OFELIA

ICT-258365

Deliverable 9.3

EXOTIC final evaluation and overall report

Editor:	*Stefano Salsano (CNIT/University of Rome Tor Vergata)*
Work Package (leader)	WP9 (Stefano Salsano, *CNIT/University of Rome Tor Vergata*)
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	30/09/2013
Actual delivery date:	05/11/2013
Version:	1.0
Total number of pages:	46
Keywords:	OFELIA, FP7, Information Centric Networking, OpenFlow

Disclaimer

Imprint

[Project title]	*OpenFlow in Europe – Linking Infrastructure and Applications*
[short title]	*OFELIA*
[Number and title of work package]	*WP9 – Support of Content centric networking functionality*
[Document title]	*D9.3 - EXOTIC final evaluation and overall report*
[Editor]	*Stefano Salsano (CNIT/University of Rome Tor Vergata)*
[Work package leader]	*Stefano Salsano (CNIT/University of Rome Tor Vergata)*
[Task leader]	*Stefano Salsano (CNIT/University of Rome Tor Vergata)*
[PM (estimated)]	*8*
[PM (consumed)]	*13*

Copyright notice

Executive summary

OFELIA WP9 is concerned with the design, implementation and validation of a solution for supporting Information Centric Networking using Software Defined Networking. In particular, the solution has been implemented and demonstrated on the OFELIA testbed.

Information Centric Networking (ICN) has been proposed as a new networking paradigm in which the network provides users with content instead of communication channels between hosts. The *Software Defined Networking* (SDN) approach promises to enable the continuous evolution of networking architectures. In this light, an SDN enabled network could support ICN functionality without the need to re-deploy new ICN capable equipment.

In our first deliverable (D9.1), we described two proposed solutions: 1) a long term solution based on SDN concepts without taking into account specific limitations of SDN standards and equipment and 2) a short term solution based on OpenFlow 1.0 equipment, taking into account the features that are currently available on the OFELIA testbed.

In the second deliverable (D9.2), we described the operations used to setup and run the first experiment of our short term solution, and the goals and planned topology of a larger scale demonstration.

This deliverable D9.3 first includes the description of the extended testbed scenario and of the enhancements to the "short term" solution needed to manage these more complex topologies. These topologies have been deployed across different islands of the OFELIA testbed. We introduced more advanced caching strategies that improve the efficiency of ICN over SDN. According to the SDN philosophy these strategy have been implemented in the OpenFlow controller. The deliverable reports on the web GUI that controls the experiment and on the performance comparison among different caching strategies obtained from the experiments over the OFELIA testbed.

Finally we include the specification and implementation details of an extensions to OpenFlow protocol that allow to easily define and transport arbitrary information. This extension is useful for our "long term" approach of extending OpenFlow to support ICN over SDN, as it drastically simplify the process of extending the semantic of the OpenFlow protocol.

List of authors

Organisation/Company	Authors
CNIT	Stefano Salsano, Luca Veltri, Stefano Brogi, Alessandra Agosta, Giacomo Morabito, Nicola Blefari-Melazzi, Fabio Patriarca, Pier Luigi Ventre
Lancaster Univ.	Matthew Broadbent

Table of Contents

List of figures and/or list of tables

Abbreviations

CONET	COntent NETwork
ICN	Information Centric Networks
NDN	Named Data Networking
OF	OpenFlow
TBF	Tag Based Forwarding
TBFF	Tag Based Forwarding – Filter

1 Introduction

In our deliverable D9.1 [1], we proposed and discussed solutions to support ICN using SDN concepts. We focused on an ICN framework called CONET, which grounds its roots in the CCN/NDN architecture. We addressed the problem in two complementary ways. First we discussed a general and "long term" solution based on SDN concepts. This "long term" solution does not take into account specific limitations of current SDN standards and equipment. Then we focused on a "short term" solution to support ICN functionality over the OFELIA large scale SDN testbed, based on OpenFlow. The current OFELIA testbed is based upon OpenFlow 1.0 equipment. Therefore, we designed the experiment to use only those features that are currently available.

In our deliverable D9.2 [2] we planned the performance evaluation of the proposed "short term" solution in more complex scenarios. We have identified our general methodology for experimentation and performance evaluation and defined the performance metrics of interest.

In this deliverable D9.3 we report on the extensions of the solution to work on more complex scenarios, which required several enhancements in our logic implemented in the OpenFlow controllers. We also report on the performance evaluation on a multi-site experiment over the OFELIA testbed.

We refer the reader to the deliverable D9.1 [1] for all the details of the solution. We also assume background knowledge in the principles of ICN and SDN. On the other hand, we preferred to include in this document the most relevant information coming from D9.2 [2], in order to reduce the number of documents that gives the full vision of our work. An overview of our solution can also be found in the journal paper [4].

The content of this document is as follows. In section 2 we present the extended testbed scenario based on a multi-island OFELIA experiment. In section 3 we describe the basic solution for ICN support using OFELIA/OpenFlow, focusing on the extensions we have designed and implemented with respect to the simple experiment described in D9.1. Section 4 provides the definition of advanced caching strategy that we have designed exploiting the SDN/OpenFlow approach. Section 5 introduces the GUI used to control the experiment. Section 6 provides the results of the experiment. Finally in section 8 we report our work on the design and implementation of a flexible extension mechanism for the OpenFlow protocol. An appendix reports the complete instructions to replicate the experiments.

2 Testbed scenarios

Figure 1, which describes the first simple ICN over SDN scenario implemented in the OFELIA testbed, can also be used as reference for the description of the architectural elements. In our scenarios we have a set of “Content Clients” making requests to download a set of “contents” using an ICN approach. This means that the ICN applications in the Content Clients will make requests to download a certain content identified by its “content name”. According to the ICN paradigm, these requests will be served by the origin Content Server, or by a node in the path, which may have cached the content beforehand. The application running on the Content Client is based on the CCNx implementation [3]. In our experiments, the requests are initially served by the Content Servers. While the content data is provided back to the Content Client, the OpenFlow Controller can instruct the OpenFlow switches to forward a copy of the packets to the Cache Server. The Cache Server will inform the OpenFlow Controller when the “chunks” of the contents become available in its cache. The Controller can instruct the OpenFlow Switches to direct further requests for cached content to the Cache Server, rather than the Content Server. For the above procedure to work correctly over an OpenFlow 1.0 testbed, flows are identified by a “tag”. In this case, each flow represents the delivery of a different content chunk. This is carried in the source and destination port fields of UDP packets. Therefore a mapping functionality from ICN content name to the 4 bytes tag is implemented at the border of the SDN network, in our case in the Content Client itself. Moreover, regular UDP packets and UDP packets carrying ICN requests/ICN content are identified using a different set of IP addresses.

A key element for the running of our experiments is the management server that is shown in Figure 1. This server provides the overall control of the experiment, offers a GUI to the experimenter, and collects the results by monitoring the interfaces. It interacts with the OpenFlow Controller over a Northbound API based on the HTTP REST approach.

Under the control of the management server, the OpenFlow Controller can activate “Tag based forwarding” in the OpenFlow switches. This means routing ICN packets based on their tag. In this case, the different strategies for cache management that will be described in this document can also be controlled. Alternatively, the Controller may simply instruct the switches to use default layer 2 forwarding based on MAC address learning.

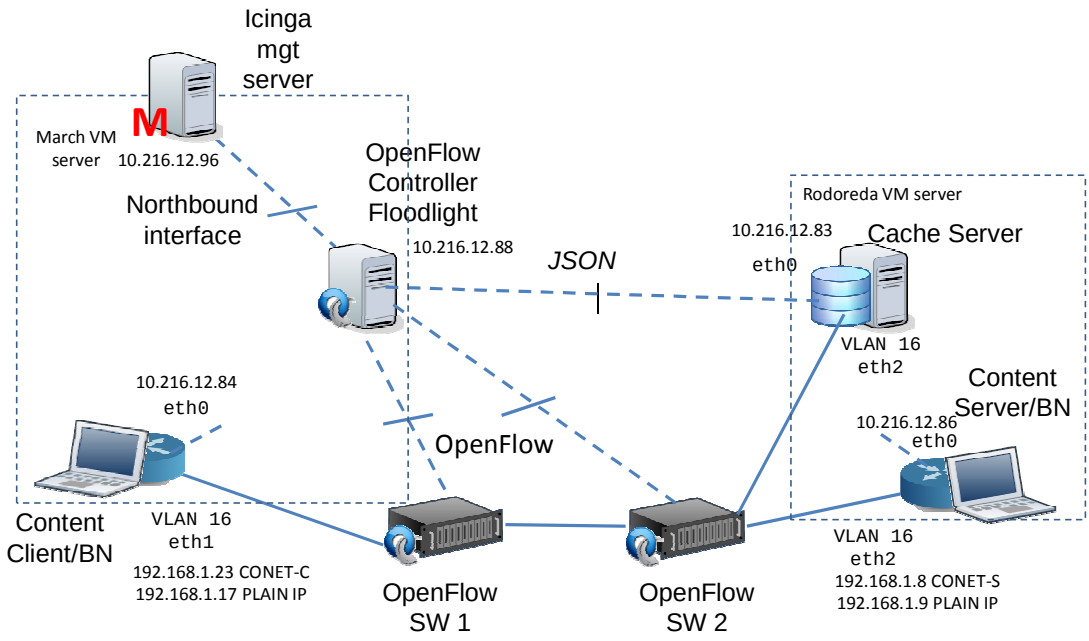


Figure 1- First ICN testbed in OFELIA

The setup for the simple demo reported in D9.1 is shown in Figure 1. The main results that were obtained from this simple experiment are: i) verification of the functionality of the system and ii) measurement of the load shift from the Content Server interface toward the Cache Server interface when the ICN TBF mechanism is activated.

2.1 Extended testbed scenarios

After developing the basic experiment described in deliverable D9.1, we have generalized our ICN/SDN solution to work over more complex topologies. These topologies allows us to experiment ICN routing/caching strategies that can be driven by the OpenFlow Controller. A target topology for the experiments was proposed in deliverable D9.2 and is reported on Figure 2. Such a relatively large topology can be built using inter-island federation of OpenFlow resources, and is made possible by the recent versions of the OFELIA Control Framework. In Figure 2 we have 5 federated islands: Trento (1), Barcelona (2), Catania (3), Zurich (4) and Berlin (5).

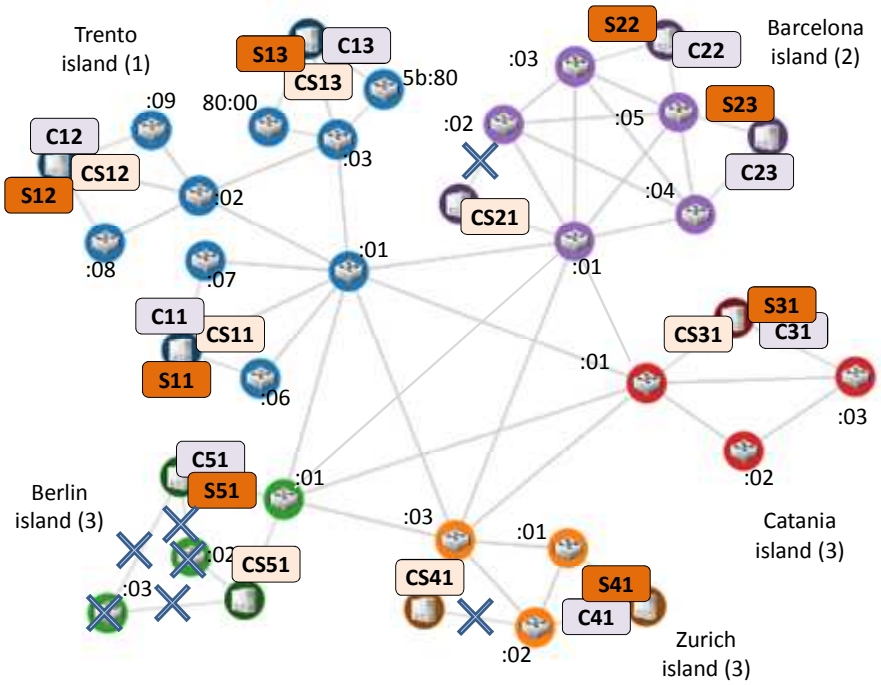


Figure 2- Initial target topology for ICN traffic engineering experiment

For practical reasons, related to the ease of performing the experiments and to the inter-island topology that has been deployed in the OFELIA testbed, we have scaled down the experiment to the topology described in Figure 3. In this topology we have 4 federated islands Trento (1), Barcelona (2), Zurich (3) and Gent (4). The Gent island provides inter-island connectivity with an “hub-and-spoke” topology.

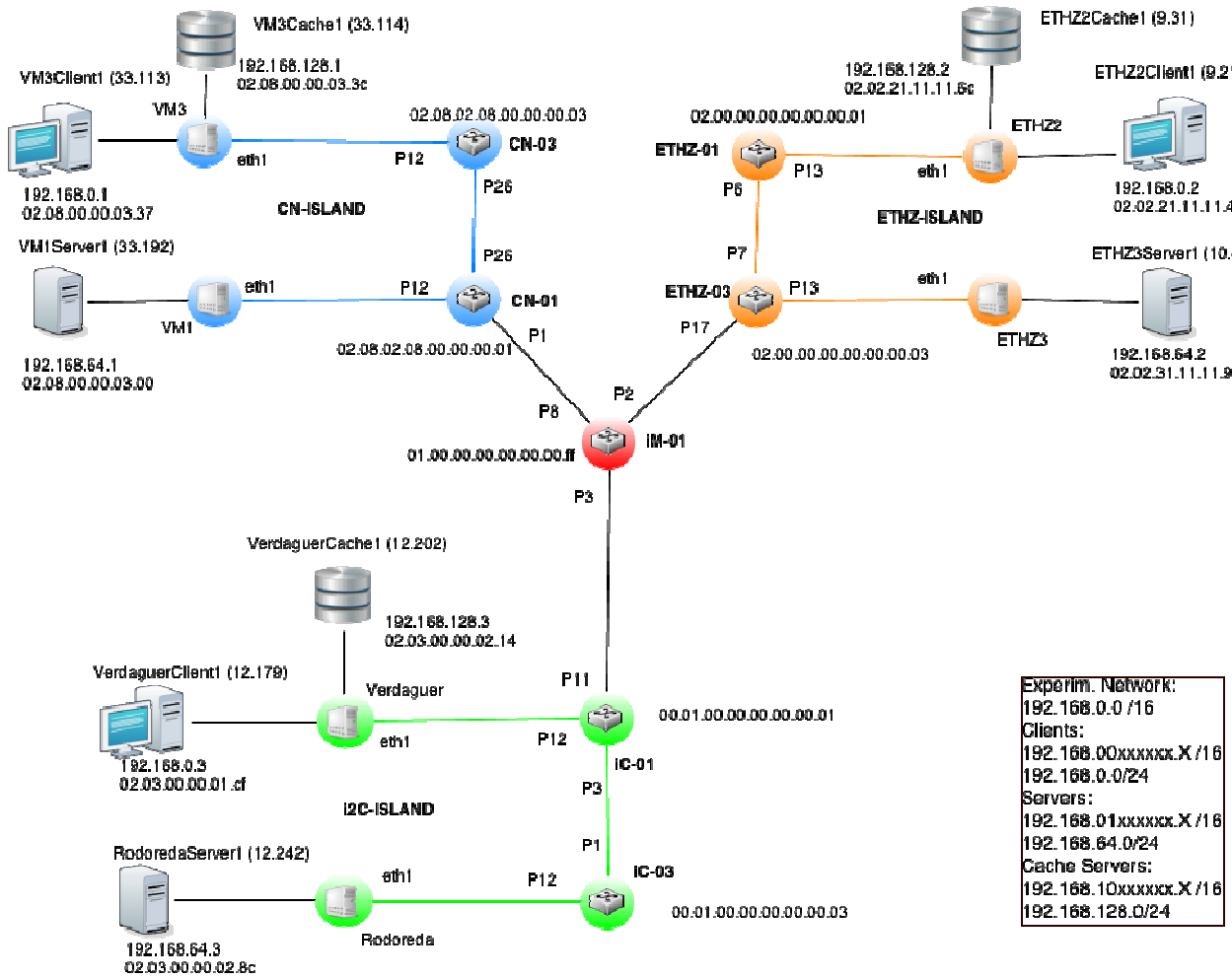


Figure 3- Deployed topology for ICN traffic engineering experiment

2.2 Mapping the experiment topology into a realistic scenario.

Let us discuss how the realized experimental topology can be mapped into a realistic scenario. We start from the assumption that a large scale ICN capable network is deployed, composed of ICN nodes offering ICN based distribution of content to ICN clients and ICN servers (end-nodes). We keep this distinction between client and server for clarity, but obviously any end-node can be at the same time both client and server. As shown in Figure 4, a set of ICN "edge" nodes can be interconnected by means of a data center or a provider network based on SDN. These nodes are called "edge" as they operate at the border of the domain that operates using SDN mechanisms. Within the SDN domain, a set of OpenFlow switches and an OpenFlow controller are represented. In our experiments we restrict our attention to the SDN domain, including the edge nodes. The ICN clients and ICN server in our experiments are located on the edge nodes. An ICN client in the testbed therefore represents all the clients whose requests flow across the edge node and enters the SDN domain. Likewise, an ICN server in the testbed represents all the data coming from servers that inject information through the edge node. An ICN client or ICN server represent a potentially large set of ICN clients and ICN servers respectively.

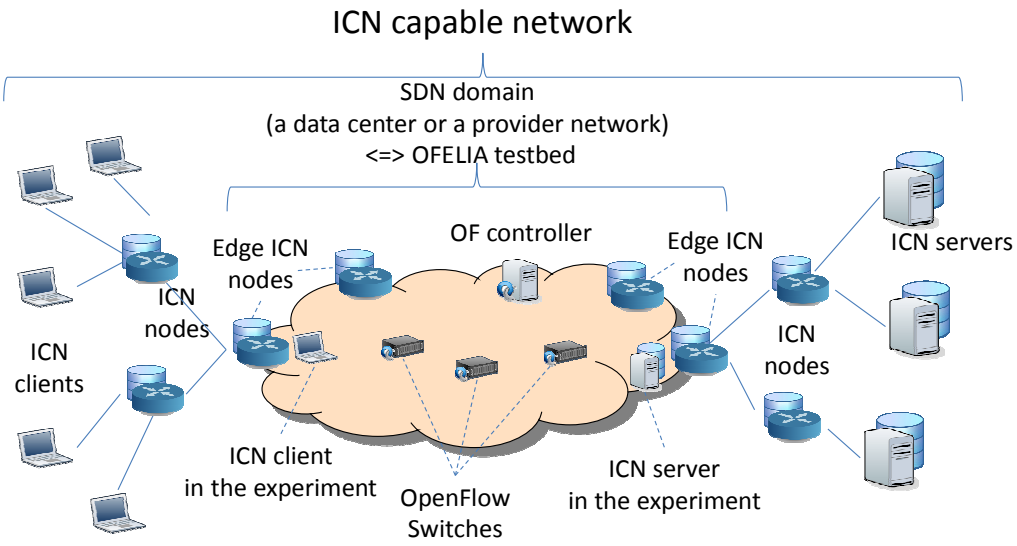


Figure 4- Mapping our experimental topology into a realistic scenario

3 Description of the Tag Based Forwarding solution

3.1 Description of the basic behavior

The Tag Based Forwarding mechanism can be seen as split into two phases. In the first phase the default paths towards ICN servers (for ICN interest packets) and towards ICN clients (for ICN data packets) are installed in the switches. Also the duplication rules for ICN data packets towards the Cache Servers are setup in this first phase. In the second phase the specific redirection rules for the interest packets related to the content chunks that are stored in the Cache Server are established.

In our solution the first phase is driven by packets coming from ICN client and ICN servers, and the default paths are discovered with a mechanism identical to the “auto-learning” of Ethernet switches. This is obviously done under the direction of the OpenFlow controller. A TBF enabled switch reports to the controller packets received with IP address belonging to an ICN server or an ICN client when no rules are present, then they are instructed by the OpenFlow controller to set the default path toward ICN servers (which will be followed by the Interest packets) and toward ICN clients (which will be followed by the ICN Data packets). Though this first phase is not preconfigured (it is driven by the received packets), for simplicity we refer to the resulting set of entries as “static TBF rules”. The static TBF rules will not expire, as we assume that ICN clients and servers are static (they represent ingress/egress points in a SDN enabled data center or provider network offering ICN services).

When the data packets cross the TBF capable switches, they are duplicated by the TBF capable switches and sent towards the Cache Servers. The Cache Server reconstructs the chunks and when a full chunk is available, stores it and notifies the controller. The controller will install a rule in the switch to redirect future interests towards the Cache Server. We refer to this second set of entries as “dynamic TBF rules”. These dynamic rules will be variable in time, as they reflect the content that is cached in the Cache Server.

To implement the above-described scenario, we have the need to differentiate among ICN interest and data packets arriving to the switches, only using OpenFlow 1.0 capabilities. The idea for doing it efficiently is to associate a set of contiguous IP addresses to ICN clients and another set to ICN servers. This approach also allows differentiation between ICN packets and regular IP packets: the ICN client and server addresses are additional addresses that are used only for ICN communications. If a device needs to play the role of ICN client or ICN server, it will respectively have an ICN client or ICN server address on its interface in additions to the regular IP address. The Interest packets flow from ICN clients towards ICN servers. Their IP source address is an ICN client address, whilst their IP destination address is an ICN server address. Likewise, the ICN Data packets flow from ICN servers towards ICN clients. Their IP source address is an ICN server address, whilst their IP destination address is an ICN client address.

3.2 Dealing with limited resources for caching

In general, when caching mechanisms are used, proper cache management strategies need to be devised to cope with the limited amount of caching resources. The ultimate goal of such strategies is to maximize the advantages arising from the use of the cache: for example the probability of finding content in the cache (cache hit ratio) should be maximized.

A peculiar aspect of our cache management strategy is that it needs to be coordinated among three elements: the Cache Server, the OpenFlow switch and the Controller.

In our specific scenario, during the operation of the TBF mechanism the Cache Server could fill up its local storage space, or the number of possible entries in the switch flow table can be exhausted. Therefore we potentially have to cope with two bottlenecks (storage space and flow table entries).

In the scenario that we deployed on the OFELIA testbed the limited resource is the number of flow table entries, which can be easily depleted by the dynamic TBF rules for the interest forwarding towards Cache Servers. In theory we have one entry per cached chunk in a switch. The goal of this entry is to redirect an interest request belonging to a given chunk to the Cache Server. For this entry, the ingress port does not care and it is "wildcarded" in the OpenFlow terminology. In practice, the FlowVisor [5] module takes care of "slicing" the switches among a set of independent experiments. If an experiment topology is composed by a subset of the switch ports, a request to set an entry that refers to "all ports" is translated by Flowvisor into a set of entries that refer to the specific ports under the control of the given slice. This will multiply the number of entries by a factor that depends on the number of ports used by a switch in our topology.

We performed an experiment adding more and more dynamic TBF rules until we got a "Table full" error from the OpenFlow switches. The results are reported in Table 1. It shows that the maximum number of entries is 1518 and this corresponds to a different maximum number of cached items, depending on the number of switch ports that we have used in our slice.

DATAPATH	Entries	Ports	Cached Items	Switch Model
02:08:02:08:00:00:00:03 (CN-03)	1518	2	755	AX-3640-24T2XW-L
00:10:00:00:00:00:00:01 (i2C-01)	1518	3	503	AX-3640-24T2XW-L
02:00:00:00:00:00:00:01 (ETH-01)	1518	2	755	AX-3640-24T2XW-L

Table 1 – Maximum number of entries in the switch flow table for a slice.

Taking into account the results of the experiment, we assumed that the Cache Server storage space is large enough to contain a number of chunks corresponding to the maximum number of entries in the flow table. In other words we focused on the scenario in which the logical bottleneck of the system is the number of flow table entries in the switch.

We have designed a simple strategy to be deployed in our testbed. The idea is to let entries expire in the flow table if they are not used after a time-out, using standard built-in OpenFlow mechanisms. Thanks to these, the expiration of a flow entry causes a notification to be sent back to the controller.

If the same chunk is requested again at a later time, the switch will simply forward the interest packet to the origin server and when the data packets are sent back, the Cache Server can cache the chunk again.

In order to remove the chunk from the Cache Server local storage, different options are possible: 1) the Controller can instruct the Cache Server to delete the corresponding chunk in the Cache Server storage (in this case a new message between the Controller and the Cache Server); 2) the Cache Server could independently implement an expiration mechanism, with a timeout greater than the switch timeout for flow entries, in order to delete chunks that have not been requested for a given time. The second solution is preferable, as it does not require an explicit message from the controller.

The procedure could be extended In order to consider the case in which the cache storage space is a limiting factor (this is still for further study and has not been realized). In the simplest solution, the Cache

Server can simply stop caching new chunks when the cache storage space is full. A more complex approach could be to make room in the cache storage by deleting the Least Recently Used chunk (LRU preemptive strategy), but this requires coordination with the controller, as the controller should be informed of the deletion in order to remove the entry in the flow table.

4 Experimenting different ICN caching strategies

In the scenario described in Figure 3, we run 3 ICN clients, 3 ICN servers and 3 Cache Servers, deployed on 6 different VM hosting servers (two for each of the 3 testbed islands in Trento, Barcelona and Zurich).

In the topology used for our scenario, we control 7 OpenFlow switches. 3 switches have an associated Cache Server. We refer to this set of switches as "ICN Tag-Based-Forwarding capable" or simply TBF-capable. A TBF capable switch is able, under the direction of the controller, to duplicate the ICN data packets towards the Cache Server and to redirect ICN interest packets toward a Cache Server. The remaining switches that are not "TBF capable" will be used as simple layer 2 switches in our scenario. It is our decision to make a switch TBF capable or not, because it is the controller that provides the forwarding logic for the OpenFlow switches.

In the implemented scenario an OpenFlow switch can be associated with a single Cache Server. This assumption is related to the hypothesis that in the long term the Cache Server will be co-located with the OpenFlow switch or even integrated within the switch itself. In this case there is no need to redirect to an external remote Cache Server.

In this section we define a more advanced caching strategy on top of the TBF mechanism described in the above solution and provide a comparison between the basic and the advanced strategy.

4.1 TBFF (TBF with Filter) caching strategy

The "basic" ICN Tag Based Forwarding mechanism foresees that all content that flows across a TBF capable switch is copied to the associated Cache Server and that for each cached chunk the interest requests are redirected toward the Cache Server. This default strategy implies that the content coming from a server is "local" with respect to the TBF capable OpenFlow switch at which they are cached. A local server could be a server located in the same POP. In our testbed we can identify as local the server located in the same testbed island. Caching only local content reduces the local server load, but it does not reduce the load on the WAN links that interconnect the POPs (or testbed islands). If cache resources are limited, using the cache resources for caching this local contents is less efficient than caching the "remote" contents coming from remote servers. Starting from this assumption, we have designed and implemented a mechanism to selectively cache content coming from remote servers. This new caching logic is implemented with and driven by the OpenFlow controller. We refer to this mechanism as TBFF: Tag Based Forwarding – Filter. In fact, the controller is able to instruct the OpenFlow switches to filter the ICN data packets, replicating towards the Cache Server only those that are coming from remote servers. The filtering mechanism is based on the classification of source IP addresses in ICN data packets coming from ICN servers.

In our testbed, we first verified the correctness of the implementation. To begin with, we published a set of contents in ICN servers located in different islands. Then we setup a script in an ICN client to periodically request a set of contents located in local ICN server (i.e. in the same testbed island of the client) and in remote ICN servers (i.e. in different testbed islands). At the beginning of the experiment, the ICN interest packets flow towards all the ICN servers and the ICN data packets return using the inter-islands links. After a short while, we could see that all inter-island traffic has disappeared and that the only server that continued to receive interest packets and forward data packets was the server located in the same island of the requesting client.

We also setup a second more complex experiment to show the effectiveness of the TBFF strategy in saving the WAN links in presence of limited caching resources. We assumed a time dependent pattern of requests coming from the ICN client. The client starts requesting a set of contents and then changes this set: some contents are not requested anymore and new contents enter the "active set" of requested contents. It is important to note that the ICN client in our experiment represents an ingress node in our SDN based content distribution network. The requests coming from our ICN client represent the aggregate set of requests coming from the downstream ICN clients that are connected to this ingress node. We can design the request pattern so that the active set of requested content exceeds the resources available for caching. Therefore only a subset of the active sets of contents will be cached. The cache hit probability will depend on the fraction of the active set that it is possible to cache.

Thanks to our entry expiration mechanisms, the contents that are cached in the Cache Server tend to follow the active set of requested content. In fact, unused entries will expire, making room for new entries for new contents. In the basic TBF case, the limited cache resources will be used randomly for local and remote content. This is in comparison to the advanced TBFF case, where the cache resources will be used only for remote content. For a given request pattern exceeding the available caching resources, composed of a mix of local and remote requests, we can measure the inter-island bandwidth usage and compare the TBF and TBFF solutions.

4.2 Methodology for performance evaluation

The overall methodology for the performance evaluation of the proposed solutions starts with the generation of synthetic ICN traffic using a set of ICN client and server applications. This synthetic ICN traffic will be processed inside the ICN/SDN network. From this, we will gather performance metrics. These are used to evaluate different strategies for controlling resources. This process is graphically represented in Figure 5.

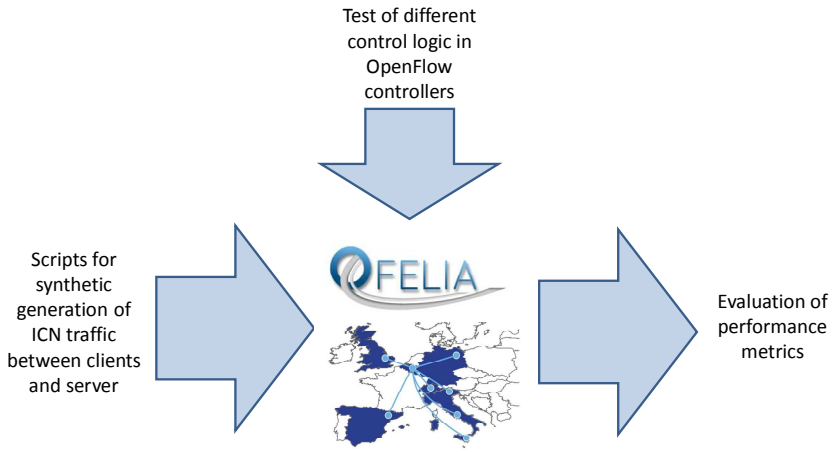


Figure 5 – Methodology for evaluation of performance metrics

We will take as reference a solution that does not use in-network caching and only uses shortest path routing between ICN client and ICN servers (NO-TBF). Then we will introduce the in-network caching capability provided by Cache Servers, with the basic TBF mechanism. Finally we will enhance the solution by using a strategy to selectively cache content on a Cache server in order to maximize the gain obtained by caching (TBFF).

4.3 Traffic patterns of ICN requests for our experiments

The effectiveness of the Tag based forwarding scenario depends on the pattern of the requests and on the capacity of caching the content in the local caches. The cache hit ratio is defined as the ratio between content requests that can be served by a cache and total number of requests. As for the request pattern, if all clients request the same content, the cache hit ratio will be close to 1 even with a minimal cache capacity. If all clients request different contents the hit ratio will be 0 for any size of the cache and whatever caching strategy is used. We do not focus on the dependency between the traffic pattern and the cache effectiveness, which requires some knowledge or assumptions about the statistical distribution of the requests and a probabilistic analysis. We made very simple assumptions on the traffic pattern by defining some deterministic traffic patterns and studied the performance of the system under these deterministic patterns. Our results could be used as input to more sophisticated models involving probabilistic characterization of traffic patterns, but this is for further study and we leave it out of the scope of the present analysis.

We considered two types of traffic patterns as input for our experiments, a “static” pattern and a “dynamic” pattern. The static pattern type foresees a continuous repetition of a set of content requests, which is only characterized by the number (N) of contents in the set. In dynamic pattern types, the “active set” of content requests changes over time, and can also be characterized by the number (NA) of different contents in the active set.

In the static pattern, from a given ICN client (we recall that an ICN client represents a potentially large set of ICN clients) we request a set of N different content objects, each one composed by M chunks (the size of each object will be M*chunk_size). Note that our basic unit of storage is the chunk. Therefore each client will periodically generate requests for N*M different content chunks. Let us call cycle time Tc [s] the duration of the period. According to the TBF logic, the chunks will be stored in the Cache Server and the following requests will be forwarded to it. If we assume that the expiration time-out To [s] of the entries in the switch is longer than the cycle time Tc, the entries in the switch flow table will not expire in our experiment.

If in the dynamic pattern the client periodically (with cycle time Tc [s]) requests a set of NA different contents, then after a duration Ta [s] (with Ta > Tc) it will request a different set of NA contents.

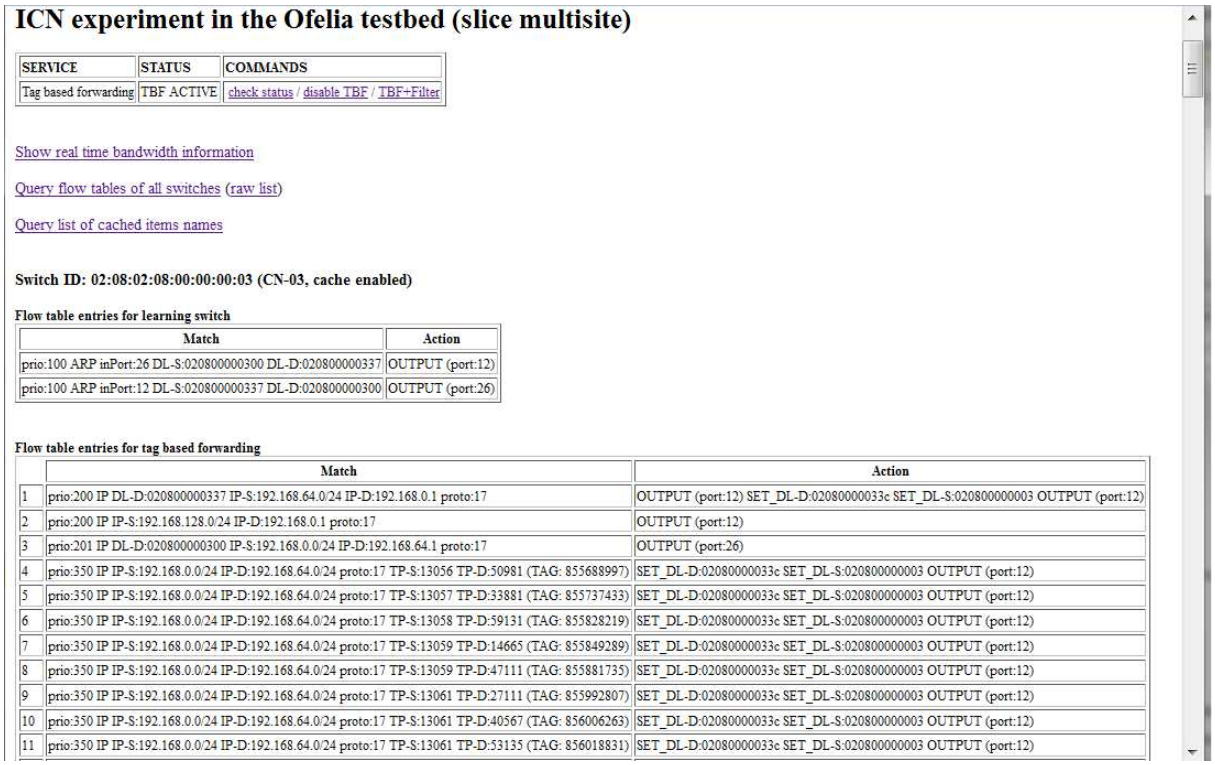
5 Experiment control GUI

In this section we provide a short report about the GUIs for controlling the experiments.

The main web page for controlling the experiments is reported in Figure 6. This page provides the capability to select the three different modes of operations for the ICN packets: No TBF (corresponding to a regular learning switch), TBF (Tag Based Forwarding), TBFF (TBF with filter). It also displays the current mode.

Figure 6 – ICN over SDN experiments home page

From the main page, it is possible to display the flow tables in the OpenFlow switches (Figure 7). Each entry reports the OpenFlow “match” and “action” in a human readable format. Two types of visualization are offered, a raw visualization that reports all entries listed by the controller and a “compact” visualization that summarizes the entries that have been duplicated by FlowVisor into a single entry. For both visualization types, the flow table entries are divided into the subset of entries related to TBF mechanisms and the subset of entries related to regular IP traffic managed by a classical OpenFlow “learning switch” logic.



ICN experiment in the Ofelia testbed (slice multisite)

SERVICE	STATUS	COMMANDS
Tag based forwarding	TBF ACTIVE	check status / disable TBF / TBF+Filter

Show real time bandwidth information

Query flow tables of all switches (raw list)

Query list of cached items names

Switch ID: 02:08:02:08:00:00:00:03 (CN-03, cache enabled)

Flow table entries for learning switch

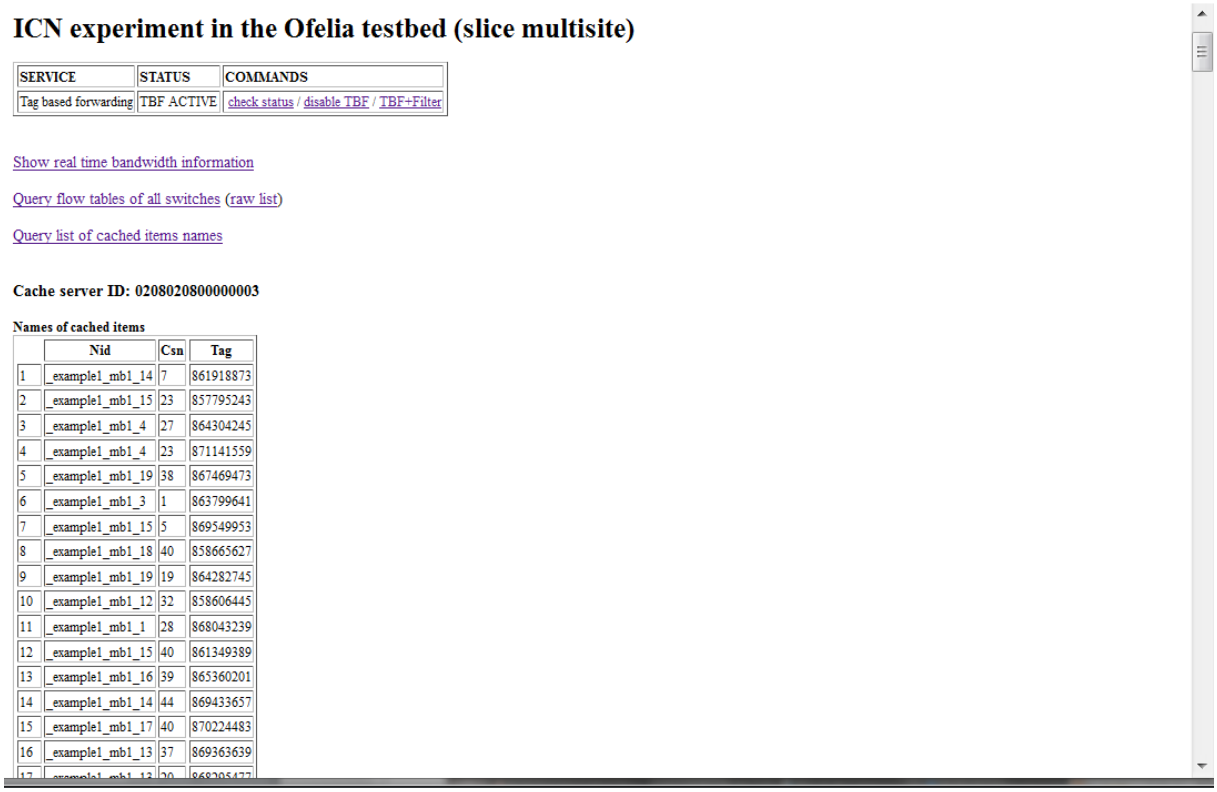
Match	Action
prio:100 ARP inPort:26 DL-S:020800000300 DL-D:020800000337	OUTPUT (port:12)
prio:100 ARP inPort:12 DL-S:020800000337 DL-D:020800000300	OUTPUT (port:26)

Flow table entries for tag based forwarding

	Match	Action
1	prio:200 IP DL-D:020800000337 IP-S:192.168.64.0/24 IP-D:192.168.0.1 proto:17	OUTPUT (port:12) SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
2	prio:200 IP IP-S:192.168.128.0/24 IP-D:192.168.0.1 proto:17	OUTPUT (port:12)
3	prio:201 IP DL-D:020800000300 IP-S:192.168.0.0/24 IP-D:192.168.64.1 proto:17	OUTPUT (port:26)
4	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13056 TP-D:50981 (TAG: 855688997)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
5	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13057 TP-D:33881 (TAG: 855737433)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
6	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13058 TP-D:59131 (TAG: 855828219)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
7	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13059 TP-D:14665 (TAG: 855849289)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
8	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13059 TP-D:47111 (TAG: 855881735)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
9	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13061 TP-D:27111 (TAG: 855992807)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
10	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13061 TP-D:40567 (TAG: 856006263)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)
11	prio:350 IP IP-S:192.168.0.0/24 IP-D:192.168.64.0/24 proto:17 TP-S:13061 TP-D:53135 (TAG: 856018831)	SET_DL-D:02080000033c SET_DL-S:020800000003 OUTPUT (port:12)

Figure 7 – Showing the flow tables of OpenFlow switches

From the main page, it is also possible to display the list of cached items seen by the controller (Figure 8).



ICN experiment in the Ofelia testbed (slice multisite)

SERVICE	STATUS	COMMANDS
Tag based forwarding	TBF ACTIVE	check status / disable TBF / TBF+Filter

Show real time bandwidth information

Query flow tables of all switches (raw list)

Query list of cached items names

Cache server ID: 0208020800000003

Names of cached items

	Nid	Csn	Tag
1	_example1_mb1_14	7	861918873
2	_example1_mb1_15	23	857795243
3	_example1_mb1_4	27	864304245
4	_example1_mb1_4	23	871141559
5	_example1_mb1_19	38	867469473
6	_example1_mb1_3	1	863799641
7	_example1_mb1_15	5	869549953
8	_example1_mb1_18	40	858665627
9	_example1_mb1_19	19	864282745
10	_example1_mb1_12	32	858606445
11	_example1_mb1_1	28	868043239
12	_example1_mb1_15	40	861349389
13	_example1_mb1_16	39	865360201
14	_example1_mb1_14	44	869433657
15	_example1_mb1_17	40	870224483
16	_example1_mb1_13	37	869363639

Figure 8 – Showing the list of cached contents

The performance analysis page is shown in Figure 9. This page is automatically refreshed every 10 seconds, allowing to monitor in real time the relevant set of parameters of the ICN over SDN solution. It monitors the Incoming and Outgoing traffic on the experimental network interfaces of all ICN client, ICN servers and Cache Server of an experimental slice. It also monitors the number of cached items in each Cache Server, and the total number of cached items on all Cache Servers.

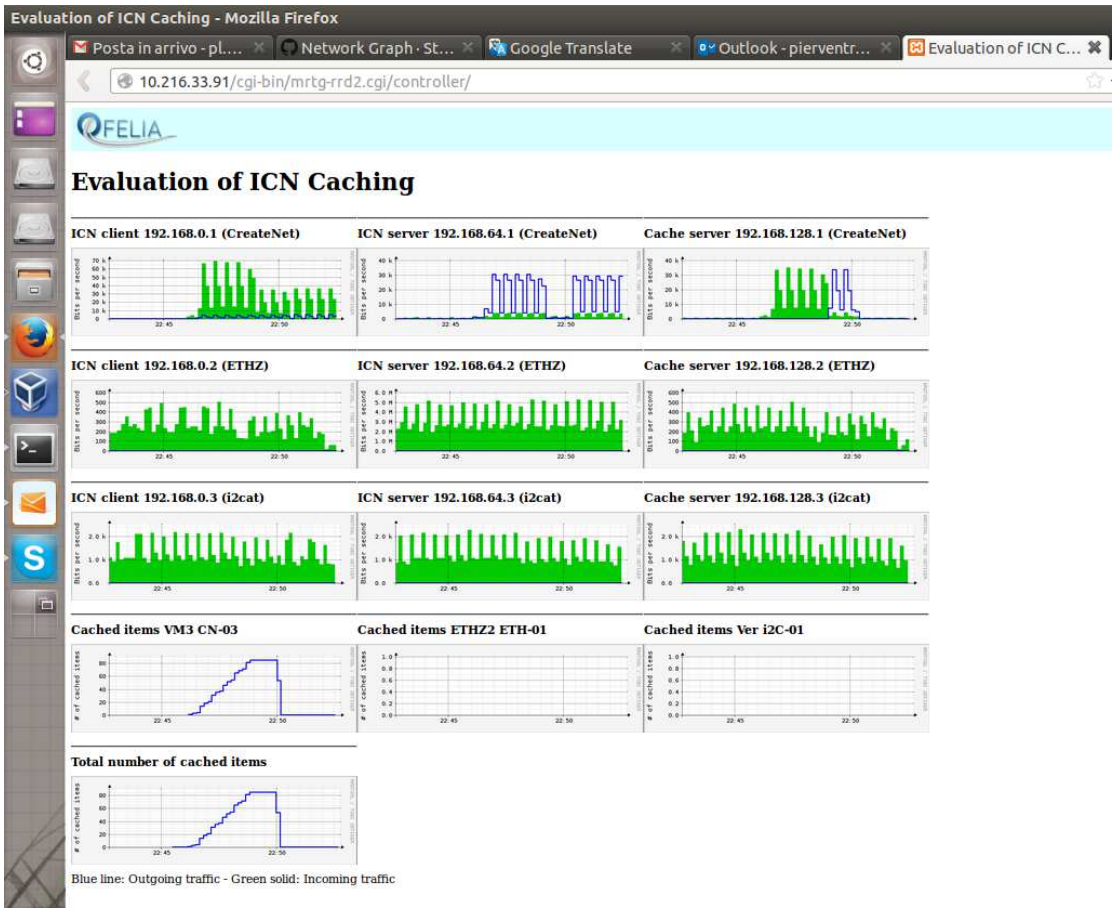


Figure 9 – Performance analysis page

6 Experiment results

In this section we present some experimental results.

In the first experiment we compare No-TBF with TBF, in a scenario where the cache resources are able to cache all the set of requested contents. We defined a static request pattern for the ICN clients, periodically requesting N contents to the ICN servers. In particular, each client request content from a remote server. If No-TBF is used, the requests and the data keep flowing on the inter-islands links. In Figure 10 it is possible to see the load on the ICN servers interfaces, while the Cache servers are unloaded (note that the scale of the graph is dynamically adapted and that for Cache Server the scale is set to bit/sec, wile for the ICN server it is set in the order of 10kbit/s).

Figure 10 – Experiment 1, No-TBF

When TBF is activated, the number of cached items increases, the load on remote servers decreases as well as the inter-island. After the duration of a cycle of requests, all contents are cached in the Cache Servers and the load on remote servers as well as the inter-island traffic becomes null. In Figure 11 the start time for the clients to start performing their requests is marked by the vertical red line. Initially, the set of N different contents are requested and provided by the ICN servers. During the data transfer, the packets are copied to the Cache servers, that cache the content and notify the controller (the number of cached items increases). When the first round of requests ends, the clients start requesting again the same set of content. From now on the contents will be served by the Cache servers: the load of the ICN servers and on the inter-island links goes to zero, while the number of cahed items remains constant.

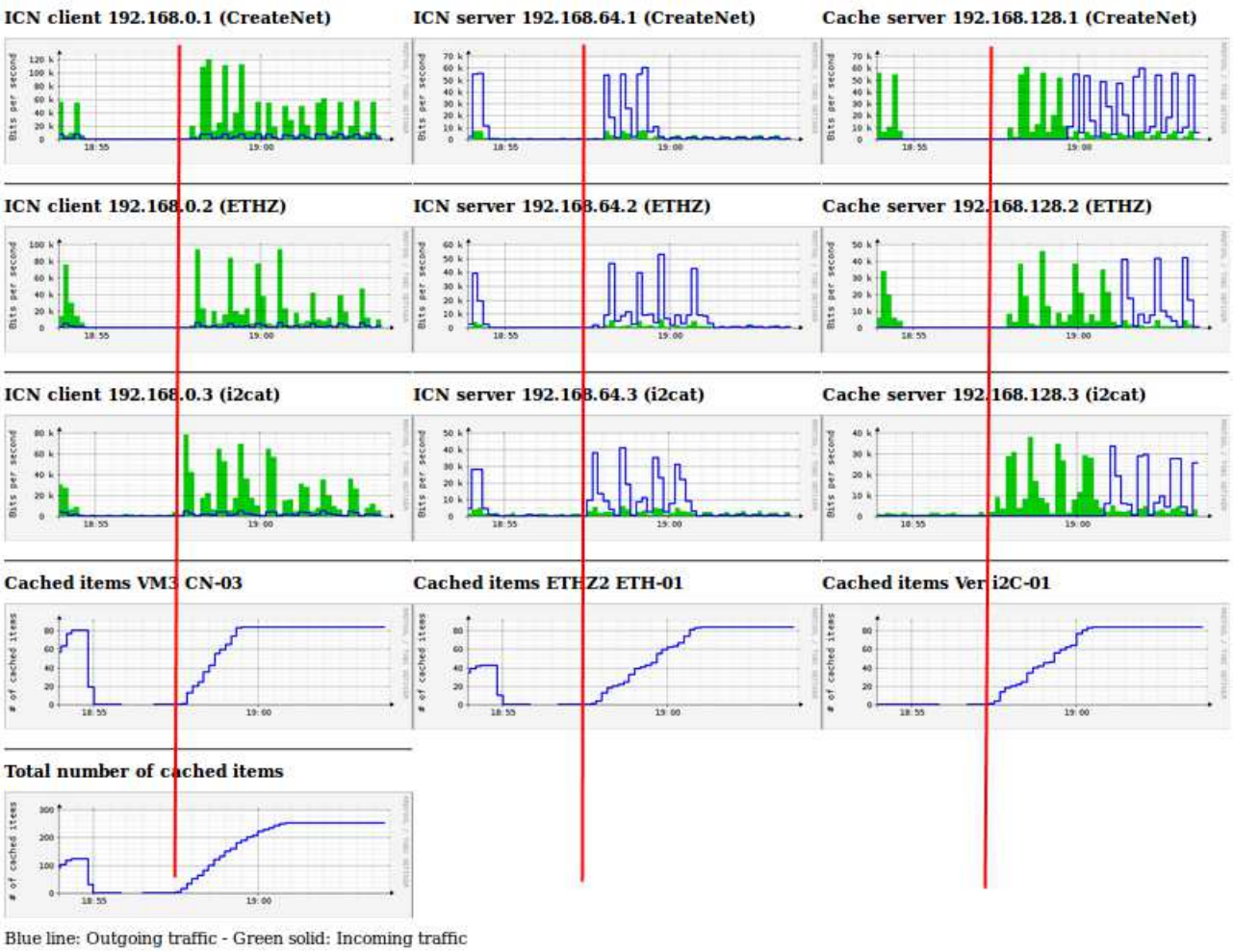


Figure 11 – Experiment 1, TBF enabled

In the second experiment we compare the basic TBF with the advanced caching strategy TBFF for a static pattern of requests, exceeding the flow table entries capacity. We assume that an ICN client located in CreateNet island is periodically making a set of requests to three different servers, one located in CreateNet islands, and the other two in the remote islands (i2Cat and ETHZ). In Figure 12, we report the "steady state" of the experiment, after the first cycle of requests has been performed and the maximum number of entries in the switch flow table is reached (it can be seen that the number of cached items in the cache is slightly more than 1K. The client continues to request the set of contents. The first part of this set is served by the local cache server in CreateNet, and the load on the three ICN server is null. Then, for the final part of the content set, we see traffic in all the three ICN server interfaces, due to the content requests that were not redirected to the Cache Server. The traffic coming from ICN servers in ETHZ and i2Cat is crossing the inter-island links. In this phase the traffic in the Cache Server becomes incoming rather than outgoing. The reason is that the OpenFlow switch copies the data packets coming from the ICN servers to the Cache Server.

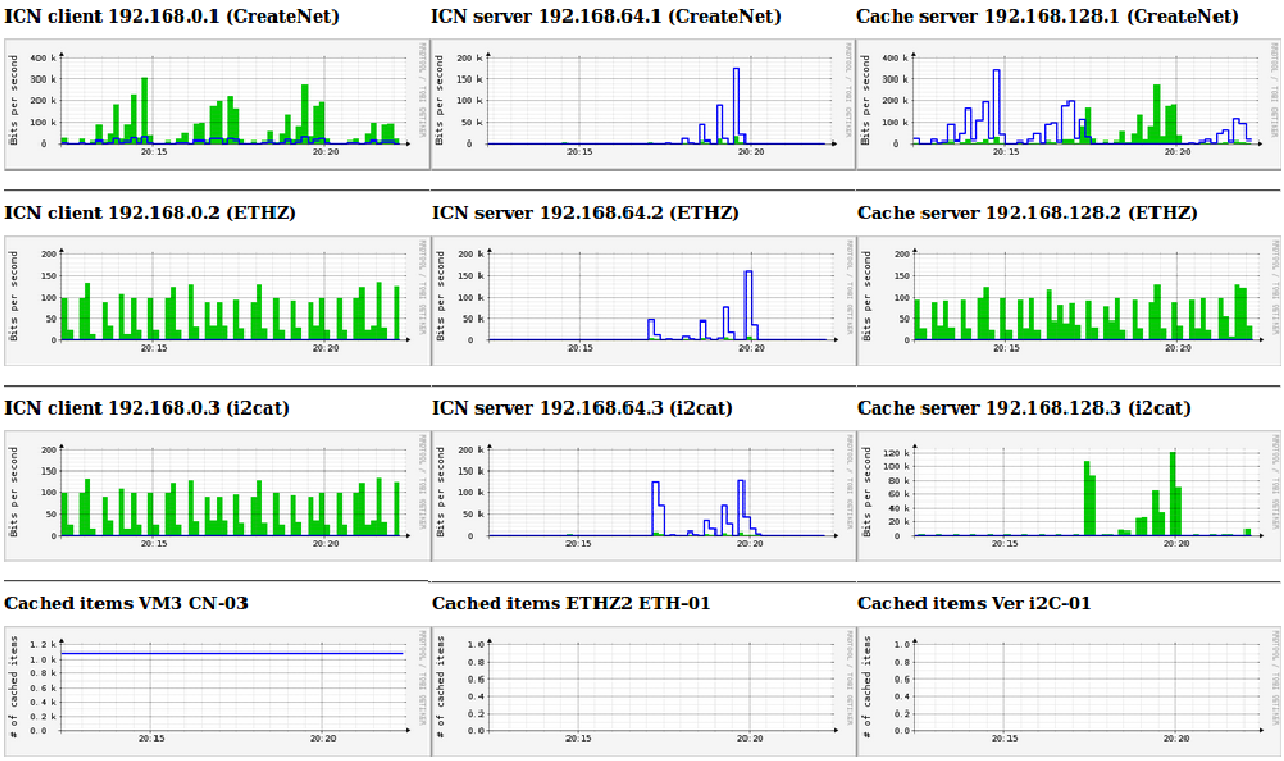


Figure 12 – Experiment 2, basic TBF, maximum number of cached entries reached

We repeated the experiment with the "advanced" TBFF strategy, to verify that we are able to make a better use of caching resources. According to the TBFF strategy, the contents coming from the local ICN server in CreateNet are not cached in the CreateNet Cache Server. The steady state of the experiment is reported in Figure 13. The ICN client in CreateNet is using the same request patterns, but the number of cached items in the cache is now around 700, below the limit imposed by the maximum number of entries in the OpenFlow switch flow table. It can be seen that the local ICN server in CreateNet is still loaded, as the contents that it provides has not been cached. On the other hand, the load on the remote ICN servers in ETHZ and i2Cat island has been reduced to zero (note the different scale of the graphs, we are only seeing some background traffic in the remote ICN servers).

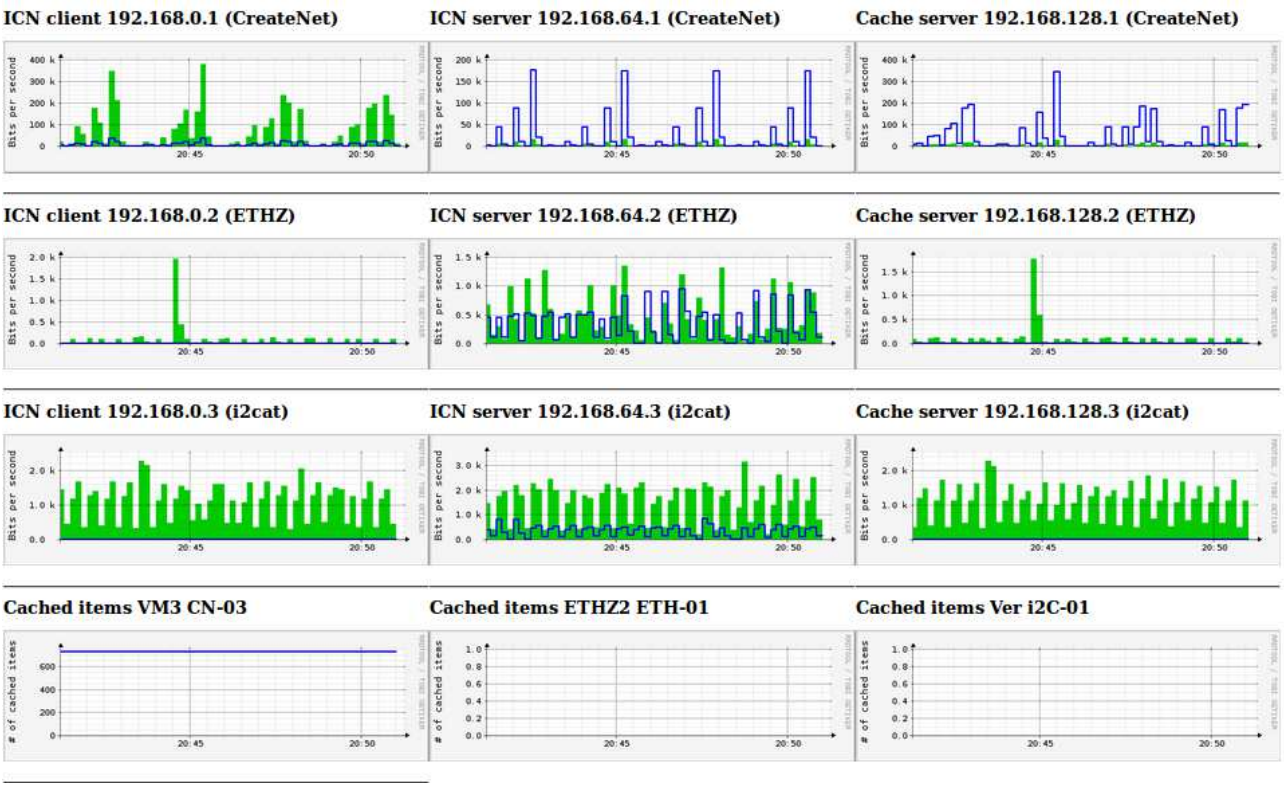


Figure 13 – Experiment 2, advanced TBFF caching strategy

Finally, we report a third experiment to validate the defined mechanisms using a dynamic request pattern. This experiment will show the entry expiration mechanism implemented in the OpenFlow switches to make room to new entries when the old ones are not used. We defined a dynamic request pattern so that the cache resources are not enough to cache all the contents, but are capable to cache the contents in the "active" set of requests. Therefore we need to make room to new contents and remove the entries related to "old" unused contents. Figure 14 reports the results of this third experiment. The ICN client in CreateNet island start performing a set of requests towards the ICN server in ETHZ island (vertical red lines at time T1). The requests are initially served by the remote ICN server, in the mean time they are cached by the local Cache Server as it can be seen from the increasing number of cached items. The ICN client repeats the same set of requests for three times, now the requests are server by the Cache Server and the ICN server load goes to zero. Then at time T2 (second vertical red lines) the client starts requesting a different set of requests, i.e. the active set of requests is changing. The new contents are requested to the remote ICN server, as they are not in the Cache Server. The Cache Server starts caching the new contents and the cached item number increases. After a short while, the entries related to the old contents start to expire, reducing the number of cached items. After a transient phase, the number of cached items is the same as before (because in our request pattern the second active set has an identical size to the first active set). The requests are now forwarded to the local cache server and the load on the remote server goes again to zero as desired.

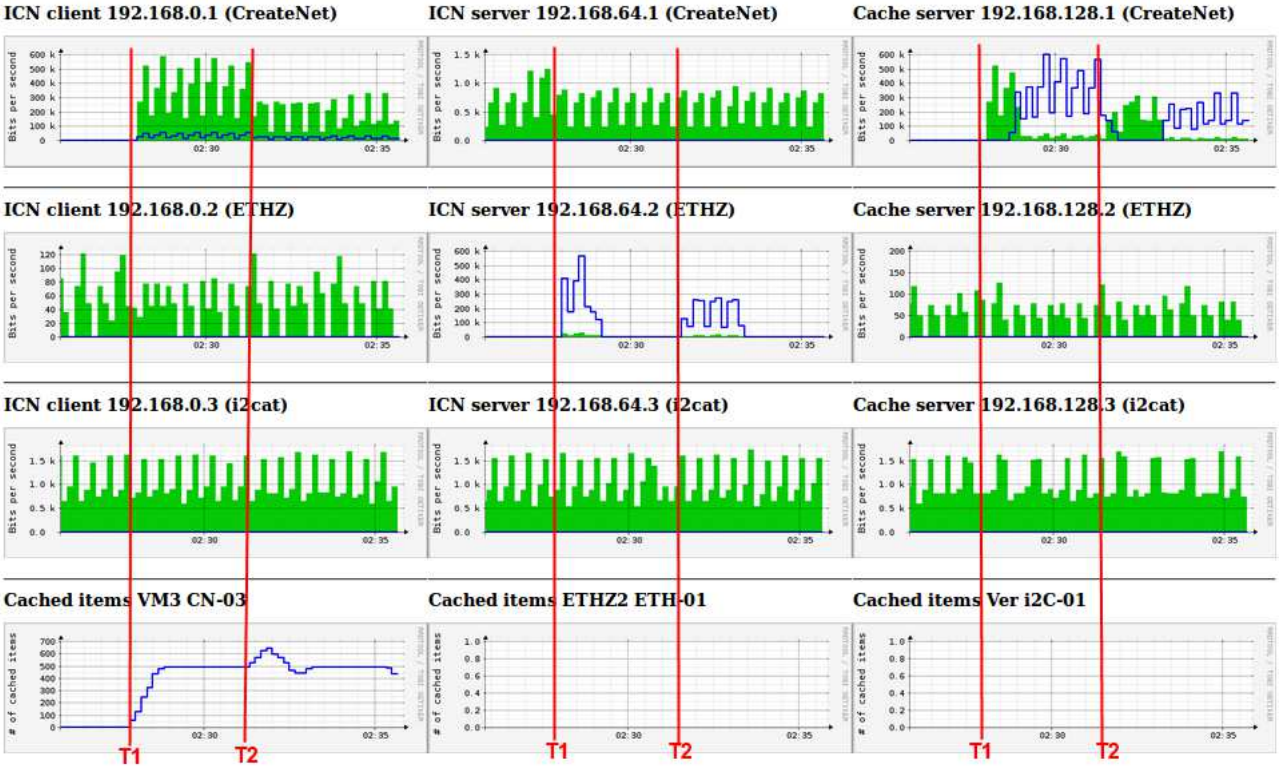


Figure 14 – TBF/TBFF under a dynamic request pattern

7 OpenFlow extendible signaling channel

Software Defined Networking and the OpenFlow protocol have been conceived and designed in order to foster innovation. The experimentation of new disruptive ideas is facilitated by the “openness” of the switch and by its capability to modify the behavior with matches and actions.

On the other hand, the OpenFlow protocol itself is difficult to modify, both at a design level and at an implementation level. In fact, OpenFlow is a binary protocol and in order to define new messages, a rather complex definition is needed. Typical implementations in switches aim at efficiency and do not facilitate the addition of new features.

The extension mechanism that was included in the definition of the OpenFlow protocol is based on the so called “vendor message” (since renamed experimenter message). This message provides the means to extend the protocol with new features. We relied on this message and defined a generic container that can carry textual messages (we will use JSON encoding) of arbitrary semantic meaning between the Controller and the OpenFlow switches (and vice versa).

We have implemented this for Open vSwitch and for the Floodlight controller. Using this proposed generic signaling channel between the controller and the switches drastically simplifies the introduction of new messages. For this reason, the proposed generic signaling channel is a useful tool for the implementation of the “long term” solution for the integration of ICN and SDN.

The definition of the Vendor extension message on OpenFlow 1.0 is the following

```
/* Vendor extension. */
struct ofp_vendor_header {
struct ofp_header header; /* Type OFPT_VENDOR. */
uint32_t vendor; /* Vendor ID:
                  * - MSB 0: low-order bytes are IEEE OUI.
                  * - MSB != 0: defined by OpenFlow
                  * consortium. */
/* Vendor-defined arbitrary additional data. */
};
OFP_ASSERT(sizeof(struct ofp_vendor_header) == 12);
```

In the Open vSwitch implementation, we define the field values in our message as follows:

```
uint32_t vendor = 0xAABBCCDD
```

the first 4 bytes in the vendor-defined arbitrary area represent a message code, and we have defined it as follows:

```
public static final int GEN_OPEN_MSG = 10
```

In the Floodlight implementation, we define a new class for our “vendor” extension:

```
public class OFExperimVendorData implements OFVendorData {
   public static final int EXP_VENDOR_ID = 0xAABBCCDD;
```

We set the message code for our open messages as follows:

```
/**
 * The data type value for an OpenMsg
 */
public static final int EXPERIM_ANY_MSG = 10;
```

We have also added a new method to the class that represent an OpenFlow switch in Floodlight:

```
public int sendOpenMsg(String message) {
```

Creating a new message and sending it to the OpenFlow switch is now easy as shown in the code snipped below:

```
JSONObject json_message= new JSONObject();
json_message.put("type", "HelloRequest");
((OFSwitchImpl)iofSwitch).sendOpenMsg(json_message.toString());
```


8 References

[1] S. Salsano (editor) “EXOTIC final architecture and design”, Deliverable D9.1, Project FP7 258365 “OFELIA”

[2] S. Salsano (editor) “EXOTIC evaluation plan and methodology”, Deliverable D9.2, Project FP7 258365 “OFELIA”

[3] CCNx project web site: www.ccnx.org

[4] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, L. Veltri, “Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed”, Computer Networks, Volume 57, Issue 16, 13 November 2013, Pages 3207-3221, ISSN 1389-1286

[5] FlowVisor Home Page - http://onlab.us/flowvisor.html

[6] “Tobi Oetiker's MRTG - The Multi Router Traffic Grapher” http://oss.oetiker.ch/rrdtool/

[7] “About RRD” http://oss.oetiker.ch/rrdtool/

[8] “Floodlight REST API”, http://www.openflowhub.org/display/floodlightcontroller/Floodlight+REST+API

9 Appendix: detailed setup instructions for the experiment

```
##############################################
####### OFELIA ICN over SDN testbed  #########
##############################################

this file is publicy available on dropbox at URL:
https://www.dropbox.com/s/u0a6v1umzos69g2/alien-ofelia-testbed.txt

#############################################################################
#####  Addresses for the demo in slices alien-ofelia-2/ basic-slice ######
#############################################################################
+-------------------------------------------------------------------------+
|       host            |  control IP  |  experim. IP | exp. VLAN |server |
|controller             | 10.216.33.93 |              |           | vm2   |
|server1                | 10.216.33.57 | 192.168.1.8  | eth1.200  | vm2   |
|cacheser1              | 10.216.33.58 |              | eth1      | vm2   |
|client1                | 10.216.33.56 | 192.168.1.23 | eth1.200  | vm1   |
|management             | 10.216.33.91 |              |           | vm1   |
+-------------------------------------------------------------------------+

NB the controller and the openflow resources are in slice alien-ofelia-2
server cacheser client1 management are is in basic-slice

Topology

+---+                 +----+                     +----+                  +---+
|vm2|-eth1<--->port12-|sw02|-port 25<--->port 25-|sw01|-port 12<--->eth1-|vm1|
+---+                 +----+                     +----+                  +---+

this part is optional but it has been added to the flowspace
                        | port 26 of switch 2
                        ^
                        | port 25
+---+                 +----+
|vm3|-eth1<--->port12-|sw03|
+---+                 +----+

#############################################################################
#####  Addresses for the demo in slice multisite1                    ######
#############################################################################

management 10.216.33.91  (in slice basic-slice)
multisitecontroller 10.216.33.109

vm3client1 10.216.33.113 192.168.0.1
vm3cache1  10.216.33.114 192.168.128.1
vm1server1 10.216.33.102 192.168.64.1
et2client1 10.216.9.21 192.168.0.2
et2cache1  10.216.9.31 192.168.128.2
et3server1 10.216.10.45 192.168.64.2
verclient1 10.216.12.179 192.168.0.3
vercache1  10.216.12.202 192.168.128.3
rodserver1 10.216.12.242 192.168.64.3

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Installing the client on the ofelia testbed
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

# apt-get install vlan

edit /etc/rc.local adding
sh /root/vlan_conf.sh
(before exit 0 !!)

vim /etc/rc.local

(previously the script was in other folders such as
sh /ofelia/users/stefanosalsano2/vlan_conf.sh)

vim vlan_conf.sh
```



```
----------<start of file vlan_conf.sh>
#!/bin/bash
#uncomment following lines for all vm configuration
#cache server
#IP_ADDR="192.168.1.248"
#icn client
IP_ADDR="192.168.1.1"
#icn server
#IP_ADDR="192.168.1.8"
ETH="eth1"
VLAN="3001"

modprobe 8021q
vconfig add $ETH $VLAN
ip addr add $IP_ADDR/24 dev $ETH.$VLAN
ip link set $ETH up
ip link set $ETH.$VLAN up
----------<end of file vlan_conf.sh>

chmod +x vlan_conf.sh

---------------------------------------
apt-get update
apt-get install libpcap-dev

<--- DOWNLOAD FROM GIT REPOSITORY
cd /root
git clone https://github.com/StefanoSalsano/alien-ofelia-conet-ccnx
cd alien-ofelia-conet-ccnx
--->

<--- DOWNLOAD FROM .tgz (DEPRECATED)
wget https://www.dropbox.com/s/5uqhwhexpkfowjm/ccnx071-conet-20130510.tgz
tar zxvf ccnx071-conet-20130510.tgz
cd ccnx071-conet-20130510
--->

vim csrc/include/conet/conet.h

check these parameters and set them correctly
define CONET_IFNAME "eth1.16"
define CONET_VLAN_ID 200
check that IS_CLIENT is set

NEW VERSION
vim csrc/include/conet/conet.h
check these parameters and set them correctly
define CONET_VLAN_ID 200
check that IS_CLIENT is set

vim conet.conf
check these parameters and set them correctly
if_name = eth1.16


./mymake

if there is an error in /root/alien-ofelia-conet-ccnx/javasrc
cd /root/alien-ofelia-conet-ccnx/javasrc
make -d
if it complains that ../csrc/conf.mk is older than Makefile
just do touch ../csrc/conf.mk

if it exits without apparant reasons (may be compiling javasrc folder)... it could be that memory is
over
check with top in a second shell if the memory goes 100% during compilation
check with swapon -s if you have swap space
follow instructions at http://www.thegeekstuff.com/2010/08/how-to-add-swap-space/ to add swap space
# dd if=/dev/zero of=/root/myswapfile bs=1M count=1024
# chmod 600 /root/myswapfile
# mkswap /root/myswapfile
# swapon /root/myswapfile


when installing from .tgz there is a problem with symbolic links in ccnx071-conet/apps

cd ccnx071-conet/apps
I have manually deleted the file ccnx071-conet/apps/include
rm include
```

```
ln -s ../csrc/generic.mk generic.mk
ln -s ../csrc/conf.mk conf.mk
ln -s ../csrc/subr.mk subr.mk

./mymake

if there are problems, please check also CCNx library prerequisite at
https://www.ccnx.org/wiki/CCNx/InstallingCCNx

you may want to change .bashrc for root so that it changes dir to
/root/alien-ofelia-conet-ccnx

vim /root/.bashrc
add the following line:
cd /root/alien-ofelia-conet-ccnx


in order to simplify the startup of remote console we allowed su without password for any needed
username:
* first time on a new machine

groupadd nopw && usermod -a -G nopw username  (replace with your username!!!)

then edit /etc/pam.d/su
vim /etc/pam.d/su

# Uncomment this if you want wheel members to be able to
# su without a password
# auth       sufficient pam_wheel.so trust
auth       sufficient pam_wheel.so group=nopw root_only trust

(no need to reboot!)

* adding a new user to a machine that has already been configured
usermod -a -G nopw username  (replace with your username!!!)



<--
(optional) running X applications
installl kate (graphical text editor): apt-get install kate
when asked about overridind your pam.d config file you can answer yes
but then check that /etc/pam.d/su includes the change made above

ssh -X utente@remote_IP
$ xauth list $DISPLAY
$ xauth list
You'll get something like
somehost.somedomain:10 mit-magic-cookie-1 4d22408a71a55b41ccd1657d377923ae

Then, after having done su just copy'n-paste the output of the above 'xauth list' onto 'xauth add'
$ su
# xauth add somehost.somedomain:10 MIT-MAGIC-COOKIE-1 4d22408a71a55b41ccd1657d377923ae

# kate &
-->


SNMP on the managed side:

apt-get install snmpd

(http://www.debianhelp.co.uk/snmp.htm)

in order to allow snmp monitoring on interfaces (for icn_server icn_client cache_server) modify
/etc/snmp/snmpd.conf as follows
#agentAddress  udp:127.0.0.1:161
agentAddress udp:161

#rocommunity public  localhost
#rocommunity public  default    -V systemonly
rocommunity public  10.216.0.0/16
/etc/init.d/snmpd restart


test with :
```



```
#snmpwalk IP_ADDRESS -c public -v1
as this does not work:
#snmpwalk localhost -c public -v1

modify /etc/snmp/snmp.conf as follows
# loaging them by commenting out the following line.
#mibs :

Install Nagios Plugin and nrpe Installation of NRPE (Nagios Remote Plugin Executor), follow instruction on http://www.howtoforge.com/icinga-monitoring-solution-installation-and-configuration-on-centos-p2
Installing Nagios Plugin and nrpe in Ubuntu 192.168.1.40

hint: cd /root before downloading with wget the Nagios plugin and the nrpe

pay attention at this command: apt-get install openssl libssl-dev (add a blank space between openssl and libssl
./configure is missing before make all
you have to add user nagios with password nagios2012 before launching make install-plugin

# tar -zxvf nrpe-2.12.tar.gz
# cd nrpe-2.12/
# apt-get install openssl libssl-dev
# ./configure
# make all

#adduser nagios

# make install-plugin
# make install-daemon
# make install-daemon-config
# apt-get install xinetd
# make install-xinetd


°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Run the icn client
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
check if ccn is running
ps ax | grep ccn

kill -9 <pid of ccnd>

cd /root/alien-ofelia-conet-ccnx
./startclient_loop


°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Installing the icn server
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

repeat the same steps as above for the client
check for the correct eth interface !!

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Run the icn server
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
check if ccn is running
ps ax | grep ccn

kill -9 <pid of ccnd>

cd /root/alien-ofelia-conet-ccnx
./singleserver

verify that the ccn is running
ps ax | grep ccn



°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Installing a cache server
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

you may want to change .bashrc for root so that it changes dir to
/root/alien-ofelia-conet-ccnx/cache_server/

vim /root/.bashrc
add the following line:
```

```
cd /root/alien-ofelia-conet-ccnx/cache_server/

in order to simplify the startup of remote console we allowed su without password for any needed
username:
* first time on a new machine

# groupadd nopw && usermod -a -G nopw username  (replace with your username!!!)

then edit /etc/pam.d/su
vim /etc/pam.d/su

# Uncomment this if you want wheel members to be able to
# su without a password
# auth       sufficient pam_wheel.so trust
auth       sufficient pam_wheel.so group=nopw root_only trust

(no need to reboot!)

# apt-get update

install and configure VLANs as for client
CAREFULLY CHECK VLAN IS, AP ADDRESS AND INTERFACE NAME TO BE USED

# apt-get install vlan

edit /etc/rc.local adding
sh /root/vlan_conf.sh

vim /etc/rc.local

(previously the script was in other folders such as
sh /ofelia/users/stefanosalsano2/vlan_conf.sh)

vim vlan_conf.sh

----------<start of file vlan_conf.sh>
#!/bin/bash
#uncomment following lines for all vm configuration
#cache server
IP_ADDR="192.168.1.218"
#icn client
#IP_ADDR="192.168.1.1"
#icn server
#IP_ADDR="192.168.1.8"
ETH="eth1"
VLAN="16"

modprobe 8021q
vconfig add $ETH $VLAN
ip addr add $IP_ADDR/24 dev $ETH.$VLAN
ip link set $ETH up
ip link set $ETH.$VLAN up
----------<end of file vlan_conf.sh>

chmod +x vlan_conf.sh

apt-get install libpcap-dev

<--- DOWNLOAD FROM GIT REPOSITORY
cd /root
git clone https://github.com/StefanoSalsano/alien-ofelia-conet-ccnx
cd alien-ofelia-conet-ccnx
./configure
--->

<--- DOWNLOAD FROM .tgz (DEPRECATED)
wget https://www.dropbox.com/s/5uqhwhexpkfowjm/ccnx071-conet-20130510.tgz
tar zxvf ccnx071-conet-20130510.tgz
cd ccnx071-conet-20130510
--->

vim csrc/include/conet/conet.h

check these parameters and set them correctly
define CONET_VLAN_ID 200
check that IS_CACHE_SERVER is set
```



```
vim cache_server/conet.conf



cd cacheServer
./make_all.sh

if there is an error in /root/alien-ofelia-conet-ccnx/javasrc
cd /root/alien-ofelia-conet-ccnx/javasrc
make -d
if it complains that ../csrc/conf.mk is older than Makefile
just do touch ../csrc/conf.mk




apt-get install valgrind
(valgrind is a C debug tool)

edit
vim /root/alien-ofelia-conet-ccnx/cache_server/go_listener.sh

replace the IP and Mac address of the cache server (not the one on eth0, the one on the interface
ethx.YY)
(they need to be consistent with conetcontroller.conf in the controller)

(Note that the go_listener.sh script changes the LD_LIBRARY_PATH in order to dynamically load some
libraries)

install snmp on cache server
see SNMP on the managed side: for the icn client

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Run the cache server
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
cd /root/alien-ofelia-conet-ccnx/cache_server
./go_listener.sh


°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Installing the controller
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

currently we have two development versions of the controller:
the first version does not include the floodlight sources (floodlight is included as a jar)
the second version includes the floodlight sources

VERSION ONE - NOT INCLUDING FLOODLIGHT SOURCES
# cd root
# git clone https://github.com/StefanoSalsano/alien-ofelia-controller
# cd alien-ofelia-controller
# make all

edit conetcontroller.conf
change VLAN_ID

# ./start.sh (this will start in background and with no output)

if needed stop.sh, restart.sh are available

you can launch interactively the controller as follows
java      -cp      lib/conetcontroller.jar:lib/conet.jar:lib/json-simple-1.1.1.jar:lib/floodlight.jar
net.floodlightcontroller.core.Main


VERSION TWO - INCLUDING FLOODLIGHT SOURCES
your-homedir$ git clone https://github.com/StefanoSalsano/my-floodlight/
your-homedir$ cd my-floodlight
my-floodlight$ git checkout 0.90

in order to compile it
my-floodlight$ ant

edit conetcontroller.conf
change VLAN_ID
```

```
in order to run it:
my-floodlight$ java -jar target/floodlight.jar

----


check if controller is running
   * ps ax | grep java

look at controller output log in in real time
VERSION ONE - NOT INCLUDING FLOODLIGHT SOURCES
   * # tail -f /root/alien-ofelia-controller/log/conetcontroller.log
VERSION TWO - INCLUDING FLOODLIGHT SOURCES
   * # tail -f /root/my-floodlight/log/conetcontroller.log



if you are setting up eclipse in your development machine
follow the instructions for Setting up eclipse in
http://www.openflowhub.org/display/floodlightcontroller/Installation+Guide

your-homedir$ cd my-floodlight
ant eclipse


°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Installing the management server
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
NB see also "Running the management server after a reboot" below

apt-get install snmp scli tkmib
apt-get install snmpd

apt-get install snmp-mibs-downloader

**************************************************
icinga installation

(NB the main reason to have icinga is to have the plugin that periodically checks the controller for
the number of cached items)

install icinga
first install xammp
http://prnawa.wordpress.com/2012/01/15/how-to-run-xampp-on-64-bit-linux/
apt-get install ia32-libs

http://www.howtoforge.com/icinga-monitoring-solution-installation-and-configuration-on-centos
pay attention with copy paste because hyphen '-' may be copied as a different char

I used icinga 1.7.4 rather then 1.2.1

#cd /root
#wget http://sourceforge.net/projects/icinga/files/icinga/1.7.4/icinga-1.7.4.tar.gz/download
#mv download icinga-1.7.4.tar.gz
#tar zxvf icinga-1.7.4.tar.gz
#cd icinga-1.7.4
#  ./configure  --prefix=/opt/icinga  --with-icinga-user=daemon  --with-icinga-group=daemon  --with-
httpd-conf=/opt/lampp/etc

Note: please make sure you do not get any error while compiling. If you are getting errors make sure
the required packages are installed.


FROM http://docs.icinga.org/1.7/en/quickstart-icinga.html
 #> apt-get install apache2 build-essential libgd2-xpm-dev
 #> apt-get install libjpeg62 libjpeg62-dev libpng12 libpng12-dev
 #> apt-get install snmp libsnmp5-dev
BUT MAYBE NOT ALL ARE NEEDED AS WE ALREADY HAVE XAMPP

you may try without and you will likely find this as output of the configure command:
------------------------------------------------------------
*** GD, PNG, and/or JPEG libraries could not be located... *********

Boutell's GD library is required to compile the statusmap, trends
and histogram CGIs.  Get it from http://www.boutell.com/gd/, compile
```



```
it, and use the --with-gd-lib and --with-gd-inc arguments to specify
the locations of the GD library and include files.

NOTE: In addition to the gd-devel library, you'll also need to make
      sure you have the png-devel and jpeg-devel libraries installed
      on your system.

NOTE: After you install the necessary libraries on your system:
      1. Make sure /etc/ld.so.conf has an entry for the directory in
         which the GD, PNG, and JPEG libraries are installed.
      2. Run 'ldconfig' to update the run-time linker options.
      3. Run 'make clean' in the Icinga distribution to clean out
         any old references to your previous compile.
      4. Rerun the configure script.

NOTE: If you can't get the configure script to recognize the GD libs
      on your system, get over it and move on to other things.  The
      CGIs that use the GD libs are just a small part of the entire
      Icinga package.  Get everything else working first and then
      revisit the problem.  Make sure to check the icinga-users
      mailing list archives for possible solutions to GD library
      problems when you resume your troubleshooting.
-------------------------------------------------------------

I did not compile the source from http://www.boutell.com/gd/, rather I followed the guide at
http://www.unix.com/unix-linux-applications/128614-gd-installation-guide-asking.html Debian/Ubuntu:
#apt-get install libgd2-xpm
To get the development package: Debian/Ubuntu:
#apt-get install libgd2-xpm-dev
To get the PHP package: Debian/Ubuntu:
#apt-get install php5-gd


# make all
# make install
# make install-init
# make install-config
# make install-commandmode
# make install-webconf

# cd /opt/lampp/etc/
# vim /opt/lampp/etc/httpd.conf
add at the end: Include etc/icinga.conf

# chmod 777 /opt/
# chmod 777 /opt/icinga/
# chmod 777 /opt/icinga/var/
# chmod 777 /opt/icinga/var/rw/
# chmod 777 /opt/icinga/var/rw/icinga.cmd

#cd /opt/lampp/bin
#/opt/lampp/bin/htpasswd -c /opt/icinga/etc/htpasswd.users icingaadmin

check configuration before launching
#/opt/icinga/bin/icinga -v /opt/icinga/etc/icinga.cfg

(it will check the default configuration)

launch icinga in the default configuration (later we will install our configuration files and will
restart icinga)
#/opt/icinga/bin/icinga -d /opt/icinga/etc/icinga.cfg

xampp splash page
http://ip-address

icinga home page
http://ip-address/icinga
login icingaadmin

************************************************
setup of mrtg and mrtg-rrd

mrtg: apt-get install mrtg
(answer NO to have the file readable by mrtg-rrd)

apt-get install mrtg-rrd
apt-get install librrds-perl
apt-get install rrdtool
```

```
configuring mrtg
create folder /home/mrtg/cfg/ as root

mrtg.cfg sample files are available on dropbox
barcelona: https://www.dropbox.com/s/whugvusqvaj1kx9/mrtg-i2cat.cfg
trento: wget https://www.dropbox.com/s/25p05ueot4qumat/mrtg-trento.cfg
to be renamed after download:
mv mrtg-trento.cfg mrtg.cfg
mv mrtg-i2cat.cfg mrtg.cfg

copy it into folder /home/mrtg/cfg/
edit for all the different interfaces (very tedious...)

the Target keyword you tell mrtg what it should monitor. The Target keyword takes arguments in a
wide range of formats:
We are using interfaces by IP:
Target[10.216.12.86_10.216.12.86]: /10.216.12.86:public@10.216.12.86:
<br>or we are using interfaces by port index: Target[icnclient_eth1]: 3:public@10.216.33.56:

see http://oss.oetiker.ch/mrtg/doc/mrtg-reference.en.html (target configuration)

in order to check the interface numbers run from the management server
snmpwalk 10.216.12.84 -c public -v1 | grep ifName
snmpwalk 10.216.33.113 -c public -v1 | grep ifName


with the IP address of the monitored node

RUNNING MRTG
#kill mrtg if it is running ps ax | grep mrtg
#/usr/bin/mrtg --user=nobody --group=nogroup /home/mrtg/cfg/mrtg.cfg

--------------------------------------------------------------------
ERROR: Mrtg will most likely not work properly when the environment
       variable LANG is set to UTF-8. Please run mrtg in an environment
       where this is not the case. Try the following command to start:

#env LANG=C /usr/bin/mrtg --user=nobody --group=nogroup /home/mrtg/cfg/mrtg.cfg
--------------------------------------------------------------------

# cd /var/lock
# mkdir mrtg
# chown -R nobody:nogroup mrtg

# cd /opt/lampp/htdocs/
# mkdir mrtg
# chown -R nobody:nogroup mrtg

# cd /home/
# chown -R nobody:nogroup mrtg

# cd /var/lib
# chown -R nobody:nogroup mrtg

move the mrtg icons in the right folder
cp /usr/share/mrtg/*.png /opt/lampp/htdocs/mrtg/

edit /etc/mrtg-rrd.conf and replace the existing file with /home/mrtg/cfg/mrtg.cfg

copy the cgi bin
copy /usr/lib/cgi-bin/mrtg-rrd.cgi into /opt/lampp/cgi-bin/
cp /usr/lib/cgi-bin/mrtg-rrd.cgi /opt/lampp/cgi-bin/

test with
http://10.216.12.85/cgi-bin/mrtg-rrd.cgi/  (barcellona)
http://10.216.33.91/cgi-bin/mrtg-rrd.cgi/  (trento)

#cd /opt/lampp/cgi-bin/
(only if updating:)#rm mrtg-rrd2.cgi
#wget https://www.dropbox.com/s/nu8nm8ebh3f4l0e/mrtg-rrd2.cgi
#chmod +x mrtg-rrd2.cgi

set up the configuration for i2cat or trento
#vim mrtg-rrd2.cgi
my $global_conf = "i2cat";
#my $global_conf = "trento";
```



```
------> perl module prerequisite for myrrd.pl
#mkdir /root/perl
#cd /root/perl

search WWW::Curl::Easy on http://search.cpan.org/

#wget http://search.cpan.org/CPAN/authors/id/S/SZ/SZBALINT/WWW-Curl-4.15.tar.gz
#tar zxvf WWW-Curl-4.15.tar.gz
#cd WWW-Curl-4.15

apt-get install libcurl4-gnutls-dev

#perl Makefile.PL
#make
#make test
#make install

search common::sense on http://search.cpan.org/
#wget http://search.cpan.org/CPAN/authors/id/M/ML/MLEHMANN/common-sense-3.72.tar.gz
#tar zxvf common-sense-3.72.tar.gz
#cd common-sense-3.72
#perl Makefile.PL
#make
#make test
#make install

search JSON-XS on http://search.cpan.org/
#wget http://search.cpan.org/CPAN/authors/id/M/ML/MLEHMANN/JSON-XS-2.34.tar.gz
#tar zxvf JSON-XS-2.34.tar.gz
#cd JSON-XS-2.34
#perl Makefile.PL
#make
#make test
#make install


#mkdir /home/daemon/
#mkdir /home/daemon/myscripts
#cd /home/daemon/myscripts
#wget https://www.dropbox.com/s/k4nj6e7oyfe1ybr/myrrd.pl
#chmod +x myrrd.pl ; chown -R daemon:daemon /home/daemon

#vim myrrd.pl
change configuration for i2cat / trento

#mkdir /opt/lampp/htdocs/mrtg/controller_
#chown nobody:nogroup /opt/lampp/htdocs/mrtg/controller_
#chmod 777 /opt/lampp/htdocs/mrtg/controller_
#chmod 777 /opt/lampp/htdocs/mrtg

the cached item RRD file needs to be manually created
#/home/daemon/myscripts/myrrd.pl -i 10 -create /opt/lampp/htdocs/mrtg/controller_/cacheditems.rrd
#chown nobody:nogroup /opt/lampp/htdocs/mrtg/controller_/cacheditems.rrd
#chmod 777 /opt/lampp/htdocs/mrtg/controller_/cacheditems.rrd


in the management server 10.216.12.85 we also have
#mkdir /opt/lampp/htdocs/mrtg/controller255_
#chown nobody:nogroup /opt/lampp/htdocs/mrtg/controller255_
#chmod 777 /opt/lampp/htdocs/mrtg/controller255_
#chmod 777 /opt/lampp/htdocs/mrtg
#/home/daemon/myscripts/myrrd.pl -i 10 -create /opt/lampp/htdocs/mrtg/controller255_/cacheditems.rrd
#chown nobody:nogroup /opt/lampp/htdocs/mrtg/controller255_/cacheditems.rrd
#chmod 777 /opt/lampp/htdocs/mrtg/controller255_/cacheditems.rrd



mkdir multisite
cd multisite
mkdir 02.08.02.08.00.00.00.03
mkdir 02.00.00.00.00.00.00.01
mkdir 00.10.00.00.00.00.00.01

mkdir multi12cat
cd multi12cat
mkdir 02.00.00.00.00.00.00.01
```

```
...

mkdir i2cat
cd i2cat
mkdir 00.10.00.00.00.00.00.01
...

/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/multisite/02.08.02.08.00.00.00.03/cacheditems.rrd
/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/multisite/02.00.00.00.00.00.00.01/cacheditems.rrd
/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/multisite/00.10.00.00.00.00.00.01/cacheditems.rrd

/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/multi12cat/02.00.00.00.00.00.00.01/cacheditems.rrd
/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/multi12cat/00.10.00.00.00.00.00.01/cacheditems.rrd

/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/i2cat/00.10.00.00.00.00.00.01/cacheditems.rrd
/home/daemon/myscripts/myrrd.pl                    -i                    10                    -create
/opt/lampp/htdocs/mrtg/i2cat/00.10.00.00.00.00.00.03/cacheditems.rrd

chown -R nobody:nogroup multisite
chmod -R 777 multisite
chown -R nobody:nogroup multi12cat
chmod -R 777 multi12cat
chown -R nobody:nogroup i2cat
chmod -R 777 i2cat

        $switch_human_readable{'02.08.02.08.00.00.00.03'} = "CN-03, cache enabled";
        $switch_human_readable{'02.08.02.08.00.00.00.01'} = "CN-01, no cache";
        $switch_human_readable{'02.00.00.00.00.00.00.01'} = "ETH-01, cache enabled";
        $switch_human_readable{'02.00.00.00.00.00.00.03'} = "ETH-03, no cache";
        $switch_human_readable{'00.10.00.00.00.00.00.01'} = "i2C-01, cache enabled";
        $switch_human_readable{'00.10.00.00.00.00.00.03'} = "i2c-03, no cache";
        $switch_human_readable{'01.00.00.00.00.00.00.ff'} = "iM-01, no cache";

now copy the images
#cd /opt/icinga/share/images/
#wget https://www.dropbox.com/s/wcp0hhb1eyspb9a/cnit-logo.png
#wget https://www.dropbox.com/s/7fnnwua8o0shnjr/ofelia-logo.png


test with
http://10.216.12.85/cgi-bin/mrtg-rrd2.cgi/  (barcellona)
http://10.216.33.91/cgi-bin/mrtg-rrd2.cgi/  (trento)

if mrtg.cfg is changed or if something is not working well, the solution is to stop mrtg, delete the
.rrd files (e.g. in /opt/lampp/htdocs/mrtg/cacheserver_)
and then restart mrtg

************************************************
set up the experiment control page

search HTML::QuickTable on http://search.cpan.org/

#cd /root/perl
# wget http://search.cpan.org/CPAN/authors/id/N/NW/NWIGER/HTML-QuickTable-1.12.tar.gz
#tar zxvf HTML-QuickTable-1.12.tar.gz
# cd HTML-QuickTable-1.12


#cd /opt/lampp/cgi-bin/
(only if updating:) #rm icn_experim.cgi
#wget https://www.dropbox.com/s/01okpfcqlmz1zty/icn_experim.cgi
#chmod +x icn_experim.cgi

set up the configuration for i2cat or trento
#vim icn_experim.cgi
my $global_conf = "i2cat";
#my $global_conf = "trento";


test with
```



```
http://10.216.12.85/cgi-bin/icn_experim.cgi/  (barcellona)
http://10.216.33.91/cgi-bin/icn_experim.cgi/  (trento)

if there are problems, one can test it on the shell
#perl icn_experim.cgi


*******************************
icinga cfg files

#cd /opt/icinga/etc/

for i2cat
#wget https://www.dropbox.com/s/ub7tuy64rhdj8ap/icinga-cfg-files-i2cat.tar
#tar xvf icinga-cfg-files-i2cat.tar

for trento
#wget https://www.dropbox.com/s/jdr1d221co0km3f/icinga-cfgfiles-trento.tar

NB the address of the ICN capable switch is coded in the /opt/icinga/etc/objects/controller.cfg

00:10:00:00:00:00:00:05 for i2cat
02:08:02:08:00:00:00:02 for trento


if you need to recreate a tar file from all the useful cfg file:
#cd /opt/icinga/etc/
#tar     -cvf     cfgfiles.tar     icinga.cfg     objects/cache_server.cfg     objects/icn_client.cfg
objects/icn_server.cfg      objects/controller.cfg      objects/localhost.cfg      objects/switch.cfg
objects/commands.cfg

create the link to the script that updates the cached item value in the .rrd file:

#ln -s /home/daemon/myscripts/myrrd.pl /opt/icinga/libexec/check_cached_items
#cd /opt/icinga/libexec/


-----------------------------------------
 we added a (perl) plugin to perform monitoring of cached items in the controller and writing
results in the RRD database, it is /home/daemon/myscripts/myrrd.pl (it was already used above to
create the .rrd file)

in cfg_file=/opt/icinga/etc/objects/commands.cfg we added

# 'check_cached_items' command definition
# it is used to retrieve the number of cached items from the controller using RESTful interface
# and to write the data in the RRD database
define command{
        command_name    check_cached_items
        command_line    $USER1$/check_cached_items -H $HOSTADDRESS$ -s $ARG1$ $ARG2$
        }

then    we    made    a    simbolic    link    from    /opt/icinga/libexec/check_cached_items    to
/home/daemon/myscripts/myrrd.pl
(after doing su daemon, as the link should be owned by daemon:daemon)

-H $HOSTADDRESS$ works because this is executed in the context of the scripts related to a specific
host
under the icinga control
the definition of the address within /home/daemon/myscripts/myrrd.pl is overrided by this command
line parameter


command line to execute the plugin if you want to test it (should do it with the same user as
icinga, i.e. daemon)

su daemon
#/opt/icinga/libexec/check_cached_items -H 10.216.33.109 -v -s all multisite
#/opt/icinga/libexec/check_cached_items -H 10.216.12.255 -v -s all multi12cat
#/opt/icinga/libexec/check_cached_items -H 10.216.12.88 -v -s all i2cat


in the management server 10.216.33.91 we added the following service definition
in /opt/icinga/etc/objects/controller.cfg

# Define a service to check the number of cached items
define service{
        use                             local-service         ; Name of service template to use
        host_name                       controller
```

```
        service_description             CACHED_ITEMS
        check_command                   check_cached_items!all!multisite
        }

in the management server 10.216.12.85 we have
/opt/icinga/etc/objects/controller.cfg
check_command                   check_cached_items!all!i2cat
/opt/icinga/etc/objects/controller255.cfg
check_command                   check_cached_items!all!multi12cat

/opt/icinga/etc/objects/controller255.cfg needs to be added in /opt/icinga/etc/icinga.cfg


        ----------------------------------------



(NB the configuration file consider the use of icinga embedded perl interpreter
http://docs.icinga.org/latest/en/embeddedperl.html
which has not been included in the configure before make, but no problem for now)

check configuration before launching
#/opt/icinga/bin/icinga -v /opt/icinga/etc/icinga.cfg

launch icinga
#/opt/icinga/bin/icinga -d /opt/icinga/etc/icinga.cfg

in case of icinga problems, have a look at the log file
#tail -200 /opt/icinga/var/icinga.log | more

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
Running the management server after a reboot
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
i've tried to disable automatic restarting of apache
sudo update-rc.d -f  apache2 remove
not sure if it worked


manually restarting:
su
check if apache is running
#ps ax | grep apache
if apache is running, stop it
#apachectl -k stop

start xampp
#/opt/lampp/lampp start
check that lampp is running
#ps ax | grep lampp

run icinga
# /opt/icinga/bin/icinga -d /opt/icinga/etc/icinga.cfg
check that icinga is running
#ps aux | grep icinga

check if mrtg is running
ps aux | grep mrtg

in order to kill mrtg
kill -9 `cat /home/mrtg/cfg/mrtg.pid`

in any case, before starting mrtg, check that there are no lock files
#cat /home/mrtg/cfg/mrtg.pid
if present, delete it
#rm /home/mrtg/cfg/mrtg.pid

check also this folder
#ls /var/lock/mrtg
if there is any file, delete it
#rm /var/lock/mrtg/*

run mrtg
#/usr/bin/mrtg      --user=nobody      --group=nogroup      /home/mrtg/cfg/mrtg.cfg      --logging
/home/mrtg/logs/mrtg.log
check that mrtg is running
#ps aux | grep mrtg
```



```
if you get this error
ERROR: Mrtg will most likely not work properly when the environment
       variable LANG is set to UTF-8. Please run mrtg in an environment
       where this is not the case. Try the following command to start:

#  env  LANG=C  /usr/bin/mrtg  --user=nobody  --group=nogroup  /home/mrtg/cfg/mrtg.cfg  --logging
/home/mrtg/logs/mrtg.log


if mrtg does not start it may go in "out of memory", this is logged in the dmesg
in this case, try to delete all RRD files like
#rm /opt/lampp/htdocs/mrtg/icnserver_/*.rrd
#rm /opt/lampp/htdocs/mrtg/controller_/*.rrd
#rm /opt/lampp/htdocs/mrtg/icnclient_/*.rrd
#rm /opt/lampp/htdocs/mrtg/cacheserver_/*.rrd



------------------------------------------------------------------------
installing floodlight
# apt-get update
# apt-get install build-essential default-jdk ant python-dev


$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ ant
DOES NOT WORK !!

# wget http://floodlight.openflowhub.org/files/floodlight-source-0.90.tar.gz
# tar zxvf floodlight-source-0.90.tar.gz
# cd floodlight-source-0.90
# ant
#  java -jar target/floodlight.jar


------------------------------------------------------------------------

http://www.openflowhub.org/display/floodlightcontroller/Installation+Guide

setting up the floodlight on eclipse including the conet module

copy src/local and src/org into floodlight src folder

copy lib/json-simple-1.1.1.jar and lib/conet.jar into floodlight lib folder and add them to build
path

edit src/main/resources/floodlightdefault.properties

floodlight.modules = net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\
net.floodlightcontroller.jython.JythonDebugInterface,\
net.floodlightcontroller.counter.CounterStore,\
net.floodlightcontroller.perfmon.PktInProcessingTime,\
net.floodlightcontroller.topology.TopologyManager,\
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
local.conet.ConetModule

edit src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule

and remove these two lines
net.floodlightcontroller.core.test.MockFloodlightProvider
net.floodlightcontroller.core.test.MockThreadPoolService

create log folder in floodlight base folder
copy conetcontroller.conf in floodlight base folder

##################################################
##  Addresses for the multi site demo create net ##
##################################################

+-------------------------------------------------------------------------+
|       host           |  control IP  |  experim. IP | server          |
|multisitecontroller   | 10.216.33.109|              | create-net vm3  |
+-------------------------------------------------------------------------+

exp VLAN 3001
```

```
http://10.216.33.109:8080/ui/index.html
(WEB GUI not working)

*****************************************************************************
vm1server1 10.216.33.102 192.168.64.1
vm3client1 10.216.33.113 192.168.0.1
vm3cache1  10.216.33.114 192.168.128.1
et3server1 10.216.10.45  192.168.64.2
et2client1 10.216.9.21   192.168.0.2
et2cache1  10.216.9.31   192.168.128.2
rodserver1 10.216.12.242 192.168.64.3
verclient1 10.216.12.179 192.168.0.3
vercache1  10.216.12.202 192.168.128.3

switches datapath

02:08:02:08:00:00:00:01  CN-01
02:08:02:08:00:00:00:03  CN-03
01:00:00:00:00:00:00:ff  iminds
02:00:00:00:00:00:00:01  ETHZ-01
02:00:00:00:00:00:00:03  ETHZ-03
00:10:00:00:00:00:00:01  i2C-01
00:10:00:00:00:00:00:03  i2C-03


##################################################
##  Addresses for the multi site demo i2cat      ##
##################################################

*****************************************************************************
msi2c_et3server1 10.216.10.31  192.168.64.2    02:02:31:11:11:6c
msi2c_et2client1 10.216.9.38   192.168.0.2     02:02:21:11:11:81
msi2c_et2cache1  10.216.9.37   192.168.128.2   02:02:21:11:11:7e
msi2c_rodserver1 10.216.12.116 192.168.64.3    02:03:00:00:01:12
msi2c_verclient1 10.216.13.1   192.168.0.3     02:03:00:00:02:b9
msi2c_vercache1  10.216.13.0   192.168.128.3   02:03:00:00:02:b6

switches datapath

01:00:00:00:00:00:00:ff  iminds
02:00:00:00:00:00:00:01  ETHZ-01
02:00:00:00:00:00:00:03  ETHZ-03
00:10:00:00:00:00:00:01  i2C-01
00:10:00:00:00:00:00:03  i2C-03
```

