An Incrementally Deployable Network Architecture to Support Both Data-centric and Host-centric Services

Hongbin Luo, Hongke Zhang

School of Electronic and Info. Eng.
Beijing Jiaotong University
Beijing China, 100044
Email: {hbluo, hkzhang}@bjtu.edu.cn

Moshe Zukerman

Electronic Engineering Dept.
City University of Hong Kong
Hong Kong SAR
Email: m.zu@cityu.edu.hk

Chunming Qiao

Department of Computer Science & Engineering
SUNY at Buffalo
Buffalo, NY 14260-2000, U.S.A.
Email: qiao@computer.org

Abstract
Most Internet traffic is associated with applications where users are interested in the data and not in the source where the data reside. On the other hand, the current Internet architecture is host-centric rather than data-centric. This motivates a new network architecture that can efficiently support both data-centric and host-centric services. In this paper, we describe in detail an implementation of an incrementally deployable Data-Centric Network Architecture (DCNA) for the Internet. DCNA is based on the inclusion of a shim layer between the application layer and the transport layer and the appropriate interfaces to efficiently connect these layers. In addition to being data-centric and incrementally deployable, the proposed DCNA can effectively support mobility, multi-homing, and ease the adoption of new applications and networking technologies.

Keywords:
Network architecture, future Internet, data-centric.

I. Introduction

The current Internet architecture was designed in the 1970s. At that time, all applications (such as file transfer and remote login) focused strictly on host-to-host communication. That is, users explicitly direct the source to communicate with another host and the network's role is to carry packets to the destination address.

However, along with the rapid development and the tremendous success of the Internet in the past few decades, the vast majority of the current Internet usage has changed to data retrieval and service access. For instance, it is reported that about 85% of Internet traffic in North America is real-time entertainment traffic, peer-to-peer (p2p) file sharing traffic, and web browsing traffic [1]. This indicates that users are mainly interested in certain data or service, but do not know or care on which machine the desired data or service resides. For ease of presentation, as in [2, 3], we use the term “data” to represent “data or service” in the rest of the paper. In order to locate the data, the current Internet uses

domain name system (DNS) name resolution system to resolve uniform resource locators (URLs) into Internet Protocol (IP) addresses. However, DNS alone is not sufficient to effectively support the data-centric model.

More specifically, the current Internet does not have a mechanism for direct and persistent naming data. Instead, it names data relative to the hosts on which they reside. For example, a photo with name alice.jpg hosted by Google may have a name http://www.google.com/alice.jpg. When the same photo is hosted by Baidu, it may have another name http://www.baidu.com/alice.jpg. Using URLs to name data overloads the names and rigidly associates them with specific domains or network locations, making it inconvenient to move service instances and data, as well as to replicate them [2].

In order to address these issues, there is a growing consensus that the Internet should be enhanced to support the data-centric model [2] – [4]. Accordingly, several radically new data-centric network architectures have been proposed [2], [3]. However, the host-to-host model is here to stay as almost all existing applications, transport layer protocols, end hosts, and routers are host-centric. Therefore, it is not practical to deploy a pure data-centric architecture. Instead, it is desirable to design a new network architecture that can efficiently support both data-centric and host-centric services.

The new architecture should also be incrementally deployable and flexible enough to satisfy future requirement (i.e., to be "future proof"). There are billions of end hosts and servers around the world, and it is impossible to upgrade them at once. Also, the Internet has millions of routers, and it would be extremely expensive to replace them. An evidence of the resistance to radical changes is that, only few Internet service providers transited their networks from IPv4 to IPv6, although IPv6 has been standardized for more than ten years now. To achieve incremental deployment, we plan to set up small networks of the new architecture, test them thoroughly, and then slowly grow the size of these networks. In addition, the protocols that implement the new architecture must be able to inter-operate with existing protocols.

In this paper, we describe an incrementally deployable data-centric network architecture (DCNA) that can efficiently support both data-centric and host-centric services. The basic ideas of the proposed architecture are:

1) to insert a shim layer between the application layer and the transport layer of the current Internet architecture and;
2) to use appropriate interfaces to efficiently connect these layers.

A proof-of-concept implementation of DCNA in a Linux environment demonstrates that, in addition to being data-centric and incremental deployable, DCNA efficiently supports mobility, multi-homing, and can also facilitate the deployment of future new applications.

The rest of this paper is organized as follows. Section II describes the proposed architecture and Section III discusses its benefits. A proof-of-concept implementation is described in Section IV and we conclude the paper in Section V.

II. The Proposed Architecture

There are various alternatives for the Internet to move towards becoming data-centric. One possible approach is for every application in the application layer to be data-centric. For example, BitTorrent is data-centric since a user does not need to know where a desired data/service locates. Instead, BitTorrent would search, using certain application specific techniques, the end hosts that provide the desired data/service. However, as envisioned in today's Internet, only limited applications are data-centric. The majority of applications such as the hypertext transfer protocol (HTTP) are host-centric. If we want these applications to be data-centric, we have to revise them application-by-application, which is not efficient. Notice that these revisions are specially tailored to existing applications. If a new application emerges, the work is repeated. It is more efficient to design a novel architecture that applies to all applications (existing and future) instead of to specific ones.

Another possibility is to design a new transport protocol that is both data-centric and host-centric, and is employable by all upper layer applications. This again, however, limits applications to a specific transport layer protocol and makes it impossible for existing applications to enjoy the benefits of any novel transport layer protocols. As an example, the Stream Control Transmission Protocol (SCTP) [5] is reported to be superior to transmission control protocol (TCP). However, it is not widely used in the current Internet since many existing applications (such as HTTP) are bound to TCP, and SCTP is not compatible with TCP.

In this paper, we propose a network architecture called DCNA that can efficiently support both data-centric and host-centric applications. DCNA inserts a shim layer between the application layer and the transport layer of the current Internet architecture. The inserted shim layer is called the service binding (SB) layer.

After the insertion of the SB layer, the functions of all the layers below it (i.e., from the transport layer to the physical layer) in DCNA remain unchanged as in today's Internet architecture. In our implementation, we implement the network layer by employing the Locator/ID separation protocol [10], an incrementally deployable identifier/locator split mechanism, to further enhance the scalability of the routing system while keeping the function of the network layer unchanged.

Mapping Data/Service to IP Addresses

The function of the application layer in DCNA can be both backward compatible with the current architecture, and be enhanced in order to reap the benefit of a data-centric model. More specifically, its main function is to identify data/services that users want, and/or to identify end hosts where the desired data/services reside. In particular, we assume that, as a main enhanced functionality at the application layer, every data has a unique name, called service identifier (SID). The style of SIDs may be in a form similar to these in [2], or in a form similar to these used in BitTorrent (i.e., the name of a data is the hash for the content of the data), or any other forms. However, the choice of the best name style is

beyond the scope of the paper and is left for future research. The (enhanced) application layer informs the SB layer in the same end host the SID of a desired data/service.

The SB layer is in charge of obtaining/caching desired data/services for applications. Based on the information received from the application layer, the SB layer chooses appropriate transport protocols for applications. The SB layer also determines from how many sources it should obtain a desired data/service, based on the quality-of-service requirement of an application. For example, if there are three servers that can provide the same desired data, the SB layer may choose to obtain the desired data from a single server, two servers, or all three servers. In addition, the SB layer needs to maintain a certain transport status so as to support efficient host/service mobility, as will be discussed later in this paper.

In the case that an application simply sends the SID of its desired data/service to the SB layer, the SB layer should also map the SID onto one or more IP addresses (end hosts) on which the desired data/service resides. In this paper, we use an approach proposed in [2] to resolve SIDs onto IP addresses.

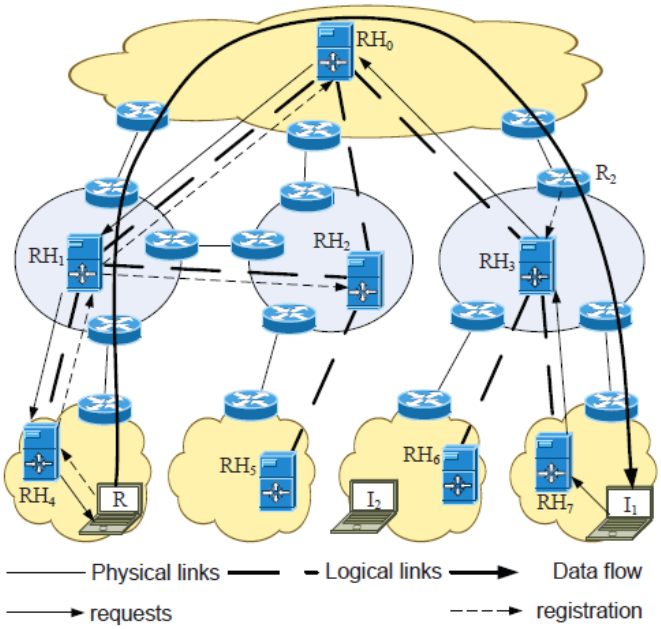


Fig. 1. Illustration of resolving SIDs onto IP addresses.

As in [2], we assume that every domain or administrative entity has one logical resource handler (RH). In addition, we denote the RH associated with an administrative entity X by RH_X. Furthermore, RH_X is the provider/customer/peer (or, alternatively, parent/child/peer) of RH_Y if X is the provider/customer/peer of Y in terms of autonomous system (AS)-level relationships. The RHs then form a hierarchical overlay infrastructure. For example, every AS in Fig. 1 maintains a logical RH. RH_0 is the provider of RH_1, RH_2, and RH_3, while RH_1 and RH_2 are peers. We also assume that each client knows the

location of its local RH through some local configuration (much like they know about their local DNS server).

Each RH maintains a registration table to store mappings, each of which maps an SID onto both a next hop RH and the distance to the server hosting the data. This way, an RH is able to forward requests based on SIDs. In particular, if a host *A* wants to provide service for a data with an SID, it registers the data to its local RH. The local RH then stores a mapping (from the SID onto *A's* location) into its registration table. The local RH then forwards the registration to its providers/peers, if it did not register for the SID to its providers/peers before. When a provider/peer receives a registration message from the RH, it stores a mapping (from the SID onto the RH's location) into its registration table if 1) no such item exists, or 2) the registration message comes from a server closer to the previous one. This registration process is illustrated by thin dashed lines in Fig. 1.

When an initiator (e.g., I_1 in Fig. 1) wants to obtain a data with SID, it sends a request to its local RH. When a RH receives such a request, it forwards the request to the next-hop RH if there is a mapping item for the SID in its registration table; otherwise, it forwards the request to its provider. This way, the request should be forwarded to a responder (e.g., *R* in Fig. 1) closest to the initiator, as illustrated by thin arrow lines in Fig. 1. The holder then sends the desired data to the initiator, as illustrated by the bold arrow lines in Fig. 1.

In order to reduce the number of requests that higher layer RHs need to process, we assume that the network employs the in-network cache mechanism used in [8] and [9]. In particular, when a router receives data, it locally caches the data if it did not cache the data before. In addition, it registers it in its local RH in order to let the RH know that it can provide the data. For example, in Fig. 1, when router R_2 receives the data from *R*, it locally caches the data and registers it in RH_3 so that RH_3 knows R_2 can provide the data. When another initiator I_2 wants to obtain the same data, it sends a request to the overlay infrastructure. When the request arrives at RH_3, it forwards the request to R_2, which then sends the desired data to I_2. Note that, in order to reduce the required storage space, routers in an administrative entity may cooperate with each other, so that the entire administrative entity caches only one copy of each data [8].

Consistent Identifier for Data/Service Binding

An important novelty in DCNA is the introduction of what we call the *consistent identifier* (CID) namespace in the SB layer. In particular, the CID uniquely identifies an application instance at a particular time. Specifically, every time an application in the application layer initiates a request for a given data/service, we take it as an application instance. For example, if an application requires a data SID_1, the SB layer considers it as an application instance. If the same application requires another data SID_2, the SB layer considers it as another application instance. For each application instance at an end host, the SB layer of the end host assigns it a unique CID. In addition, this CID is not changed until the desired data is obtained or the SB layer cannot find a host providing the desired data during a certain period (e.g., ten seconds). Furthermore, CIDs are locally unique. That is, they are used to differentiate application instances at an end host and would not be used to differentiate application instances at different end hosts. When assigning a

CID, the SB layer allocates adequate physical resources to the CID in order to cache data corresponding to the desired data, or to store the transport status, or for other uses. As will be shown later, the introduction of CIDs makes it possible to separate the upper layer application states and the transport layer states. This makes DNCA not only data-centric, but also capable of supporting efficient mobility and multi-homing.

In order to efficiently link these layers, we use an interface between the network and the transport layers (denoted as NT interface), an interface between the transport and the SB layers (denoted as TS interface), and an interface between the SB and the application layers (denoted as SA interface). In order to be incrementally deployable, the NT and the TS interfaces are in formats consistent with interfaces formats used in the current Internet, namely, <source IP address, destination IP address> and <source IP address, destination IP address, source port, destination port>, respectively.

The SA interface is in the form of <SID, source IP address, destination IP address, source port, destination port, …>. There are several reasons for using this form of SA interfaces. First, as stated above, we use the Internet mainly for purpose of obtaining services. If services are named by service identifiers, applications may only know the SID of a service and do not know where the service is hosted. In this case, an application sends the SID of the desired service to the SB layer. Second, there may still be some applications that want to communicate with a given end host. For example, when a person is in travel, he/she may want to access his/her personal computer in order to obtain some desired data that is only hosted on that personal computer. Third, almost all applications in today's Internet use the form of <source IP address, destination IP address, source port, destination port> to communicate with the lower transport layer. In addition, there should be some spare spaces reserved for future use. We use "…" to represent the reserved spare spaces. Therefore, in order to be incrementally deployable, a SA interface should in the above mentioned form.

Note that, however, the application layer does not need to fill in all the fields of the SA interface. Instead, it may only fill in some of them. For example, if an application knows the SID of the desired data, it fills in the fields of SID and source port. The purpose of the source port field is to differentiate between two applications that require the same data. If an application knows the IP address of the end-node that hosts the desired data, it fills in the fields of source IP address, destination IP address, source port, and destination port, as is done in the current Internet. More details about the communication process are provided in [6].

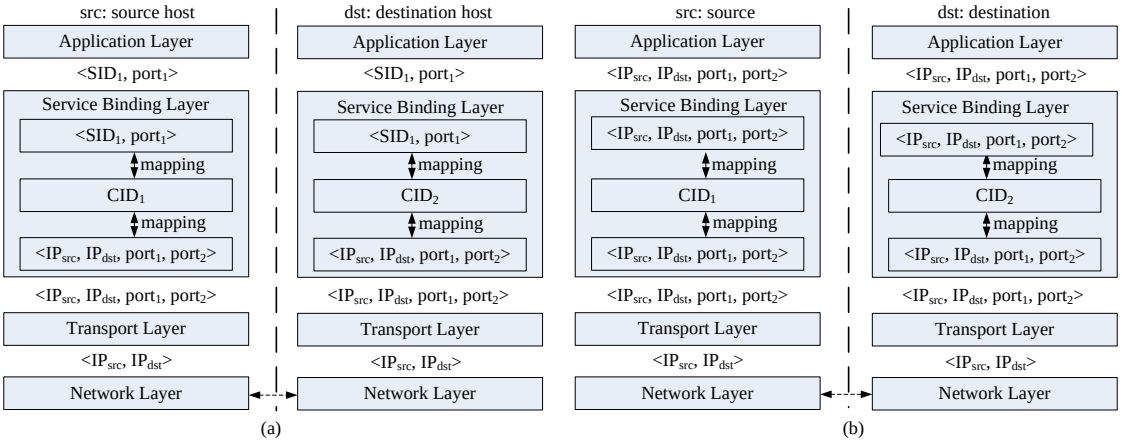


Fig. 2. Illustration of a basic communication in DCNA. (a) An application only knows the SID of the desired service; (b) An application only knows the IP address of the node that host the desired service.

Fig. 2 (a) illustrates how a destination host obtains a desired data from a source host in DCNA, assuming that the destination initiates the communication and an application only knows the SID of the desired service. Here, without loss of generality, we call a host requesting a desired data a *destination* and a host providing the desired data a *source*.

1) An application in the application layer of the destination fills in the SID field and the source port field (i.e., SID_1 and $port_1$ in Fig. 2 (a)) in the SA interface, and sends a request to the SB layer of the destination. SID_1 and the source port $port_1$ indicate the desired data and the desired application, respectively.

2) When the SB layer of the destination receives the request, it first assigns to it a CID (i.e., CID_2 in Fig. 2 (a)) that is not used by other services. At the same time, the SB layer allocates physical resources to the CID. It then maps the binary $<SID_1, port_1>$ onto CID_2.

3) Since the SB layer of the destination does not know which end-node(s) host(s) the desired service, it should map SID_1 onto one or more IP addresses of the end-nodes that host the desired service.

4) The SB layer of the destination fills in the TS interface using the IP addresses of the source and the destination (i.e., IP_{src} and IP_{dst}, respectively, in Fig. 2 (a)), the source port (i.e., $port_1$ in Fig. 2 (a)), and the destination port (i.e., $port_2$ in Fig. 2 (a)). It then maps CID_2 onto the filled in TS interface and sends the request to the transport layer of the destination. Note that the SB layer of the destination may choose the desired transport layer protocols based on the requirement of upper layer applications.

5) The transport layer of the destination encapsulates the data from the SB layer using a transport header, and sends the data to the network layer of the destination through the NT interface of the destination. This is same as what the transport layer does in today's Internet.

6) The network layer of the destination encapsulates the data with a network header and sends out the packet, which is routed to the source. Note that this is also same as what the network layer does in today's Internet.

7) When the network layer of the source receives the packet, it strips the network header of the packet and sends the decapsulated packet to the transport layer of the source.

8) The transport layer of the source strips the transport header when it receives the packet from the network layer of the source. It then sends the packet to the SB layer of the source through the TS interface of the source.

9) When the SB layer of the source receives the packet, it assigns a CID (i.e., CID_1 in Fig. 2 (a)) for the service if there is not such a CID corresponding to the quadruple of $<IP_{src}, IP_{dst}, port_1, port_2>$. If there is such a CID for the quadruple, the SB layer of the source simply maps the quadruple onto the existing CID.

10) After that, the SB layer of the source maps CID_1 onto the binary of $<SID_1, port_1>$ if this is the first packet for the service. The SB layer of the source then sends the data to the application layer of the source through the SA interface of the source by filling in the fields of SID and source port.

Note that, in order to facilitate in-network cache, every packet needs to carry the SID of the corresponding data. When an old router receives a packet, it can ignore the SID field and forwards the packet to the next hop. In addition, it is possible for a data to be divided into multiple segments, each of which is represented by a unique SID. For example, in [12], if segment *B* is the subsequent segment of segment *A*, the SID of a segment *A* may contain a pointer to the SID of the segment *B*. In this case, the SB layer of the destination should map the SID of every segment onto a corresponding IP address. In addition, once the SB layer has assigned a CID to the desired data, the CID remains unchanged until the desired data is obtained (or no source providing the desired data), although both the SIDs for different segments and the corresponding IP addresses are different.

Note also that, in today's Internet, when an application requests certain data (e.g., a web page corresponding to a URL), the web page often contains different data objects (such as JPEG and text files). In this case, the current Internet uses only one persistent TCP connection to obtain these data objects for enhanced efficiency. In DCNA, this may be achieved in a similar way by treating the URL as a SID and letting the SID pointing to other SIDs (each corresponding to a single data object contained in the web page), as in [12]. This way, when an application wants the data, it only sends the SID corresponding to the URL to the SB layer. The SB layer then uses only one CID for the application. In addition, the SB layer may resolve the other SIDs onto corresponding servers/routers. If these SIDs do map to the same server with the SID corresponding to the URL, the SB layer may also use a common TCP connection to obtain the desired data objects for improved efficiency.

In the case that the destination host knows the IP addresses of the end-nodes that host the desired service, most of the steps of a communication process are identical to those in the above case. Fig. 2 (b) illustrates the communication process in this case, focusing on the differences between the two cases.

1) The application of the destination fills in the fields of IP_{src}, IP_{dst}, $port_1$, and $port_2$ in the SA interface of the destination. It then sends a request to the SB layer of the destination.

2) When the associate layer of the destination receives the request from the SA interface of the destination, it assigns for the service a CID (i.e., CID_2 in Fig. 2 (b)) that is not used by other services. It then maps the quadruple <IP_{src}, IP_{dst}, $port_1$, $port_2$> onto CID_2.

3) - 8) in the second case are same to 4) - 9) in the first case, respectively.

9) The SB layer of the source maps CID_1 onto the quadruple <IP_{src}, IP_{dst}, $port_1$, $port_2$> if this is the first packet for the service. The SB layer of the source then sends the data to the application layer of the source through the SA interface of the source by filling in the files of IP_{src}, IP_{dst}, $port_1$, and $port_2$.

Since the SB layer has obtained the IP address of the source, it is not necessary for the SB layer to map an SID onto IP addresses. In addition, it is not necessary for the destination to send a CID to the source in both cases. Instead, CIDs are locally maintained.

Note that the second case makes it convenient for DCNA to use existing applications since they employ interfaces of the type <IP_{src}, IP_{dst}, $port_1$, $port_2$>. This in turn makes DCNA incrementally deployable, as will be shown later in Section IV. Furthermore, mapping a quadruple (with IP addresses) to CID and then mapping the CID back to a quadruple makes DCNA easy to support efficient mobility and multi-homing, as to be explained in Section III.

III. Benefits of DCNA

So far, we have described how DCNA works. We will now show that DCNA can provide data-centric, mobility support, multi-homing support, and is incrementally deployable. In addition, it can facilitate the deployment of new applications.

Data-centric

An important merit of DCNA is that it is data-centric. We describe it in more detail using the example illustrated in Fig. 3, where we assume that a destination with IP_{dst} wants to obtain a service with SID hosted by two sources, i.e., source 1 with IP_{src}^1 and source 2 with IP_{src}^2. In order to obtain the service denoted by SID, the application layer of the destination sends a request to the SB layer of the destination. The SB layer of the destination then assigns an unused CID (i.e., CID_{dst} in Fig. 3) and maps the SID onto IP_{src}^1 (or IP_{src}^2, or both). The destination then opens a connection to communicate with source 1 in order to obtain the desired service denoted by SID, by following the Steps 4 - 10 of the first case described in Section III.

We now assume that source 1 fails. In this case, the SB layer of the destination would detect this failure. If the destination has obtained the IP address of source 2 (i.e., IP_{src}^2), the SB layer of the destination opens a connection with source 2. If the destination does not know IP_{src}^2, the SB layer of the destination should first map the SID onto IP_{src}^2 and then open a connection with source 2. In both cases, the CID_{dst} is kept unchanged, as illustrated in Fig. 3. Therefore, the upper layer application would not be affected by the failure of source 1.

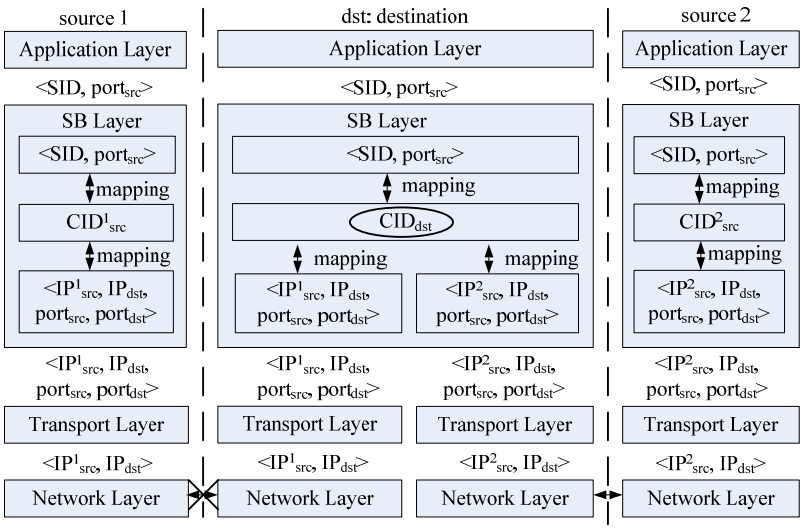


Fig. 3. Illustration of DCNA for data-centric services.

Note that the case the SB layer of the destination first resolves IP_{src}^2 is similar. In addition, if the SB layer of the destination resolves both IP_{src}^1 and IP_{src}^2, it may open a connection with source 1 and another one with source 2 simultaneously. In this case, the destination can obtain data from both sources simultaneously and discard those duplicated packets. If any one of them fails, the destination can still obtain the desired service, thus increasing the robustness of the service. The destination can also obtain a given percent (e.g., 40%) of the desired data from source 1 and the rest from source 2. Similarly, if there are multiple sources that host the same service, the destination can obtain its desired service from a part or all of these sources.

Notice that some existing applications such as BitTorrent [6] in the current Internet can also share the above described benefit. However, only very limited applications can share this benefit and the majority of existing applications cannot share this benefit. While one may argue that we can modify existing applications to share this benefit, it is time-consuming to modify them, and furthermore, these modifications have to be made one-by-one.

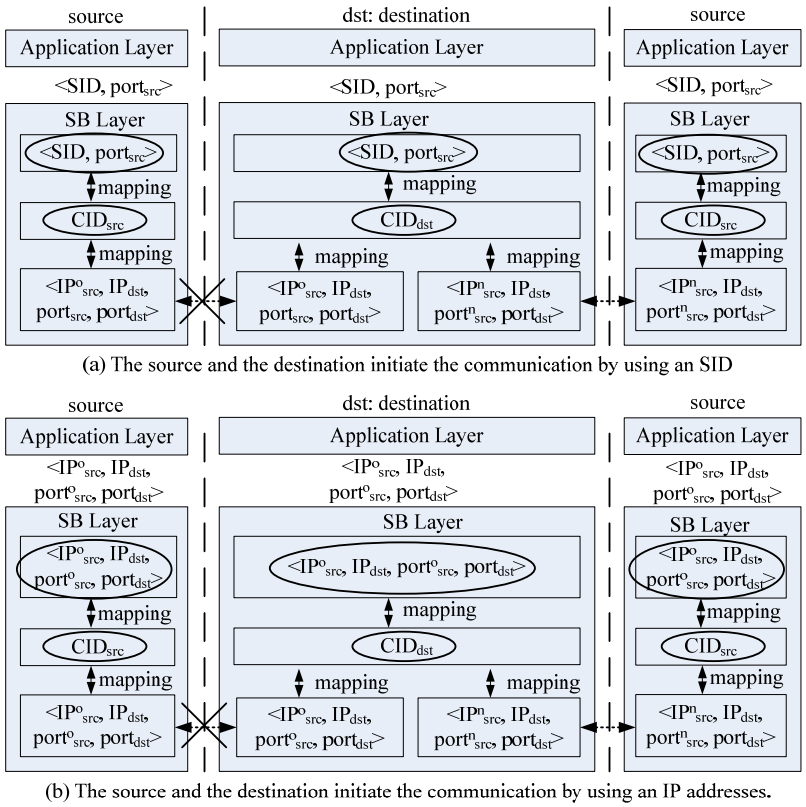


Fig. 4. Illustration of mobility support in DCNA.

<u>Mobility Support</u>

Another benefit of DCNA is its support for mobility, including service mobility and host mobility. Host mobility allows a device to move between IP subnets, while continuing to be reachable for incoming requests and maintaining sessions across subnet changes [7]. By contrast, service mobility allows users to maintain access to their services while moving or changing devices and network service providers [7].

Fig. 4 (a) illustrates how DCNA supports host mobility when a source and a destination initiate a communication using an SID. As illustrated in Fig. 4 (a), when the source host moves from one subnet to another and changes its IP address from IP_{src}^{o} to IP_{src}^{n}, the source and the destination simply 1) record the current transport status, 2) tear down the old connection identified by the old quadruple $<IP_{src}^{o}, IP_{dst}, port_{src}^{o}, port_{dst}>$, 3) re-establish a new connection identified by a new quadruple $<IP_{src}^{n}, IP_{dst}, port_{src}^{n}, port_{dst}>$, 4) map the new quadruple onto CID_{src} and CID_{dst}, respectively, and 5) continue communication from the recorded transport status. This way, the CIDs maintained at the source and the destination do not change, as illustrated by the circles in Fig. 4 (a). Therefore, upper layer applications will not be interrupted when a host changes its subnet.

We note that, in order to properly forward some old data sent before the new transport layer connection is established, some network layer mechanisms are still needed.

However, even if no such mechanisms are available, the SB layer can still retransmit these old data through the new transport layer connection.

Similarly, Fig. 4 (b) illustrates how DCNA supports host mobility when a source and a destination initiate a communication by using their IP addresses. Again, when the source changes its IP address from IP_{src}^{o} to IP_{src}^{n}, CID_{src}, CID_{dst}, the SA interfaces at both the source and the destination do not change. As a result, upper layer applications would not be interrupted when a host changes its subnets.

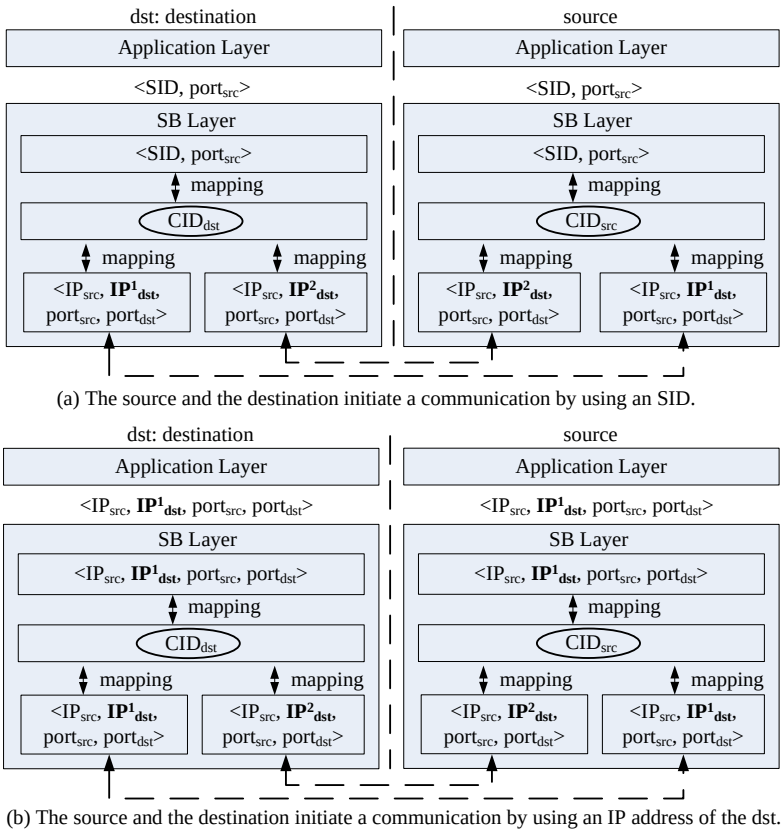
DCNA is also capable of supporting service mobility. Indeed, when a user roams by using the same host, DCNA can efficiently support host mobility, as described above. In addition, the case that a user changes its devices may be viewed as the case where a user moves from a subnet to another one. Similarly, the change of network service providers may correspond to the case that the source moves from a subnet to another one. We note that there are security issues that need to be addressed. However, our focus here is on the functional aspect and the problem of how to secure connections/services is left for further research.

Multi-homing Support

DCNA is also capable of efficient multi-homing support. We illustrate this by using the example shown in Fig. 5, where we assume that a destination is multi-homed to two subnets and has two IP addresses denoted by IP_{dst}^{1} and IP_{dst}^{2}. We also assume that the destination wants to obtain a service denoted by SID hosted by a source denoted by IP_{src}.

Fig. 5 (a) illustrates how DCNA supports multi-homing when the source and the destinations initiate a communication using the SID. In this example, the destination may first establish a connection using IP_{dst}^{1} with the source, and establish another one using IP_{dst}^{2} when IP_{dst}^{1} becomes unavailable. In this case, multi-homing is used for redundancy. The destination may also simultaneously establish two connections with the source and obtains a given percent of data from one connection and the rest from the other. In this case, multi-homing is used for load-balancing. In both cases, the CIDs assigned for the service at the source and the destination do not change.

Fig. 5 (b) illustrates how DCNA supports multi-homing when the destination initiates a communication with the source by using IP_{dst}^{1}. If IP_{dst}^{1} becomes unavailable, the destination can still communicate with the source by re-establishing a new connection using IP_{dst}^{2}, while keeping the CIDs in both the destination and the source unchanged. Similar to the above case where a communication is initiated by an SID, the destination may also simultaneously setup two connections with the source by using IP_{dst}^{1} and IP_{dst}^{2}.



(a) The source and the destination initiate a communication by using an SID.

(b) The source and the destination initiate a communication by using an IP address of the dst.

Fig. 5. Illustration of multi-homing support in DCNA.

Facilitating the Development of New Applications

An important benefit of DCNA is that it facilitates the development of new applications. Since the SB layer can provide a set of functions such as segmentation, shaping and caching, application developers do not need to develop codes that realize such functions. This makes it possible for application developers to develop new applications.

Enabling New Networking Technologies

In the current Internet, the Application Program Interface (API) of most applications is bound to IPv4 addresses. Thus, although IPv6 has been widely deployed in routers and end hosts, it is not widely used. On the other hand, if applications in DCNA only use SID, the SB layer can flexibly map CIDs onto IPv4 or IPv6 addresses based on their performance such as delay and throughput, as long as a client and a server support both IPv4 and IPv6. Similarly, if another new network technology emerges, as long as the network layer and transport protocols support such new technology, the SB layer can flexibly choose to use the one with the best performance. This way, an improved new technology can compete with existing technologies and gradually replaces. This in turn encourages innovations of new network layer technologies.

Incremental Deployment

As stated previously, DCNA is incrementally deployable. This is illustrated by Fig. 6, which shows how a node with the current Internet architecture (called “old node” for ease

of presentation) communicates with another node with DCNA (designated as “new node”). Since the source cannot support SID, the destination node can only initiate a communication using the IP address of the source. In this case, an application in the destination fills in the fields of IP_{src}, IP_{dst}, $port_1$, and $port_2$ in the SA interface of the destination and sends a request to the SB layer of the destination. The SB layer then assigns an unused CID (i.e., CID_1 in Fig. 6) to the service and sends the request to the transport layer in the same way as an application send a request to the transport layer in today’s Internet. Note that the SB layer of the destination does nothing to the request from the application layer. As a result, the application layer of the source receives a request in the same way as a request is sent from an application in today’s Internet. Therefore, the destination and the source can communicate with each other.

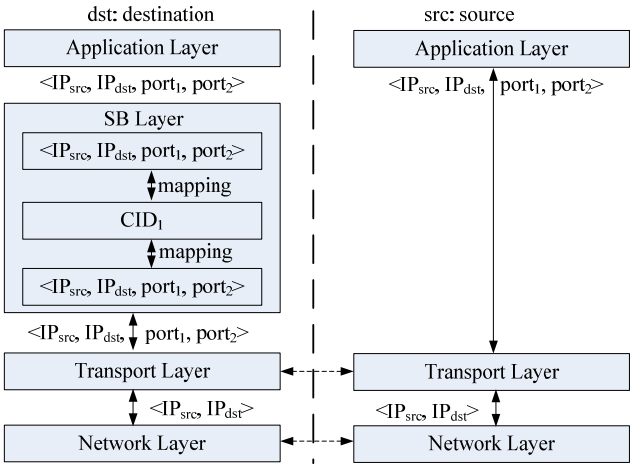


Fig. 6. Illustration of communications between a node with the current Internet architecture and a node with DCNA.

More importantly, a new node can also enjoy the benefit of data-centric when it obtains service from old nodes. In order to obtain a service, a new node still can establish multiple connections with multiple old nodes and maps these connections onto the same CID assigned for the service. In addition, if any one of these connection fails, the new node can obtain service from the remaining old nodes. Therefore, DCNA is incrementally deployable.

IV. Demonstration

We have validated the design of DCNA by implementing a simple Linux-based prototype with kernel version 2.6.28. Fig. 7 illustrates the topology of the prototype, which comprises three routers, three ingress/egress tunnel routers (xTRs), two mapping servers, a RH, a multi-homed server implemented with DCNA, a server with DCNA, a multi-homed host with DCNA, a mobile host implemented with DCNA, and a traditional host.

The implementation of the hosts with DCNA consists of about 22,300 lines of code and comprises the following main modules:

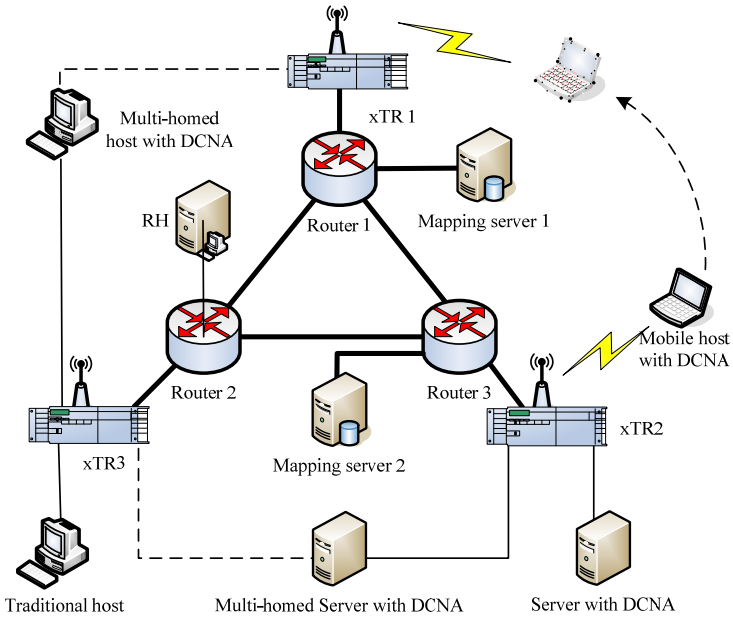


Fig. 7. The topology of the prototype.

- Application module: In order to verify that DCNA can support both host-centric and data-centric model, we modified the Apache HTTP server (version 2.2) so that it can send a request to the SB layer when we simply input an SID instead of a URL.

- Session Binding module: It generates CIDs for application instances, resolves IP addresses for SIDs, and manages transport layer connections used for obtaining a desired data.

The xTRs are used to split identifiers from locators (see [10] for details) to improve efficiency of Internet routing. The implementation of the xTRs consists of about 5,100 lines of code. The mapping servers are used to store mapping items of end hosts and their implementation consists of about 3,700 lines of code.

The RH is used to map SIDs onto identifiers. Unlike the implementation of RHs in [2], we implement the RH by running a database in a Linux server and letting the database maintain the mappings from SIDs onto identifiers.

The implementation shows DCNA's main benefits including data-centric, mobility support, and multi-homing support. For example, in our experiment, the multi-homed host firstly opened a session for a movie hosted by the single-homed server attached to xTR2. After a certain period, we successfully migrated the ongoing video streaming session to the mobile host. This was accomplished by letting the multi-homed host send the server a command that 1) tells the server the IP address of the mobile host and, 2) indicates that we want the movie be sent to this IP address. We refer interested readers to the video linked in [6].

V. Conclusions

It has been widely recognized that a future Internet should be data-centric, since users use the Internet mainly for data retrieval but do not care in which machine the desired data resides. In this paper, we have proposed an incrementally deployable data-centric network architecture (DCNA) for the Internet. We have described DCNA's basic framework and outlined a range of benefits offered by DCNA. We note that, despite our proof-of-concept implementation, there are still a lot of open issues including large-scale experiments and we expect this work will inspire many research activities in this and related future internet architectures.

Acknowledgment
This work was supported in part by National Basic Research Program of China ("973 Program") under Grant No. 2007CB307100, in part by the Natural Science Foundation of China (NSFC) under Grant no. 60833002, and in part by the "111 Program" of China under Grant No. B08002.

References

[1] Sandvine Global Internet Phenomena Report: Spring 2011. http://www.sandvine.com/news/global_broadband_trends.asp.

[2] T. Koponen, M. Chawla, B.-G. Chun, et al., "A data-oriented (and beyond) network architecture," In *Proc. of ACM SIGCOMM'07*, Kyoto, Japan, Aug. 2007.

[3] Named Data Networking. http://named-data.org/index.html.

[4] S. Ganapathy, and T. Wolf, "Design of a network service architecture," in *Proc. IEEE ICCCN'07*, Honolulu, Hawaii, USA, Aug. 2007, pp. 754 - 759.

[5] R. Stewart, Q. Xie, K. Morneaut, et al., "Stream control transmission protocol," *IETF RFC 2960*, Oct. 2000.

[6] H. Luo, H. Zhang, M. Zukerman, and C. Qiao, "An Incrementally Deployable Network Architecture to Support Both Data-centric and Host-centric Services – extended version," Available at: http://iplab.bjtu.edu.cn/~hbluo/.

[7] H. Schulzrinne, E. Wedlund, "Application-layer mobility using SIP," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 4, no. 3, July 2000, pp. 47 - 57.

[8] B. Ahlgren, et al., "Second NetInf architecture description," Deliverable D6.2, 4WARD EU FP7 Project, Jan. 2010. FP7-ICT-2007-1-216041-4WARD/D6.2.

[9] T. Wolf, "In-network service for customization in next generation networks," *IEEE Network*, vol. 24, no. 4, August 2010, pp. 6 - 12.

[10] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/ID separation protocol (LISP)," *IETF Internet Draft*, draft-ietf-lisp-15.txt (work in progress), July 2011.