Received May 13, 2021, accepted July 4, 2021, date of publication July 9, 2021, date of current version July 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3096042

Fast Content Delivery Using a Testbed-Based Information-Centric Network

NAZIB ABDUN NASIR AND SEONG-HO JEONG
Department of Information and Communications Engineering, Hankuk University of Foreign Studies, Cheoin-gu, Yongin-si, Gyeonggi-do 17035, South Korea
Corresponding author: Seong-Ho Jeong (shjeong@hufs.ac.kr)

This work was supported in part by the Ministry of Trade, Industry, and Energy (MOTIE) and the Korea Institute for Advancement of Technology (KIAT) through the International Cooperative Research and Development Program, in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT), and in part by the Hankuk University of Foreign Studies Research Fund of 2020.

ABSTRACT Key requirements for a better performance of multimedia applications typically include lower latency, improved security, faster content retrieval, and adjustability to the traffic load. However, the current Internet often fails to meet the requirements due to the drawbacks of the host-oriented communication architecture. Those drawbacks can be overcome by a well-recognized networking paradigm, called Information-Centric Networking (ICN), that offers name-based and information-centric communication rather than host-oriented communication. This paradigm uses an in-network caching policy and may provide enhanced security along with user mobility. Therefore, the ICN architecture can outperform the current Internet in many aspects, including, but not limited to, content transfer time, traffic load control, mobility support, and efficient network management. Most of the existing research validating the ICN paradigm's superior performance compared to the current Internet is based on simulation. In this paper, we propose a novel ICN-based testbed, where we used several modified functions for naming, routing, and caching and developed two new functions that provide mobility support and auto-playing of the retrieved content so that the performance of the ICN network is enhanced even more in terms of content delivery time or response time. The proposed testbed architecture is based on Content-Centric Networking (CCN), a prominent branch of the ICN concept. We present the testbed development procedures and the functions in detail and demonstrate that the testbed-based architecture outperforms the current Internet architecture and the basic CCN-based environment.

INDEX TERMS Content-centric networking, information-centric networking, testbed development.

I. INTRODUCTION

Internet data traffic usage has increased rapidly in recent times as the number of people who access the Internet grew massively. Similarly, the number of wireless and mobile devices to access the Internet to watch or download streaming contents is on a remarkable rise. Several applications keep the users busy surfing the web and entertaining themselves via mobile devices even while they are on the move. Accordingly, the demands on faster content access and seamless video streaming have increased tremendously. Users interact with video data traffic more often than the other forms of data traffic, including audio data and text data. The CISCO forecast of Global Internet Growth and Trends predicts that by 2022 there will be approximately 4.8 Billion Internet users connected with 28.5 billion different devices [1]. Those users will be able to access the Internet with an increased average broadband speed of 75.4 Mbps, and it is estimated that most of the exchanged IP traffic will consist of video data. The basic features of video data typically include massive volume and high bandwidth. The video data can be live or pre-recorded streaming video contents, various video-on-demand contents such as movie files and video clips. CISCO also predicts that this enormous number of people with a vast number of connected devices and higher connection speed will exchange approximately 400 Exabytes per month, and the percentage of video data traffic among all other application types will be more than 82%.

This rapid increase of users and usage of different online-based technologies were not foreseen during the development of the original Internet architecture primarily based on immobile wired devices. Besides, the communications in the current Internet are based on location-oriented host-to-host data transmissions. Subsequently, the current Internet

The associate editor coordinating the review of this manuscript and approving it for publication was Jose Saldana.

confronts several downsides, including data traffic congestion, higher content delivery time, and connection failure due to user mobility. A novel networking paradigm was proposed to eliminate these stumbling blocks, shifting the communication strategies from host-centric to content-centric. This new networking paradigm is called Information-Centric Networking (ICN) [2]. In this approach, the communicating host machine becomes less focused than the expected data, and as the requested content is fetched based on the name of the content itself instead of the host's location where the content is available, the delivery process becomes faster. In this information-centric architecture, the security of the requested content and user mobility are also addressed and improved. As this concept is becoming popular, several distinctions were proposed that are discussed briefly in the following section. Although these approaches vary in terms of actual implementation, they have an identical principle to improve the end-user experience, which provides the desired contents faster using the name of the contents rather than utilizing the host's location that contains the contents. Among them, Content-Centric Networking (CCN) [3] is one of the most prominent variations.

ICN and CCN both emphasize name-based communications as opposed to host-based communications. Features of the CCN paradigm are briefly described in the following subsection. In contrast, an IP-based system called content delivery network (CDN) or content distribution network (CDN) deploys geographically distributed proxy servers and delivers the requested contents from a data center located at an optimal distance from the end-user. The optimal distance may be measured based on the number of hops, latency, server resource availability, or cost-efficiency. Albeit being a location-based service, the benefits of CDN may include an increase in performance due to having cached contents nearby the end-users, a reduction in operating costs, and high scalability. Interested readers can find more information in [4]. Both CCN and CDN have the advantage of cached contents. However, the CCN paradigm handles various other concerns not covered by CDN and provides additional benefits. This includes controlling network congestion, performing active forwarding strategies, supporting user mobility, securing the content itself, multicasting, and higher performance caching, and others.

Historically, researchers focused on using various tools to simulate and validate their proposals and theories. There are several popular simulation tools available, such as NS-3 [5], MATLAB [6], and ndnSIM [7]. Nevertheless, the present advanced technology era allows the researchers to authenticate and verify their proposals in a configuration that is as close to the real world as possible using a real testbed architecture. As a testbed architecture consisting of real machines allows to test the system in reality, testbed-based experiments and performance evaluations can be more pragmatic and complement the simulation-based experiments. Thus, we chose to develop a testbed based on the ICN concept and provide performance results of several experiments on the developed testbed architecture in terms of content delivery time or response time. As of late, various testbeds have been depicted in research articles while just some are open, and even less are accessible, which are based on the ICN paradigm. Among the few available alternatives for developing the testbed architecture from scratch (one, for example, CCN-LITE [8]), we have picked the CCNx Distillery Software Distribution 2.0 [9] as the base to prepare the proposed testbed. The rationale behind using this software is that it offers solutions to experiment with the ICN architecture as closely as possible, and it can be configured extensively. We have modified and updated several functions that handle the naming, routing, and caching features. Additionally, we added two newly developed functions into the testbed architecture, one of which provides mobility support, and the other auto-plays the downloaded contents automatically. All these functions contribute to the enhancement of the performance of the basic ICN network.

Therefore, the main contributions of this paper are to develop an ICN-based testbed architecture utilizing the CCNx Distillery Software Distribution 2.0, which can work as the base framework for many future research based on testbed-based performance evaluations for the ICN networks and implement the developed functions, naming, routing, caching, mobility support, and auto-play, which enhances the performance of the basic CCN paradigm. In summary, the following are the main contributions of our work:

- Unlike the customary tradition of relying on performance results based on simulation, we proposed a novel testbed-based performance evaluation for the ICN paradigm.
- We introduced an ICN-based testbed architecture and provided details of the development procedures. This testbed architecture can provide the base platform for future research.
- We enhanced the performance of some of the existing functions such as naming, routing, and caching, and we also implemented new functions such as mobility support and auto-play functions in the testbed architecture.
- We demonstrated that the testbed-based architecture with the developed functions outperforms the basic CCN and the current Internet significantly in terms of the content delivery time or response time for various sizes and different types of contents.

The rest of the paper is organized as follows. In Section 2, the related state-of-the-art technologies are addressed. Section 3 then introduces and describes the testbed development process and the procedures executed by the added functions. After that, we examine the diverse performance outcomes in Section 4 before concluding the paper with Section 5.

II. RELATED WORK

Our research work is related to two fields: Information-Centric Networking and existing ICN-based testbed architectures. In this section, we introduce various ICN concepts and describe the features of CCN. We also describe existing work closely related to our proposal. Additionally, we briefly describe the basic software that was used to build the proposed testbed architecture.

A. VARIETY OF ICN

The first name-based information communication theory was introduced by TRIAD [10], and since then, researchers have proposed multiple architectures. In 2006, researchers at UC Berkeley proposed DONA [11], which uses a name resolution system to authorize the content storage points and update the mobility information. It was mostly an improved version of TRIAD in terms of security and architecture. Then, funded by the EU Framework 7 Program (FP7), PSIRP [12], [13] was proposed that replaces the IP protocol stack with a publish/subscribe protocol stack. This project was continued further by the name PURSUIT [14], [15]. NetInf [16] was also based on a name resolution service like DONA, and it supported content searching based on meta-data. This concept was initially proposed by the FP7 project 4WARD [17], and further development was made by SAIL [18]. Additionally, Content-Centric Networking (CCN) [3] was proposed by PARC in 2007, and the performance of this architecture was enhanced by Named-Data Networking (NDN) [19] project.

The CCN paradigm is considered the most notable one among the proposed ICN concepts. In this paradigm, for efficient delivery of the requested content, the name of the content itself is focused on rather than the location or address of the content repositories. A consumer requests a specific content by sending an Interest packet, including the name of the requested content, to all appropriate neighbor nodes. The serving node sends back a Data packet containing the desired content. In the basic CCN, all contents that pass through a node are cached in that node. Therefore, the requested content can be provided from an intermediate node instead of the original remote server. Furthermore, the CCN architecture provides packet-level security and incorporates a basic mobility support mechanism as well. Nevertheless, the basic CCN has a few critical drawbacks. For example, it may flood the network with the Interest packets as it looks for the nearest content source. As a result, congestion may still occur in the network. Furthermore, the basic CCN concept is not focused on a sophisticated caching mechanism that accounts for inefficient caching results in the real environment. Owing to these, several duplicate contents may coexist in a particular topology, and cache overflow may happen often.

B. RELATED WORK ON ICN-BASED TESTBEDS

Although a considerable amount of research is already going on to evaluate and validate the ICN concepts, most of them revolve around either simulation or emulation strategy to prove this paradigm's efficiency. From the realization of the requirement of a scalable, configurable, and low-cost testbed for fast prototyping, a "Control and Management Framework for a scalable CCN testbed" was proposed and developed in [20] that includes controlling and management of multiple CCN nodes scattered across different places. The authors hoped that other researchers could expand their experiments without being bothered regarding the tiresome procedure of setting up the network itself. The core purpose is to let other experimenters utilize the authors' framework to validate and evaluate the experimenters' ideas promptly. However, this approach's drawbacks include a limited number of parameters to experiment with, such as the number of publishers, routers, and subscribers. Furthermore, this testbed does not consist of real machines; instead, virtual machines are used. Therefore, this approach may be limited in providing an absolute perception of the real-world environment.

Video traffic transmission over the Internet is showing a brisk upsurge in recent years. The current Internet, which uses the Internet Protocol (IP), was predominantly meant for the stationary machines exchanging mostly time-insensitive text or audio data. This architecture is rather quickly becoming inadequate to meet the demands and requirements of future Internet users who demand lower latency, improved security, faster content retrieval, and adjustability to the traffic load while using multimedia applications. As the ICN paradigm gained popularity, significant research work based on ICN is going on. Thus, deploying ICN on the radio access network (RAN) and validating the concept with a testbed can strongly complement theoretical and simulation-based works. Inspired by this idea, a testbed that provides testament for the ICN-RAN using 4G was presented in [21]. The experiments demonstrated that in the cases of real-time applications and video distributions, the proposed testbed achieves a lower-latency result than presently available options. Nevertheless, the absence of accelerated hardware and failure to optimize the codes limited the bit rate that could have been achieved in real-time situations. Despite being a viable option as a testbed for the ICN network, a few limitations in this paper need to be addressed, including the incapability of taking different paths for the content that would allow the evaluation of multicast routing benefits into account. Moreover, adding more eNodeBs would increase the acceptance of this testbed, which is missing in this paper, giving a better evaluation of the mobility events.

C. THE BASIC SOFTWARE

The CCNx Distillery Software Distribution 2.0 [9] is referred to as basic software from hereafter, and it can be easily reconfigured. This project's evolution was driven by the ICN Community [22], and its scope was based on implementing CCNx 1.0, a software release based on the CCN architecture, by a harmonization effort. The primary purpose of the basic software is to pull together all the necessary modules to build a full CCNx software suite. The modules can be used for various requirements, and each of the modules is independent by itself. Furthermore, each of those modules may come from a different author or institution and may have its own set of requirements. The basic software provides features for building CCNx software and tools for writing, testing, and evaluating code. Although it can run on several OS platforms, this software is most stable while using the Ubuntu 14 OS, and therefore, we have used that OS platform. This software is the most promising tool based on the CCN paradigm that allows developing various testbed-based topologies and experiment with the CCN network architecture as closely as possible. However, it needs vigorous modifications, as we did, to be used as a standard platform for sophisticated experiments.

III. ICN-BASED TESTBED DEVELOPMENT

This section provides the implementation architecture and the development procedure of the proposed testbed from the basic software and then presents the detailed descriptions of the testbed development and the execution procedures of the developed functions. We proposed the preliminary ideas on a testbed-based performance evaluation of the ICN in [23]. This paper is a continued and significantly extended work where we added more details of the testbed development process as well as the newly added functions. Furthermore, we added the algorithms using pseudo codes of the functions and introduced a new function to support mobility in this paper.

A. OVERVIEW OF THE TESTBED DEVELOPMENT

The testbed development procedures involve two parts. First, the basic software had to be installed on all the testbed machines, and then some of the existing functions were modified and improved, and other new functions were added. The primary responsibility of the basic software is to assemble the required modules for building an ICN architecture. Therefore, it brings together several independent modules from different developers to build the complete ICN architecture, and that can be configured and customized using the CCNx software. The essential modules used for performing the experiments of the testbed are LongBow - a C language software framework to combine the fail-fast philosophy of an offensive-stance of program development and xUnit style unit testing, Libparc - a C runtime library developed by PARC that provides an array of features and capabilities for C programs and programmers, Libccnx-common - a set of functions and data structures for CCNx, Libccnx-transport-rta - is a CCNx networking base stack used with a forwarder running underneath it and a set of APIs above it, Libccnx-portal - a simple API to communicate via Interests and Content Objects connected to a transport stack, and Athena - a CCNx Forwarder. These modules are used to execute the content exchange experiment. However, the major drawback of the basic software is that it can be used within one machine only. By default, it is configured to recognize content requests from a machine's folder and deliver them to another folder. Thus, the basic software does not need to create a routing table with other machines, although it supports the ICN routing mechanisms. Therefore, the basic software is not adequate to create a testbed architecture of several machines by itself and evaluate the performance of the ICN networks realistically in terms of content delivery time.

B. DEVELOPED FUNCTIONS AND THEIR TASKS

The primary design objective of the testbed is to accomplish multimedia applications requirements such as faster, secure, seamless, efficient, and reliable content retrieval and delivery. Therefore, we modified the existing naming, routing, and caching functions and added newly developed mobility support and auto-play functions within the proposed testbed architecture. All the machines of the testbed architecture, clients and servers, are featured with the naming function. This is an enhanced function from the existing one within the basic software and handles two tasks, allocating names to all the machines that comprise the testbed architecture and providing unique names to all the available contents. Then, the routing function is implemented within all the server machines. Considering that the basic software does not create any routing table, the existing routing function needed to be modified and updated. Routes among all the connected server machines are created by utilizing this routing function. Moreover, this function aids in mapping the names of all the machines with their IP addresses so that the machines outside of the testbed architecture can recognize these machines, and it is also helpful when experiments are performed with the current Internet environment instead of employing the ICN paradigm. Subsequently, the existing caching scheme that caches all the flying contents at all the in-network server machines adopted by the basic CCN and the basic software is replaced by an enhanced caching scheme developed for the proposed testbed architecture. This scheme selectively caches some of the contents that go through a server machine, and some of the contents are discarded.

Similarly, while removing a content when there is a lack of space, a set of updated algorithms are followed by the modified caching scheme. Next, an auto-play function is added along with all the client machines of the testbed architecture to allow those machines to auto-play the retrieved content as soon as the download is completed. Finally, another new function was developed and implemented within the testbed client machines that provides seamless mobility support while the clients are moving away from one server to another server during an active download process. The primary responsibility of this function is to locate an appropriate server that can continue to deliver the requested content as quickly as possible when the hand-off is necessary. The architectural framework of the testbed and the task table of the functions are given in Fig. 1.

C. CONTENT REQUEST AND RETRIEVAL PROCEDURE

We propose to use a real testbed architecture to show that the ICN-based environment delivers a requested content faster than the current Internet, and we developed several functions to enhance the performance and reduce the content delivery time even more than the basic CCN paradigm. Experimental procedures involving content requests and retrievals follow a similar flow to that of the basic CCN paradigm. The experiment begins with a client sending a request for a particular content to its connected server. As soon as the server receives

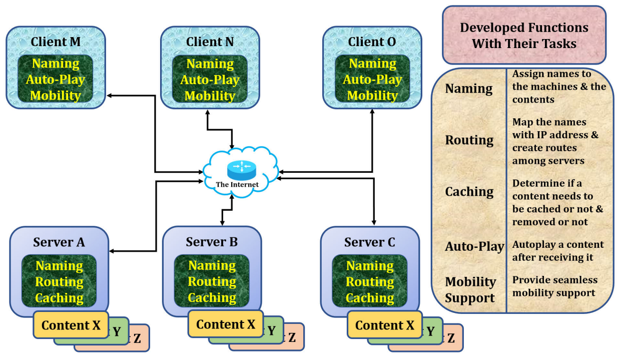

Client M
Naming
Auto-Play
Mobility
Client N
Client O
The Internet
Server A
Server B
Server C
Routing
Caching
Content X
Y
Z
Developed Functions With Their Tasks
Naming — Assign names to the machines & the contents
Routing — Map the names with IP address & create routes among servers
Caching — Determine if a content needs to be cached or not & removed or not
Auto-Play — Autoplay a content after receiving it
Mobility Support — Provide seamless mobility support

FIGURE 1. **Architectural framework of the testbed along with the tasks of the functions.**

the content request, the naming function operates and identifies the desired content. The server checks its local content store (CS) if the content is readily available. If the content exists in the CS, the server directly sends back the file, and the content delivery time is measured after the retrieval is completed. Responsibilities and the relative positions of the functions, naming, auto-play, and mobility support during the content request and retrieval procedure are shown in Fig. 2.

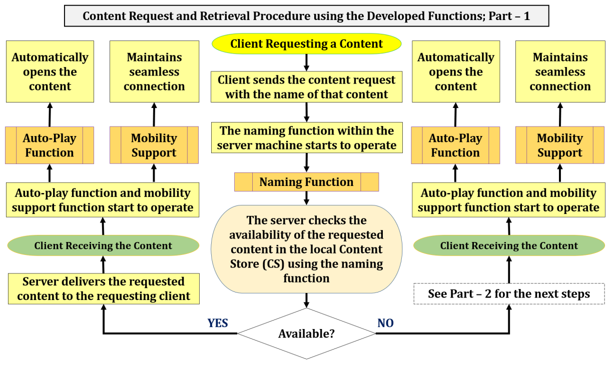

Content Request and Retrieval Procedure using the Developed Functions; Part – 1
Client Requesting a Content
Client sends the content request with the name of that content
The naming function within the server machine starts to operate
Naming Function
The server checks the availability of the requested content in the local Content Store (CS) using the naming function
Available?
YES
NO
Server delivers the requested content to the requesting client
Client Receiving the Content
Auto-play function and mobility support function start to operate
Auto-Play Function
Mobility Support
Automatically opens the content
Maintains seamless connection
See Part – 2 for the next steps

FIGURE 2. **Content request and retrieval procedure using the developed functions; Part – 1.**

However, if the requested content is not available in the CS, it has to fetch that content from another server that might have it. Before forwarding the content request, the server first cross-checks the pending interest table (PIT) if it has prior request(s) of that same content. The PIT keeps track of all the content requests that could not be satisfied with the CS; hence it had to forward to another server. Accordingly, if there is already a request for that content and the server is waiting to receive the requested content, it only updates the PIT list with the latest client's request. Otherwise, it has to consult the forwarding information base (FIB) for the information regarding where that content is available. The routing function comes in handy in this step as the FIB table is built up with the help of this function. Following the FIB table, the connected server locates the proper target server, forwards the content request to that server, updates the PIT information, and waits for the response. However, if there is no routing information available for the requested content, the server has no other option but to discard the content request. Although the basic CCN paradigm floods the network with the content request at this step, we consider this rare circumstance as a failed attempt to download the content, and the client continues to request other contents as the experiment keeps going on. In our graphical representation of the experimental results, delivery times are shown only for the successful content retrievals. Moving on to the next steps, and after receiving the requested content, the server has to decide whether to keep the content in the CS or not. Unlike the basic CCN paradigm, which always caches the requested content, the updated caching scheme implemented in the testbed architecture helps the server make smart decisions. The contents are ranked in terms of content popularity, and based on that, the server either stores or discards that content. Finally, the content is delivered to the requesting client by the connected server, and the responsibility of the server ends here. Responsibilities and the relative positions of the functions, routing, and caching during the content request and retrieval procedure are shown in Fig. 3.

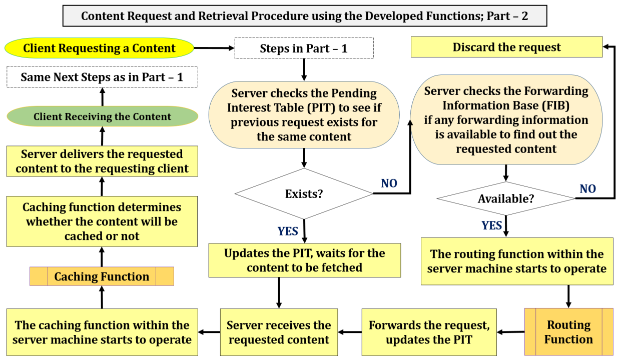

Content Request and Retrieval Procedure using the Developed Functions; Part – 2
Client Requesting a Content
Steps in Part – 1
Discard the request
Same Next Steps as in Part – 1
Client Receiving the Content
Server delivers the requested content to the requesting client
Caching function determines whether the content will be cached or not
Caching Function
The caching function within the server machine starts to operate
Server checks the Pending Interest Table (PIT) to see if previous request exists for the same content
Exists?
NO
YES
Updates the PIT, waits for the content to be fetched
Server receives the requested content
Server checks the Forwarding Information Base (FIB) if any forwarding information is available to find out the requested content
Available?
NO
YES
The routing function within the server machine starts to operate
Routing Function
Forwards the request, updates the PIT

FIGURE 3. **Content request and retrieval procedure using the developed functions; Part – 2.**

As soon as the client starts to receive the requested content, the mobility support function comes into motion and maintains a seamless and continuous connection with a server. This function locates the next best server that can continue to deliver the requested content if the client moves away from the currently connected server and sends a request for the desired content in advance to the target server. On the contrary, if the client is not on the move, this function does not have any further responsibility. At last, after the retrieval process is completed, the auto-play function takes over the program and starts auto-playing the audio or video file using appropriate software installed on the client device, and if the retrieved content is a text file, the auto-play function opens up that file for the client to read. Responsibilities and the relative positions of these two functions are shown in Fig. 2 earlier. The whole procedure keeps on repeatedly running as long as the experiment is not finished.

D. DEVELOPED FUNCTIONS FOR NAMING

One of the distinguishing features of the ICN paradigm is identifying the contents using a unique name. Assigning unique names to all the available contents is essential for the content request and retrieval process. Therefore, new functions were developed and implemented within the testbed architecture, which performs the tasks described below.

The naming function in the basic CCN can name the contents. However, the machines of the testbed architecture also need to identify and recognize each other for exchanging various control information. Therefore, we propose to assign distinguishable names to the machines of the testbed architecture. The function NAME_MACHINES () shown in Algorithm 1 assigns unique names to all the machines of the testbed architecture as soon as they become available. These machines initially broadcast their assigned names to neighboring machines so that each of the machines can identify the other machines for exchanging various control information. We call this name the *DisplayName*, as this name provides a way to recognize the MAC addresses of the testbed machines by a human-readable name. The names are generated by mapping the physical MAC address with an alphabetically ordered name when a machine joins the network. For example, a machine with the MAC address of *94-M2-05-B9-37-F5* can be assigned the *DisplayName* of *client-A*. Similarly, the machines can have various names, including *server-Y*, *server-AC*, and *client-ZX*. Therefore, the server machines can identify each other using their *DisplayNames*. Then, Algorithm 1 invokes Algorithm 6 and executes the function MAP_NAMES (*DisplayName*) that maps the *DisplayNames* of the server machines with their IP addresses so that machines outside of the testbed architecture can recognize them.

The second function for naming creates and provides distinct *alias* names to the *given* names of the contents uploaded in the server machines. This function is called NAME_CONTENTS (), and it is presented in Algorithm 2. Each content uploaded to the server repositories gets a unique name assigned by the naming mechanism of the basic CCN. We call these names the *given* names. The testbed architecture uses the same naming mechanism for assigning the *given* names. Additionally, the developed function takes these *given* names of the cached contents as an input and generates related *alias* names using the longest prefix matching algorithm for quick identification whenever requested by the client machines. Blank spaces between the words, capitalizations of the alphabets, and special characters from the *given* names are not considered while creating the *alias* names. For example, the same content may have diverse *given* names when it is uploaded to different server machines such as *Movie 1*, *movie_1*, or *Movie - 1*. However, the server machines create an *alias* for all these names of the same content as *movie1*. This *alias* name is then mapped with the *given* name of the content in the server repository. Hence, whichever name the client uses when requesting that content, the servers can always identify the desired content using the naming function and the *alias* name. The developed functions NAME_MACHINES () and NAME_CONTENTS () for the naming purpose are shown using the pseudo code below in Algorithm 1 and Algorithm 2.

Algorithm 1 Algorithm for Naming the Machines Using Function **NAME_MACHINES ()**

```
INPUT: MAC-addresses
OUTPUT: DisplayNames
1.  DETERMINE machine_type (server | client)
2.  GET MAC-address
3.  CREATE a DisplayName
4.      SET DisplayName: = machine_type-
        incrementing_alphabets
5.  MAP DisplayName ← MAC-address
6.  CALL MAP_NAMES (DisplayName)
```

Algorithm 2 Algorithm for Naming the Contents Using Function **NAME_CONTENTS ()**

```
INPUT: given names
OUTPUT: alias names
1.  GET given name
2.      Content name assigned by the basic CCN
3.  IF given name CONTAINs blank spaces
4.      DISCARD blank spaces
5.  ELSE IF given name CONTAINs capitalizations
6.      DISREGARD capitalizations
7.  ELSE IF given name CONTAINs special characters
8.      REMOVE special characters
9.  END IF
10. CREATE alias name from the updated given name
11.     USE longest prefix matching algorithm
12.     MAP alias name ← given names
```

E. DEVELOPED FUNCTIONS FOR ROUTING

After understanding the name of the server machine and the names of the available contents at the local storage, the server machine requires the information regarding its connected neighbor servers and the list of accessible contents at those servers to locate and deliver a requested content that is not available at the CS. For that reason, the routes among the connected servers needed to be created, and this information is stored in the FIB. New functions were developed and implemented within the testbed architecture so that the tasks performed by the functions developed for naming can be recognized and complemented in addition to supporting the operations of the developed testbed architecture.

Albeit the basic software supports the CCN-based routing mechanisms, it does not create any routing information as it performs the content exchange procedure within one machine only. Therefore, one of the functions developed for routing, CREATE_ROUTES (), creates random connections with the available neighboring servers shortly after a new server machine enters the network. This function is presented in Algorithm 3. It creates FIB tables and makes routes among random servers using the same algorithms as in the basic CCN for all the machines that comprise the testbed architecture. For example, if a new server machine, *server-Y*, becomes alive, it broadcasts its name so that other servers can identify the *server-Y*. In return, the existing servers, such as *server-A* and *server-AC*, adds new routing information to reach the new server machine. On the other hand, the existing server machines periodically update their FIBs by exchanging control messages with the neighboring server machines. When a server machine drops out of the network, it cannot respond to those control messages. Therefore, all the routing information concerning this machine can be removed, and the FIB is updated accordingly. Moreover, new routes may need to be formed among the existing servers in some cases to complete the networking routes. Another function for routing, REMOVE_ROUTES () presented in Algorithm 4, handles these tasks. For example, if the *server-Y* from the earlier example becomes unavailable within the network, a direct connection between *server-A* and *server-AC* may be created after removing all the routing information to and from *server-Y* from the FIB of *server-A* and *server-AC*.

Additionally, the list of all the available contents in the CS of each server machine is renewed by another developed function for routing whenever there is any change in the network topology. This function is called UPDATE_CONTENTS_LIST (), and it is shown in Algorithm 5. Whether a new server with loads of contents comes into the network or some of the contents become unavailable due to an existing server leaving the network, FIBs of all current server machines will always remain up to date with the help of this function. For example, if a new server, *server-Y*, becomes available, existing servers, such as *server-A* and *server-B*, will update the list of newly available contents and include *movie_1,* which is available in *server-Y*. The lists of contents are continually updated by control message exchange using this routing function. Using the function, the number of the same contents can also be managed efficiently.

Furthermore, we have created a resolution server with a name resolution database to store several necessary and helpful information. The IP addresses of all the server machines within the testbed architecture are mapped and stored with their *DisplayNames* in the resolution server by a developed function called MAP_NAMES (name) given in Algorithm 6. As mentioned before, Algorithm 1 invokes this function in Algorithm 6. This feature of the developed function for routing can be utilized by the machines outside of the testbed architecture in order to recognize the testbed servers and execute the content exchange procedure without a hitch. Additionally, this feature was also helpful when we conducted the content delivery experiments using the current host-based Internet environment without further ado. The Algorithms 3-6 followed by the developed functions for the routing purpose are shown using the pseudo code below.

Algorithm 3 Algorithm for Creating Routes Among the Server Machines Using Function **CREATE_ROUTES ()**

```
1.  IF server-Y JOINs the topology
2.  CREATE FIB for the new server
3.      ADD random routes to the neighbor servers
4.          USE basic CCN mechanism
5.  UPDATE FIB of the neighbor servers
6.      ADD random routes to the new server
7.          USE basic CCN mechanism
8.  END IF
```

Algorithm 4 Algorithm for Removing Existing Routes Between the Server Machines Using Function **REMOVE_ ROUTES ()**

```
1.  IF server-Y LEAVEs the topology
2.  FOR each server-X having server-Y in the FIB
3.      DELETE FIB entry to server-Y
4.      IF server-Y was an intermediate server between 2 other
         servers
5.          UPDATE FIB of the neighbor servers
6.          ADD direct connection between them
7.      END IF
8.  END FOR
9.  END IF
```

Algorithm 5 Algorithm for Updating the List of Available Contents Using Function **UPDATE_CONTENTS_LIST ()**

```
1.  IF server-Y JOINs the topology
2.      FOR each server-X in the FIB of server-Y
3.          FOR each content in server-Y
4.              GET given name
5.              ADD new content info to the FIB of
                 server-X (server-Y $\rightarrow$ server-X $\leftarrow$ given
                 name)
6.          END FOR
7.      END FOR
8.  ELSE IF server-Y LEAVEs the topology
9.      FOR each server-X having server-Y in the FIB
10.         FOR each content entry via server-Y
11.             REMOVE content info from the FIB
12.             DELETE entry (server-X $\rightarrow$ server-Y $\leftarrow$
                given name)
13.         END FOR
14.     END FOR
15. END IF
```

F. DEVELOPED FUNCTIONS FOR CACHING

The caching scheme has a vital role in maintaining network performance as it can be simultaneously responsible for network congestion and faster content distribution. Caching the contents at a nearer hop can reduce the delivery time and network resource consumption as well. On the contrary, a poorly organized caching scheme may cause recurrent cache overflow and cache-miss. The basic software uses the same caching schemes for content caching and content removal used by the basic CCN paradigm. For a faster content delivery procedure, the basic CCN uses an in-network caching policy where Leave Copy Everywhere (LCE) caching scheme is used, and all intermediate machines cache all the contents that transit via those machines.

Algorithm 6 Algorithm for Mapping the *DisplayNames* With the IP Addresses Using Function **MAP_NAMES (Name)**

```
INPUT: DisplayNames, IP Addresses, Server Locations
OUTPUT: Information uploaded to the name resolution database
1.  SET DisplayName: = name
2.  GET IP address of the server
3.  MAP DisplayName ← IP-address
4.  UPLOAD to the name resolution database
5.      DisplayNames mapped with the IP addresses
6.      Server locations that will be used by the
         TARGET_SERVER() function
```

Therefore, an intermediate machine can speed up the subsequent content delivery process by providing the requested content from its CS. Simultaneously, the basic CCN uses the Least Recently Used (LRU) caching scheme to replace contents whenever there is a lack of space. According to this scheme, the least active content in recent times is removed from the CS. However, both these caching schemes, LCE and LRU, have their drawbacks and bear pitfalls causing cache overflow and cache-miss. The LCE scheme replicates all the accessed contents at all the machines, resulting in several duplicate contents that may quickly fill up the repository spaces of the servers. On the other hand, the LRU scheme removes the contents based on its access time, which is expensive in keeping track of which content was requested when and may result in an inefficient content replacement scheme. As a result, the basic CCN network may often face both cache overflow and cache-miss disadvantages. Therefore, we improved the existing ICN-based caching mechanism by developing functions that execute an entirely different set of algorithms for content caching and cache removal. The details of the caching functions used in the testbed architecture will be included in our future work. In this paper, we briefly introduce them below.

The developed functions for caching are based on a new algorithm developed for the testbed architecture, and we named this algorithm the content popularity ranking (CPR) algorithm. The details of this algorithm will be presented as a separate proposal in our future work. The CPR algorithm follows an updated and improved mechanism of our previous work published in [24] that essentially tries to identify the content popularity based on various labels assigned to the contents. The efficiency of the previous mechanism was evaluated in NS-3, where different contents were distinguished by their labels that were allocated during the content registration process. For the testbed architecture, the algorithms are made more precise, and in summary, the CPR algorithm takes into account the number of times a content is requested and delivered, the file type of that content, and the duration since the content was published to calculate the popularity ranking of each content. Then, it sorts the available contents at the CS of each server in real-time and chooses the appropriate content to either cache or discard based on its popularity ranking. A *PopularityThreshold* is calculated for each server individually that holds the edge to determine whether the incoming content falls on the safe side of the boundary or on the other side to be removed. Individual content popularity of all the available contents at a server is calculated, and the average popularity value of the available contents at a server is considered the *PopularityThreshold* of that server. After a server fetches a requested content that was not available at the CS beforehand, the caching function runs in the background, uses the CPR algorithm to rank and sort the incoming content, and utilizes the *PopularityThreshold* value to conclude the fate of that content. A content is cached if its popularity is higher than the *PopularityThreshold* of the server machine; otherwise, it is discarded. If the content needs to be cached, but there is not enough space in the CS to store it, then the caching function removes as many existing contents as necessary using the content removal feature of the CPR algorithm. This time, the existing contents are sorted in terms of their popularity ranking, and the content having the lowest popularity ranking is removed from the repository one by one until there is enough space for caching the new content. The research work conducted so far indicates that the servers can select more popular contents for caching and less popular contents for erasing when necessary by following this CPR algorithm. In this way, all contents are not duplicated at all the machines; instead, repeatedly requested contents are cached at the machines that are nearer to the clients, and they are stored for a longer time than the rarely requested contents. This benefit also impacts the overall content delivery time and improves the performance of the network topology. The developed functions for the caching purpose, CACHE_CONTENTS () and REMOVE_CONTENTS (), are shown briefly in Algorithm 7 and Algorithm 8 using the pseudo code below.

G. DEVELOPED FUNCTIONS FOR MOBILITY SUPPORT

These days, people are using wireless devices a lot more than before, and they are surfing the Internet while on the move. Therefore, a client may move closer towards a different server than the currently connected one from which it is downloading the requested content. If the client keeps moving for a certain amount of time at a vehicular speed, a scenario may arise where the client gets disconnected from the serving server before completing the content retrieval process. Hence, the client needs to connect with another server to continue receiving the previously requested content as quickly as possible. Mobility support is a process that creates a connection with a target server seamlessly without disrupting the continuity according to the clients' perception. Although the basic CCN supports user mobility, the basic software does not include any mobility support function as the content exchange procedure is done within one machine only. Therefore, we developed new functions for providing mobility support that is implemented within the testbed architecture. These new functions locate a target server in advance, send a request for the same content to the target server, make a quick connection with the target server shortly before the

Algorithm 7 Algorithm for Caching a New Content Using Function **CACHE_CONTENTS ()**

```
INPUT: $CPR$, $PopularityThreshold$
OUTPUT: Decision on whether to cache or not a new content $C$ at a server-$N$
1.  CALCULATE $CPR$ of each content at the server-$N$
2.      USE $download\_counter$, $file\_type$ (video | audio | text), $age\_of\_content$
3.      SET $cpr := CPR_C$
4.  CALCULATE $PopularityThreshold$ of the server-$N$
5.      AVERAGE $CPR$ value of all the available contents at the server-$N$
6.      SET $pt := PopularityThreshold_N$
7.  IF $cpr \geq pt$
8.      CACHE the content $C$ at the server-$N$
9.  ELSE IF $cpr < pt$
10.     DISCARD the content $C$
11. END IF
```

Algorithm 8 Algorithm for Removing an Existing Content Using Function **REMOVE_CONTENTS ()**

```
INPUT: $CPR$ of all the contents at the server-$N$
OUTPUT: Selection of a content $Z$ for removal from the server-$N$
1.  FOR all the contents at the server-$N$
2.      CALCULATE $CPR$ of each content
3.  END FOR
4.  SORT all the contents at the server-$N$
5.      in terms of their $CPR$
6.  WHILE (lack of space)
7.      REMOVE a content $Z$
8.          having the lowest $CPR$
9.  END WHILE
```

handover, and help the client machines continue receiving the requested content seamlessly. All the ICN nodes are initially provided with information about the name resolution server, and the information about the content servers can be obtained from the name resolution database.

The functions developed for providing seamless mobility support begin to execute after the client starts to receive the requested content from the server. At first, MOBILITY_SUPPORT () function determines whether it is necessary to execute the rest of the functions or not. Because if the client device is a fixed desktop computer or even if the device is a wireless one but not in motion, it is not worth keeping running this function as there is no possibility of a handover. Additionally, when the client device moves away at a vehicular speed and a significant portion of the requested content is yet to be received, a new target server may be needed to be located at that time. Therefore, this function is not triggered if the clients are fixed machines or wireless machines that are not moving, and the content is almost downloaded. Procedures followed by this function are shown in Algorithm 9. On the contrary, if a handover is necessary, the TARGET_SERVER () function described in Algorithm 10 looks for a proper target server that can seamlessly provide the requested content by creating a quick connection with the client machine. As the client moves on, if this function can locate only one target server, it directly goes to the function HANDOVER (new_server) that executes a seamless handover after the handover decision is triggered, which is expressed by H. However, if multiple potential target servers are found, it executes several more steps in order to select a target server.

The procedures followed by the TARGET_SERVER () function are a reformed version of the procedures presented in one of our previous works [25]. It is modified to fit with the context of the experiments carried out using the testbed architecture. This function collects the necessary information such as locations of the current server and nearby potential target servers, the location, speed, and direction of movement of the client machine. The potential target servers are those machines that already have the content requested by the client in their CS. As mentioned before, the routing function UPDATE_CONTENTS_LIST () allows the server machines to share the lists of available contents, so it is possible to identify which server machines can act as the potential target servers. Additionally, since the current content server can inform the target server of the current content being served, the client device can continue receiving the previously requested content without interruption. It is preferable if the potential target server is in a similar direction as the direction in which the client is moving. The position information of the content servers, which was uploaded using the function MAP_NAMES (name), can be obtained from the name resolution database, and the client device can also get the location of itself. These locations are used by the function TARGET_SERVER () explained in Algorithm 10. Assuming that the locations of the client, the connected server, and the potential target server are, (X_C, Y_C), (X_S, Y_S), and (X_T, Y_T), respectively. The distance between the client and the currently connected server and the distance between the client and a potential target server are calculated according to (1) and (2), respectively, expressed by D_S and D_T. The handover decision is triggered if the value of H becomes bigger than 0.5 in (3).

$$D_S = \sqrt{(X_C - X_S)^2 + (Y_C - Y_S)^2} \quad (1)$$

$$D_T = \sqrt{(X_C - X_T)^2 + (Y_C - Y_T)^2} \quad (2)$$

$$H = \frac{(D_S - D_T)}{D_S} \quad (3)$$

The current server sends out IPERF messages to the potential target servers to find the performance of the network by measuring the throughput and assigns a value for the *server_priority* variable to each of the potential target servers. The motive behind this is to prioritize a target server with the highest throughput at the time of handover. The *server_priority* variable is expressed by p, and its value is taken from (4).

$$p = \begin{cases} 0.9, & if\ through\ put\ is\ high \\ 0.5, & if\ through\ put\ is\ average \\ 0.1, & if\ through\ put\ is\ low \end{cases} \tag{4}$$

Finally, one target server is selected among the available potential target servers that achieves the highest value of the *server_preference* variable, S from (5).

$$S = \delta * H + \Phi * p \tag{5}$$

where, $\delta = 0.8$ and $\Phi = 0.2$, and they are tunable parameters. The target server is selected such that it has the shortest distance from the client and the highest throughput among the other potential target servers. As soon as a target server is picked, the HANDOVER (new_server) function is invoked, and the steps followed by this function are described in Algorithm 11. This function sends a new request to that target server to prepare the desired content of the client in advance for delivering it. After that, when the handover is triggered, the client is disconnected from the currently connected server after quickly connecting with the target server. The target server begins to deliver the requested content spontaneously, and as a result, the client keeps on receiving the requested content seamlessly. The Algorithms 9-11 followed by the developed functions for the mobility support purpose are shown using the pseudo code below.

Algorithm 9 Algorithm for Determining Whether Handover Is Necessary or Not Using Function **MOBILITY_SUPPORT ()**

```
INPUT: Clients' information (device_type, speed, download_status)
1.  IDENTIFY device_type (wired or wireless)
2.  IF device_type = wired
3.      BREAK
4.  ELSE IF device_type = wireless
5.      GET speed (vehicular | non-vehicular)
6.      IF speed = non-vehicular
7.          BREAK
8.      ELSE IF speed = vehicular
9.          CHECK download_status (over 70% | at or
            below 70%)
10.          IF download_status = over 70%
11.              BREAK
12.          ELSE IF download_status = at or below 70%
13.              CALL TARGET_SERVER ()
14.          END IF
15.      END IF
16.  END IF
```

H. DEVELOPED FUNCTION TO AUTO-PLAY

Another new feature was implemented within the testbed architecture that gives the client machines the capability to auto-play the retrieved contents automatically after receiving the requested contents completely. A new function was developed for this purpose. The content delivery times are

Algorithm 10 Algorithm for Selecting the Target Server for the Handover Using Function **TARGET_SERVER ()**

```
INPUT: Clients' information (location, requested_content)
INPUT: Servers' information (target_servers (using info from function
UPDATE_CONTENTS_LIST () - requested content by the client is avail-
able), locations (from the name resolution database))
1.  GET location, requested content name
2.  GET number_of_target_servers and the locations of
    current_server and potential target_servers
3.  IF number_of_target_servers = 1
4.      CALL HANDOVER (target_server)
5.  ELSE IF number_of_target_servers > 1
6.      CALCULATE distances ($D_S$, $D_T$)
7.          $D_S$ = distance between client and current
            server (using (1))
8.          $D_T$ = distance between client and potential
            target servers (using (2))
9.      DECIDE handover_trigger decision
10.         CALCULATE $H$ (using (3))
11.     ASSIGN values to the server_priority variable
12.         SEND IPERF messages
13.         SET $p$ = (0.9 | 0.5 | 0.1) (using (4))
14.     FOR each target server having $H > 0.5$
15.         CALCULATE server_preference variable, $S$
            (using (5))
16.     END FOR
17.     SORT target servers in terms of $S$
18.     SELECT the server with the highest value of $S$ as the
        target_server
19.     CALL HANDOVER (target_server)
20.  END IF
```

Algorithm 11 Algorithm for Executing the Handover Using Function **HANDOVER (New_Server)**

```
1.  SET target_server: = new_server
2.  RESEND request for requested_content to target_server
3.  CONNECT with the target_server
4.  START receiving the requested_content
5.  DISCONNECT from the current_server
```

already measured, and this function is not a part of the graphical results. However, we deduce that the content auto-play function can be highly convenient for the clients who have requested a content, whether it is an urgent situation or even just for entertainment. This function enables the client machines to distinguish various file extension categories, including text, audio, and video, from the miscellaneous retrieved contents. Then, it helps the client machines to select the appropriate software for running the retrieved content. Therefore, the client machines can automatically open a text file and auto-play an audio or video file right after downloading the expected content from the server machines. Fig. 4 illustrates that a client machine is auto-playing a video content right after retrieving it from the server without any manual intervention from the user. The developed function

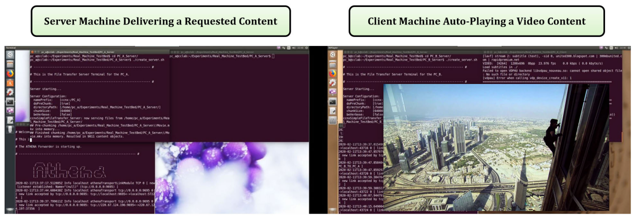

Server Machine Delivering a Requested Content
Client Machine Auto-Playing a Video Content

FIGURE 4. Auto-play of a video content by a client machine.

called AUTO_PLAY () for auto-playing a retrieved content is shown in Algorithm 12 using the pseudo code below.

Algorithm 12 Algorithm for Auto-Playing a Retrieved Content Using Function **AUTO_PLAY ()**

```
1. IDENTIFY the retrieved content type
2.     text | audio | video
3. FIND software to run the content
4. AUTO-PLAY the retrieved content
```

IV. PERFORMANCE ANALYSIS

This section presents the testbed configurations, explains the topology setup procedure, illuminates the experimental scenario, and demonstrates that in terms of content delivery time. The proposed testbed-based architecture outperforms the current Internet architecture and the basic CCN-based environment by utilizing the developed functions.

A. TESTBED CONFIGURATIONS, TOPOLOGY SETUP, AND THE EXPERIMENT

The primary purpose of developing the testbed architecture is to provide a framework where content request and retrieval experiments can be performed in a real ICN-based environment. Therefore, we built a network topology that resembles the real world as closely as possible. Although the developed testbed architecture can be used in a multitude of scenarios, in this paper, we analyzed the performance of the ICN-based testbed architecture assisted by the developed functions in terms of content delivery time, as it is a primary and the most crucial performance indicator from the user perspective.

The testbed architecture consists of five server machines that are regular desktop computers. Among them, three are considered remote servers, and two of them acted as cloud servers. These five machines were placed in different geographical locations at a distance so that they fall under two different cellular base stations. The base stations are needed for the handover of the moving client devices. Otherwise, they do not have any impact on the testbed operations. We used 10 different client devices that include desktop computers and laptops. All these devices had access to the Internet via Ethernet, Wi-Fi, or 4G cellular connection. Additionally, all these devices were equipped with the basic software and the developed functions as well. The basic software runs best on the OS platform Ubuntu 14. Therefore, the operating system of all the machines of the testbed architecture was Ubuntu 14. Various random contents were used that are of

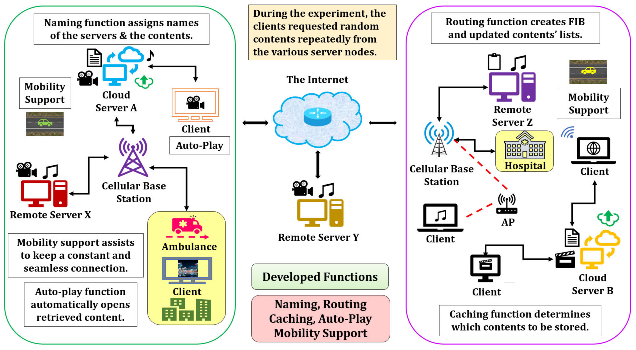
TABLE 1. The testbed configurations.

Software		Hardware	
Basic Software	CCNx Distillery 2.0	Processor	Intel(R) Xenon(R)
OS Platform	Ubuntu 14	System Type	64-bit
Internet Connection	Ethernet, 4G, Wi-Fi	RAM & CPU	32 GB 3.30 GHz

Nodes		Contents	Sizes
Servers	Clients	Text	50; 100 KB ~ 4000 KB
5 Desktop Machines	10 Different Devices	Audio	20; 1 MB ~ 15 MB
3 Remote Servers, 2 Cloud Servers	Desktop Computers, Laptops	Video	15; 1 MB ~ 200 MB

different types and sizes as well. We used 50 text files that ranged from 100 Kilo-bytes (KB) to 4000 KB, 20 audio files that ranged from 1 Megabyte (MB) to 15 MB, and 15 video files that ranged from 1 MB to 200 MB. The cache size of the repository was fixed initially for each of the servers, but that size varied throughout the experiment in order to evaluate the performance of the caching function using various cache sizes. The cache size of the repository started from 10 MB, and the maximum size was 250 MB. The testbed configurations are summarized in Table 1. Although we used five server machines only, the testbed architecture is entirely scalable, and it is possible to add more server machines and client machines if available. The testbed architecture can be configured in different topologies easily, and it can also be extended by adding more machines. Additionally, the testbed architecture can be developed using different hardware than what we used.

The contents were stored randomly within the two cloud servers at the beginning of the experiment. Each client device was randomly attached to a server machine and can request 100 times for random contents. Besides containing the names of the available contents, the generated names of the requested contents also included names of contents that are not available, misspelled names, and names spelled with different capitalizations. Therefore, requests for the unavailable contents were unsuccessful, requests for the mismatched names of the contents needed the help of the naming function, and some of the content requests needed to be forwarded by the routing function, as they were not readily available in the connected server. For those requests which needed the contents to be fetched from other servers, the caching function was executed and decided whether to cache the retrieved contents or not. We took the client devices in a car and requested various contents while the car was moving. The client devices used the cellular network in order to connect with the Internet while they were on the move. As a result, the physical locations of the content servers and the client devices are utilized only in the mobility support function when a target server was searched and selected. Hence the mobility support function was needed, and it helped maintain a seamless connection with the server machines. After a content request is successfully completed, the auto-play function auto-played the downloaded contents. The servers and clients are dispersed, and therefore it is quite challenging to capture and draw the whole picture and topology. Therefore, we illustrated a testbed topology in Fig. 5 to better inform the possible scenarios of utilizing the testbed architecture. The testbed servers are in the fixed locations; however, the client devices can move.



Naming function assigns names of the servers & the contents.
Mobility Support
Cloud Server A
Client
Auto-Play
Cellular Base Station
Remote Server X
Mobility support assists to keep a constant and seamless connection.
Ambulance
Client
Auto-play function automatically opens retrieved content.
During the experiment, the clients requested random contents repeatedly from the various server nodes.
The Internet
Remote Server Y
Developed Functions
Naming, Routing Caching, Auto-Play Mobility Support
Routing function creates FIB and updated contents' lists.
Remote Server Z
Mobility Support
Cellular Base Station
Hospital
Client
AP
Client
Client
Cloud Server B
Caching function determines which contents to be stored.

FIGURE 5. A Depiction of the testbed architecture.

The content delivery time is measured when a client sends a content request and receives that content completely. However, if the requested content is not available and the client does not receive the content, it is considered a failed attempt, and the delivery time for this attempt is not stored in the result output file. The available contents are of various types and sizes. The content request and retrieval process continued until each client requested various contents 100 times, making it one complete experiment, and the performance evaluation graphs are given based on the average result of the whole process. However, we separated the performance results into two different graphs as the ranges of both the axes were very high. Additionally, we intended to make the performance trend in terms of content delivery time clearer using the two different graphs. In the 1st graph, we plotted the delivery times of the contents: all text files and audio and video files below 7 MB. In the 2nd graph, we plotted the delivery times of the audio and video files only and no text files. All the content delivery times were initially measured in milliseconds (ms). However, in the 2nd graph, the unit is shown in seconds (s) as most of the delivery times were over 1000 ms. The performance in terms of the content delivery time of the proposed testbed architecture assisted by the developed functions was compared against the performance of the current Internet and the basic CCN. Although the content delivery times increased in all the network architectures as the size of the contents increased, the proposed testbed architecture outperformed the two other architectures in terms of content delivery time, as shown by the following graphs.

B. EXPERIMENTAL RESULTS WITH MISCELLANEOUS SMALLER CONTENTS

This section provides the results of content exchange experiments where diverse types of contents were used. The contents include all the 50 text files of size from 100 KB up to 4000 KB and some audio and video files under 7 MB in size. As the units of the contents are different, we normalized the unit by converting them into MB. Fig. 6 summarizes the 1st part of the outcome of the experiments.

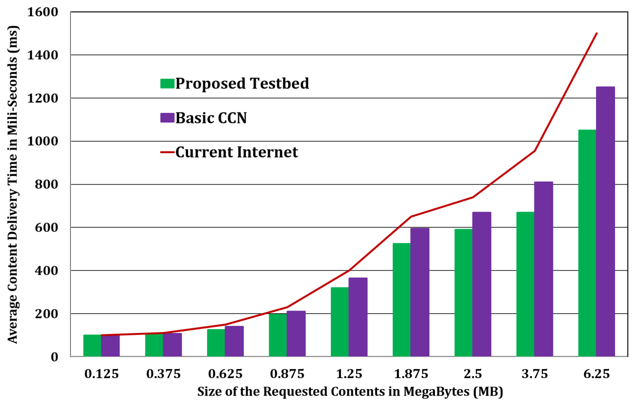


FIGURE 6. Experimental results with small-sized miscellaneous types of contents.

Initially, the content delivery times were similar for all the networks as it does not take much time to deliver a short amount of data. However, as the content size increased, the basic CCN environment required more time to deliver the requested contents than the proposed testbed architecture, and the current Internet environment took an even longer time. To deliver a requested content of size 1.25 MB, the server machines from the testbed architecture, basic CCN paradigm, and current Internet environment required 320 ms, 365 ms, and 400 ms on average, respectively. They needed as much as 1050 ms, 1250 ms, and 1500 ms on average to deliver a content of size 6.25 MB.

The purpose of Fig. 6 is to show the trend in performance in terms of the content delivery time for all the three networks. A longer time was required to deliver the requested contents as the size of the contents increased. The main takeout is that, even though the proposed testbed architecture also took a longer time to deliver the requested contents when the size got bigger, the required time is still much less than the other two architectures. The experiment continued, and we collected further data that shows the time required for delivering contents of even bigger size than 6.25 MB. From the graph, the trend is that the testbed-based architecture enhanced by the developed functions will continue to outperform the two other networks in terms of content delivery time.

C. EXPERIMENTAL RESULTS WITH LARGER-SIZED AUDIO AND VIDEO CONTENTS

We present another graphical demonstration of content delivery time from the same continued experiment. We plotted the delivery times for the contents, which were audio and

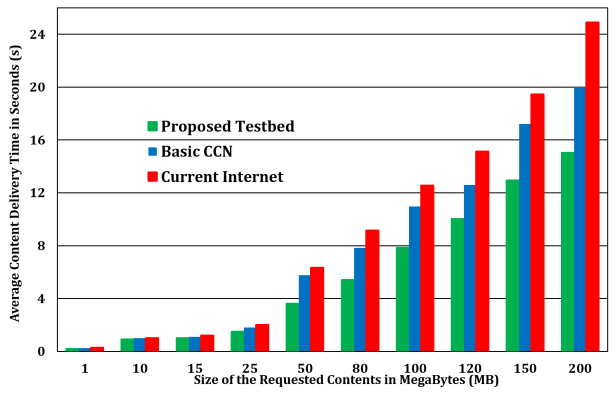


FIGURE 7. Experimental results with larger sized audio and video contents.

video only. This graph's primary purpose is to show the performance trend for the three architectures in terms of content delivery times for larger-sized audio and video contents. The size of the contents ranged from 1 MB up to 200 MB. Although the content delivery times were initially measured in milliseconds, the unit shown in the graph is seconds, as most of the delivery times were over 1000 ms. Fig. 7 shows that the requested contents were delivered faster when the server machines used the developed functions within the testbed architecture than when the server machines used the current Internet environment and the basic CCN architecture.

Like the previous experimental results, servers from the testbed architecture with the developed functions outperformed the servers from the basic CCN architecture and the servers from the current Internet environment in terms of the content delivery times. Figuratively speaking, on average, respectively 3 s, 5 s, and 6 s were needed to deliver the contents of 50 MB size. As soon as the requested contents grew to 200 MB, they took 15 s, 20 s, and 25 s to complete the delivery process of the requested content.

This graph also shows the performance trend in terms of the delivery time needed for the requested contents in all the three architectures. It indicates that the testbed-based architecture enhanced by the developed functions delivers the requested contents faster than the basic CCN and the current Internet. Furthermore, the client machines automatically played all the received audio and video contents in the testbed architecture right after retrieving those contents. Therefore, a few more seconds can be saved to complete watching the requested contents in real life.

V. CONCLUDING REMARKS

This paper has introduced a novel ICN-based testbed architecture and presented several functions added to the testbed in order to improve the performance of the basic CCN network in terms of content delivery time. The rationale behind choosing a testbed-based approach is to provide a framework for evaluating the ICN-based networks, representing the real world as closely as possible. The primary design objective of this testbed architecture is to achieve various requirements, such as fast, secure, seamless, efficient, and reliable content delivery and retrieval for multimedia applications. To accomplish that, several functions were implemented within the testbed architecture. We modified the existing naming, routing, and caching functions and provided new functions such as mobility support and auto-play capability. We have described the testbed development procedure and the operations of the functions in detail. We believe that this testbed architecture will provide the base for many future researches in similar areas. We have presented the testbed configuration, explained the topology setup procedure and the experimental scenario, and provided the performance results. The illustrated results prove that the testbed-based architecture outperforms the basic CCN and the traditional Internet in terms of the content delivery time/response time for various sizes and different types of contents. Additionally, the testbed architecture can be extended, and various other performance metrics can be evaluated. As future work, we will continue to work on updating the testbed architecture and developing the functions.

REFERENCES

[1] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index, complete forecast update, 2017–2022," Tech. Rep., 2018.

[2] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, and C. Tsilopoulos, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, May 2014.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2009.

[4] G. Ma and Z. Chen, "Comparative study on CCN and CDN," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 169–170.

[5] *NS-3*. Accessed: Oct. 15, 2014. [Online]. Available: https://www.nsnam.org/

[6] *MathWorks—Makers of MATLAB and Simulink*. Accessed: Sep. 2, 2020. [Online]. Available: https://www.mathworks.com/

[7] *ndnSIM Documentation*. Accessed: Jan. 13, 2027. [Online]. Available: https://ndnsim.net/current/

[8] *CCN-Lite—A Lightweight CCNx Implementation*. Accessed: Aug. 21, 2018. [Online]. Available: https://github.com/cn-uofbasel/ccn-lite

[9] *CCNx Distillery Software Distribution 2.0*. Accessed: Aug. 21, 2018. [Online]. Available: https://github.com/parc-ccnx-archive/CCNx_Distillery

[10] D. R. Cheriton and M. Gritter, "TRIAD: A new next-generation Internet architecture," Tech. Rep., 2000.

[11] T. Koponen, M. Chawla, and B. G. Chun, "A data-oriented (and beyond) network architecture," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, vol. 7, 2007, pp. 181–192.

[12] D. Lagutin, K. Visala, and S. Tarkoma, "Publish/subscribe for internet: PSIRP perspective," in *Future Internet Assembly*. 2010, pp. 75–84.

[13] *PSIRP*. Accessed: Aug. 21, 2018. [Online]. Available: http://www.psirp.org/

[14] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[15] *FP7 PURSUIT Project*. Accessed: Aug. 21, 2018. [Online]. Available: http://www.fp7-pursuit.eu/

[16] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (NetInf)–an information-centric networking architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721–735, 2013.

[17] *FP7 4WARD Project*. Accessed: Aug. 21, 2018. [Online]. Available: http://www.4ward-project.eu/

[18] *FP7 SAIL Project*. Accessed: Aug. 21, 2018. [Online]. Available: http://www.sail-project.eu/

[19] L. Zhang, A. Afanasyev, J. Burke, and V. Jacobson, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[20] L. Hyunwoo, D. Kim, J. Suh, and T. Kwon, "ICN-OMF: A control, management framework for information-centric network testbed," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2015, pp. 416–417.

[21] P. Batista, I. Araújo, N. Linder, K. Laraqui, and A. Klautau, "Testbed for ICN media distribution over LTE radio access networks," *Comput. Netw.*, vol. 150, pp. 70–80, Feb. 2019.

[22] *Cicn*. Accessed: Jul. 3, 2019. [Online]. Available: https://wiki.fd.io/view/Cicn

[23] N. A. Nasir and S.-H. Jeong, "Testbed-based performance evaluation of the information-centric network," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 166–169.

[24] N. A. Nasir and S.-H. Jeong, "Content popularity based fast content transmission in the wireless environment," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2017, pp. 229–231.

[25] N. A. Nasir and S.-H. Jeong, "A prediction-based mobility support in the content-centric mobile networks," in *Proc. 9th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2017, pp. 613–616.

NAZIB ABDUN NASIR received the B.S. degree in computer science and engineering from the University of Dhaka, Dhaka, Bangladesh, in 2014, and the M.S. degree in information communications engineering from the Hankuk University of Foreign Studies, South Korea, in 2016, where he is currently pursuing the Ph.D. degree in information communications engineering.

Since 2014, he has been a Research Assistant with the Computer Communications Laboratory, Department of Information Communications Engineering, Hankuk University of Foreign Studies. His major research interests include information-centric networking, content-centric networking, 5G internet, eHealth, and android development.

SEONG-HO JEONG received the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA.

From 1990 to 2001, he worked as a Senior Member of Research Staff with the Electronics and Telecommunications Research Institute (ETRI). He is currently a Professor with the Department of Information and Communications Engineering, Hankuk University of Foreign Studies (HUFS). He was the Vice Dean of HUFS and the Vice President of The Korean Institute of Information Scientists and Engineers. He is the Vice President of the Korean Institute of Information and Communications Sciences. He has been working as a consultant for various institutes and companies. He has led various industry projects as a principal investigator in the areas of multimedia communications and applications including eHealth, wireless/mobile networks, 5G/future networks, QoS, QoE, home networks, cross-layer design, networked robot, mobility management, signaling, CCN/CDN, and others. He has many journal/conference publications, patents, and standards in the areas of communications, multimedia systems, and computer networks. He also served as a TPC member for numerous international conferences. He has been serving as the Vice Chairman of ITU-T SG12 (Performance, QoS, and QoE) and the Chairman of ITU-T SG16 WP1 (Multimedia Content Delivery). He has also been serving as an Editor of various ITU-T Recommendations on multimedia systems and QoS/QoE in ITU-T SG16 and ITU-T SG13, since 2002. In addition, he has contributed to standardization in other SDOs, including IETF and 3GPP. He served as the General Chair for ICOIN 2017 and ICUFN 2019 and a Technical Program Committee Chair for international conferences, including ICUFN 2011, ICOIN 2012, ICTC 2013, ICTC 2014, and ICTC 2015 technically co-sponsored by IEEE and IEICE.