CoLoRSim: A Highly Scalable ICN Simulation Platform

Hongyi Li[1(✉)] and Hongbin Luo[2]

[1] Beijing Jiaotong University, Beijing, China
l6l20084@bjtu.edu.cn
[2] Beihang University, Beijing, China
luohb@buaa.edu.cn

Abstract. In the past few years, Internet usage is shifting from host-to-host communication to content distribution. As a result, information-centric networking (ICN) is emerging as a promising candidate paradigm for the future Internet. Meanwhile, many researchers have developed specified simulation platforms for the ICN architecture they proposed, to evaluate the performances in the research stage. In this paper, we present a simulation platform for a recent proposed ICN architecture named CoLoR. With the help of this simulation platform, performance for CoLoR can be evaluated through large-scale simulations. New protocols designed for CoLoR can also be studied.

Keywords: CoLoRSim · CoLoR architecture · Information–centric networking · INET

1 Introduction

As a promising candidate paradigm of the future Internet, information-centric networking (ICN) is envisioned to advocate the efficient usage of the network resource. Over the past few years, many researchers put lots of efforts on ICN and a lot of different ICN architectures [1] have been proposed in order to fit the requirements of different networking environments and address a series of limitations of the current Internet, including the efficient delivery of information to the users, mobility management and security enforcement.

Instead of assigning IP addresses to hosts, ICN assigns a globally unique name to a piece of content. To obtain content, a consumer sends out a request carrying the name of the desired content to the network. The network will forward the request to the closest caching node. The caching node returns the content by using data packets carrying the same content name. The intermediate nodes along the forwarding path can cache the content for serving subsequent requests for the same content.

In our previous work, we have proposed a new ICN architecture called CoLoR [2], whose intrinsic idea is to couple service location with inter-domain routing, but to decouple them from forwarding. Taking the advantages of CoLoR, many issues can be addressed. For example, the in-network caching of CoLoR can help Internet Service Providers (ISPs) reduce content retrieval latency and improves users' Quality-of-Experience (QoE), since the desired content may be retrieved from a nearby caching

S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 473–485, 2019.
https://doi.org/10.1007/978-3-030-22971-9_40

node instead of a remote server [3]. CoLoR can also help address the mobility issues [4] and improve network security [5].

In our previous research work, we put lots of efforts into developing a real-world prototype to evaluate the feasibility and performance of CoLoR. However, due to the limitations of machines and the complexity of implementations, the prototype size is restricted in 40-60 nodes, which has not reached the realistic Internet scale. As such, there is a need to develop a simulation platform for CoLoR in large-scale network topologies.

To address the above issues, in this paper, we present CoLoRSim, a highly scalable simulation platform for CoLoR based on the well-known simulator of OMNeT++ and INET framework [6, 7]. Especially, we make the following main contributions.

First, we designed and realized the main components of CoLoRSim, which includes a topology generator, a platform configuration module, the resource manager, the border routers and other components.

Second, we evaluate the relationship between the time consumption and topology scale. We draw the conclusion that when the topology contains 10000 autonomous systems (Ases, or domains) simulation will consume about four hours and 16.5G memory.

The rest of this paper is organized as follows. In Sect. 2, we summarize several ICN simulators. In Sect. 3, we briefly introduce CoLoR architecture. In Sect. 4, we introduce the system-level architecture of the developed simulation platform and the node structure of resource manager and border router. We also present how to extend this simulator. Finally, we conclude the paper in Sect. 6.

2 Related Work

For researchers, building a prototype system for a new ICN architecture is hard and expensive, especially when a large-scale network is needed. In such a situation, simulation is a better choice. Various simulation platforms have been developed for different ICN architectures. ndnSIM [8], which implements Named Data Network (NDN)/Content-Centric Network (CCN) [9] communication model, has been built based on ns-3. ccnSIM [10] for NDN/CCN has been developed based on OMNeT++. In Table 1, we list several ICN open source simulators.

Table 1. Summary of several simulators

Simulator	Architecture	Language	Operation system
ICNsim	Others	OMNeT++, C/C++	Linux, FreeBSD
Icarus	Caching	Python	Linux, Mac
CCN-Lite	CCN/NDN	OMNeT++, C	Linux, Mac, Win
CCNPL-sim	CCN/NDN	C++	Linux
NDNsim	CCN/NDN	NS-3, C++, Python	Linux, Mac
ccnSim	CCN/NDN	OMNet++, C++	Linux

3 A Brief Introduction to CoLoR

CoLoR architecture assumes that the future Internet is still comprised of ASes. CoLoR architecture separates inter-domain routing from intra-domain routing explicitly. Every domain in CoLoR architecture can choose its network architectures according to its demand, such as IPv4, IPv6, and others. Meanwhile, inter-domain routing is based on path identifier (PID). Every inter-domain path connecting two domains will be assigned a unique PID negotiated by the two domains. Border routers located in the border of domains need to store the intra-domain and inter-domain routing tables. For example, R2 in Fig. 1 has an inter-domain routing table like the one shown in the down left corner of Fig. 1. Each inter-domain routing entry contains the destination domain of the PID, the value of the PID, the prefix length of the PID, and preference value if multiple inter-domain paths exist.

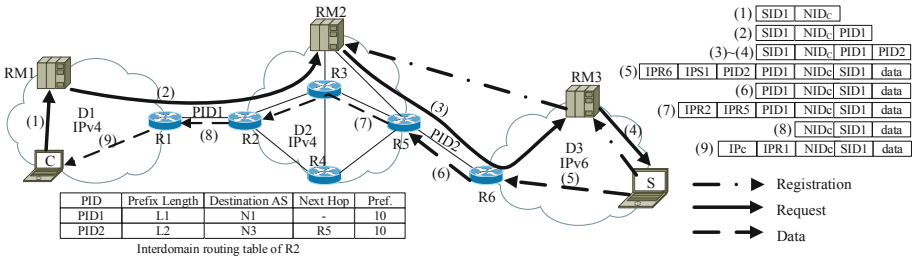


Fig. 1. Illustration of basic mechanisms of CoLoR architecture.

In particular, the PID of a path is not advertised through the Internet. It will only be kept secret by the two connected domains. By default, a user needs to send a request message which is routed to the receiver through the network, and then the receiver sends the desired content back to the user. Such a default-off communication model is more secure than the default-on communication model employed by the current Internet.

A globally unique node identifier (NID) is assigned to every component, and a globally unique service identifier (SID) is assigned to every piece of content in CoLoR. A logic centralized resource manager (RM) is deployed in every domain to manage node information, service information and inter-domain path information.

Figure 1 illustrates how a request message is routed from a user to the content provider. We assume that service provider S1 has registered service whose SID is SID1 in the network. The procedure is illustrated by dash lines in Fig. 1. When a user C wants to obtain content, whose SID is SID1, it sends a request message to its local resource manager (RM1) by using local routing mechanism, such as IPv4 in Fig. 1. Request message carries SID1 and NIDc of C. When RM1 receives the request message, it looks up the service information table and finds out that this request message should be forwarded to RM2 in D2. According to Fig. 1, RM1 appends PID1 at the end of the request message and sends the request message to RM2. When RM2 receives the request message, it looks up the service information table and finds out that this

message should be forwarded to RM3 in D3. RM2 appends PID2 at the end of the request message and sends the request message to RM3, as illustrated by (3) in Fig. 1. While RM3 receives the request message, it looks up in its service information table and finds out that the desired service is hosted by service provider S1. It sends the request message to S1, as illustrated by (4) in Fig. 1.

When S1 receives the request message, it sends the requested content back by using several data packets. Each of these data packets contains the information of SID1, NIDc, PID2, PID1. RM3 encapsulates an IPv6 header to the data packets and sends them to the border router R6, as illustrated by (5) in Fig. 1. When R6 receives the data packet, it strips the IPv6 header and PID2. Then R6 sends the data packet to R5, as illustrated by (6) in Fig. 1. While R5 receives a data packet, it forwards the data packet to R2 after encapsulating an IPv4 header to the data packet, as illustrated by (7) in Fig. 1. R2 strips the IPv6 header and PID1 and then sends the data packet to R1 as illustrated by (8) in Fig. 1. Finally, R1 sends the data packet to user S1 by using local routing mechanism, as illustrated by (9) in Fig. 1.

4 Simulation Model Design

A new simulation platform is developed based on OMNeT++ and INET framework in order to evaluate the performance of the CoLoR architecture.

4.1 Design Goals

The development of CoLoRSim takes following goals into consideration.

- Full support for CoLoR architecture
- Compatibility with original INET modules
- Auto-configuration for parameters
- Extensible architecture
- Applicable to large-scale network simulations

4.2 Simulator Architecture

As introduced in previous sections, the main system elements of CoLoR architecture include resource manager, border router, client/server. In this section, we describe the system-level architecture and node structure of CoLoRSim.

Figure 2 shows the architecture of CoLoRSim. There are eight blocks in it: topology generator, platform configuration, communication protocol, user-defined modules, mapping information tables, OMNeT++ platform interface, and internal/external analysis tools. The advantage of this architecture is that resource manager and border router can share the same communication protocol block and carry out different operations by replacing different user-defined modules.

Three different types of flows are defined, including internal input/output flow, external input/output flow, and control flow. Internal input/output flow represents communications via message or inter-module communication mechanism provided by

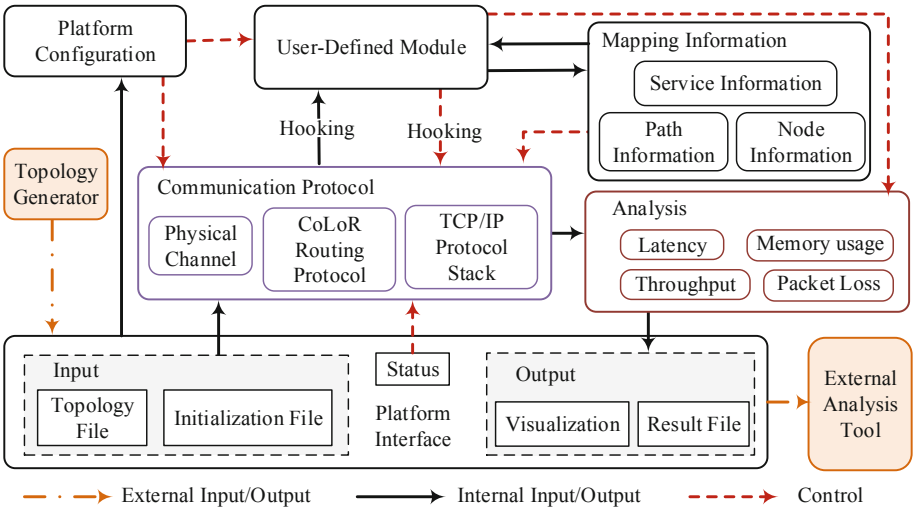


Fig. 2. The architecture of CoLoRSim

OMNeT++. External input/output flow represents communications via regular text files, JSON files, XML files, and even network connections. The control flow represents communications via signals.

Three different types of flows are defined, including internal input/output flow, external input/output flow, and control flow. Internal input/output flow represents communications via message or inter-module communication mechanism provided by OMNeT++. External input/output flow represents communications via regular text files, JSON files, XML files, and even network connections. The control flow represents communications via signals.

1. Topology Generator Block

New protocols should be tested under different topologies. To this end, a topology generator is developed, to avoid the errors in manual topology generation. The topology generator accepts flexible network parameter settings, including the number of ASes and the degree distribution of the network. The output files of the topology generator include an AS-level network description file and corresponding initialization file. Each node in the network description file represents an autonomous system consisting of a resource manager, border routers, clients, servers and other IPv4 or IPv6 routers.

2. Platform Configuration Block

A large number of parameters need to be set before starting a simulation. If all parameters are placed in the initialization file, the size of the initialization file will explode with the number of nodes, increasing the time of initialization. In addition, multiple initialization files need to be generated if the protocols are tested under different situations. Therefore, in CoLoRSim, the platform configuration block can

generate parameters according to users' requirements or read parameters from users specified files. The configuration block of the current version enables users to set the inter-domain path prefixes through prefix generating algorithms or pre-generated prefix assignment files.

3. OMNeT++ Platform Interface Block

The OMNeT++ platform includes input and output interfaces. The input interface enables users to import a topology file and the corresponding initialization file. The output interface offers a graphical display, performance chart and statistics that can be saved in a file for future analysis.

4. Mapping Information Block

There are three different types of mapping information tables to store routing information used by user-defined-module blocks and CoLoR routing protocol module. These blocks provide several basic operations: insertion, deletion, update, query, and verification. Service information table stores the mapping between SIDs and service providers. Node information table stores the mapping between NIDs and local routing identifiers, such as mapping between NIDs and IPv4 addresses. The path information table stores the mapping between PIDs and destination autonomous systems. The key shared between two adjacent ASes will also be stored in path information module.

5. Analysis Block

Two different types of analysis blocks are provided in CoLoRSim: internal analysis block (Analysis block) and external analysis tools block. Internal analysis block collects critical statistics from user-defined-module blocks and communication protocol block, such as latency, throughput, and packet loss. These statistics can be stored by the output interfaces provided by OMNeT++ platform interface block. An in-depth understanding of all collected data is gotten by using external analysis tools.

6. Communication Protocols Block

Communication protocols block provides protocols from physical layer to network layer. The basic structure of this block is shown as Fig. 3. Communication protocol block includes physical layer protocols, link layer protocols, and network protocols. We use physical layer modules, link layer modules, and some existing network layer modules. Only wired network is taken into consideration, where Point-to-Point (PPP) and Ethernet protocols are supported.

The network layer supports different protocols. First, we implement a CoLoR routing protocol module which supports an inter-domain routing protocol based on PIDs and an intra-domain routing protocol based on NIDs. CoLoR routing protocol block also translate NIDs and PIDs into local routing identifiers. Second, routing protocols used in the current Internet is also supported. Because local routing protocols are selected by network administrators, different autonomous systems may choose different protocols. Thus, we simplify the change of local routing protocol module, without modifying other modules. The CoLoR routing protocol module detects the type of local routing protocol and encapsulates correct information into packets sent to the down layers.

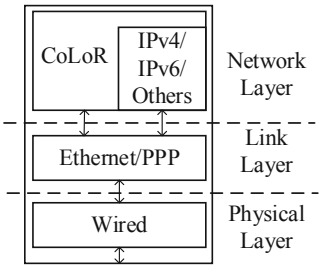


Fig. 3. The structure of communication protocols block

7. User Defined Modules Block

Communication protocols block only focuses on routing and transmitting messages. As a message is detected by communication protocol block, the message will be forwarded to a user-defined module according to the type of the message and the destination port number. User-defined module will call its own handleMessage() function when receiving the message. A user-defined module is usually designed for one type of message. Several related functions have been implemented:

Client module is used to generate request flow subject to a probability distribution, such as Poisson distribution. This module is also used to handle received data packets.

Server module is used to handle received request packets and generate data flow corresponding to the requested service. The characteristics of data flows are subject to probability distributions or characteristics inferred from real Internet flow data.

Request packet handler module of border routers and resource managers performs request packet processing, request packet verification, and any other operations on request packets. This module may modify requests based on its local policy or drop illegal packets if the packet verification fails. For example, when a request packet should be forwarded to another autonomous system, a PID should be encapsulated into it. When a border router receives a request, the router calculates a message authentication code and drop the packet if the calculated code is not the same as the code carried in the request packet.

Data packet handler module behaves similarly to request handler module. This module is equipped with border routers and resource managers. This module performs data packet processing, data packet verification, and any other operations on data packets. This module needs to modify packet and drop illegal packets. For example, when a data packet is forwarded to another border router located in another domain, a PID is popped. This module will also use message authentication code to detect if a packet is modified by attackers.

Service registration packet handler module is equipped with servers, border routers, and resource managers. Servers use this module to generate service information and announce it to its local resource manager. When receiving the service advertisement messages, resource manager adds new entries into service information table and make announcements to the parent autonomous systems and peer autonomous systems, according to the local policy defined by users.

Node registration packet/management handler module performs automatic configuration during the early period of simulation. Node advertisements will be generated and send to resource manager and border routers. When node advertisements are received, the mapping between NIDs and local routing identifiers can be established based on the information carried in the advertisements. This module can also interchange inter-domain information between two border routers and send the information to the resource manager.

PID updating handler module performs all operations about path prefixes, such as periodic negotiation between two adjacent autonomous systems, advertisements for newly negotiated path prefixes.

Those modules can be equipped according to the types of nodes. For example, a resource manager is equipped with all these modules except client module and server module. For different types of nodes, the same type of user-defined modules has some differences. For example, request packet handler module of resource manager needs to query its database to decide how and where to forward this request, while requests packet handler module of border routers do packet verification instead.

8. Extensibility

Extensibility provides the ability to add new functions without modifying existing modules. In our simulation platform, we provide hooking interfaces between communication protocols block and user-defined module block.

The first step to use hooking module is to specify the hooking function. When the hooking module receives a message from other modules, the hooking module will call the function we specify previously. By specifying different functions, we can change one node's behavior. For example, to record the PID sequences carried by a request, the client module can be modified to extract PID sequence and save it to a file. But such a method requires recompiling simulation platform, which may take long time and cause new issues. A hooking module can address this issue. We can develop new functions separately without modifying existing modules. Then, by modifying the parameters of hooking module and the hooking module will load the functions automatically and perform the corresponding functions.

4.3 Node Structure

In the previous sections, we describe the system-level design of CoLoRSim. This section further shows the node structure design based on the CoLoRSim. Each node contains the communication protocol block, the user-defined module block, and information table block. However, different user-defined module blocks are defined for different types of nodes.

1. Resource manager

The structure of resource manager is shown as Fig. 4. Resource manager is equipped with five types of user-defined modules, i.e., request packet handler, service registration handler, PID updating handler and node management handler.

When a message arrives at a resource manager, the physic layer and linker layer module process this message firstly. Then the message will be sent to the IPv4 or IPv6

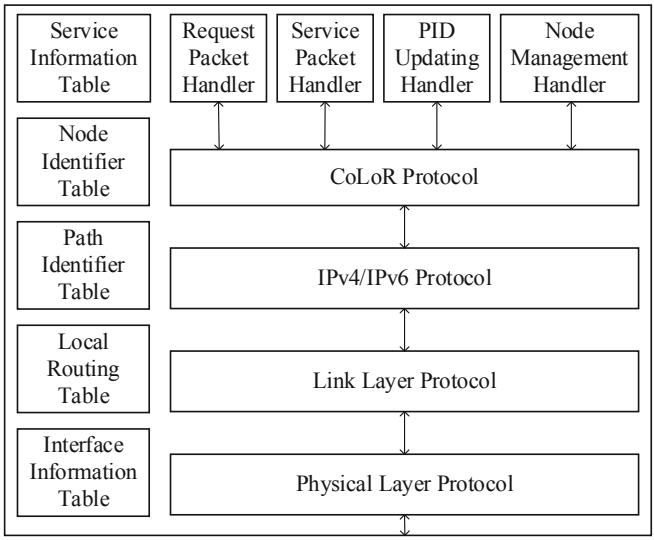


Fig. 4. Structure of resource manager

protocol module. The message will be sent to the CoLoR protocol module if the destination IP address belongs to this resource manager. Message received by CoLoR protocol module will be sent to a user-defined module according to the type of message. For example, if a request packet is received, this message is sent to request packet handler. As the message arrives at a user-defined module, these modules will take actions.

When user-defined modules send out messages, the CoLoR protocol module will receive those messages and get the destination of those messages. CoLoR protocol module will encapsulate corresponding control information into packets according to its local routing protocols. For example, if the local routing protocol is IPv4, then the IPv4 address of the destination will be encapsulated into control information. IPv4 or IPv6 protocol modules can process received messages from CoLoR protocol module according to the control information. After processed by link layer and physical layer module, the message will finally be sent to other nodes.

Request packet handlers in RMs get requested SID and query it in the service information table. In addition, this module decides which node or which domain this message should be forwarded to.

Service registration packet handlers in RMs will get the SID, NID of provider, the autonomous system the provider is located in and other information about this service. A new service entry will be added into service information table. Service registration handler will also send an acknowledgment (ACK) message to the source of the received registration message and register this service to its parent autonomous systems if possible.

PID updating handlers in RMs are used to negotiate new PIDs with the adjacent ASes and distribute new PIDs to nodes in the same domain. When receiving a PID

updating message, this module detects the type of the message. If the message is a negotiation message, the module checks if the new PID is available. If available, a new entry should be added and an ACK message should be sent back. Otherwise, an available PID should be generated and this module tries to re-negotiate PID with the other one. After adding new entry, new PID should be distributed to other nodes in this domain by sending distribution messages which carry the old and new PIDs.

Node management handler in RMs is used to process node notification message. When this module receives a message, it will add an entry to node information table. This entry contains the NID and its local routing identifier.

2. Border Router

Border router has a similar structure with resource manager, as shown in Fig. 5. The biggest difference is that border router can run over link layer protocol instead of network layer. Packets from intra-domain links will be handled by IPv4/IPv6 protocol module, while those packets from inter-domain links can be handled by CoLoR protocol module directly. The incoming packets will be handled from physical layer to application layer as the same as the procedure in resource manager. The outgoing packets will be handled in the reversed order.

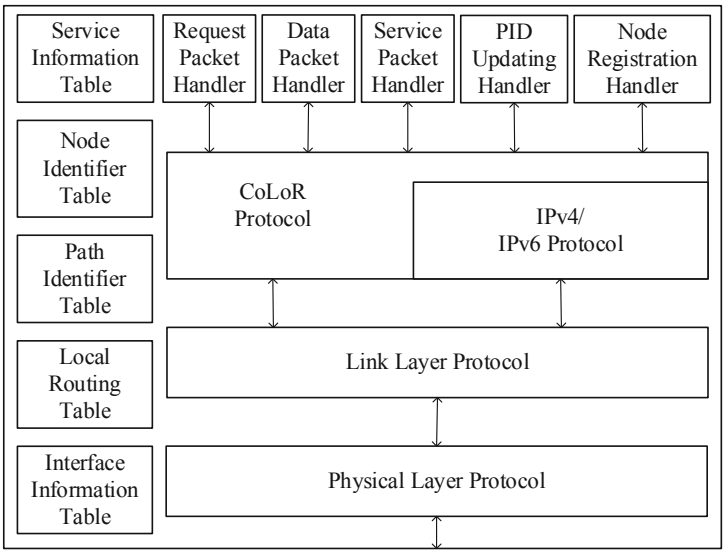


Fig. 5. Structure of border router

Request packet handlers in BRs forwards request packets according to the result querying from path information table. Data packet handler in BRs checks if data packets are legal and forward legal packets according to the routing entry queried from path information table. Service packet handlers in BRs only forwards registration message. PID updating handlers in BRs receives PID distribution message and update

corresponding entries in its path information table. Node management handlers in BRs announce its information including its own NID, its local routing identifier to the RM and other border routers. Node management handlers also update the node information table as receiving advertisements.

3. Client/Server

Structure of client module, as shown in Fig. 6, is similar with border router. The request packet handler is replaced by a client module or server module. A service registration packet handler is replaced with service generating handler. In client/server, all received requests should be forwarded to server module and all data packets should be forwarded to client module. Service registration handler module generates service registration messages and sends them to its local resource manager.

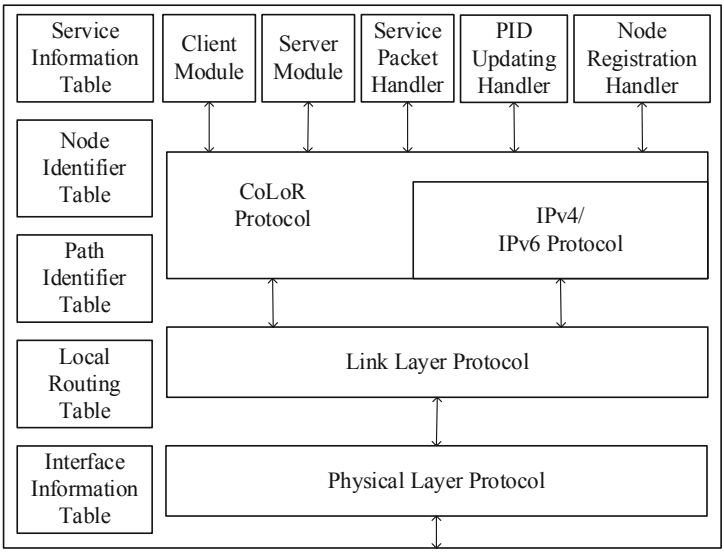


Fig. 6. Structure of client/server

5 Evaluation

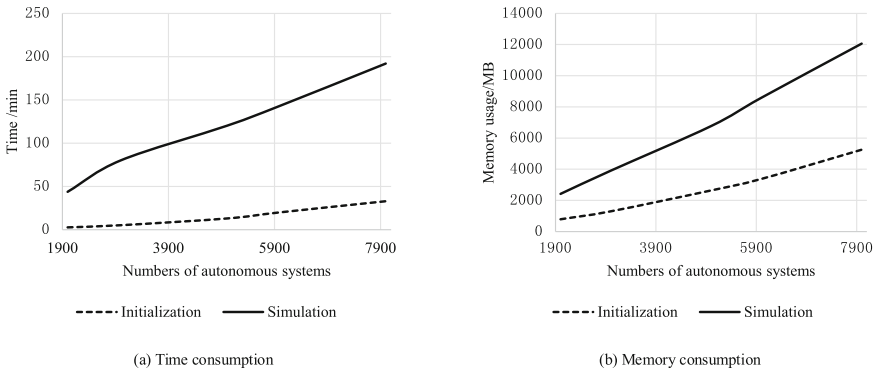
In this section, we evaluate the performance of the CoLoRSim. The CPU of the server is Intel Xeon E7-4820 @ 2.0 GHz. The total memory size is 128G and the capacity of hard drives is 900 GB.

We define two different phases of simulation, initialization phase, and simulation phase. The initialization phase begins with reading topology from files and ends with the first event occurring. The simulation phase begins with the first event and end with the last event.

Different topologies are adopted, containing different numbers of autonomous systems. Each node in our topology will generate 500 requests and corresponding 500 data packets whose length is 1000 bytes.

We measure the time usage and peak memory usage of the initialization process and simulation process. The simulation time is set to 8000 s.

Figure 7(a) and (b) show that both time usage and memory usage are almost proportional to the size of the simulation networks in both initialization and simulation phase. When we simulate a network that contains 8,000 autonomous systems, about 56,000 nodes, CoLoRSim consumes about 12G memory and 200 min. As the number of autonomous systems reaches 10000, we need 16.5 G memory and 4 h.



(a) Time consumption

(b) Memory consumption

Fig. 7. Time and memory consumption during initialization and simulation

6 Conclusions and Future Perspectives

In this paper, we present the design of a simulation platform for the CoLoR architecture based on OMNeT++ and INET framework. The system-level architecture of the whole simulation platform and the node structures of resource manager, border router, client and server are introduced in detail. The performance of CoLoRSim is evaluated, and the relationship between resource consumption and topology scale is also demonstrated.

In the future, we will enhance the compatibility of wireless link in CoLoRSim, such as IEEE 802.11. The parallel simulation will also be introduced to speed up the simulation, and the memory usage will be optimized.

References

1. Xylomenos, G., et al.: A survey of information-centric networking research. IEEE Commun. Surv. Tutor. **16**(2), 1024–1049 (2014)
2. Luo, H., Chen, Z., Cui, J., Zhang, H., Zukerman, M., Qiao, C.: CoLoR: an information-centric internet architecture for innovations. IEEE Netw. **28**(3), 4–10 (2014)
3. Zhang, M., Tang, J., Rao, Y., Luo, H., Zhang, H.: Degree-based probabilistic caching in content-centric networking. China Commun. **14**(3), 158–168 (2017)

4. Ying, R., Hongbin, L., Deyun, G., Huachun, Z., Hongke, Z.: LBMA: a novel locator based mobility support approach in named data networking. China Commun. **11**(4), 111–120 (2014)
5. Luo, H., Chen, Z., Li, J., Vasilakos, A.V.: Preventing distributed denial-of-service flooding attacks with dynamic path identifiers. IEEE Trans. Inf. Forensics Secur. **12**(8), 1801–1815 (2017)
6. OMNeT++. https://omnetpp.org/. Accessed 20 Dec 2018
7. INET Framework. https://inet.omnetpp.org/. Accessed 20 Dec 2018
8. Mastorakis, S., Afanasyev, A., Zhang, L.: On the evolution of ndnSIM. ACM SIGCOMM Comput. Commun. Rev. **47**(3), 19–33 (2017)
9. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.: Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies - CoNEXT 2009, Rome, Italy, pp. 1–12. ACM Press (2009)
10. Chiocchetti, R., Rossi, D., Rossini, G.: ccnSIM: an highly scalable CCN simulator. In: 2013 IEEE International Conference on Communications (ICC), pp. 2309–2314. IEEE, Budapest, Hungary (2013)