



Deliverable D6.1

Definition of Evaluation Scenarios – Phase I

Public deliverable, Version 1.3, 1 February 2011

Authors

FT

AL-BELL Werner Van Leekwijck

IBBT Tim Wauters

IDATE

N2Nsoft

UVS

FRAUNHOFER

TP

UERT

Reviewers

Abstract

This document describes a selection of technical use cases for the PHASE I emulations, based on the actual status of deliverables and algorithms worked out in other work packages.

Further, the objectives used for validation, and the requirements and platforms (middleware, OS versions, end-user devices, etc.) for the test bed are described.

© Copyright 2010 OCEAN Consortium



Project funded by the European Union under the
Information and Communication Technologies FP7 Cooperation Programme
Grant Agreement number 248775



EXECUTIVE SUMMARY

This document describes a selection of technical use cases for the PHASE I emulations, based on the actual status of deliverables and algorithms worked out in other work packages. Premium VOD or catch-up TV services are chosen as the service use cases, delivered over a network with 1 or 2 levels of hierarchical caches, using both traditional cache algorithms like LRU and LFU and novel cache algorithms developed in OCEAN. AVC and SVC based HTTP adaptive streaming will be used as delivery mechanism.

Further, the objectives used for validation, and the requirements and platforms (middleware, OS versions, end-user devices, etc.) for the test bed are described. On one hand, the performance criteria include criteria that were also taken into account in the simulation experiments in WP4 and WP5, with the aim to verify the simulation results in an emulation environment with real data files, chunks, flows, and network protocols. The most important criteria are maximization of cache hit ratio (and associated reduction of average bit rate on the network links), and the reduction of the peak bit rate on the network links, where especially the link from the origin server to intermediate caches is very important. On the other hand, specific criteria that have not been taken into account in the simulations like processor load required for running the specific caching and streaming algorithms will be investigated and compared for the different algorithms.

The IBBT Virtual Wall will be used to execute the experiments, and several functional components for HTTP adaptive streaming, caching, and emulation will be integrated.



TABLE OF CONTENTS

EXECUTIVE SUMMARY	2
TABLE OF CONTENTS	3
INTRODUCTION	4
1. TECHNICAL USE CASE SELECTION	5
1.1 Service Use Cases	5
1.2 Architecture.....	6
1.2.1 <i>Interconnection Models</i>	6
1.2.2 <i>Architectural Components</i>	6
1.2.3 <i>Content Delivery</i>	7
1.3 Caching	7
1.3.1 <i>Network Topology</i>	7
1.3.2 <i>Cache Algorithms</i>	8
1.3.3 <i>Cache Collaboration Strategies</i>	10
1.4 Adaptive Delivery	12
2. VALIDATION SCENARIOS.....	14
2.1 Experimental Set-up	14
2.2 Performance Criteria	14
2.3 Objectives and Targets	15
3. EMULATION PLATFORM	16
3.1 Virtual Wall	16
3.2 Functional Components.....	17
3.2.1 <i>HTTP Adaptive Streaming Client</i>	17
3.2.2 <i>Client Emulator</i>	18
3.2.3 <i>Network Proxy</i>	18
3.2.4 <i>Content Server, Encoder and Segmenter</i>	18
3.3 Validation Components	18
3.3.1 <i>NS-2 experiment description</i>	19
3.3.2 <i>NetBuild GUI</i>	20
3.3.3 <i>Extended experiment management GUI</i>	21
3.3.4 <i>Monitoring</i>	22
3.3.4.1 Monitoring of Hardware performance	22
3.3.4.2 Monitoring of Network Performance	22
REFERENCES	24
ACRONYMS	25



INTRODUCTION

The activities in WP6 focus on the design, construction, integration and evaluation of various experimental tools, in a two-iteration approach, in order to validate the OCEAN project concepts and prototype implementations, as well as to feed project dissemination actions.

In task 6.1 “Definition of Evaluation Scenarios”, a first activity identifies the interesting use cases for validation, based on the business and architectural requirements determined in WP2 and WP3. The actual validation of the work, in addition to the in-lab simulations performed on the single technological components in WP4 and WP5, will be performed via emulations. The emulation driven experimentations in Task 6.3, using the integrated prototype software developed in Task 6.2, will verify the scalability of the approach taken in WP4 and WP5 in a distributed hardware environment.

This document describes a selection of technical use cases for the PHASE I emulations. Based on the actual status of deliverables and algorithms worked out in other work packages, it defines the use cases for PHASE I, with the rationale for their selection. Obviously, the number of use cases selected for emulation testing is smaller than the number of cases that can be handled in simulations in the other work packages. Selection criteria include the (business) relevance of certain cases, promising simulation results, and emulation feasibility.

Further, it describes the objectives and scenarios used for validation, and the requirements and platforms (middleware, OS versions, end-user devices, etc.) for the test bed.

1. TECHNICAL USE CASE SELECTION

1.1 Service Use Cases

The selection of use cases for the Phase I emulations is based on D2.1 “Current state of the OCEAN market (service, usage, regulation)”. The five key service use cases identified for OCEAN in [1] are:

- Free short clips represent the bulk of online video traffic and generate massive volumes of traffic and major audiences. However, they represent only a limited market compared to other online services in value, as most services struggle to monetize their inventories. This is not the most attractive use case for the OCEAN architecture, with limited end-user requirements in terms of QoS and a lot of unclear business models. The rationale would be limited to a requirement for more QoS in the future, and some dissatisfaction among end-users with respect to the lack of seamless experience and the time to start.
- Streaming platforms are progressively replacing P2P consumption of (mostly infringed) video, with the advantage of immediate access. Its potential audience is also bigger than P2P, as it requires less technical skills by end-users. While used only by a third of the Internet users, the overall volumes of traffic are significant, almost in the same range as free short clips in most countries, far ahead of catch-up TV and VoD. But with a business model mostly relying on premium paid accounts for less than 10% of users and consumption of mainly infringed content for all types of users, this use case is, at first sight, not necessarily the most attractive one for OCEAN.
- Catch-up TV has become a real popular phenomenon. This strategy is deployed with some good returns by all the major broadcasters in all major countries, with content already broadcast being available on the web for a few days. With more and more consumption on PC and an extension progressively to Internet TV devices, catch-up TV will soon require additional QoS to cope with the current moderate level of satisfaction. Catch-up TV also benefits currently from a clear business model with limited rights acquisitions around TV series that allows developing sustainable business models. With the highest level of adoption among services with a real business model and room for improvement in terms of encoding levels, this use case is therefore very attractive in the context of future OCEAN services.
- Usage of premium VoD is very marginal for now, but a lot of people are already ready to pay for content in the offline world (DVD rentals or chases). Some of those users may shift towards VoD when the content catalogue will be big enough and when services will be easier to use. Current VoD services are already requiring advanced QoS, as most of the offerings are moving towards HD quality. This use case is therefore obviously very attractive for OCEAN, as it provides a sound business model and large amounts of traffic per video content around a single stream.
- Live streaming is mostly used for sports, which has some major implications in terms of QoS and QoE (fast movement, high degree of detail, etc...). Many users are still unsatisfied while providers try to offer more and more very advanced features (multi-view, multiple angles, etc...) and high encoding rates. This use case is very attractive for the development of OCEAN architectures, as sports and video are already able to generate significant revenues and offer profitable operations, both with broadcasters or sports right holder services. However, this use case also illustrates the need for huge technical requirements, especially for large-scale events attracting numerous simultaneous viewers. As the current OCEAN focus is more on

on-demand non-linear content, OCEAN may require additional technical elements to optimally support near real-time delivery.

Based on the above analysis, **Catch-up TV** and **premium VOD** are the selected use cases for Phase I emulation. They provide clear business models, and the technical requirements have been taken into account in the initial OCEAN work items and scope.

Further deliverables D2.2 “Final requirements for Open Content Aware Networks”, D2.3 “Use cases definition and market scenarios for future OCEAN services”, and D2.4 “Business models for future OCEAN architectures” will impact Phase II.

1.2 Architecture

The architecture for the Phase I emulations is based on MS3.1 “Preliminary functional OCEAN architecture specification” [2].

1.2.1 Interconnection Models

Several interconnection models of interest to OCEAN have been identified:

- **Global CDN to Telco CDN:** A Global CDN (A) is the source CDN. It has authority over the content, which it distributes also over the Telco CDN (B). The Global CDN can decide to redirect end-user request to the Telco CDN to be served.
- **CDN to cache network:** A Global CDN (A) is the source CDN. It has authority over the content, which it distributes also over the Telco CDN which has proxy servers deployed that cache content (that is marked as “allowed to be cached”). End-user request get served by the proxy servers if they contain the content, or forwarded to the Global CDN otherwise.
- **Content Provider to Telco CDN:** The Content Provider has authority over the distributed content and is connected directly to a Telco CDN. End-user requests to the Content Provider are redirected to the Telco CDN to be served.

For Phase I, an architecture in which users are connected to a **cache network** is selected. The proxies in the cache network either serve the end-users directly, or forward the request upstream to the next level of caches, or to the origin server. (The origin server can be seen as a simplification of a Global CDN acting on behalf of a content provider, or as the local content provider when the Telco is also playing to role of content provider). In this architecture there is no need for a request-routing mechanism or the (still to be defined) OCEAN open interface between Global CDN and Telco CDN. Requests are forwarded until an element has the content available to serve the end-user. Focus is on the caching algorithms and adaptive streaming behaviour.

1.2.2 Architectural Components

OCEAN has identified several key building blocks in the Content Delivery Module (CDM), residing on top of the IP layer, and responsible for content ingestion, content deployment, request routing, and content delivery. At macro level, distinction is made between Control Plane and Transfer Plane, and between Front Office and Back Office functions. Functions that are in scope for Phase I emulations are:

- **Session Control:** Responsible for receiving and processing delivery requests sent by end-users at the application level. Then, if the request is in the scope of cache policies, it forwards the requests to a Delivery Control instance. In other cases it discards the request or let it pass through without any change.

-
- **Delivery Control:** Responsible for receiving and processing delivery requests sent to it by a Session Control instance. Selection of a Delivery Transfer instance for the delivery.
 - **Delivery Transfer:** Responsible for performing requested content delivery down to the end-user terminal. It interacts with Population Control, to get directions about how to get the requested content, and transfers content to end-user terminals from local storage (Cache Storage).
 - **Content Population Control:** Responsible for managing contents inside Cache Storage (check if a request triggers a Cache-hit or a Cache-miss), track popularity for each content, and provide information useful for acquisition (Retrieve or Retrieve and Store) and deployment of contents (or chunks).
 - **Cache Storage:** Responsible for storing content files.

As there will be no Telco to Global CDN interface in Phase I, the OCEAN Content Internetworking Gateway (OCIG), and associated Network-to-Network Interface (NNI), are out of scope. The Transport Network Interface (TNI) is also out of scope for Phase I.

1.2.3 Content Delivery

The User-to-Network Interface (UNI) and Content delivery to end user terminals will be based on HTTP for Phase I. HTTP will be used in **HTTP Adaptive Streaming** mode. HTTP Adaptive Streaming can be seen as the successor of HTTP Progressive Download and recently received support (in different flavours) from Microsoft, Apple, and Adobe. Terminals will be limited to PC and TV sets.

Further deliverables D3.1 “OCEAN functional architecture and open interface specification” and D3.2 “OCEAN architecture for FTTH and DSL access networks” will impact Phase II architecture, including the open Network-to-Network Interface (NNI) specification and Telco CDN to Global CDN interconnection model.

1.3 Caching

The caching architectures, algorithms, and criteria for evaluation are based on M4.1 “Simulation Environment and Parameters for the Evaluation of Caching Algorithms” [3] and draft versions of D4.2 “Highly Dynamic and Distributed Caching – Report” [5], and D4.5 “On-line Learning Mechanisms for Distributed Caching” [6].

1.3.1 Network Topology

The network for distributing the requested objects to the users is tree-based. The origin server sits at the root of the tree and the users at the leaves. In between, at least at one level, possibly at two levels, caches are deployed. If there are two levels of caches the caches closest to the user are referred to as leaf caches, while the caches further away from the users, residing between the origin server and the leaf caches are referred to as intermediate caches. Each user is served by a primary cache (i.e., the leaf cache closest to the user in the tree). For each request of this user it is first checked whether or not this primary cache has the requested object. If this cache does not contain the requested object, the cache collaboration mechanism may ask one or more other local caches, referred to as secondary caches. Secondary caches may be intermediate caches or other leaf caches. Note that the terms “leaf” and “intermediate” cache refer to the location of the caches in the tree, i.e., to how close they are to users (leaf cache are closer than intermediate caches), while the terms “primary” and “secondary” caches pertain to the functionality of the caches, i.e., in which order they are consulted following a user request (primary caches are consulted first, secondary caches subsequently).

Both pure tree-based networks and (partly) meshed tree networks are candidate network topologies. In the latter case, the logical link between leaf caches would be implemented on a tree-like network architecture, and not necessarily imply a physical link connecting both leaf caches. From[5]:

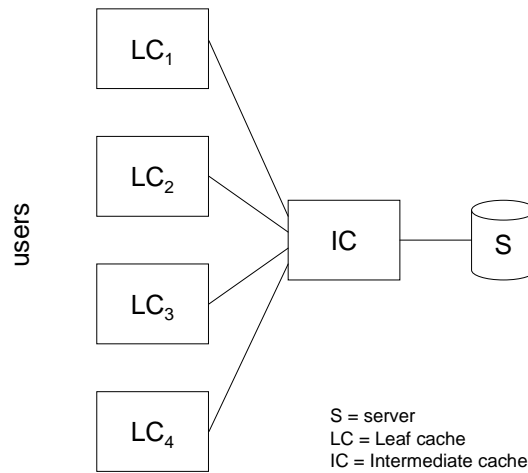


Figure 1: Example of a tree topology.

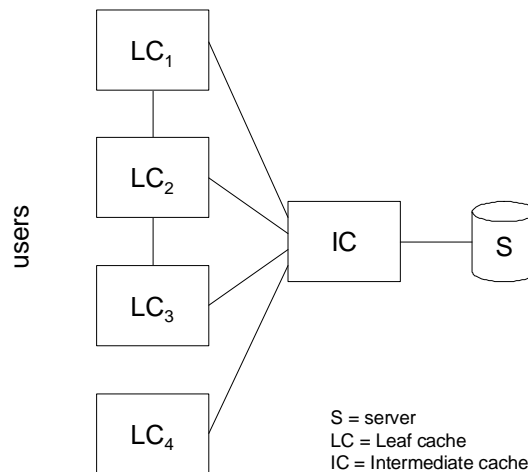


Figure 2: Example of a partly meshed topology

For the first phase emulation experiments, the structure will be limited to 2 levels of caches, as shown in Figure 1 and Figure 2.

1.3.2 Cache Algorithms

Several caching algorithms will be used in the Phase I emulation: the theoretically optimal caching strategy (at least for movie-level caching, as opposed to chunk-level caching), 2 existing and widely used caching algorithms (as benchmark, and to investigate their performance for video), and a selected set of new caching algorithms optimized for streaming content.

- **OPT (Optimal strategy):** This algorithm is an optimal caching strategy in terms of cache hit ratio. When an object is requested that is not part of the cache and the cache is full, it replaces the object in the cache whose next request occurs furthest in the future. It is impractical as it assumes perfect knowledge of future request arrivals, but serves as a benchmark.

- **LRU (Least Recently Used):** This is the most popular cache replacement policy nowadays. In theory, at each request arrival time, the least recent content is discarded and replaced by the last requested content. In practice, the cache eviction algorithm would not be run at each request, but only at certain points in time (e.g. when a certain filling of the cache is reached) and a number of items are discarded simultaneously. The filling level of the cache, and the amount of space freed are parameters that could be tuned for an optimized mode of operation. The main advantage of this algorithm is its simplicity. On the other hand, it suffers from overly attaching importance to unpopular objects. (Even an unpopular object jumps to the first rank when it is requested once and subsequently takes a long time to be evicted from the cache).
- **LFU (Least Frequently Used):** This is another popular cache replacement algorithm. It assumes that we maintain statistics of the request frequency for each object. It requires one parameter (time window for frequency measurement if a fixed window is used, or the coefficient of the moving average if this last technique is used) to be tuned and the optimal value for this parameter may depend on the characteristics of the request traffic. Basic LFU is sluggish as it attaches as much importance to ancient as to recent history (within the time window).

Several variations and completely new algorithms for streaming content are being developed in the OCEAN project, and – based on the results obtained in [5] and [6]– two or more of the following will be retained for Phase I of the emulation testing, with particular focus on applicability for **HTTP**

Adaptive Streaming specific caching:

- **Modified LRU:** Modified LRU is different from normal LRU in the fact that a requested object jumps in the cache, but to a position where it is not difficult to get the object evicted from the cache if it turns out that the object only gets very few requests. In particular, a requested multimedia object does not jump to rank 1. Instead a request results in upgrading the rank of the requested object by J positions and increasing the rank number of the objects between the new and old rank of the requested object with 1. If the requested object did not reside in the cache it jumps to a rank J positions up from the last rank and the object that had the last rank is evicted from the cache. Choosing J (considerably) smaller than the number of places L in the cache will make the modified version of LRU less nervous. (Setting J=L boils down to using pure LRU.)
- **Scoring-based Caching (SC):** SC is based on keeping a score S_k for each video k ($k=1, 2, \dots, K$). Each time video k is requested, it gets a bonus, i.e., its score is increased by an amount A, while the score of all other videos is decreased by 1. When a new video is requested for the first time its score is initialized to a value B. Moreover, the score is kept within the interval $[-C, C]$, by truncating it to these bounds if adding a value A or decreasing it by 1 would move it outside this interval. A simplified version of the algorithm re-ranks at each request time the videos based on these scores S_k . The first L ranked videos are cached. In a practical set-up, the re-ranking process has to be scheduled at particular time intervals T. A, B, C and T are parameters to be tuned.
- **Chunk –level Caching (CC):** The chunk level version of SC also maintains the values S_k in just the same way as described above. Moreover each video is segmented in M chunks and each chunk inherits the score S_k from the video it belongs to. For each chunk m of video k a value $N_{k,m}$ is maintained that accumulates the number of guaranteed hits this chunk will have knowing which videos are currently watched by the users (and assuming that no user aborts watching a video). The values $N_{k,m}$ are increased by 1 for all values of the index m, each time

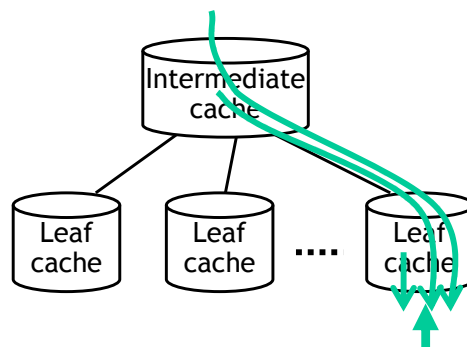
video object k is requested by a user. The value of $N_{k,m}$ is decreased by 1 after a user watching video object k has consumed chunk m .

- Predicting Least Recently Used (P-LRU): Has perfect knowledge of the future and uses LRU within a time window in the future to calculate the theoretical maximum gain of using predictions.
- Predicting Least Recently Used – Real (P-LRU-R): The P-LRU-R strategy is the prediction-based variant of LRU. Contrary to the more theoretical P-LRU algorithm, the predictions of future request arrivals are made by extrapolating each file's popularity distribution, obtained through fitting with template popularity models.
- Cache Management based on Temporal Pattern Solicitation (CMTPS): CMTPS takes into account the daily temporal pattern of solicitation when the algorithm must choose the content to be evicted from the cache. It makes the hypotheses that if an item was solicited on a time period during the previous day, it will be solicited *approximately* at the same period during the current day. Hence it will avoid to evict the content item from the cache just before the supposed solicited period, to avoid a miss. It will predict the future solicitation and will put the content in the cache just before the supposed solicited time.

1.3.3 Cache Collaboration Strategies

A cache collaboration strategy is characterized by two decisions it needs to make: first, for each request it needs to decide from which cache it will serve the request, and second, it needs to decide which cache(s) to update. Several types of collaboration strategies can be distinguished:

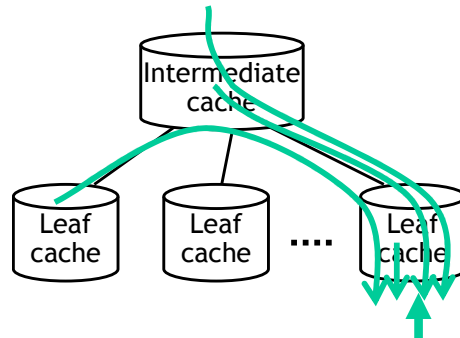
- In the case of hierarchical caches, illustrated in Figure 3, the intermediate cache never asks for assistance of the leaf caches: if a request of a user connected to the intermediate cache cannot be served from this cache the object is retrieved from the origin server. On the contrary leaf caches can ask the intermediate cache for assistance: if the requested object does not reside in the leaf cache to which the requesting user is connected, the leaf cache (after updating its ranking) first attempts to retrieve it from the intermediate cache (updating the ranking there too). If the requested object does not reside there either, it is retrieved from the origin server.



```
if(hit on LC)serve from LC
else{
  if(hit on IC)serve from IC
  else serve from origin
  update IC
}
update LC
```

Figure 3: Hierarchical caches.

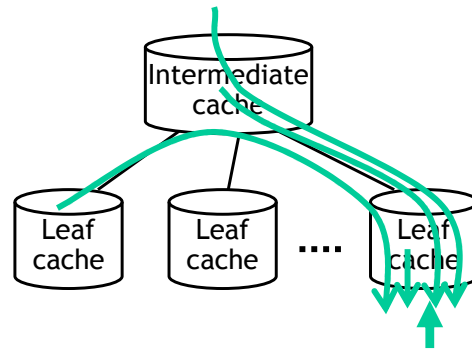
- For borrowing caches, illustrated in Figure 4, there are two cases. First, if the request is on the leaf cache to which the requesting user is connected and it does not have the requested object, all other local caches can serve the requested object. The intermediate cache is consulted first (as in the case of hierarchical caches), and if this intermediate cache cannot serve the content, the other leaf caches are consulted. Second, if the requesting user is connected to the intermediate cache and this cache does not have the requested object, it is checked if one or more leaf caches have it. In case they do, the content is served from those caches. These decisions may lead to some upstream traffic from the leaf caches.



```
if(hit on LC)serve from LC
else{
  if(hit on IC)serve from IC
  else{
    if(hit on LC's)serve from LC's
    else serve from origin
  }
}
update LC
```

Figure 4: Borrowing caches.

- Federated caches, depicted in Figure 5, essentially treat all leaf caches as one large virtual cache. The content is served in the same way as in borrowing caches, but the cache update is done such that content duplication in the leaf caches is avoided. In the federated cache collaboration strategy, for each request it is first checked if the content is available in the cache to which the requesting user is connected. If it is, the request is served from that cache and the ranking (and hence, the set of cached multimedia objects) of that cache is updated, while the ranking of other caches remains untouched. If it is not, the other caches are consulted, the intermediate cache first. Only the cache from which the object is served is updated, with one exception: if the request is on a leaf cache, and neither the leaf cache, nor the intermediate cache has the object, the intermediate cache is updated too. In that way, even if there are no users connected to the intermediate cache, its cached content has a chance of being updated. The consequence of the federated cache collaboration strategy is that cached objects have the tendency to reside only in one leaf cache, because once they are cached somewhere each new request reinforces this.



```
if(request is cached locally){
  if(hit on connected cache){
    serve from that cache
    update that cache
  }
  else{
    serve from other cache, with priority to IC
    update other cache and IC
  }
}
else{
  serve from origin
  update the cache on which the request was asked
}
```

Figure 5: Federated caches

For Phase I emulations, **hierarchical caching** is selected, as it constitutes the easiest strategy with no upstream traffic. Other more advanced strategies are candidates for Phase II.

Further deliverables D4.4 “On-line Content Popularity Estimation Techniques” and D4.7 “Optimized On-line Learning Algorithms for a highly distributed CDN” will impact Phase II architecture, for more advanced caching algorithms and learning techniques.

1.4 Adaptive Delivery

The adaptive delivery mechanisms and buffering strategies are based on D5.1 “Congestion avoidance using pre-congestion measurement and local buffering mechanisms in media-aware elements” [7], and preliminary results on SVC, to be worked out in D5.2 “Evaluation of congestion control via the scalable video codec”[8].

Several strategies to combine (pre-) congestion measurement and notification mechanisms with use of buffering and rate adaptation in media-aware elements to achieve a better streaming behaviour have been investigated in[7]:

- TCP-friendly client server mechanisms (such as TFRC, and TEAR) have shown promising results to provide a smooth sending rate, allowing adaptive applications to better adapt the media stream to the available bandwidth by using graceful quality degradation mechanisms. However, as more detailed analysis will be performed to compare TFRC with TEAR (in combination with SVC), Phase I of the emulations will focus on **regular TCP transport in HTTP Adaptive Streaming**.
- An adaptive PCN algorithm with automatic rate adaptation for video delivery has been shown to be superior to the basic PCN architecture and significantly improve on a static rate adaptation system for quality scalable video streams. But as the Transport Network Interface (TNI) is out of scope for Phase I of the emulations, adaptive PCN is a candidate for Phase II only.



-
- Further, Priority based Media Delivery (PMD) of streams using SVC over RTP was investigated and it was shown that SVC outperforms H.264/AVC-based EDF and PMD methods, achieving much better results in terms of received video quality and playable frame rate over a wide range of operating points. Although PMD using SVC in HTTP streaming scenarios will be further investigated in the simulation work packages, Phase I of the emulations will include both **AVC and SVC-based HTTP adaptive streaming**.
 - Finally, an empirical algorithm for graceful video rate adaptation was defined, and initial performance comparisons indicate potential gains of over 100% when compared to traditional RAC systems. **Graceful video rate adaptation for SVC-based HTTP adaptive streaming** is part of the Phase I emulations.

Phase II of the emulations will include more detailed SVC elements based on D5.2 “Evaluation of congestion control via the scalable video codec” [8], and possibly aspects from D5.5 Distributed delivery methods using Content-aware Network Codes” [9].

2. VALIDATION SCENARIOS

2.1 Experimental Set-up

The emulation experiments are different from the simulations of WP4 and WP5 as they use real data files, processors, cache memories, network links, switches and routers, and (TCP) video flows. While the simulator is essentially event-driven, fed by a list of events, and using a time-stepped (e.g. 10 sec) simulation approach to speed up the simulations, the emulation is running experiments in real time on a real network, be it constrained and scaled-down in bandwidth and possibly memory sizes to maximize the number of experiments that can be carried out in a given time period.

Figure 6 shows the functional components of the emulation set-up. Clients, proxies, and servers have to support both AVC-based and SVC-based HTTP adaptive streaming functionality. Each component will be briefly described in section 3.2.

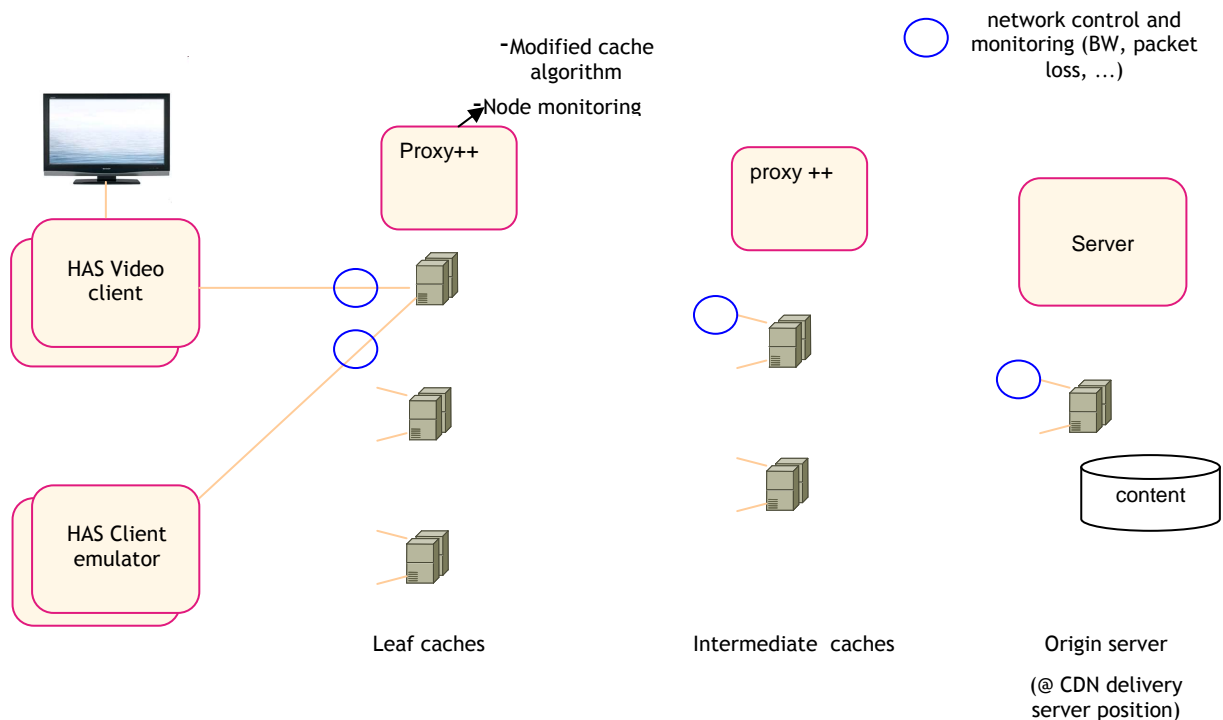


Figure 6: Functional blocks for emulation set-up

2.2 Performance Criteria

The performance criteria can mainly be divided in two categories:

- 1) The (functional) criteria that are also taken into account in the simulation experiments in WP4 and WP5: Aim here is to verify simulation results in an emulation environment with real data files, chunks, flows, and network protocols. The most important criteria are maximization of cache hit ratio (and associated reduction of average bit rate on the network links), and the reduction of the peak bit rate on the network links, where especially the link from origin server to intermediate caches is very important, as described in [3]. Furthermore



evaluations need to take the received video quality of the users and the number of supportable clients into account, where latter is influenced by the cache hit ratio, the load on the last mile and the target video quality.

- 2) Specific criteria that are not being taken into account in the simulations. The prime example here is processor load required for running the specific caching and streaming algorithms.

For both categories also a sensitivity analysis is important.

2.3 Objectives and Targets

The Phase I emulation experiments aim at providing an answer to the following topics:

- Verify the (simulated) performances of traditional LRU and LFU caching algorithms for HTTP adaptive streaming in a single and double level tree topology of caches.
- Assess the feasibility of the novel OCEAN caching algorithms, and compare the results with those of using LRU and LFU. This includes efficiency of the implementation and associated processor loads.
- Compare AVC-based HTTP adaptive streaming with SVC-based HTTP adaptive streaming, in terms of caching efficiency, streaming adaptation capabilities (in various network conditions of delays, loss, and jitter), processor load (for the network caches), and bandwidth usage.

A detailed description of the components used in the experiments, as well as the scenarios and tests that are performed, will be reported in D6.3 and D6.5.

3. EMULATION PLATFORM

3.1 Virtual Wall

The evaluations are performed on the IBBT Virtual Wall from its iLab.t Technology Center (<http://ilabt.ibbt.be>). iLab.t interconnects its test network (>200 nodes) and measurement equipment to this Virtual Wall facility, a generic emulation environment for advanced network, distributed software and service evaluation, supporting scalability research.

The Virtual Wall facilities consist of 100 nodes (dual processor, dual core servers, 6x1 Gb/s interfaces per node) interconnected via a non-blocking 1.5 Tb/s VLAN Ethernet switch, and a display wall (20 monitors) for experiment visualization. Each server is connected with 4 or 6 gigabit Ethernet links to the switch. The Virtual Wall nodes can be assigned different functionalities ranging from terminal, server and network node to impairment node. The nodes can be connected to test boxes for wireless terminals, generic test equipment, simulation nodes (for combined emulation and simulation) etc. The Virtual Wall features Full Automatic Install for fast context switching (e.g. 1 week experiments), as well as remote access.



Figure 7. IBBT's emulation environment: the iLab.t Virtual Wall

Full control over the network is ensured through the Emulab¹ management software]. Emulab allows for repeatable, dedicated and confined experiments and is responsible for swapping in required

¹ <https://www.emulab.net/>

operating system images, dynamically interconnecting the (virtual) nodes through VLANs on a network topology with emulated network links (with proper capacity, delay and packet loss characteristics).

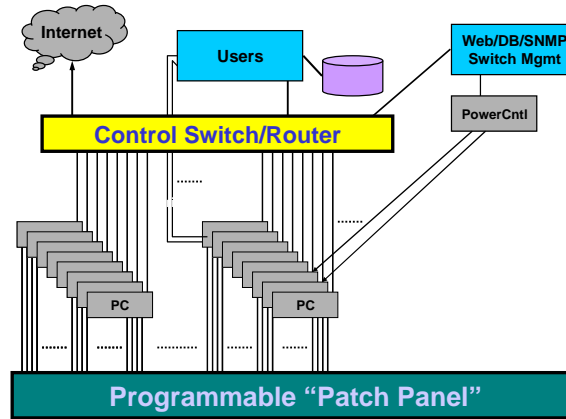


Figure 8. Emulab architecture on the Virtual Wall

The Virtual Wall also provides the necessary network monitoring tools to examine network traffic and consumption of memory, CPU, disk load and energy on the different nodes.

In addition to the network layer functionality, applications can be deployed on the Virtual Wall nodes controlled by the researchers through VM images. These instances can also be monitored (bandwidth, CPU and memory consumption, response times and availability over time) and can be invoked through request generators. These request generators can be tuned in a variety of different parameters (e.g. spatial request distribution, temporal request distribution, popularity of service instances) and assign the client locations through automated generation of scripts.

Dedicated accounts for accessing the Virtual Wall's internal network² through OpenVPN³ are available in the project.

3.2 Functional Components

3.2.1 HTTP Adaptive Streaming Client

Clients for both AVC-based and SVC-based HTTP Adaptive Streaming will be used.

For the AVC-based emulation experiments, the Microsoft implementation of adaptive HTTP streaming - called Smooth Streaming – will be used. The adaptive streaming intelligence is located in the client. Based on the perceived local bandwidth and its video rendering performance, the client decides which video quality level will be downloaded and displayed. During the ployout, the client continually optimizes the playback of the content by switching between the different available video quality levels in real-time. As a Smooth Streaming client, the Microsoft Silverlight client is made available by Microsoft as a plugin to web-browsers such as Internet Explorer, Mozilla Firefox, etc.

² <http://wall.test/>

³ <http://openvpn.net/>

For SVC, software developed by one of the OCEAN partners will be used.

3.2.2 Client Emulator

For multi-client emulation, one of the OCEAN partners will develop software that can emulate the Smooth Streaming client (in terms of algorithm used, but without video output), and can emulate up to 100 clients simultaneously. By running this emulator on several hardware nodes, up to a thousand users could be emulated.

For the SVC-based emulation tests, a similar client emulator will be developed to run similar tests as with AVC and compare the results.

3.2.3 Network Proxy

A key property of HTTP adaptive streaming is its use of standard HTTP delivery infrastructure (no specialized streamers required) such as network proxies. The experiments will use the widely used proxy Squid [10] with its built-in LRU and LFU algorithms for both AVC and SVC experiments.

For the novel caching algorithms of OCEAN, either Squid will be extended, or an alternative light-weight HTTP proxy server will be developed by one of the OCEAN partners. Both AVC and SVC will be supported.

3.2.4 Content Server, Encoder and Segmenter

For the AVC-based experiments, content is encoded in multiple quality levels, each corresponding to an average video encoding bit rate. The encoding of content can be accomplished using the Microsoft Expression Encoder. At the moment of ingest, the same source content is encoded at different quality levels. Smooth Streaming uses the MPEG-4 Part 14 (ISO/IEC 14496-12) file format as its disk (storage) and wire (transport) format.

Additionally, the encoded content is divided in temporal segments, also called “chunks”, of typically 2 seconds starting with an I-frame. Microsoft offers a Smooth Streaming plugin to the Microsoft IIS Media services web-server. When the server receives a segment request, it dynamically creates the required segments from the encoded video files. The server then makes these segments available for retrieval under the form of HTTP data.

For SVC, content will be encoded in layered format, where a higher quality can be achieved by combining multiple layers. The encoded content will also be divided in segments, which can be separately addressed and retrieved by the client. Software provided by one of the OCEAN partners will be used.

Besides real video content, also pseudo content will be generated for the emulation experiments. A pseudo content generator will be able to generate segments with sizes and size distribution comparable to that of real encoded content. These segments will contain random data and be consumed only by the client emulator software. (The real clients obviously will play the real video content.)

3.3 Validation Components

This section describes the functionality of the different components on the Virtual Wall emulation environment.

3.3.1 NS-2 experiment description

In Emulab, experiments are described using NS-2 scripts, for which advanced tutorials⁴ are available.

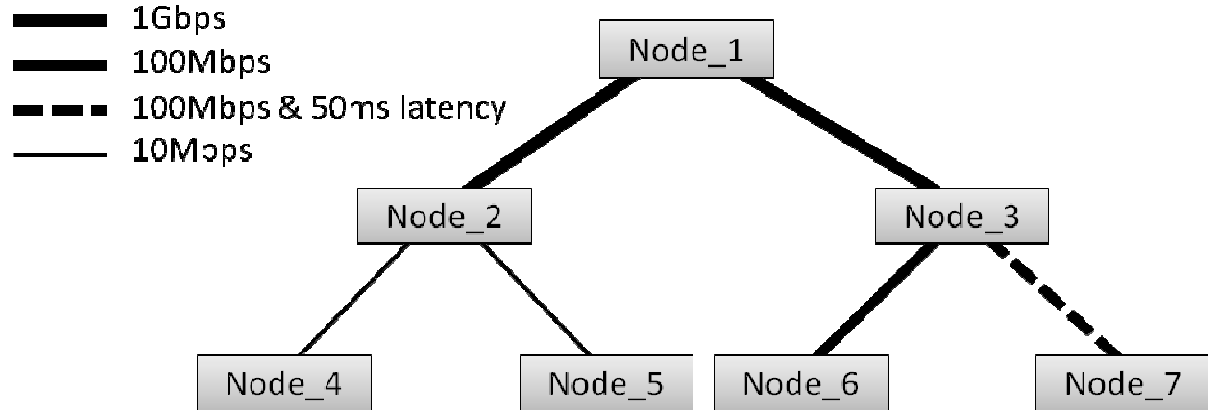


Figure 9. IBBT's emulation environment: the iLab.t Virtual Wall

An example network topology as in Figure 9, where a server (Node_1) is connected to several client nodes (Node_4, Node_5, Node_6 and Node_7) through proxy caches (Node_2 and Node_3) could result in the following NS script:

```

set ns [new Simulator]
source tb_compat.tcl

set node1 [$ns node]
tb-set-node-os ${node1} OS_IMAGE_SERVER
set node2 [$ns node]
tb-set-node-os ${node2} OS_IMAGE_PROXY
set node3 [$ns node]
tb-set-node-os ${node3} OS_IMAGE_PROXY
set node4 [$ns node]
tb-set-node-os ${node4} OS_IMAGE_CLIENT
set node5 [$ns node]
tb-set-node-os ${node5} OS_IMAGE_CLIENT
set node6 [$ns node]
tb-set-node-os ${node6} OS_IMAGE_CLIENT
set node7 [$ns node]
tb-set-node-os ${node7} OS_IMAGE_CLIENT

set network1 [$ns duplex-link ${node1} ${node2} 1000Mb 0ms DropTail]
set network2 [$ns duplex-link ${node1} ${node3} 1000Mb 0ms DropTail]
set network3 [$ns make-lan "${node2} ${node4} ${node5}" 10000kb 0ms]
    
```

⁴ <http://users.emulab.net/trac/emulab/wiki/AdvancedExample>



```
set network4 [$ns duplex-link ${node3} ${node6} 100000kb 0ms DropTail]
set network5 [$ns duplex-link ${node3} ${node7} 100000kb 50ms DropTail]
```

```
tb-set-ip-link ${node1} ${network1} 172.16.1.2
tb-set-ip-link ${node1} ${network2} 172.16.2.2
tb-set-ip-link ${node2} ${network1} 172.16.1.3
tb-set-ip-lan ${node2} ${network3} 172.16.3.2
tb-set-ip-link ${node3} ${network2} 172.16.2.3
tb-set-ip-link ${node3} ${network4} 172.16.4.2
tb-set-ip-link ${node3} ${network5} 172.16.4.4
tb-set-ip-lan ${node4} ${network3} 172.16.3.3
tb-set-ip-lan ${node5} ${network3} 172.16.3.5
tb-set-ip-link ${node6} ${network4} 172.16.4.3
tb-set-ip-link ${node7} ${network5} 172.16.4.5
```

```
$ns rproto Static
```

```
$ns run
```

The software that has to be deployed in the different nodes is installed through Virtual Machine (VM) images (OS_IMAGE_SERVER, OS_IMAGE_PROXY and OS_IMAGE_CLIENT).

3.3.2 NetBuild GUI

Besides an XML RPC⁵ interface, a graphical user interface is available on the Virtual Wall platform as well. The same setup is in the previous section on the NetBuild GUI is presented in Figure 10.

⁵ <https://www.emulab.net/xmlrpcapi.php3>

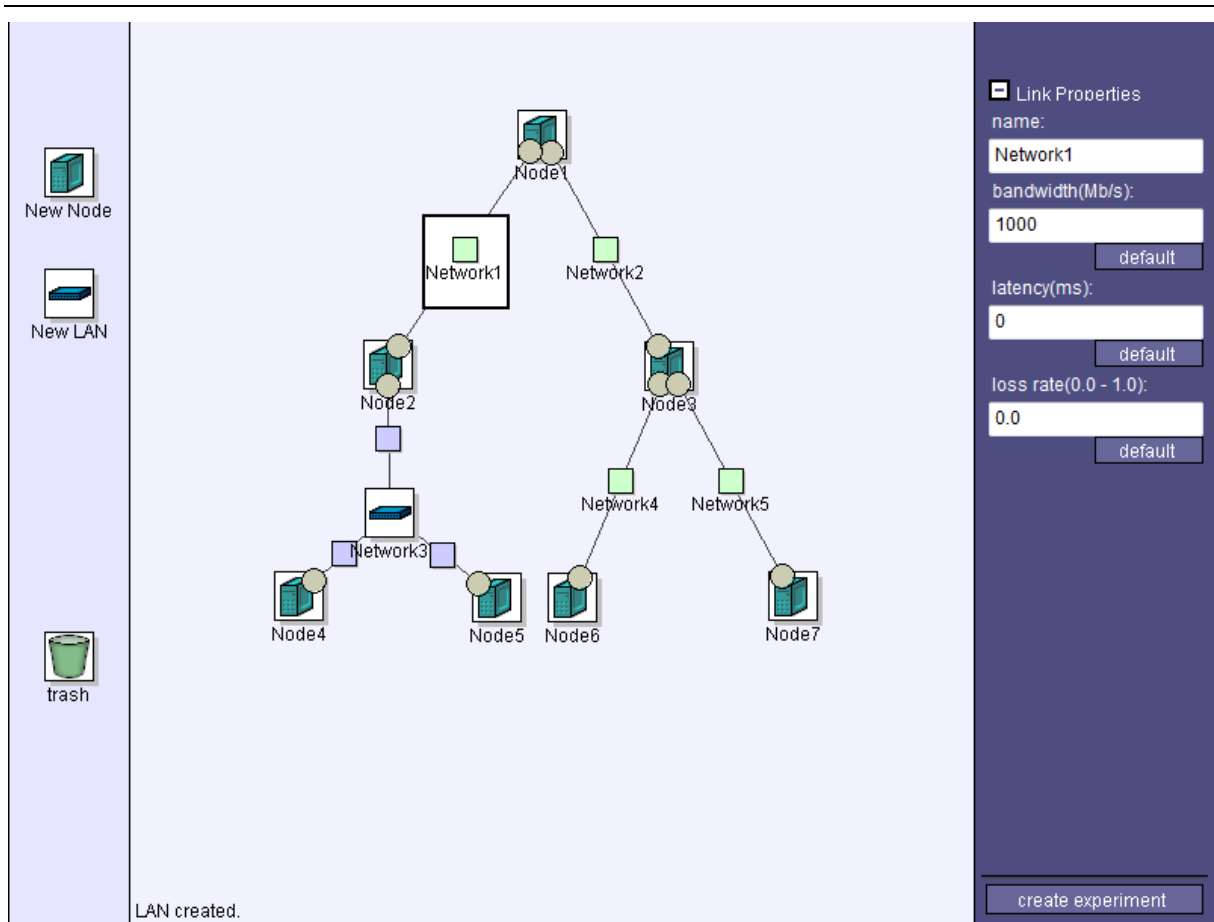


Figure 10. NetBuild GUI in Emulab

3.3.3 Extended experiment management GUI

Working with the abovementioned NetBuild GUI might be cumbersome for multiple reasons:

- Experiments with large network topologies require many drag-and-drop and configuring operations.
- Batches of experiments have to be configured, scheduled and deployed on the Virtual Wall one by one.
- Log files have to be collected by the experimenter from the repository after each experiment.

Therefore, an extended experiment management GUI is under development in the OCEAN project so that larger setups, with parameter settings specific to the envisioned experiments, can be setup, scheduled and deployed automatically. Data collection and monitoring is available in this application as well. An early prototype version is shown in Figure 11.

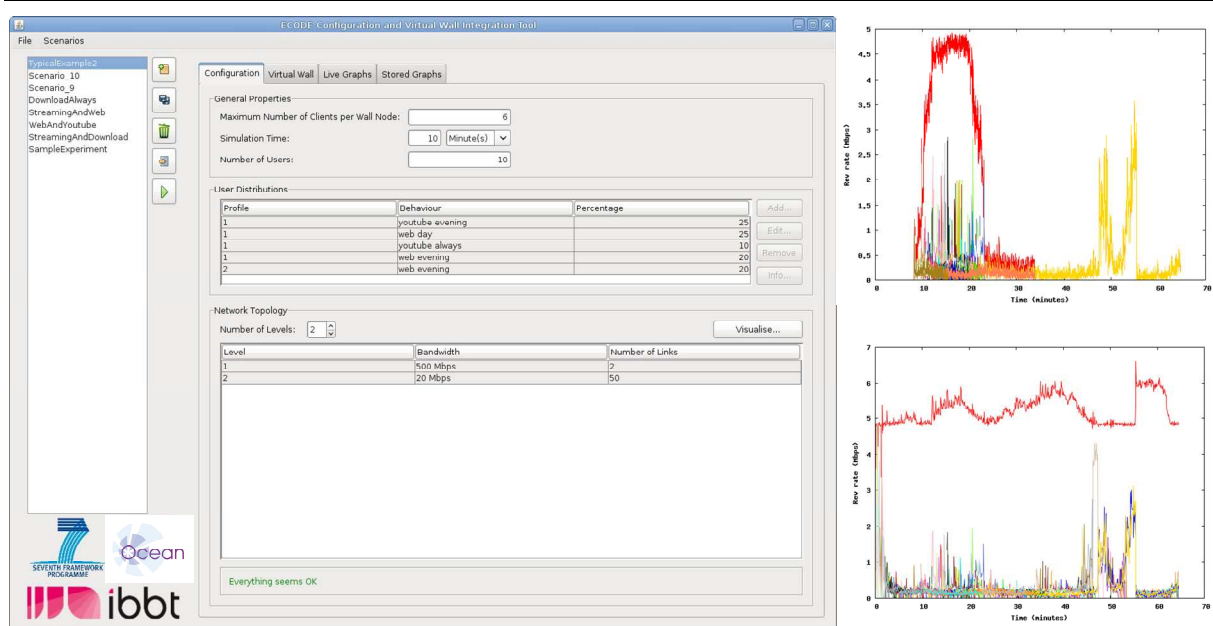


Figure 11. Prototype of the extended experiment management GUI

3.3.4 Monitoring

The Virtual Wall has several components that allow monitoring network and application performance. These components all have a stable version but are still work in progress. Hence, they can be extended during the project. In some cases, some minor integration work with the Virtual Wall can be needed.

3.3.4.1 Monitoring of Hardware performance

To monitor the hardware performance of a server, a distributed monitoring architecture exists on the Virtual Wall. This monitoring architecture consists of several small monitoring probes that measure power consumption, memory usage, etc. and transmit it to a central server. Each monitoring probe contains a transactional database, to store the measured data in a cyclic manner, and a web interface, to provide visual information (e.g. through real-time graphs). The following metrics can be measured through the architecture:

- Memory usage
- CPU usage
- Transfer rate on the hard drive in terms of read and write operations
- Incoming and outgoing bit rate on a network interface
- Actual voltage
- Total power consumption
- Environmental temperature (e.g. temperature close to the device)

3.3.4.2 Monitoring of Network Performance

Two distinct types of network monitoring probes exist on the Virtual Wall. A Click Modular Router based monitoring probe, and a libpcap based monitoring probe. The first one can be used when



integration with Click is advisable (e.g. when existing sessions need to be altered through shaping and policing). The latter is optimized for application specific monitoring situations (e.g. the monitoring of video quality).

Both monitoring probes monitor typical QoS parameters such as packet loss ratio, delay and jitter in real-time. Furthermore, the Click based monitoring probes contain different bandwidth measurement algorithms to monitor the used resources. Depending on the envisaged use case, monitoring can occur on session, subscriber and node level.

Besides monitoring the actual network performance, the Click based monitoring probe can estimate packet loss, delay and jitter by monitoring intermediary in the network. This estimation can be interesting if user-experienced network performance must be measured in a network environment where the operator has no control on the end devices (e.g. a laptop in the home network). The carried out estimation has an accuracy between 95% and 100%.



REFERENCES

- [1] "Current State of the OCEAN Market", D2.1, October 2010.
- [2] "Preliminary Functional Architecture", MS3.1, v1.0, September 2010.
- [3] "Simulation Environments and Parameters for the Evaluation of Caching Algorithms", M4.1, v2.0, August 2010.
- [4] "Evaluation Criteria for Online Popularity Estimation Techniques", M4.2, v2.1, October 2010.
- [5] "Highly Dynamic and Distributed Caching – Report", D4.2, draft v1.0, December 2010.
- [6] "On-line Learning Mechanisms for Distributed Caching", D4.5, draft v1.0, December 2010.
- [7] "Congestion avoidance using pre-congestion measurement and local buffering mechanisms in media-aware elements", D5.1, v1.0, December 2010.
- [8] "Evaluation of congestion control via the scalable video codec", D5.2, draft notes v0.x, December 2010.
- [9] "Distributed delivery methods using Content-aware Network Codes", D5.5, v1.0, December 2010.
- [10] <http://www.squid-cache.org>

ACRONYMS

AVC	Advanced Video Coding
CC	Chunk-level Caching
CDM	Content Delivery Module
CDN	Content Delivery Network
CMTPS	Cache Management based on Temporal Pattern Solicitation
CPU	Central Processing Unit
DSL	Digital Subscriber Line
DVD	Digital Video Disc
EDF	Earliest Deadline First
FTTH	Fiber To The Home
GUI	Graphical User Interface
HD	High Definition
HTTP	Hyper Text Transfer Protocol
LFU	Least Frequently Used
LRU	Least Recently Used
MPEG	Moving Pictures Expert Group
NNI	Network to Network Interface
OCEAN	Open ContEnt Aware Networks
OCIG	OCEAN Content Interworking Gateway
OS	Operating System
PCN	Pre Congestion Notification
P-LFU	Predicting Least Frequently Used
PMD	Priority based Media Delivery
QOS	Quality Of Service
RAC	Resource Admission Control
RPC	Remote Procedure Call
RTP	Real-time Transport Protocol
SC	Scoring-based Caching
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
TEAR	TCP Emulation At Receiver



TFRC	TCP Friendly Rate Control
TNI	Transport Network Interface
UNI	User to Network Interface
VLAN	Virtual LAN
VM	Virtual Machine
VOD	Video On Demand
WP	Work Package
XML	eXtensible Mark-up Language