constrained in terms of physical bandwidth, computing capability, and power supply. At the same time, their network typologies vary over time due to the node mobility, which requires fast response from routing protocols. In this case, although software-defined networking (SDN) architecture [5] [6] has been deployed in Internet and emerging in some MANETs [7]–[9], per-mobile-node routing is still a practical solution in resource-constrained MANETs.

However, current operating systems do not well support the implementation of routing protocols in MANETs [14]. Existing routing protocols can be classified into two categories: proactive and reactive routing protocols. They are forced to be implemented either in the user space or inside the kernel of the operating systems. In proactive (or table-driven) routing protocols (e.g., DSDV [13]), each mobile node maintains a routing table which contains the information of the routes to all the possible destination nodes. Since routing tables can be updated by collecting network state information periodically in application layer, proactive routing protocols are usually implemented in the user space. Whereas, in reactive (or on-demand) routing protocols (e.g., AODV [10], DSR [11]), a route is discovered upon it is required. Due to many operations in network layer, reactive routing protocols are usually implemented in the kernel, suffering from sophisticated low-level system programming. Different implementation approaches make it challenging to integrate both categories of routing protocols in one framework. In addition, each routing protocol features an individual setup command set, which makes it difficult for a network manager to remember all the commands of the routing protocols and configure the network accurately, increasing the risk of networking faults.

To ease the implementation of the routing protocols, Kawadia et. al. [14] propose an architecture and a generic API. The API is provided as a shared user-space library, the ad-hoc support library (ASL), based on Linux 2.4 Kernel. Some common functionalities in reactive routing protocols are abstracted out and implemented in the kernel. New routing protocols are enabled to be implemented in the user space and interact with kernel functions via ASL. However, it only provides system services facilitating the implementation of routing protocols. An unified development and management framework is still an open issue.

Quagga [15] [16] is a popular routing software package that incorporates common unicast routing protocols for Internet, such as ISIS [17], OSPF [18], and RIP [19]. Quagga provides an integrated user interface shell and supports common client commands for all the routing protocols. New protocol daemons can be added to Quagga easily without affecting any other software. However, it lacks support for routing protocols in MANETs.

In this work, we propose a unified routing framework for routing protocols in MANETS. The goal is providing a customizable and extensible routing software package for MANETs, supporting both proactive and reactive routing protocols. Analyzing the routing functionalities common in MANET routing protocols, we design programming abstractions (APIs) and network components under the framework.

*Zenghua Zhao is the corresponding author.

The network components provide system services implemented in Linux 5.4 kernel, which deliver network information to the routing daemons in the user space via APIs. With the unified routing framework, researchers are able to choose appropriate components to build MANET routing protocols according to their specific requirements, significantly reducing low-level programming efforts. In addition, under the unified framework routing protocols can be setup using the same command set. To demonstrate the viability of the framework we implement AODV routing protocol using the APIs and network components. Our source codes are publicly available for further research [1].

II. Overview of the Unified Routing Framework

As illustrated in Fig. 1, the unified routing framework consists of two parts: the system service set and the application set. The system service set is composed of the network components that provide system services for MANET routing protocols. The applications set includes two types of applications. One is the general application involving MANET routing protocol daemons and the interactive terminal. The other is the agent daemon managing the system services and the general applications. The application set communicates with the system service through the APIs encapsulated within the network components, while the general applications communicate with the agent daemons through the inter-process communication (IPC).

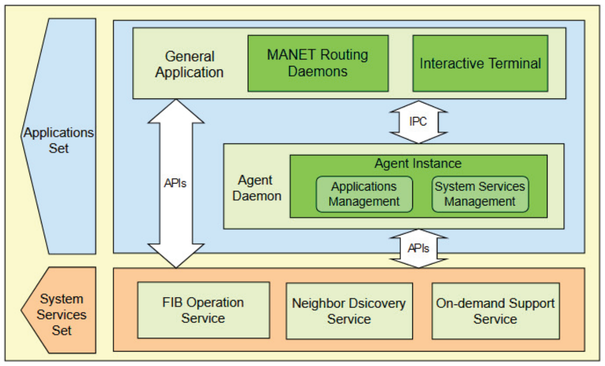

Fig. 1. The unified routing framework.

In the system service set, the primary network components consist of FIB (Forwarding Information Base) operation service, neighbor discovery service, and on-demand support service (ODS). They work in the kernel of the OS, and provide the common functionalities for routing protocols. For example, the FIB operation service offers the kernel space with the forwarding table operation services; the neighbor discovery service provides the service of discovering the neighbors. The network components are independent of each other, allowing for flexible compositions of each component. Currently, the system services can be implemented by two methods in operating systems. One is the library that provides services via APIs, while the other involves background daemons that provide services through the IPC. We implement the system services by libraries due to their efficiency. The network components in the system service set hide the low-level programming details and provide common functionalities to the routing daemons in the user space.

In the applications set, the agent daemon is a special category of applications that run in the background. The daemons take in charge of initializing, configuring, and managing the network components, as well as providing the system services to the general applications. The agent daemon acts as the intermediaries between the system service set and the general applications. The general applications consist of the interactive terminal and MANET routing daemons, which can access system services either directly through the APIs provided by the network components or through the agent daemons.

The routing framework provides a unified software package for both the proactive and reactive routing protocols in MANETs. The two types of routing protocols differ in the implementation, where the proactive routing protocols are implemented in the user space, and the latter in the kernel. Moreover, each routing protocol is set up independently with their own setup commands. Under the unified routing framework, all the routing protocols in MANETS can be implemented in the user space using the system services with the same APIs, without the need of the low-level system programming. In addition, they can be setup by the same command set through the interactive terminal.

III. Design of the Components in the Framework

In this section, we will describe the design of the system service set and application set in detail.

A. *FIB Operation Service*

The FIB operation service facilitates the data exchange between the forwarding table and the routing table. We differentiate the two tables here. The forwarding table works in the kernel according to which a packet is forwarded to the next hop. Whereas, the routing table is maintained by the routing daemon in the user space. Apart from the information required in the forwarding table, the routing table involves other data assisting the routing algorithm, such as the cost of a link.

When the routing table is updated by the routing protocol, its data has to be transferred to the forwarding table in time for accurate operation. Similarly, when the forwarding table is updated in the kernel, its information has to be delivered to the routing table notifying the routing daemon.

To facilitate the data exchange between the forwarding table and the routing table, we design the high-level programming abstractions (APIs) in the user space. The APIs are designed as follows:

- `int fo_add_a_route(uint32 dst, uint32 gateway, int ifindex, int metric, uint32 src);`
- `int fo_delete_a_route(uint32 dst, uint32 gateway, int ifindex, int metric, uint32 src);`

[1]https://github.com/WirelessGroupTJU/UnifiedRoutingFramework

The two functions are used to add and delete the forwarding table entries. The function parameters correspond to the key attributes of the data structure of the forwarding table. The parameter "dst," "gateway," and "ifindex" are the basic items in a forwarding table, while "metric" and "src" are optional to meet the requirements of some specific routing protocols, such as the source routing.

- `int fo_get_routes(route_entry* table, int* len);`

The function gets the route entries in the forwarding table and store them in the `struct route_entry`, which data structure is:

```
struct route_entry{
  uint32 dst;
  uint32 gateway;
  int ifindex;
  int metric;
  uint32 src;
  int table;
};
```

B. Neighbor Discovery Service

Current operating systems have system services for neighbor discovery, such as the Address Resolution Protocol (ARP) [20] for IPv4 and the Neighbor Discovery Protocol (NDP) [21] for IPv6. ARP operates between the network layer and the link layer while NDP resides in the network layer. Although these protocols provide the neighbor information, they work in the kernel and can hardly be customized to meet the requirements of the specific applications. Therefore, we design the neighbor discovery mechanism in the user space to facilitate the customization, scalability, and maintenance.

Considering the communication link asymmetry caused by the characteristics of the wireless channel, we adopt the neighbor discovery mechanism described in RFC 6130 [22]. This mechanism periodically broadcasts information about known neighbors and inform them of the local node's neighbor information. It also maintains the link status and link symmetry information of each neighbor. The APIs ofthe neighbor discovery module are give by:

- `void nd_get_one_hop_neighbor_information (char* name, vector<pair<uint32,uint8>>& neighbor_list);`

The function returns to the caller a list of one-hop neighbors discovered on the specified network interface, along with the link status.

- `int nd_set_hello_interval(char* name, uint64 new_hello_interval);`

The function sets the neighbor discovery period on the specified network interface. When the discovery period value is set to less than or equal to 0, the network interface stops sending periodic probing packets.

C. On-demand Support Service

In current operating system, when a packet in the network layer fails to match any forwarding rule for the corresponding destination network address in the forwarding table, it will be directly discarded. These packets may originate from the upper-layer applications of the local node or from neighboring nodes requesting the local node to act as a relay. However, these discarded packets are the source of route demand in reactive routing protocols. Therefore, the on-demand service can be described as follows:

1) Caching packets that fail to match any forwarding rules.
2) Analyzing the source of the packets and generating route requests for the packets.
3) Informing the corresponding routing daemon about the route requests for further route computation
4) Processing the cached packets based on the computation results from the routing daemon.

we add some sub-components to the operating system to build on-demand services. The on-demand service is illustrated in Fig. 2.

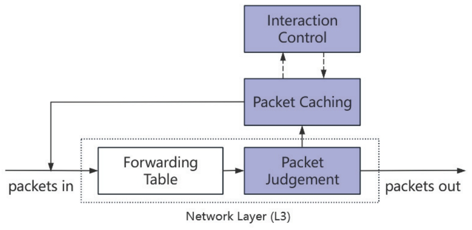

Fig. 2. The diagram of the on-demand service model.

The on-demand service consists of three modules: packet judgement, packet caching, and interaction control. The packet judgement module examines all the packets passing through the L3 layer, identifying packets that cannot match any entry in the forwarding table and hand them over to the packet caching module. The packet caching module caches the packets and generates the route requests, which are delivered to the interaction control module. The interaction control module interacts with the service users, obtains route results, and delivers the route results to the packet caching module. There are two processing methods for cached packets based on the route results: if the route is alive, the cached packets are sent back to the network stack for processing; otherwise, the cached packets are cleared. The following APIs are provided by the on-demand service:

- `int ods_get_route_request(route_request* now_request);`

The function gets the route request, which has the following data structure:

```
    struct route_request { uint32
    dst_ip; uint32 src_ip; unsigned
    char protocol; int input_ifindex;
  };
```

The destination network address, source network address, and protocol are basic information included in the packet header.

The input network device index identifies the input device when the packet enters the network stack. This attribute is primarily used for routing request allocation.

- `int ods_feedback_route_reply (route_reply* reply);`

This function passes the route discovery result to the on-demand service module. The route discovery result has the following data structure:

```
    struct route_reply { uint32 dst_ip;
    int result;
  };
```

When "result" indicates the discovery and establishment of the destination path, the packet is re-injected to the network stack. In contrast, when "result" signifies the absence of a discovered destination path, the packet will be removed from the cache.

D. Agent Daemon

Agent daemon is a special type of applications. It serves as a unified access and management tool for specific local routing services, facilitating development and administration. The specific architecture of Agent Daemon is illustrated in Figure 3.

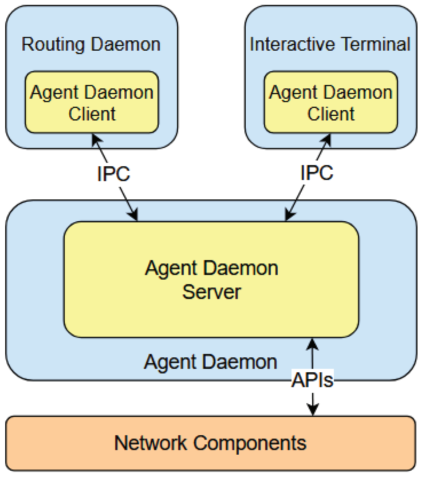

Fig. 3. Architecture of Agent Daemon.

Agent daemon forms a client-server (C/S) architecture with general applications. Within the agent daemon, a server is maintained internally. Meanwhile, a general application wishing to connect to Agent Daemon needs to maintain a client. On a local node, the client and server interact using IPC.

The contents are exchanged through IPC including requests from specific routing daemons, configurations made by network administrators through the interactive terminal on the local node, and the responses provided by the agent daemon to these requests or commands. The interaction through IPC requires a well-defined communication protocol, which can range from simple string-based queries and responses to complex data structures. In our demo, we provide a simple signaling format as a reference.

Agent daemon cannot process requests, but can utilize APIs to initialize and manage network components. Thus, it can use the network components to provide responses to client applications' requests. Therefore, Agent Daemon can be easily customized and tailored, making it highly adaptable, especially in scenarios where computational resources on the nodes are limited.

IV. Implementation in Linux

We have implemented the three network components in Linux 5.4. We will introduce them sequentially in this section.

A. Implementation of FIB Operation Service

The mapping between the routing table and forwarding table needs to be implemented using the inter-process communication mechanisms provided by the operating system. In the current Linux 5.4, there are two mechanisms available for interacting with the kernel forwarding table from user space: Netlink and ioctl. The Netlink mechanism is a bidirectional communication mechanism that conforms to the BSD socket API and is the preferred mechanism for network configuration and management.

In user space, interacting with the forwarding table through the Netlink mechanism requires creating a Netlink socket. The Netlink socket uses the `AF_NETLINK` protocol family, specifically the `NETLINK_ROUTE` protocol. In the signaling interaction, we use the `RTM_NEWROUTE` flag to indicate the command for adding a forwarding table entry. The `RTM_DELROUTE` flag is used to indicate the command for deleting a specified forwarding table entry. The `RTM_GETROUTE` flag is used to indicate the command for retrieving forwarding table entries from the kernel.

B. Implementation of Neighbor Discovery Service

In the implementation of the neighbor discovery module, the first consideration is to specify the wireless network interface to be used when sending neighbor discovery packets. When creating a UDP socket for sending neighbor discovery packets, we use the `setsockopt()` function with the `SO_BINDTODEVICE` parameter to bind the UDP socket to the specified wireless network interface.

Another consideration is the probing interval in the neighbor discovery module. When we set a network interface to send neighbor discovery packets at a certain time interval, the control of the probing interval requires the use of timers. Additionally, each wireless network interface needs to maintain its own neighbor information. If no neighbor probe information is received within a certain time period, the corresponding neighbor's information needs to be removed. These time-limited information entries also require timer control. In the Linux operating system, each process can only use one system timer. To address this, we have adopted a virtual timer queue. The queue is sorted in ascending order based on the timeout time of each timer node, corresponding to the system time.

Only the timer node at the head of the queue utilizes the actual system timer for timing. The remaining timer nodes start timing only when they become the head of the queue.

C. Implementation of On-demand Support Service

As shown in Fig. 4, on-demand service consists of packet judgement, packet caching, and interaction control.

Packet judgement identifies packets in the operating system network layer that do not match any forwarding rules in the forwarding table. We use TUN/TAP technology to create a virtual TUN network device with a specified name. Then, we append a default routing rule to the forwarding table, where the output network device is set to the name of the virtual network device. When a packet in the network layer does not match any other forwarding rules, it will use this default route and be send to the virtual network device. Packet judgement checks the output device of the packet. If the output device name matches the name of the virtual network device, it is the packet that needs to be captured. Our packet judgement module needs to use Netfilter [23] to reserve hooks in the kernel. Since we need to differentiate the packet source, the judgement module mounts the judgement function on the `NF_INET_FORWARD` and `NF_INET_POST_ROUTING` hooks.

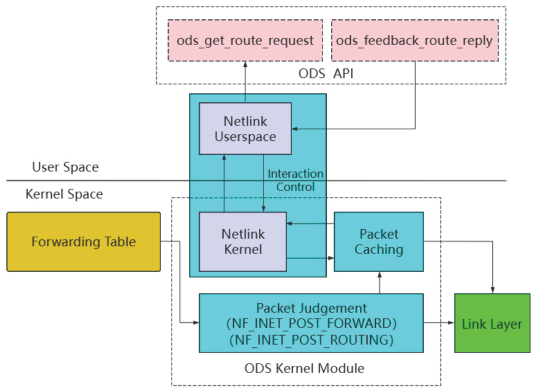

Fig. 4. Implementation illustration of ODS.

Packet caching caches and processes the packets detected by the judgement module. The processing functionality includes clearing specific packets or re-sending specific packets to the forwarding chain of the network stack based on the routing result. We use the return value `NF_STOLEN` in the HOOK function to remove these packets from the forwarding chain and hand over control to the caching module. After obtaining a successful routing result, we need to re-lookup the forwarding table for these packets' skb to populate certain parts of the `sk_buff` to guide the subsequent packet forwarding behavior. We can use programming interfaces provided by the operating system to manipulate the `sk_buff`.

Interaction control utilizes Netlink mechanism for cross-space interaction. The Netlink UserSpace component handles and encapsulates Netlink messages into function interfaces, while the Netlink Kernel component serves as the counterpart for interaction.The kernel module includes components such as Packet Judgement, Packet Caching, and Netlink Kernel.

V. Application Experience

To validate the applicability of the unified routing framework and its system services, we have implemented AODV (a reactive routing protocol) using the network components under the framework. We will discuss the experiences obtained during the deployment.

A. Implementation of AODV Routing Protocol

We have implemented AODV based on the AODV-UIUC version, which supports all the details mentioned in the AODV Draft Version 10 [24]. We rewrite and modify serveral parts of the codes in the AODV-UIUC with our network components' APIs. Its daemon is incorporated into the management of Agent Daemon. The architecture of the AODV routing protocol in our framework is shown in Fig. 5.

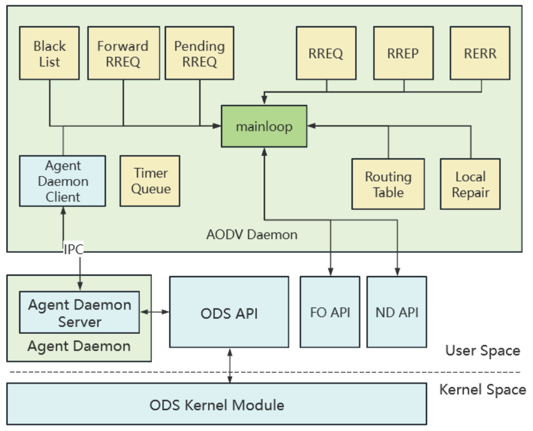

Fig. 5. Architecture of AODV implemented under the unified routing framework.

We replace the forwarding table operations and neighbor discovery operations in the AODV source code with the APIs of the FO and the ND component. For the on-demand services required by the AODV routing protocol, we delegate the initialization, configuration, and management to the Agent Daemon. The AODV routing daemon obtains on-demand system services through the Agent Daemon.

B. Discussions and Future Work

The unified routing framework can support any reactive routing protocols. Currently we have implemented AODV for illustration. More routing protocols, such as DSR and proactive routing protocols, will be implemented under the framework in the future. Furthermore, security issues have not been considered in this framework, which will be developed as a component of Agent Daemon.

VI. Conclusion

We have presented a unified routing framework for routing protocols in resource-constrained MANETs. Under the framework, we implement common network components in the kernel and provide APIs in user space. Agent daemons are designed to provide a unified network service for routing daemons, hiding the low-level system programming details in the kernel. We have implemented the network components in Linux 5.4 kernel. To demonstrate the feasibility of the framework, AODV routing protocol have been implemented under the framework.

Acknowledgement

This research was supported by National Natural Science Foundation of China (NSFC) under the grant 61972283.

References

[1] Liu, S., Dong, C., Zhu, X., Tang, J., & Zhang, L. (2022). Performance Evaluation of BATMAN-adv Protocol on Convergecast Communication in UAV Networks. GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 5105-5110.

[2] Liu, Q., Zhu, X., Zhou, C., & Dong, C. (2023). Advanced Fast Recovery OLSR Protocol for UAV Swarms in the Presence of Topological Change. 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 709-714.

[3] Luo, J., Chen, Y., Wu, M., & Yang, Y. (2021). A Survey of Routing Protocols for Underwater Wireless Sensor Networks. IEEE Communications Surveys & Tutorials, 23, 137-160.

[4] Saravanan, M., R, C.M., T, M., & Sukumaran, R.K. (2021). Routing strategies for underwater wireless communication: a taxonomy. Int. J. Commun. Networks Distributed Syst., 27, 147-177.

[5] Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., & Rao, N. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine, 51, 36-43.

[6] McKeown, N., Anderson, T.E., Balakrishnan, H., Parulkar, G.M., Peterson, L.L., Rexford, J., Shenker, S., & Turner, J.S. (2008). OpenFlow: enabling innovation in campus networks. Comput. Commun. Rev., 38, 69-74.

[7] Yu, H.C., Quer, G., & Rao, R.R. (2017). Wireless SDN mobile ad hoc network: From theory to practice. 2017 IEEE International Conference on Communications (ICC), 1-7.

[8] Bellavista, P., Dolci, A., & Giannelli, C. (2018). MANET-oriented SDN: Motivations, Challenges, and a Solution Prototype. 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), 14-22.

[9] Poularakis, K., Qin, Q., Marcus, K.M., Chan, K.S., Leung, K.K., & Tassiulas, L. (2019). Hybrid SDN Control in Mobile Ad Hoc Networks. 2019 IEEE International Conference on Smart Computing (SMARTCOMP), 110-114.

[10] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, New Orleans, LA, Feb. 1999.

[11] D. A. Maltz. On-Demand Routing in Multi-hop Wireless Mobile Ad Hoc Networks. PhD thesis, Carnegie Mellon University, 2001.

[12] Clausen, Thomas H. and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR).RFC 3626 (2003): 1-75.

[13] Perkins C E, Bhagwat P. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers[J]. ACM SIGCOMM computer communication review, 1994, 24(4): 234-244.

[14] Kawadia V, Zhang Y, Gupta B. System services for ad-hoc routing: Architecture, implementation and experiences[C]//Proceedings of the 1st international conference on Mobile systems, applications and services. 2003: 99-112.

[15] Jakma, P., & Lamparter, D. (2014). Introduction to the quagga routing suite. IEEE Network, 28, 42-48.

[16] Quagga Software Routing Suite, http://www.quagga.net/.

[17] Oran, D. (1990). OSI IS-IS Intra-domain Routing Protocol. RFC, 1142, 1-517.

[18] Moy, J. (1998). OSPF Version 2. RFC, 1247, 1-189.

[19] Malkin, G.S. (1998). RIP Version 2. RFC, 2453, 1-39.

[20] Atkinson, R.J., & Bhatti, S.N. (2012). Address Resolution Protocol (ARP) for the Identifier-Locator Network Protocol for IPv4 (ILNPv4). RFC, 6747, 1-12.

[21] Narten, T., Nordmark, E., & Simpson, W.A. (1998). Neighbor Discovery for IP Version 6 (IPv6). RFC, 1970, 1-82.

[22] Clausen, T.H., Dearlove, C., & Dean, J.W. (2011). Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP). RFC, 6130, 1-88.

[23] Welte H. The netfilter framework in Linux 2.4[C]//Proceedings of Linux Kongress. 2000.

[24] C. E. Perkins, E. M. Royer, and S. R. Das. Ad hoc on demand distance vector (AODV) routing. IETF Internet-Draft, draft-ietf-manet-aodv-10.txt, work in progress, Jan. 2002.