Supporting COntent NETworking in Software Defined Networks

Technical Report - Version 0.3, July 2012

S. Salsano(1), N. Blefari-Melazzi(1), A. Detti(1), G. Mazza(1),
G. Morabito(2), A. Araldo(2), L. Linguaglossa(2), L. Veltri(3)

(1) University of Rome “Tor Vergata”, (2) University of Catania, (3) University of Parma

Available at: http://netgroup.uniroma2.it/CONET/
Contact: stefano.salsano@uniroma2.it

DISCLAIMER
This work has been partly funded by the EU in the context of IST OFELIA (http://www.fp7-ofelia.eu/) and CONVERGENCE (http://www.ict-convergence.eu/) projects, but it does not necessarily represent the official position of the project itself and of its partners. Authors are solely responsible for the views, results and conclusions contained in this work.

COPYRIGHT
"Copyright and all rights therein are retained by authors. This work may not be modified or reposted without the explicit permission of the copyright holder"
Contents

1. Introduction .............................................................................................................................................. 3
2. The CONET Architecture ..................................................................................................................... 4
3. Detailed Analysis of operations performed by Border Nodes and Internal Nodes ....... 6
   3.1 Border Nodes .................................................................................................................................... 6
       3.1.1 Arrival of an Interest. ..................................................................................................................... 6
       3.1.2 Arrival of a Data-Unit: .................................................................................................................. 7
   3.2 Internal Nodes .................................................................................................................................... 8
       3.2.1 Arrival of an Interest. ..................................................................................................................... 8
       3.2.2 Arrival of a Data-Unit: .................................................................................................................. 8
4. Analysis of OpenFlow features for CONET support ............................................................................ 10
5. Analysis of Information Centric Networking functionality ............................................................... 12
6. Long term approach for CONET support in OpenFlow ................................................................. 15
7. Short term approach for CONET support in OpenFlow ........................................................................ 16
   7.1 Analysis of possible tag solutions .................................................................................................... 16
       7.1.1 Analysis of MPLS tag solution ................................................................................................... 18
       7.1.2 Analysis of IP protocol + ports tag solution ............................................................................ 20
       7.1.3 Analysis of MAC address tag solution ..................................................................................... 21
   7.2 Other architectural aspects ............................................................................................................... 21
       7.2.1 Communications between Cache Server, Controller and Switch ............................................. 22
       7.2.2 The Cache Server application .................................................................................................. 24
       7.2.3 Implementation schema ............................................................................................................. 25
8. Details and architecture for the short term solution ............................................................................ 27
References .................................................................................................................................................. 28
1. Introduction

This document has to be considered as a continuation of the EXOTIC proposal, since we deal with practical aspects, focusing on functionalities and operations that need to be performed for a successful implementation.

In particular, this document is organized as follows: in the first section we resume basic CONET architectural aspects, highlighting operations and functionalities that are performed into different network entities. Then we deal with some practical aspects and requirements and we outline key modules needed to realize those functionalities.

Later, we separate our long term vision and solutions from a realization that could be obtained in the short term and, in the end, we propose some practical implementations of this solution.

The home page for retrieving more information about OFELIA project and Exotic proposal is:

http://netgroup.uniroma2.it/CONET/
2. The CONET Architecture

The CONET architecture is composed of a set of CONET nodes and serving nodes interconnected by CONET Sub System (CSS, defined as a generic network with homogeneous networking technology and homogeneous native addressing space). CONET nodes exchange CONET Information Units (CIUs), which are used to convey both requests of named-resources, called interest CIUs, and chunks of named-resources themselves (i.e., part of files, videos, etc.), called named-data CIUs. A CONET SubNet interconnects two or more CONET nodes, providing transfer of CIUs by using an under-CONET technology, such as point-to-point Layer 2 links, Layer 2 networks or overlay links (e.g. UDP over IP) [1].

To best fit the transfer units of an under-CONET technology, both interest-CIUs and named-data CIUs are carried in small CONET data units named carrier-packets, which also include routing information of the end-to-end communication session they serve to.

The different types of CONET nodes are shown the following figure. (see Figure1).

\[\text{End-nodes}\] are user devices that request named-resources by issuing interest CIUs. \[\text{Serving-nodes}\] store, advertise and provide named-resources by splitting the related sequence of bytes in one or more named-data CIUs. \[\text{Border-Nodes}\] interconnect different CONET SubSystems; they forward carrier-packets by using CONET routing mechanisms, may reassemble carrier-packets and cache the related named-data CIU, and may send back cached named-data CIUs.
Within an IP-CSS, optional **Name Routing System (NRS) Nodes** may be used to assist the CONET routing operations. Moreover, optional CONET **Internal-Nodes** could be deployed inside an IP-CSS to provide additional in-network caches.

The operations performed to provide a end-node with a chunk of named-resource can be described as follows:

- An end-node requests a chunk of a named-resource by issuing an interest CIU, which includes the network-identifier (i.e., the name of the resource); the interest CIU is encapsulated in a carrier-packet, named **XXX**.

- Name-based forwarding engines in the End-Node and intermediate Border-Nodes forward-by-name the packet **XXX** upward the proper Serving-Node. Forward-by-name means that, on the base of the network-identifier contained in the carrier-packet **XXX**, a name-based routing-engine singles out the CSS address of the next upward Border-Node toward the Serving-Node. A CSS address is an interface address, consistent with the CSS technology (e.g., an IPv4 address in case of an IP-CSS). Then, the name-based routing engine encapsulates the carrier-packet **XXX** in a data-unit of the underlying CSS technology and uses the CSS address as the destination address;

- The CSS address of the end-node and the set of CSS addresses of the traversed border-nodes in the “upward” path toward the serving-node are appended to the carrier-packet **XXX**, within a control field named **path-state**;

- If present, the Internal-Nodes forward the carrier-packet **XXX** by using the underlying routing engine (e.g. IP RIB), but are able to parse the carrier-packet **XXX**;

- The first in-path CONET Node, Border, Internal or Serving-Node, which is able to provide the chunk of the named-data requested within **XXX**, will send back the named-data CIU, without further propagating the interest CIU;

- The named-data CIU is encapsulated in a carrier-packet, named **YYY**. The carrier-packet **YYY** follows the same path of the carrier-packet **XXX**, but in the downward direction and will reach the requesting end-node;

- The reverse-path routing is carried out in a source-routing fashion by using a path-state control information appended to the carrier-packet **YYY**. This path-state is the copy of the one set up in the interest CIU during the upward routing;

- All the Border-Nodes and internal-nodes in the downward path may cache the named-data CIU, according to their policies and available resources.
3. Detailed Analysis of operations performed by Border Nodes and Internal Nodes

In order to better understand how all these functionalities can be implemented, it is worth to outline what is the sequence of operations that has to be performed into Internal and Border Nodes, differentiating between them, since they perform different operations. In the following we assume, for the sake of simplicity, an underlying IP-CSS technology, as we may expect that this will be the most widely deployed (at least in the near future). Please note that this is only an exposition simplification that could be easily replaced by any other underlying technology.

One of the first things we have to do is to distinguish between Border Nodes (BNs) and Internal Nodes (INs), as INs perform only classic router and cache policies, while, in addition to that, BNs do also routing-by-name [2].

There are two different types of caches we are concerned with:

- a cache for the content, i.e., for the chunks of the data;
- a cache for the routing information, as the routing table are handled with the “lookup-and-cache” approach.

Therefore we will call “content-cache” the former cache and the “route-cache” the latter one.

Since there is the need to perform different functionalities, we also distinguish between the arrival of an Interest and a Data Unit.

3.1 Border Nodes

3.1.1 Arrival of an Interest.

1. Inspection of the packet:
   - if it is not a CONET packet, perform traditional routing.
   - if it's a CONET packet

   ↓

2. Perform content-cache lookup:
   - If the packet belongs to a chunk that is in the content-cache, answer back providing the requested content and satisfy the interest.
• If the content-cache does not contain the requested content

↓

3. Lookup in the routing-by-name table (route-cache):

• If there is a correspondance between the requested content and a
destination IP address, then forward the packet to that node, inserting
into the “Path state” field his own IP address and refresh the route
cache.

• If there is not a correspondance between the requested content and a
destination IP address,

↓

4. Query the Name Resolution Server (NRS) node

5. Update the forward-by-name table (route-cache) according to the
NRS answer.

6. Forward the packet, inserting its IP address into the “Path state” field

Previous functions have to be intended as logical functions and entities. In practice, they
can be realized as one logical entity (e.g. forwarding table and cache-lookup table can be
merged in a unique table).

3.1.2 Arrival of a Data-Unit:

1. Inspection of the packet:

• if it is not a CONET packet, perform traditional routing .

• if it's a CONET packet

↓

2. Forward the packet according to normal routing table

In parallel the node should decide whether to cache the data in the content-cache,

3. The node checks if the data packet belongs to a chunk that is already in the
content-cache:

- if yes → just refresh the timer in the content-cache for the given content
  (do Content-Cache Timer Refresh)

- if not → try to insert the packet in the cache (doPrecache)
We define as Precache a system that stores data that are waiting to be reassembled in a chunk. The decision to hold a data packet in the precache system may depend on the current load (processing and memory) of the precache itself and on the number of interest/data packets received for that chunk (and may be for the NID). At the end of the precaching phase, when a full data chunk is available, security checks should be performed before storing the content in the cache. Storage in the cache (if there are cache size restrictions) will be subject to cache policy algorithms (eg. LRU).

3.2 Internal Nodes

3.2.1 Arrival of an Interest.

1. Inspection of the packet:
   - if it is not a CONET packet, perform traditional routing.
   - if it's a CONET packet

2. Perform content-cache lookup:
   - If the packet belongs to a chunk that is in the content-cache, answer back providing the requested content and satisfy the interest.
   - If the content-cache does not contain the chunk to which the packet belongs

3. Perform “normal” lookup in the IP routing table, as the IP destination address is the IP of the border node.

3.2.2 Arrival of a Data-Unit:

1. Inspection of the packet:
   - if it is not a CONET packet, perform traditional routing.
   - if it's a CONET packet && it is a data packet

2. Forward the packet according to normal routing table
In parallel the node should decide whether to cache the data in the content-cache,

3. The node checks if the data packet belongs to a chunk that is already in the content-cache:
   - if yes → just refresh the timer in the content-cache for the given content (do Content-Cache Timer Refresh)
   - if not → try to insert the packet in the cache (doPrecache)

Focusing on Internal Nodes, the first operation we have to realize is the separation between CONET packets and normal traffic – which would be routed following traditional routing by the OpenFlow switch (that implements L3 routing).

Since we chose to use the “Integration approach”, exploiting an IPv4 option to contain the header of a Carrier Packet (Network Identifier + Chunk Number + Payload Type, Interest or Data Unit), it is clear that the only difference between CONET traffic and normal packets is represented by this IPv4 CONET Option.

In particular, this option contains a double information for an Internal Node, because it identifies if the packet is a CONET one, and if it is an Interest or a Data Unit.

So an Internal node can exploit this information to:

1. decide whether the incoming packet is a CONET packet
2. distinguish between Interest and Data Unit.

In particular, the above scheme can be combined in the first part as it follows:

1. arrival of a packet
2. Inspection of the IPv4 option. Is it a CONET packet?
   - NO → do normal routing
   - YES → is it an interest or a data unit?

and then follow the above parts, depending of the type of packet.

So, the CONET Option plays a crucial role in the routing process. In addition, the first part of this option, _Network Identifier + Chunk Number_, offers a simple and clean possibility to perform a fast cache-lookup, due to its uniqueness (thanks to the Name System) and hierarchical structure.

Because of that, we suggest to use that option to address content into the cache as well as to recognize packets and decide if it’s necessary to perform cache-lookup or if the packet needs normal routing.
4. Analysis of OpenFlow features for CONET support

Focusing on the internal structure, both Border and Internal Nodes can be represented as in the figure below, where it is possible to recognize the HostPC running the controller, connected to the switch through an SSL channel [3].

The controller is responsible for the management of switch’s flow tables, which play a fundamental role in CONET architecture, as they are meant to recognize if an incoming packet is a CONET one and to perform consequent actions.

![OpenFlow Architecture Diagram](image)

**Figure 2 – The OpenFlow Architecture**

The main problem of this CONET Option is that OpenFlow 1.0 supports only the following fields into his matching entries [4]:

<table>
<thead>
<tr>
<th>OpenFlow Switch Specification</th>
<th>Version 1.1.0 Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingress Port</td>
<td></td>
</tr>
<tr>
<td>Metadata</td>
<td></td>
</tr>
<tr>
<td>Ether src</td>
<td></td>
</tr>
<tr>
<td>Ether dst</td>
<td></td>
</tr>
<tr>
<td>Ether type</td>
<td></td>
</tr>
<tr>
<td>VLAN id</td>
<td></td>
</tr>
<tr>
<td>VLAN priority</td>
<td></td>
</tr>
<tr>
<td>MPLS label</td>
<td></td>
</tr>
<tr>
<td>MPLS traffic class</td>
<td></td>
</tr>
<tr>
<td>IPv4 src</td>
<td></td>
</tr>
<tr>
<td>IPv4 dst</td>
<td></td>
</tr>
<tr>
<td>IPv4 proto / ARP opcode</td>
<td></td>
</tr>
<tr>
<td>IPv4 Tos bits</td>
<td></td>
</tr>
<tr>
<td>TCP / UDP / SCTP src port</td>
<td></td>
</tr>
<tr>
<td>TCP / UDP / SCTP dst port</td>
<td></td>
</tr>
<tr>
<td>ICMP Type</td>
<td></td>
</tr>
<tr>
<td>ICMP Code</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3 – OpenFlow matching fields**
Unfortunately, at the moment, OpenFlow cannot realize a match against the IPv4 Option field, so a node cannot recognize CONET traffic through OpenFlow rules. Furthermore, it is worth to point out that the IPv4 Option field does not have a fixed length, so the CONET option will not appear in a stable position within this field.

Therefore, even if this seems to be the cleanest and most efficient solution, the use of the IP CONET option to differentiate CCN traffic has to be considered as a long-term target, to be achieved when a new OpenFlow 2+ version, able to match against the whole IP header, will be deployed.
5. Analysis of Information Centric Networking functionality

In this section we are providing an identification of the Information Centric Networking (ICN) functionalities that needs to be implemented either in the Switch or in the Controller in an OpenFlow based architecture.

In particular, as shown in Figure 4, these are the key functionalities:

- **Forward-by-name**: it is applied to Interest packets, it is the mechanism used by ICN nodes to relay an incoming content request to an output interface. The output interface is chosen by looking up a “name-based” forwarding table.

- **Data Forwarding**: it is the mechanism that allows the content to be sent back to the device that issued a content request. Data forwarding cannot use the Forward-by-name mechanisms, because the devices are not addressed by the content routing plane of an ICN. Therefore, an ICN requires two different forwarding strategies to forward content requests and to deliver the data.

- **Content Routing**: it is the mechanism used to disseminate information about location of contents, so as to properly setup the name-based forwarding tables. For
instance, content routing could re-use IP routing mechanisms, where name prefixes are distributed instead of IP prefixes. Content routing is one of the assets of ICN, as a provider could use content routing to improve the efficiency and reliability of content access in its network.

- **Caching**: it concerns the ability of ICN nodes to cache data and to directly reply to incoming content requests rather than forwarding them towards a serving node.

It is worth to outline that these functionalities are fundamental within Information Centric Networking and they could be realized in practice by means of different implementation solutions. In fact, as shown in previous figure, these logical functionalities does not necessarily reside within ICN Nodes, but can be physically shared among several entities, since they involve also a control logic that can be implemented in separate machines. In fact, in a Software Defined Network scenario, which exploits OpenFlow, the control logic can be placed externally and instruct the data plane and the packet forwarding of the switch as well as decide which content is going to be cached or disseminate information about location of contents.

XXX Qui è possibile presentare un paio di scenari di architettura, che saranno eventualmente richiamati nella parte long/short term

XXX NRS / integrated or not in controllers…

![Diagram](image.png)

**Figure 6 – Functionalities disposition in SOLUTION X**
Figure 8 – Functionalities disposition in SOLUTION Y
6. Long term approach for CONET support in OpenFlow

As we already said, the use of the IP CONET Option to differentiate ICN traffic has to be considered as a long-term target, that we want to analyse in this section.

Our ultimate goal is to exploit the IPv4 CONET Option (or IPv6 CONET Option in case of IPv6 traffic) to distinguish CCN traffic and to decide whether an incoming packet is an Interest or a Data.

As foreseen in the CONET architecture, the Carrier Packet Header is mapped into the IPv4 (or IPv6) CONET Option, as shown in the following figure.

![Figure 5 – Mapping between CONET Header and IP CONET Option](CONET.pdf)

It is evident that in order to successfully develop such an architecture, there is the need to simply define new flows, based on the CONET Option, so that the Lookup&Cache become simpler and faster.

Because of that, we envisage the deployment of a new OpenFlow 2+, able to match against the whole IP header and, thus, able to identify the length of the IP Option field and the position of the CONET Option on the inside.

Other than the CCN enhancement of OpenFlow, for the long term we foresee also an efficient distribution of functionalities, in order to achieve best possible performances.

XXXX Se prima viene presentato un esempio di architettura, qui richiamerei la disposizione finale delle funzionalità
7. Short term approach for CONET support in OpenFlow

Being part of the OFELIA project and due to the lack of an OpenFlow version able to recognize the IP option field, this work, as the entire project, relies on the OpenFlow 1.0.0 specification and, therefore, tries to develop a solution that can run on the existing switches and API.

In order to do so, there is the need to realize a correspondence between the Carrier Packet header, which identifies CONET traffic, and another field, currently exploited by OpenFlow 1.0.0. Thus, taking into account the fields listed in Figure 3, the following possibility are envisaged:

- a MPLS tag;
- a new CONET code for “IP Protocol” and the “Ports” fields of the IPv4 header;
- locally administered MAC addresses.

7.1 Analysis of possible tag solutions

Since it is not possible to define flows exploiting the IPv4 CONET Option, for a short term implementation I decided to use the existing ability of OpenFlow 1.0.0 to match a tag in one of the three ways listed above and to define flows on those basis.

To achieve these solutions, there is the need for an application that, on the edge of a Border Node, is able to recognize CONET traffic and to map it into one of these three fields. This operation has to be considered mandatory before allowing the ingress of an ICN packet into an IP-CSS based on OpenFlow and could be realized in cooperation with the Name Routing System. In fact, as pointed out in next sections, also these local tags should be unique, raising the necessity for an entity that controls their distribution.

It is also worth to point out that, at the moment, CONET developers are not working extensively on the realization of this “mapping application”, especially because it would not be part of the long term solution, since this mapping would no longer be necessary. However, assuming a fixed position of the CONET Option within the packet header, it would be quite simple to realize an application that matches a string of bits and, carefully, maps the Network Identifier just extracted to a new, unused tag.

In case of MPLS, for example, within a Border Node it is necessary to realize a mapping between the IPv4 CONET Option and a fake MPLS tag, used not to offer label based forwarding, but to differentiate CCN packets, as shown in the following figure.
As highlighted by the previous figure, the CONET Option field is kept also within the IP Option, even if the matching is realized with the MPLS tag. I suggest to do that in order to have a structure that is “forward compatible” and will exactly fit in the long term solution, only removing the MPLS matching.

XXX Se prima vengono citate, allora vale la pena di richiamare anche qui le funzionalità, altrimenti io taglierei.

Also functionalities disposition would be a little different from the one we envisaged for the long term.
In particular CC Routing and CC Caching do not run on the switch, but on the same separate server where the controller runs.
Caches are running on a separate server where the CC Caching has to run.

IP Routing, on the contrary, is realized through virtualization, using components like RouteFlow and Quagga. Precisely, Quagga and RouteFlow provide layer 3 functionalities offering their results to the NOX Controller, which runs on the same server, that updates switch’s flow tables according to that information.

So, Lookup&Cache is realized through switch entries, decided by the controller and pushed from the server to the switch.
7.1.1 Analysis of MPLS tag solution

The use of the MPLS tag as matching field has some disadvantages but could turn out to be a good solution, if it is realized it in a way that answers to the following questions:

1. Is it possible to keep offering label switching with MPLS in CONET architecture?
2. Is there enough space to realize this correspondence without tag collisions?
3. Is it still possible to preserve a hierarchical structure, that identifies in a simple way all the chunks of a content and enables quick cache lookup?
4. Are tags re-used after some time? If yes, what should be a correct time?
5. Who is meant to realize this matching?

A first raw implementation could be done exploiting 31 bits out of the 32 the MPLS header to identify CCN packets. Doing so, it is possible to leave the highest value bit to normal MPLS usage, which means offering the possibility to \(2^{31} = 2.147 \times 10^9\) MPLS flows to go through the network and granting to the CONET other \(2^{31}\) tags to identify CCN traffic.

It is important to remember that Carrier Packet header has the structure “Network Identifier + Chunk Number + Payload Type”, so that every chunk of a content is requested by an Interest with the same name, namely “Network Identifier + Chunk Number”, of the content itself.

This means that for one logical basic entity (the chunk) that goes along the network two different tags are needed.

Considering also that the differentiation between Interest and Data Unit is one of the first operation implemented in a node, in order to decide whether perform cache-lookup or forwarding the packet and refreshing the timer, it could be useful to insert that information in a place easy to find. Because of that, I suggest the use of the least significant bit to differentiate between Interest and Data Unit, so that the identifying process in an Internal Node becomes simple in that way:

Arrival of a packet:

- Is this a MPLS packet?
  - If NOT, perform normal forwarding.
  - If YES, check the first bit.

- Is it a CONET packet?
  - If NOT, perform normal MPLS forwarding.
  - If YES, check the last bit. Is it an Interest or a Data Unit?

And so on, as described in Section 3.
Going in this direction, it is possible to have \(2^{30} = 1.073 \times 10^9\) tags to be used to identify CONET chunks of content through the network.

Considering the fact that CONET plans to have chunks of about 128-256KB and supposing an average size of contents about 32MB, it means that there are almost 120 chunks per content, so that it will be used \(2^7 = 128\) tags for every content.

This will result in \(2^{30-7} = 2^{23} = 8388608\) contents that could go through the network at the same time.

It is worth to underline the temporal aspect of this solution: in fact, a MPLS tag assigned to a chunk will be for sure re-used after a certain time, as happens in normal MPLS. Moreover it would be smart to use the MPLS tag assigned to every chunk also to address it into Internal Nodes' caches, so that cache/precache lookup becomes quick and simple.

The downside of this solution is that it is necessary to be sure that when a tag is re-assigned (so that it refers to a second content) there will be no valid copies of the first content into network caches, so that the correspondence between tag and content is effective and there are no errors in delivering contents.

A solution for this issue could be the following: when a Border Node assigns a new MPLS tag (I will expand this point later) it sets a timeout timer, refreshed every time a packet with that tag pass through it. Granting that this timer is longer than the Internal Nodes cache timeout timer would be sufficient to prevent association mistakes.

In fact, assuming that Border Nodes are the entities that assign tags, it is sure that in the process of sending an Interest and receiving a Data Unit, they would be the last to encounter a packet with a certain tag. So, if a BN's timer is over and it decides to re-use the tag, there will not definitely be any copies of previous content bound to that tag, since the cache-timeout used was shorter than Border Node's one.

Going back to the point of who is meant to assign tags, my suggestions are all about Border Nodes, eventually in cooperation with the Name Routing System. I suggest that because it is also the entity that performs routing by name, providing the so-called "Integration approach", as it allows portion of networks to act simply and unaware of content networking.

Therefore, there would not be logical differences with long-term architecture, except that now is exploited a fake MPLS tag instead of the CONET Option, to recognize CCN traffic.

Moreover, Border Nodes are the only network entities that communicate with the Name Routing System, which is centralized and could talk with all nodes simultaneously, preventing tag conflicts. In my opinion, in fact, there is the need for a centralized entity that allows BNs to assign those MPLS tags to packets, preventing the fact that two different nodes could assign the same tag to two different packets.

Therefore I suggest that Name Routing System manages those 2 tags, dividing them into sets to be assigned to Border Nodes, that have to manage them, and re-use them according to timeout policies.

If this solution is adopted, considering a network made of 32 Border Nodes, there would be \(2^{25}\) tags per BN, resulting in \(2^{18} = 262144\) contents addressable at the same time, by every BN.

Considering that this mapping is meant to be realized as a short term solution, that would be tested in small environments, the ability of addressing more that 2 millions of contents simultaneously is fully acceptable for a short term trials on a testbed.
7.1.2 **Analysis of IP protocol + ports tag solution**

It is worth to highlight the fact that the use of MPLS tag is just one of the possible implementations of a short term solution, that could be realized also exploiting another kind of mapping between the IPv4 CONET Option and one or more fields already present into the IP header which OpenFlow is aware of.

In particular, another solution that I consider useful to investigate and I will use in my work, foresees the use of a new IP Protocol Type to identify CCN traffic, with the 32 bits of the "Ports" field used as a tag for chunks.

The basic idea is to exploit one of the unassigned codes for the Protocol Type, in order to avoid conflicts with TCP or UDP datagrams and to simply identify CCN traffic. Once that distinction is done, the 32 bits originally assigned to Ports field lose their original meaning and could be exploited (together or separately) as a tag for Interest and Data chunks.

It is necessary to underline that all the considerations about tag re-use, conflict and dissemination are still valid and has to be solved in a smart way in this solution as well as in the MPLS one. Moreover, this solution could be more efficient in solving tag conflicts, since it foresee the complete use of the 32 bits originally assigned to layer 4 ports.

Because of that, there are $2^{32}$ tags available that would result, according to previous assumptions, into $2^{24}$ contents available at the same time in the network. Thus, compared to the MPLS solution, this one offers a **doubled number of contents** addressable, resulting into an even more scalable and acceptable implementation of a CONET scenario.

The downside of this solution is that it would be exploited layer 4 header fields in order to distinguish and separate CCN traffic. According to the “Integration approach”, instead, the recognition of CONET packets from normal traffic is meant to be as an enhancement of the existing layer 3, not involving the analysis of upper layers’ headers.
7.1.3 Analysis of MAC address tag solution
Instead of doing this mixing layer solution, another suggestion could be the use of MAC addresses to realize such mapping. This solution is still under investigation, but a hint could be to exploit the so called "locally administered addresses" so that there would be 48 bits available as CCN chunk’s tags. Although this solution could offer an even larger number of addressable contents, it still needs further analysis, especially regarding packet forwarding, before being successfully developed.

7.2 Other architectural aspects

The main elements that we need to realize in order to develop a working architecture in the short term are:

- **Edge Nodes** (interfaces between a non OpenFlow CONET and the OpenFlow CONET network, that take a packet with the IP CONET options and add the tag, one of the three types discussed above)
- **TAG distributors** (assist the Edge Nodes by providing a network-wide unique mapping from ICN-ID contained in the IP Option into the tag)
- **Border Node** switch elements
- **Internal Node** switch elements
- **Cache Server**
- **Controller**

The Cache Server and the Controller can be co-located in the same machine (this is the simplest configuration). On the contrary, the most efficient scenario foresees Cache Servers separated from the control logic of the switch and, eventually, shared also among different nodes.

In order to realize an efficient content delivery, we have to set up a content-advertising mechanism and a reactive way of updating caches and OpenFlow routing tables. In particular, the Cache Server needs to inform the controller about which content is stored into the cache, the controller will setup the switch table accordingly, so that an Interest corresponding to a Content present into the cache is routed properly towards the Cache Server.

That could be realized sending the interest out through the port connected to the controller and substituting MAC destination address with the address of the Cache Server. Within the Cache server we need a mechanism to read the incoming packets (i.e. the content requests). This can be based on an Hook-Up software running on the HostPC, responsible for taking the packet between layer2 and layer3 and bringing it to the application layer, preventing an early discard due to the fact that the incoming packet has an IP address that is not the HostPC one.

Other solutions, if we use a specific ethertype, could be:

- in user space, to open a raw socket
- in kernel space, to use a “protocol handler” module that registers for the specific ethertype
7.2.1 **Communications between Cache Server, Controller and Switch**

The fundamental idea of this approach is to send Interest packets to the cache, only if the corresponding data are certainly present, since the controller has been previously informed about that.

It is worth to focus on the behaviour of a node when it has to communicate with a Cache Server, defining a way in which Controller, Switch and Cache Server communicate with each other in order to perform correct packet forwarding and content delivery.

In particular, the first thing that is necessary to say is that the communication between all the three entities involved is one-way, so that the Cache Server communicates to the controller, which instruct properly the switch which sends packets to the Cache Server, according to those instructions.

More precisely, the Cache Server has to send advertising packets to the controller every time it:

- has a new content,
- refreshes a timer
- deletes an old content

So, in those packets there is the need to insert a code that uniquely identifies the action applied (one of the three listed above), the name of the content to which the action has been applied and the MAC address of the interface that is going to receive a packet (Interest or Data Unit) from the switch.

![Diagram showing communication between Cache Server, Controller and Switch](image.png)
Obviously the name of the content is the CONET one in a long term scenario, while it would correspond to one of the three solutions (MPLS, proto + ports or MAC) envisaged for the short term.

Thanks to the information received from the Cache Server, the Controller is able to push to the switch, OpenFlow messages that modify Flow Tables, reacting to the current state of the cache.
In particular it will send a “delete_datapath_flow()” message to remove an entry and a “send_flow_command()” to refresh the timer of an existing one.
Instead, if the Cache Server advertises a new content, the Controller has to send an “install_datapath_flow()” message, telling the Switch to do the following operations:

1. filter on the name/tag of the content
2. distinguish between Interest or Data Unit
3. re-write the MAC destination address with the Cache Server’s one
4. perform normal layer2 switching according to the newly inserted MAC address.

Obviously, all this considerations have to be merged with the above discussion in section “Detailed analysis of operations performed by Border Nodes and Internal Nodes”, where a more complete picture is taken.

In order to realize that behaviour, there are several operations that need to be realized, both on the controller side and, most of all, within the Cache Server. For instance, in order to successfully implement the above mentioned situation there is the need to realize, in the cache, an application able to intercept all the packets that arrive to its interfaces, even if they would normally be discarded. In fact, if all Data Units are sent also to the Cache Server, this one would receive, thanks to layer 2 forwarding performed by the switch, only packets with an IP destination address different from the one associated to its interfaces. Therefore, to prevent an early discard, there is the necessity to open a raw socket, which first receives all the packets and then filters them, recognizing CONET traffic.

Other than this raw socket application, the fundamental part of the communication that is not present at the moment is the exchange of messages between the Cache Server and the controller, realized through two dedicated interfaces.
This communication is achieved through the exchange of JSON messages, sent almost only from the cache to the controller, used to communicate the action performed over a certain content, as will be explained in next section.
We chose to use JSON messages, because it is a lightweight data interchange format, easy to read and write and, therefore, suitable for the simple information that is necessary to exchange. In fact, after two connection setup messages, the only communication that, in my opinion, was necessary to implement is made up of simple messages sent only from the cache to the controller. In particular, the only information stored within these messages would be a couple of key:value entries of the form:

```
{CONTENT : "name"; TYPE : "action"}
```
Another reason to implement this communication through JSON messages has been NOX ability to handle this kind of messages, by means of a JSONMsg_event, already present and written in C++. Therefore, after some modification to NOX, in order to make it able to rise this event even within Python controllers, it is just necessary to write a callback function to handle the event and to make the controller reactive to the received messages.

7.2.2 The Cache Server application

The first realization of the application that runs on the Cache Server has been written in Python and, essentially, establish a connection with the controller, sending some messages about the three types of actions envisaged to be performed on contents. Therefore, there is no caching policy at the moment, so that messages are sent to the controller without a precise logic and have the purpose of testing the ability of this one to react in front of different situations. It is worth to outline how there is no need to perform a particular caching policy, so that, once the application that communicates with the controller is realized, the preferred policy could be applied. In fact, from our point of view, the Cache Server should be organized as in the following figure, with three main applications: a Raw Socket application, a Cache Manager and a Sender, that communicates with the controller.

![Figure 10: Cache Server main modules](image)

This kind of structure is modular and easily extensible. In fact, at the moment, we realized the Sender application, that works without any input and also the raw socket is implemented in a way that only recognize CONET packets and answers back to the Interest with the corresponding data. These two applications communicate successfully, even if exchanging randomly generated messages. Therefore, it has been possible to realize a working testbed even without the caching logic, that could be realized later and easily placed between the sender and the raw socket.

We want to outline that the only three messages that we decided to implement during the communication between the cache and the controller are the following: stored, refreshed and deleted.
In fact, in our opinion, the controller need to modify the behaviour of the switch only if a new content is stored within the cache or if a content has been considered old and, therefore has been deleted. We chose also to add the refreshed message, used to refresh flow entries timer, in order to avoid the deletion, due to a timeout, of a switch entry responsible to send an Interest to the cache. In fact, a content that is not very popular, so that it does not hit frequently the switch entries, could anyway be on the Cache Server and, therefore, its Interest has to be forwarded there. Thus, we chose to delete a flow entry only if the cache explicitly communicates that it does not have a content anymore.

7.2.3 Implementation schema

The following schema wants to point out only the operations that would be performed into a Switch when it has to talk to the Cache Server and, obviously, has to be merged with all the functionalities and operations explained in section “Detailed analysis of operations performed by Border Nodes and Internal Nodes”.

Moreover, in the following we consider a well-known information the fact whether a content is in cache or not, since we assume that the Cache Server has already advertised the content and the Controller, in consequence, has already instructed the Switch properly.

Arrival of an Interest to the Switch:

- Is the content stored into the Cache Server?
  NO: perform normal routing
  YES: 1. re-write the MAC destination address with the Cache Server’s one
        2. re-write the ethertype of the packet with the code assigned to Interests.
        3. send the packet to the Cache Server (the complete procedure is listed above)

Arrival of a Data Unit to the Switch:

- Is this packet arriving from my Cache Server?
  YES: perform normal routing following the “path state”
  NO: 1. duplicate the packet
       - the first one follows normal routing according to the “path state”
       - the second one has to be sent to the Cache Server, so, on this copy of the Data Unit the following operations are applied:
         o re-write the MAC destination address with the Cache Server’s one
         o re-write the ethertype of the packet with the code assigned to Data Units.
         o send the packet to the Cache Server (again, the complete procedure is listed above)
So, when a packet arrives to the Cache Server, the “protocol handler” looks at the ethertype of the incoming packet and brings it to the application level, differentiating actions it has to perform in this way:

Arrival of a packet with an Interest-ethertype:
- look for the requested content into the cache
- answer back with the right content
- refresh the cache timer
- advertise this refresh to the Controller

Please note that there is no possibility that the content associated to the Interest is not into the cache, since the Cache Server itself previously advertised this content to the Controller and so forth the Switch that sent the Interest packet.

Arrival of a packet with a Data-Unit-ethertype:
- look for that chunk of content into the cache
  Is this chunk already into the cache?
  YES: - refresh the cache timer
       - advertise this refresh to the Controller
  NO:  - follow the caching policy

A simple caching policy could be to cache every content if the cache load is under a certain threshold. Assuming that kind of policy, the scheme goes on like this:

- Is the load under the threshold?
  NO:  - discard the packet
  YES: - do Precache

If there is enough space to cache all the chunks (since a content follow the “path state”, every chunk is going to arrive to the Switch and an copy of that is meant to be sent to the Cache Sever), the content would be cached and this caching event would be noticed to the Controller.
8. Details and architecture for the short term solution
References


