COntent Mediator architecture for content-aware nETworks

European Seventh Framework Project FP7-2010-ICT-248784-STREP

Deliverable D4.2
Final Specification of Mechanisms, Protocols and Algorithms for Enhanced Network Platforms

The COMET Consortium

Telefónica Investigación y Desarrollo, TID, Spain
University College London, UCL, United Kingdom
University of Surrey, UniS, United Kingdom
PrimeTel PLC, PRIMETEL, Cyprus
Warsaw University of Technology, WUT, Poland
Intracom SA Telecom Solutions, INTRACOM TELECOM, Greece

For more information on this document or the COMET project, please contact:

Dr. Ning Wang
University of Surrey
Guildford
Surrey GU2 7XH
UK

Document Control

Title: Final Specification of Mechanisms, Protocols and Algorithms for Enhanced Network Platforms

Type: Public

Editor(s): George Kamel, Ning Wang

E-mail: g.kamel@surrey.ac.uk, n.wang@surrey.ac.uk

Author(s): George Kamel, Ning Wang (UniS)

Andrzej Beben, Jaroslaw Sliwinski, Jordi Mongay Batalla, Piotr Wisniewski, Wojciech Burakowski (WUT)

Wei Koong Chai, Ioannis Psaras, João Taveira Araújo (UCL)

Doc ID: D4.2-2.0.docx

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	July 29, 2011	Ning Wang	Initial ToC
V0.2	October 24, 2011	George Kamel	Added UniS contributions (§6 & 8)
V0.3	November 2, 2011	George Kamel	Added UCL contribution (§7.2)
V0.4	November 7, 2011	George Kamel	Added WUT contributions (§4& 5 and Appendix A)
V0.5	November 15, 2011	George Kamel	Added WUT contribution (§3)
V0.6	November 16, 2011	George Kamel	Added UCL contributions (§9 & 7.1 and Appendix B)
V0.7	November 17, 2011	Ning Wang	Edited Conclusions (§10)
V0.8	November 17, 2011	George Kamel	Edited Executive Summary and Introduction (§1 & 2)
V1.0	November 17, 2011	George Kamel	Final version ready for submission
V1.01	November 18, 2011	George Kamel	Corrected problem with typesetting of equations in §9 and App. B
V1.1	December 6, 2011	George Kamel	Added amended UCL contributions (§7.1 and §9)
V1.2	December 14, 2011	Ioannis Psaras	Reviewed and amended whole deliverable
V2.0	December 19, 2011	George Kamel	Final amended version ready for submission

Table of Contents

(This page is left blank intentionally.)

1 Executive Summary

This deliverable presents the final specifications of the mechanisms, protocols and algorithms associated with content forwarding plane functionalities for end-to-end content delivery in multi-domain networks. This supersedes the corresponding interim specifications given in COMET Deliverable D4.1 [1], and complements the mechanisms, protocols and algorithms relating to content mediation specified in COMET Deliverable 3.2 [2].

Two main approaches to content-aware forwarding are proposed: an evolutionary approach and a revolutionary approach. The **evolutionary approach**, which is *stateless* in nature, aims at fast, incremental deployment of enhanced networking techniques for efficient Internet-wide content delivery without fundamentally changing the underlying legacy systems. On the other hand, the **revolutionary approach**, which is state-based, supports long-term evolution of future Internet and clean-slate design of future content-centric network architectures, which do not necessarily rely on existing network platforms. Both the stateless and stateful approaches are specified in Sections 5 and 6, respectively, in terms of both the architecture of the various content-forwarding-related entities, and the interaction between these entities.

Supporting both approaches are a number of 'core' and 'advanced' features. The first of these core features is **quality-of-service support**, which is presented in Section 3. The COMET approach for Quality of Service (QoS) engineering assumes that content is delivered using dedicated paths (content delivery paths), which support COMET Classes of Service (CoS). We define the global COMET CoS that can be used for supporting differentiated traffic treatment in end-to-end content delivery, and we also consider offline and long timescale network provisioning operations for preparing for the actual content delivery.

The way in which information about different network paths is disseminated to the COMET content delivery decision making engine is another important issue that is addressed. Towards this end, we specify a number of mechanisms and algorithms in Section 4 to support the **routing awareness** process. This process is responsible for gathering information about inter-domain paths that are used for content delivery. We specify the format of messages exchanged between routing awareness entities (RAE), basic operations performed by RAEs as well as the considerations about different strategies of RAE deployment.

One of the main benefits of information-/content-centric networks is the ubiquitous **in-network caching** which can significantly increase the efficiency of the network and reduce content delivery latency. In Section 7, we propose two novel schemes to support efficient content caching within COMET. The first scheme, presented in Section 7.1, is a probabilistic in-network caching scheme (ProbCache), which approximates the caching capability of a path and then caches contents probabilistically along the path. The second scheme, presented in Section 7.2, is a centrality-based caching scheme, where content is cached only at specific high centrality node(s) along the content delivery path. We show that such a scheme outperforms the standard ubiquitous caching.

Having specified the content resolution and delivery processes for fixed clients in Sections 5 and 6, we turn our attention to the provision of **content-aware mobility support** mechanisms for COMET in Section 8. Despite the inherent ability of information- or *content*-centric networks to support user mobility (albeit inefficiently), we observe that there is still space for additional mobility support mechanisms that aim to keep handover latency to a minimum. Therefore, Section 8 addresses mobility at both the system level, in the form of handover support, and mobility- and content-aware scheduling of content at the wireless interface, respectively.

Finally, in Section 9, we question the validity of metrics such as hop-count used to measure network congestion. We find that, sometimes, in real networks, shorter paths (in terms of autonomous systems, for instance) induce longer delays. Therefore, we design a loss-based, **congestion-aware content traffic load-balancing** algorithm, which we call PREFLEX (Path RE-Feedback with Loss EXposure) and assess its operational properties. The performance of our approach is evaluated by simulation and is shown to perform well in a variety of circumstances.

2 Introduction

The COntent Mediator architecture for content-aware NEtworks (COMET) project aims to define a novel content-oriented Internet architecture that will radically simplify content access, and will support content distribution in a network-aware fashion. Towards this objective, this deliverable specifies the mechanisms, protocols and algorithms to enhance network platforms to support content delivery in a content-aware manner. This supersedes the interim specifications given in COMET Deliverable D4.1 [1], and complements the mechanisms, protocols and algorithms relating to content mediation specified in COMET Deliverable 3.2 [2].

Two main approaches to content-aware forwarding are proposed: an evolutionary approach and a revolutionary approach. On one hand, evolutionary approaches aim at fast and incremental deployment of enhanced networking techniques for efficient Internet-wide content delivery without fundamentally changing the underlying legacy systems. In this case, existing platforms, such as DNS-like content resolution and IP based addressing, routing, and forwarding architectures are considered. On the other hand, in order to support long-term evolution of future Internet, clean-slate design of future information-/content-centric network architectures, mechanisms and protocols are also investigated, which do not necessarily rely on existing network platforms. This deliverable will present the final specification of our proposed approaches in both categories, namely the former, *stateless* approach, and the latter, *stateful* approach.

Both stateless and stateful approaches are supported by a number of common features, namely, quality-of-service provisioning, routing-awareness, congestion-avoidance through load-balancing, network-based content caching, and support for mobile COMET users. Some of these features are new to this project, such as the loss-aware traffic load-balancing and mobility support, and are presented here for the first time. Other features such caching support has been significantly extended from what was presented in the D4.1, for example proposing techniques for limiting the number of nodes at which content is cached to reduce redundancy and, in turn, increase overall network performance. This rest of this section will briefly introduce both approaches to content forwarding, as well as each of these common features.

The specifications of the two main approaches to content forwarding, stateless and stateful, are presented in Sections 5 and 6, respectively. The stateless content delivery process is related with the decoupled content resolution process described in Section 3 in D3.2 [2]. This approach assumes that Content-Aware Forwarding Entities (CAFEs) forward content based on information about the selected path that is stored in the COMET header attached to the original packet containing bits of content. As a result, CAFEs maintain only the neighbourhood (local) information, i.e., how to forward packet to the next CAFE instead of keeping routing table with all prefixes. In addition, the stateless forwarding allows the Content Mediation Plane (CMP) to select adequate path for each consumption request (in a manner similar to source routing). The stateless content delivery process covers three main actions: (1) collecting of forwarding information for selected paths, (2) preparation for content delivery and (3) content forwarding. The collecting action is performed at semi-long time scale, i.e., when a new routing path is used. The resulting configuration is valid for long durations (in the order of hours or days) and may be used by multiple content transfers as long as the selected path is the same. The result of collecting is the list of forwarding rules required to forward content along the selected path. The preparation for content delivery is invoked for each consumption request. This action is responsible for configuration of edge CAFEs and assigning appropriate COMET header containing the list of forwarding rules. Finally, content forwarding relates directly to packet handling in CAFEs.

Concerning the revolutionary/radical option, we have specified a stateful content delivery process presented in Section 6, which is to be tied to the coupled content resolution approach specified in Section 4 in D3.2 [2]. According to this approach, specific content states are required to be maintained at involved CAFEs as the ingress and egress nodes of individual domains. This type of state maintenance is similar to the principle of IP multicast, but there are some key differences. A fundamental one is that the state configuration is effectively done by the local Content Resolution and Mediation Entity (CRME) in the CMP. More specifically, during the content resolution phase,

once the CRME has determined to forward the content resolution request to its counterpart in the next hop domain towards the targeted source, it is responsible for configuring the corresponding CAFEs in its local domain for preparing for the actual delivery of the content flow back to the consumer. Due to the fact that the content resolution and delivery processes are tightly coupled, the actual *domain-level* delivery path is effectively the reverse direction of the corresponding resolution path.

As previously mentioned, both approaches to content forwarding are supported by a number of common features. The first of these is quality-of-service support, which is presented in Section 3. The COMET approach for Quality of Service (QoS) engineering assumes that content is delivered using dedicated paths (content delivery paths), which support COMET Classes of Service (CoS). We first define in Section 3 the global COMET Class of Services (CoS) that can be used for supporting differentiated traffic treatment in end-to-end content delivery. The way in which end-to-end CoS can be achieved through the mapping of specific internal QoS capabilities within participating ISP networks is detailed in this section. On the other hand, we also consider offline and long timescale network provisioning operations for preparing for the actual content delivery; specifically, the provisioning of (multi-paths) inter-domain routes across domains.

The way in which information about different network paths is disseminated to the COMET content delivery decision making engine is another important issue addressed in this deliverable. Towards this end, we specify a number of mechanisms and algorithms in Section 4 to support the routing awareness process. This process is responsible for gathering information about inter-domain paths that are used for content delivery. In principle, this process is similar to inter-domain routing but we introduce two main enhancements. First, the routing awareness process aims to find inter-domain paths that optimise delivery of content. Therefore, this process should match transfer requirements of specific type of content with the QoS capabilities offered by domains. The second enhancement corresponds to propagation and maintenance of multiple paths between the source and destination domain. This feature allows for applying smart traffic engineering algorithms, load balancing in the network as well as improving reliability of content delivery. In Section 4 we specify the format of messages exchanged between routing awareness entities (RAE), basic operations performed by RAEs as well as the considerations about different strategies of RAE deployment.

One of the main benefits of information-centric networks is the ubiquitous in-network caching which has the potential to increase the efficiency of the network and reduce content delivery latency, avoiding the need to obtain content from the source each time it is requested. In Section 7, we depart from the Content-Centric Networking (CCN) in-network caching scheme used by Van Jacobson *et al.* [3], in which content is wastefully cached at all nodes along a content delivery path, and propose two novel schemes to support more efficient content caching. The first scheme, presented in Section 7.1, is a probabilistic in-network caching scheme (ProbCache), which approximates the caching capacity of a path and then caches contents probabilistically along the path in order to: i) leave space for other flows on the same path to cache their contents, and ii) fairly multiplex contents in caches along the path from the server to the client. The second scheme, presented in Section 7.2, is a centrality-based caching scheme, where content is cached only at specific high centrality node(s) along the content delivery path. We show that such a scheme outperforms the standard ubiquitous caching (i.e., where content is cached at each and every node it traverses along the delivery path).

Having specified the content resolution and delivery processes for fixed clients in Sections 5 and 6, we turn our attention to the provision of mobility support mechanisms for COMET in Section 8. In general, information-centric networks (ICNs) are inherently able to offer better support for mobile users compared to host-centric communication models. This can be attributed to the network's awareness of content, which mitigates the need for the content server or a mobility agent located away from the client to be updated each time the client's IP address changes as a result of handover between routers. Instead, communication paths are simply 'healed', such that only the changed part of the communication path is re-established after a client handover. Nevertheless, despite ICNs' inherent mobility support, it is possible improve this feature by designing additional mobility support mechanisms that aim to keep to a minimum the disruptive effect of handovers. Therefore,

Section 8 addresses mobility at both the system and link levels. At the system level, both intra- and inter-domain handover mechanisms are specified to support within the coupled approach both user context transfer between CAFEs and smooth switching of changed communication paths. At the link level, we propose a number of novel ideas to support mobility- and content-aware scheduling of content at the wireless interface.

In D3.2 [2], a number of strategies to select the best path available from a given client to a given server were introduced and evaluated. These strategies included combinations of paths and servers based on the number of hops and the load on the servers. Although the results presented in D3.2 give a good insight on the most appropriate selection strategies, depending on the topology and the available servers, we have conducted measurements from real networks to see whether our assumptions and predictions hold. In particular, we used non-anonymised traces of inter-domain traffic, provided by the Japanese academic network, which provides connectivity for academic and research institutions in the country. In Section 9, we question the validity of metrics such as the number of hops (in terms of IP routers, or ASes). We find that, sometimes, in real networks, shorter paths (in terms of ASes, for instance) induce longer delays. Therefore, we design a loss-based, congestion-aware traffic balancing algorithm, which we call PREFLEX (Path RE-Feedback with Loss EXposure) and assess its operational properties. Equations are derived for the automatic tuning of the time scale and traffic split at a decision point. The algorithms described in this document allow the load balancer to self-tune to the network conditions. The calculations are simple and do not place a large burden on a router where the algorithm would be implemented. The algorithm is evaluated by simulation using ns-3 and is shown to perform well in a variety of circumstances. The resulting adaptive, end-to-end load balancing architecture provides the necessary framework to meet the increasing demands of users while simultaneously offering edge networks more fine-grained control at far shorter timescales.

3 Quality of Service Engineering for Content Delivery

The COMET approach for Quality of Service (QoS) engineering assumes that content is delivered using dedicated paths (content delivery paths), which support COMET Classes of Service (CoS). The COMET CoS assures adequate transfer of given type of content in each domain between server and client. The content delivery paths are organized and maintained by routing awareness processes performed by the Routing Awareness Entities (RAE). The RAE is responsible for matching the content delivery requirements with the transfer capabilities offered by particular domains. Figure 1 illustrates the idea of the investigated approach.

The domains usually differ in offered transfer capabilities, i.e., they support different intra-domain CoSs. In order to perform coherent transfer of the content from the server to the client, each domain should perform a mapping between COMET CoSs and the transfer capabilities that it offers. This mapping should be performed as a part of RAE configuration based on the domain provisioning information, e.g., domain owner policies, available resources and transfer capabilities. The specification of the RAE is provided in chapter 4.

Other important elements enabling the QoS engineering in COMET are the Content Aware Forwarding Entities (CAFEs). CAFEs handle the packets transferring the content by using a dedicated COMET header appended to the packets as shown in Figure 2. This allows the COMET system to take advantage of three main features: first, CAFEs may enforce the content delivery paths regardless of existing routing. This feature not only relaxes constraints of standard routing protocols where single shortest path is used (e.g., BGP), but it also allows to use content delivery path on demand for particular content consumption (using previously established paths). Second, CAFEs perform adequate packet classification/marking for mapping between COMET CoSs into transfer capabilities used in particular domain. Although this function seems to be simple, it is essential to provide end-to-end QoS across the multi-domain network. Both RAEs and CAFEs allow the creation of an overlay network for content delivery, which in turn enables the setting of specific interconnection agreements for content delivery, independent of those currently used for Internet traffic, with autonomous systems potentially located at hop distances higher than one AS.

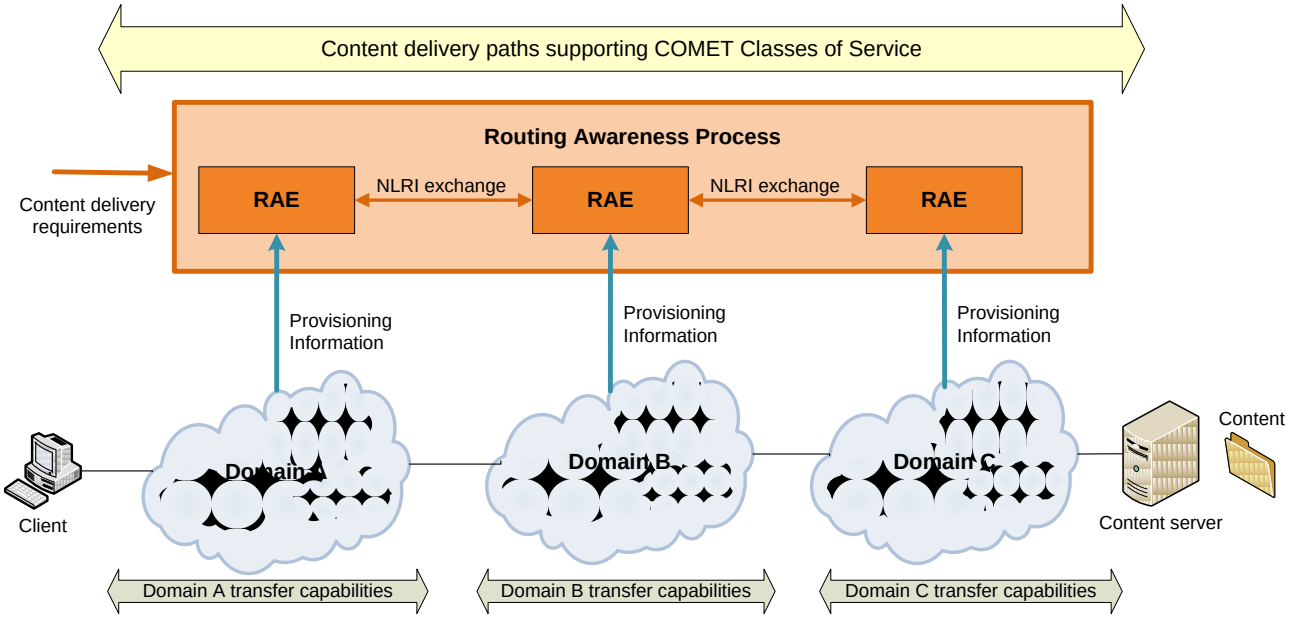


Figure 1: Approach for creating content delivery paths

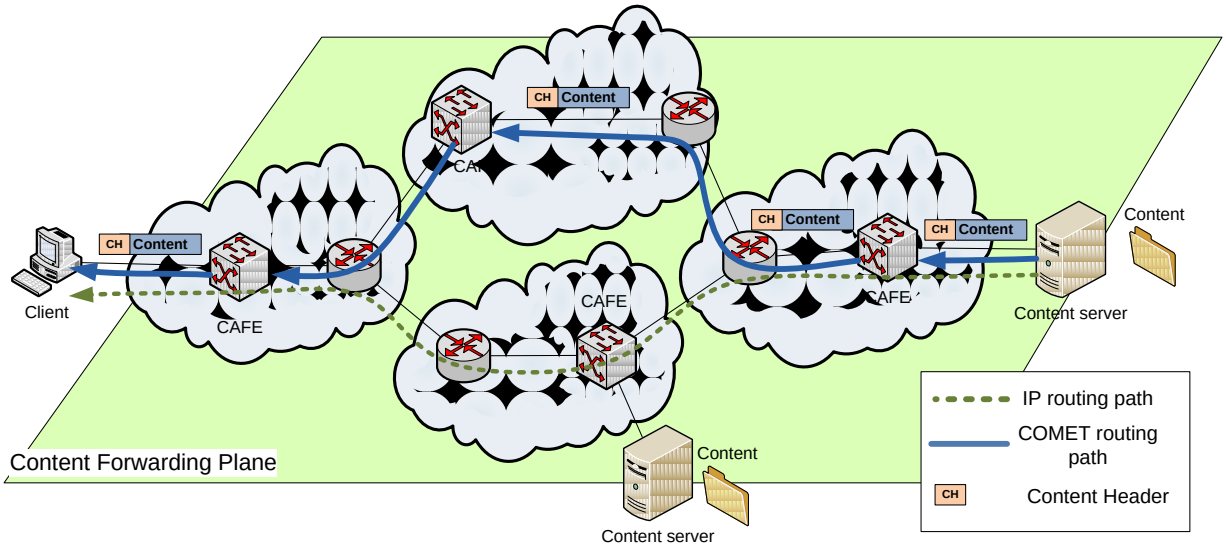


Figure 2: Approach for enforcement of content delivery paths

Briefly, even though the content delivery in COMET relies on transfer capabilities provided by particular domains, the RAEs and CAFEs allow the system to apply specific traffic control and engineering algorithms in order to optimise content delivery paths. Thanks to them, the COMET can improve both the quality perceived by content consumers and the efficiency of resource utilisation (network and server resources).

3.1 COMET Classes of Service

The COMET approach for delivering the content follows the commonly recognised approach of end-to-end Classes of Service (e2e CoS) that is widely investigated for providing QoS in a multi-domain network, see [4][5][6]. The e2e CoS defines the transfer capabilities required to meet QoS requirements of given type of traffic. The e2e CoSs are defined in a global scope, and then they are mapped into appropriate intra-domain network CoSs offered by particular domains. These intra-domain CoSs have only local meaning, because particular domains may differ in offered intra-domain CoSs and the assured QoS level, depending on service provider policies, technical constraints or business relations.

In the COMET project, we follow the end-to-end CoS approach with two main extensions. First, we define the end-to-end CoS, called COMET CoS, that are strictly oriented to content delivery. They are designed taking into account characteristics of content types considered in COMET. Similar to end-to-end CoS, the COMET CoSs are defined in the end-to-end scope and they should be properly mapped into intra-domain CoSs offered by a particular domain. The second extension is the COMET specific routing awareness function, which allows to fix inter-domain routing paths taking into account requirements of COMET CoS and transfer capabilities of intra-domain CoS offered by particular domain. Thanks to routing awareness function, we assure wide visibility of COMET CoS. The COMET CoS are used in several COMET processes. First, each content is assigned to COMET CoS during content registration based on required service level. Second, COMET CoS are used in routing awareness process to build content delivery paths based on intra-domains CoS. Finally, we use COMET CoS during path configuration process to assign appropriate forwarding rules to delivered content, e.g. marking, queuing.

We start the design of COMET CoS from the analysis of content characteristics. First of all, we classify the content considered in COMET in content types based on the content nature and the requirements for content transfer. Basically, we distinguish two content types that are: *live* and *pre-recorded*.

The main feature of the **live content** is that content producer prepares the content in an on-line manner, i.e., the content is captured, encoded and transmitted in a scale of fraction of a second. As

a result, we do not know the size of the content, although the duration can be approximated. Good examples of this content type are live transmissions of sport or cultural events.

Even though we cannot accurately describe the duration or size of the content, the important characteristic is the setting of the codec. This allows for estimation of traffic profile emitted by servers, where we distinguish 2 main types of encoding: i) constant bit rate encoding and ii) variable bit rate encoding. From the perceptive point of view the variable encoding offers better QoS (better quality at lower bit rate), while at the same time it generates bursts of data that may temporarily overload the network. As a result the buffers may overflow and encoded stream may be damaged.

Notice that the content server usually streams the content for multiple clients at the same time. This naturally brings into attention the point-to-multipoint capabilities, e.g., multicast. While there are several overlay solutions that uses TCP transport protocol (e.g. Octoshape), the transmission over UDP is more natural. In addition, transmission over UDP is able to exploit natural features of multicast in IP networks. On the other hand, it requires more stringent transfer capabilities in the network, e.g. lower packets losses.

The **pre-recorded content** is prepared in an off-line manner, i.e., the processing of content is completed and then it is ready for transfer. All properties of the content and its meta-information are known beforehand. Typical examples of such content are movies or songs.

While both duration and bit rate requirements are known, the emitted traffic profile depends on the used transport protocol. The UDP protocol reduces the load at server side and simplifies the synchronization at the receiver. The problem with traffic bursts remains the same as for live traffic. Although using TCP protocol filters the bursts produced by application and in addition, allows for retransmissions of lost packets, it is often limited by, so called, *bandwidth delay product* which bounds the maximum achievable throughput when network delay becomes noticeable. This is important limitation when the content uses high resolution fidelity and thus requires large amounts of bandwidth.

The pre-recorded content is usually available in multiple network locations, which improves scalability of traffic sources and moreover it allows for load balancing among the content sources.

The COMET CoS take into account both types of content and relate them with potential transfer capabilities offered by particular domains. We consider 3 COMET CoS:

- **Premium (PR)** CoS – it offers guaranteed services by exploiting Service Level Specification agreements between peering domains. The guarantees define the service at the level of packet transfer characteristics, i.e., packet loss ratio, packet transfer delay metrics and guaranteed throughput. Note that for achieving absolute QoS guarantees, the domains may employ resource and admission control functions.

- **Better than Best Effort (BTBE)** CoS – it offers the transfer of content over the COMET controlled paths in the network. This requires that the forwarding over a path is enforced by CAFEs. While this CoS does not introduce any kind of guarantees, it improves the ability to control the load over the paths in the network. Consequently, we enable new mechanism for traffic engineering that can be used for relative differentiation of the service. This service is intended for media-delivery applications, which traffic may be transferred without QoS guarantees, e.g., P2P based real-time applications. This CoS may be used also for delivery of live or pre-recorded content, but in this case without QoS guarantees. This may happen when domain cannot provide Premium CoS for particular content delivery.
- **Best Effort (BE)** CoS – it offers the transfer of content in best effort way using existing routing paths in the network.

Table 1 summarizes main features of considered COMET CoS.

	Best Effort	Better than Best Effort	Premium CoS
Requires CAFE	No	Yes	Yes
Ability to use COMET routing awareness information	No	Yes	Yes
Relative QoS (with traffic engineering)	No	Yes	Yes
Absolute QoS (with admission control)	No	No	Yes
Specialization for live and pre-recorded content	No	No	Yes
Multicast feature	No	Optional	Optional
Traffic description	None	Optional	Double token bucket

Table 1: Characteristics of COMET CoS

3.2 Content delivery in different network environments

3.2.1 Best effort network (Current Internet)

Current Internet offers only the best effort service, which does not provide QoS capabilities. As a consequence, the COMET system deployed over current Internet can support only two types of content delivery paths: BE and BTBE paths. The BE paths follow the paths available in the Internet, so once the server is selected, the content is delivered in the same way as in the current Internet. On the other hand, the BTBE paths are engineered and configured by the COMET system. Thanks to traffic handling mechanisms in CAFEs and traffic engineering algorithms in PMF, the BTBE paths may improve the content delivery when comparing with BE paths.

Obviously, content delivery in the current Internet can benefit from other features offered by COMET, e.g. anycast (server selection taking into account the server load).

3.2.2 Multi-service network environments

Given the existing paradigms in providing **multi-service network** environments, such as Differentiated Services (DiffServ) in packet forwarding, as well as new routing and path selection techniques for enabling service differentiation, we have investigated how the COMET content delivery platform can be gracefully built on top of these existing infrastructures, if necessary with lightweight adaptations or extensions. In addition to enabling service differentiation, it is also essential to consider overall network resource optimisation objectives in order to achieve end-to-end multi-service aware content delivery with cost-efficient network support. According to the common practice, Service Level Specifications (SLSs) are needed between the customer side and the network side which is responsible for specifying QoS parameters (such as delay and packet loss bounds). In case of end-to-end QoS across multiple autonomous networks, provider-level SLSs can be also established in order to "bind" the QoS capability of individual ISP networks. The outcome of such provider SLSs will also be used to drive the underlying (inter-domain) routing configuration for QoS enforcement. As far as the COMET paradigm is concerned, such operations are performed in an offline manner. When the content delivery services have been activated, the pre-provisioned paths (possibly multiple end-to-end paths between each source-destination network pair, each corresponding to a specific QoS level) can be dynamically used for content delivery according to dynamic network conditions.

In section 3.1 we have defined three types of Class of Services (CoS) in the context of the COMET based approach, namely *Premium (PR), Better-than-best-effort (BTBE) and Best effort (BE)*. Now the key issue is how these COMET-defined CoS can be efficiently mapped onto the underlying differentiated network capabilities. In the data forwarding plane, it is natural to use dedicated DiffServ code points (DSCP) to enforce specific packet forwarding treatments inside the network, as is the common practice today. It is important to note that it is up to individual ISP's options to define the mapping between DSCP values and provisioned QoS classes inside their own networks, although the identification of the COMET CoSs should be globally unique, i.e. recognised by all COMET-participating ISPs. As such, another level of code point mapping is necessary when COMET media flows are being delivered across network boundaries – the globally recognised COMET code point carried by the content packets should be specifically marked to the locally understandable DSCP values when they are injected into the next hop domain. As far as routing is concerned, as indicated previously, each COMET CoS is supported by dedicated offline path provisioning processes in order to enable service differentiation. Even within one single COMET CoS, it is also possible to provision multiple equivalent content delivery paths so that content packets can be adaptively transmitted across multiple paths, depending on the path quality/conditions in dynamic environments. Fundamentally, the offline engineering of the content delivery capability across multiple COMET CoSs is performed individually CoS by CoS without interference with each other. Certainly, more advanced approaches can be envisioned for more advanced usage of network resources, for instance bandwidth sharing or path sharing between different COMET CoSs. Such features will be potentially investigated in the project at a later stage.

The handling of the packets by CAFEs allows us to map the COMET CoSs into different technologies used for content delivery. In Table 2, Table 3, and Table 4, we present the exemplary mappings for **selected network technologies**.

COMET CoS	**Priority Code Point (PCP) value**	**Notes**
Premium	3 (Critical Applications) 4 (Video)	1) Live content should use PCP value 4; 2) Pre-recorded content should use PCP value 3; 3) Resource and admission control function should be applied before accepting the content delivery.
Better than Best Effort	2 (Excellent Effort)	No SLA type resource assurance – other traffic control mechanisms may apply. This service is intended for media-delivery applications which do not require strict QoS guarantees, e.g., P2P.
Best Effort	0 (Best Effort)	No resource assurance

Table 2: Mapping of COMET CoSs into Ethernet priorities [4]

COMET CoS	Service Class name	DSCP name and value	Notes
Premium	Broadcast Video	CS3 (011000)	1) Live content should use CS3; 2) Pre-recorded content should use AF3x; 3) Resource and admission control function should be applied before accepting the content delivery.
	Multimedia Streaming	AF31 (011010) AF32 (011100) AF33 (011110)	
Better than Best Effort	High-throughput Data	AF11 (001010) AF12 (001100) AF13 (001110)	No SLA type resource assurance – other traffic control mechanisms may apply. This service is intended for media-delivery applications which do not require strict QoS guarantees, e.g. P2P.
Best Effort	Standard	DF (000000)	No resource assurance

Table 3 Mapping of COMET CoSs into DiffServ Service Classes [7]

COMET CoS	Treatment Aggregate (TA)	DSCP name and value	MPLS EXP value	Notes
Premium	Real Time	CS3 (011000)	100	1) Live content should use Real Time TA; 2) Pre-recorded content should use Assured Elastic TA; 3) Resource and admission control function should be applied before accepting the content delivery.
	Assured Elastic	AF31 (011010) AF32 (011100) AF33 (011110)	010 011	
Better than Best Effort	Assured Elastic	AF11 (001010) AF12 (001100) AF13 (001110)	010 011	1) No SLA type resource assurance; 2) Domain supporting both PR and BTBE CoSes must guarantee no overlaps between 010 and 011 MPLS EXP values.
Best Effort	Elastic	DF	000	No resource assurance

Table 4 Mapping of COMET CoSs into DiffServ Treatment Aggregates with MPLS option [8]

4 Routing-awareness in the COMET System

The objective of routing awareness process is to find content delivery paths. The routing awareness is an off-line process performed in long time scale. It reacts to changes in inter-domain network reachability or re-provisioning of domains. The routing awareness is performed by Routing Awareness Entities (RAE), which should be located in every COMET domain a presented on Figure 3. Comparing to standard inter-domain routing protocol, like BGP-4 [5], the RAE offers two new features: (1) multi-criteria (QoS) routing which allows RAE to build routing paths taking into account the requirements of COMET CoSs and (2) multipath routing as RAE builds a set of routes going towards a given prefix. These features provide COMET more information about the network and inter-domain routing paths, which enable COMET to deliver content with respect to QoS requirements as well as improve utilisation of network resources, e.g. by load balancing. The RAE provides information about discovered routes and their properties to the Path Storage component of CME [1]. This information is used by decision maker during content resolution process to select the best path for content consumption.

Each RAE requires information about its domain, which among others covers: Autonomous System (AS) number, available network prefixes, IP addresses of peering RAEs, as well as supported COMET CoSs and values of their QoS parameters as maximum IP Packet Transfer Delay (IPTD), maximum IP Packet Loss Ratio (IPLR), maximum bandwidth (BW) for single content consumption assured between any ingress and egress points of its domain. Note, that values of QoS parameters should be valid in long term constituting an upper bound for actual values. This information should be provided by Domain Management System based on domain configuration, domain provisioning as well as SLA agreements between peering domains. In COMET prototype, this information would be provided by RAE configuration file. Then, the RAE exchanges UPDATE or WITHDRAW messages with other RAEs located in peering domains to build content delivery routes. Each content delivery route is characterised by the list of AS numbers, supported COMET CoS and aggregated values of QoS parameters. The QoS parameters are calculated from the egress CAFE of a source domain towards a given prefix. Moreover, RAE exchanges KEEP-ALIVE messages between peers in order to detect failures of peering RAEs. Whenever any change is detected, i.e. peer failure or domain re-provisioning, routing information is updated and propagated to the entire network.

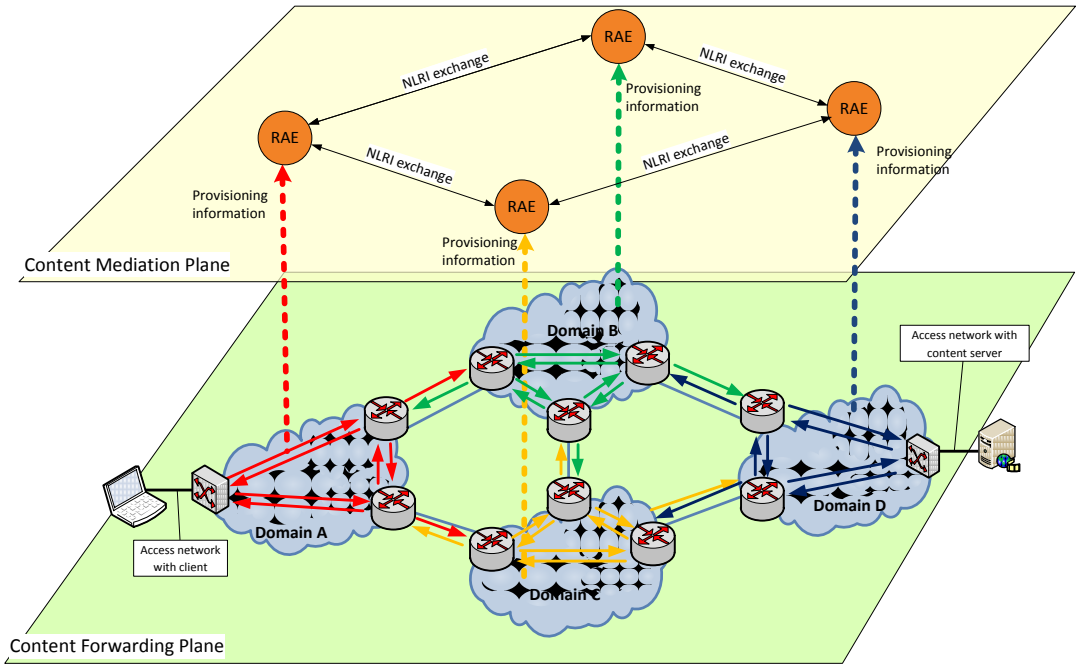


Figure 3: Routing awareness and provisioning

Each RAE maintains two routing information tables that are named: *Known Routes Table* (KRT) and *Preferred Routes Table* (PRT).The former stores information about routes that have been advertised by peering RAE, while the latter stores the routes preferred by the domain. The RAE uses a route ranking algorithm to evaluate all known routes and to select a set of preferred routes. In multi-path approach, the route ranking algorithm selects a number of preferred routes going towards the same network prefix.

Let's focus now on RAE operations, that is, the actions performed by the RAE after reception of UPDATE, WITHDRAW or KEEP-ALLIVE messages:

1. When the RAE receives UPDATE message from peering domain:
    a. It performs sanity check of UPDATE message to remove routes that contain its own AS number. This process allows us to avoid routing loops.
    b. It updates properties of the routes received in UPDATE message, i.e., path length, list of domains, aggregated QoS parameters, with values corresponding to its domain and then it stores routes in KRT. It is worth to mention that this process may also remove the previously advertised routes, when received UPDATE has fewer routes than the previous one.
    c. It invokes the route ranking algorithm to select the new preferred routes and prepare new *PRT*.
    d. If *new PRT* is identical to the old one, the process is finalized. Otherwise, the RAE updates PRT and sends UPDATE message with the currently preferred routes to its peers. Then, the process is finalized.
2. When the RAE receives WITHDRAW message from peering domain:
    a. It removes routes from the KRT.
    b. It invokes the route ranking algorithm to prepare new *PRT*.
    c. If *new PRT* is identical to the old one, the process is finalized. Otherwise, the RAE updates PRT and sends WITHDRAW message to its peers with the set of routes, which should be removed. Then, the process is finalized.
3. When the RAE receives KEEP-ALIVE message from peering domain:
    a. It restarts keep_alive timeout.

    Note that when keep_alive timer expires, the RAE treats peer as unavailable and:

    a. It removes all routes from the KRT that were received from this peer.
    b. It invokes the route ranking algorithm to prepare new *PRT*.
    c. If *new PRT* is identical to the old one, the process is finalized. Otherwise, the RAE updates PRT and sends WITHDRAW message to its peers with the set of routes, which should be removed. Then, the process is finalized.

Route ranking algorithm

The objective of the route ranking algorithm is to select the set of preferred routes from the list of known routes stored in the KRT. Note that in inter-domain network, there may be number of feasible routes that meet the requirements of COMET CoSs and have similar properties, e.g., similar length of the routes, similar QoS parameters. The route ranking algorithm evaluates the feasible routes using specific cost function, *cost_f(.)*. The latest studies related to multi-criteria routing presented in [9], [10] and [11] pointed out that the most effective cost functions are nonlinear, strict monotonic and convex functions. In our approach we consider two cost functions.

Option 1:

$$cost\_f(.) = \sum_{k=IPTD,IPLR} \frac{a_k / r_k}{\max(0, 1 - q_k / r_k)},$$

where parameter q_k is k-th parameter characterising the route, r_k is reservation level for kth parameter, and a_k is aspiration level for k-th parameter.

Option 2:

$$cost\_f(.) = \max_{i \in availalbe\ paths} \left[\min_{k=IPTD,IPLR} \frac{q_k - r_k}{a_k - r_k} \right],$$

where parameter q_k is k-th parameter characterising the route, r_k is reservation level (constraint) for k-th parameter, and a_k is aspiration level for k-th parameter.

Option 1 uses weighted sum of inverse functions related to each parameter. This function assures that the cost of path significantly increases when a given decision variable becomes close to the reservation level. This feature allows to avoid paths that are close to the limits. The second option exploits the multi-criteria decision algorithm with the reference point (similar algorithm is used in COMET decision process). This algorithm ranks paths based on decision variable, which has the worst value. The effectiveness of considered cost functions will be evaluated in simulation experiments.

The cost function ranks available routes. The set of N preferred routes will be advertised to the peering RAE entities. These routes may be similar from the point of view of the assumed cost function but they could differ essentially in other criteria, e.g. the number of disjointed domains. We argue that the set of preferred routes should be limited to 2-5 routes in order to limit the route table size. The proposed route ranking algorithm follows three steps:

1. It splits the KRT into sets of routes going towards a given prefix or domain and belonging to the same COMET CoS. For each set, the algorithm removes routes that are dominated by any remaining route.
2. The algorithm ranks the routes in each set using the cost function.
3. Finally, the algorithm prepares the subset of preferred routes based on ranking and other optimisation criteria, like, e.g., load balancing. This step finalises the algorithm.

For route ranking algorithm the following parameters should be configured: K – the number of parameters, r, a– vectors of values related to reservation, aspiration levels, the identifier of cost function, N – the number of advertised paths.

4.1 RAE interfaces

The interfaces cover: (1) configuration file, which provides information about domain configuration, provisioning as well as SLA agreements between peering domains; (2) RAE-to-RAE interface used to exchange messages between peering RAEs and (3) RAE-to-CME interface, which is used to provide discovered path to Path Storage component.

4.1.1 Configuration file

The configuration file includes all parameters required for RAE. The configuration file is stored in a binary file following *protobuf* encoding. The information included in RAE configuration file covers:

- The number of local AS,
- Local IP address and port number for running TCP listening socket,
- Value of keep-alive timeout,
- Value of reconnection timeout,
- Minimum interval for advertising update messages,
- Limit for the number of preferred routes maintained for single destination prefix,
- IP address and port number for connecting to CME interface,
- Type of route ranking algorithm,
- A list of reservation levels r_k for particular parameter k=IPTD, IPLR,
- A list of aspiration levels a_k for particular parameter k=IPTD, IPLR,
- A list of peering RAEs:

- IP address and port number of the remote RAE,
- AS number of remote RAE,
- Time-to-live value, which should be maintained for the TCP connection,
- List of provisioned QoS parameters for the traffic sent to peering domain:
  - Class of Service identifier,
  - Metric value for: IPTD, IPLR, and BW for a single content consumption.

- A list of local prefixes corresponding to local access networks:
  - IP address and a prefix length
- A list of provisioning information for intra-domain part
  - Source defines the origin of traffic. It can be either a local prefix or a peering.
  - Sink defines the destination of traffic. It can be either a local prefix or a peering.
  - List of provisioned QoS parameters for the traffic sent from source to sink:
    - Class of Service identifier,
    - Metric value for: IPTD, IPLR, and BW for single content consumption.

Note that in commercial deployment of COMET, this information should be provided by Domain Management System. The exemplary configuration file is included in Annex A.

4.1.2 RAE to RAE interface

RAEs exchange 4 messages between themselves that are: OPEN, UPDATE, WITHDRAW, KEEP_ALIVE. The UPDATE, WITHDRAW and KEEP_ALIVE messages are sent by RAE asynchronously, while the OPEN message requires confirmation from peering RAE. Figure 4 presents sequence of messages exchanged between RAEs.

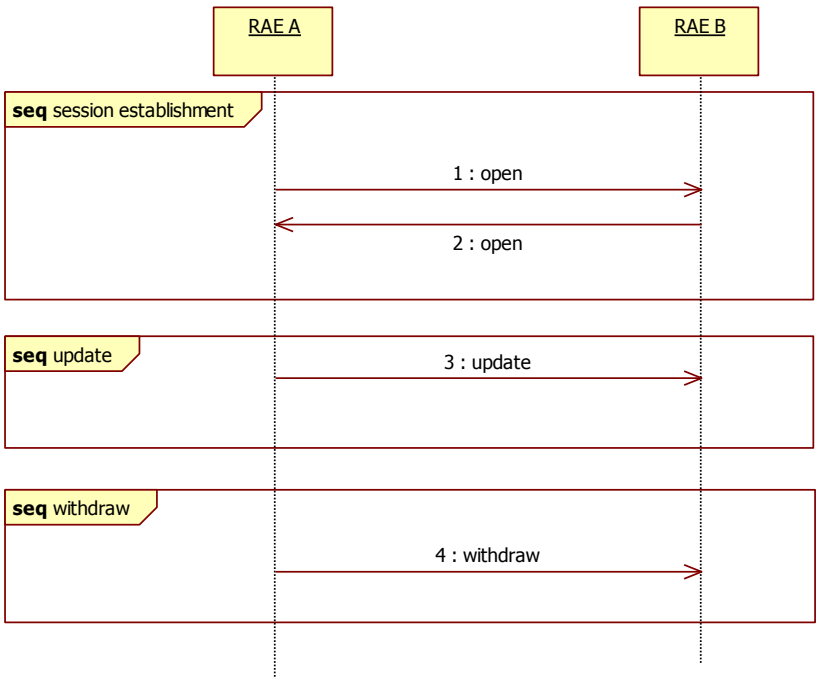


Figure 4: Sequence of messages exchanged between RAEs

The OPEN message is used to initiate the session between peering RAEs. Once the OPEN message is received, the RAE responses with its own AS_number. The structure of OPEN message is the following:

```
OPEN{
      as_number– the AS number of sending RAE
}
```

The KEEP_ALIVE message is used to keep going session between peering RAEs. Once the KEEP_ALIVE message is received, the RAE reset the keep_alive timer. The structure of KEEP_ALIVE message is the following:

```
KEEP_ALIVE{
}
```

The UPDATE message is used to advertise: (1) new prefixes, when they appear within domain, (2) new paths going towards already known prefixes, as well as (3) updates of path's properties, e.g. values of QoS parameters after domain re-provisioning. In addition, the UPDATE message may be also used for implicit path removal. The structure of UPDATE message is the following:

```
UPDATE{
      List of UpdateEntry {…}
}
      UpdateEntry {
            Prefix {} – this structure defines the network prefix
                  available through the advertised route. It
                  includes the IP address and prefix length.
            CoS_id – this is identifier of COMET CoS supported on
                  the advertised route .
            List_of_routes {…} – this structure defines the list of
                  routes going towards advertised prefix.
}
      Route {
            List of as number {…} – this structure defines route as
                  the ordered list of AS numbers,
            Metric {…} – this structure defines metric corresponding
                  to route. This matric includes the following
                  parameters: IPTD, IPLR, and BW for a single
                  content consumption. The values of these
                  parameters correspond to end-to-edge relation.
}
```

The WITHDRAW message is used to explicitly remove a given prefix or a set of paths which becomes no longer available. The RAE uses WITHDRAW message in two cases: (1) to remove a given network prefix. In this case, the source domain sends WITHDRAW message to all peering domains, or (2) to remove a set of paths belonging to a given branch of destination sink tree. In this case, the domain at the beginning of the branch sends WITHDRAW message uphill along the branch.

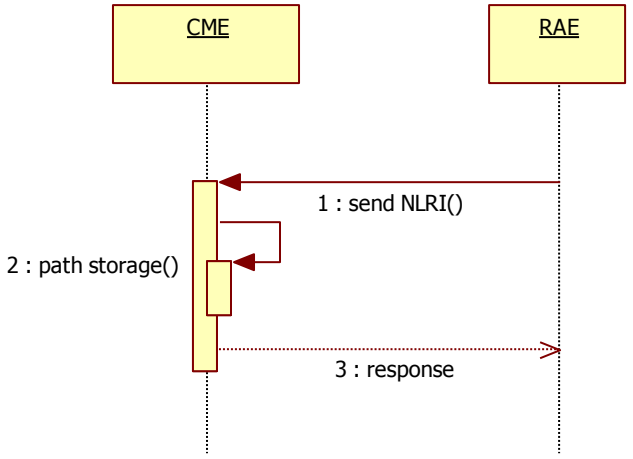


Figure 5: Message sequence chart exchanged between RAE and CME

The structure of WITHDRAW message is the following:

```
WITHDRAW{
      List of WithdrawEntry {…}
}
      WithdrawEntry {
            Prefix {} – this structure defines the network prefix
                  available through the advertised route. It
                  includes the IP address and prefix length.
            CoS_id – this is identifier of COMET CoS supported on
                  the advertised route.
            List_of_routes {…} – this structure defines the list of
                  routes going towards given prefix.
}
      Route {
            List of AS numbers {…} – this structure defines route as
                  the ordered list of AS numbers.
}
```

4.1.3 RAE to CME interface

RAE provides information about routes and local domain provisioning to the Path Storage component in CME. Figure 5 presents the message sequence diagram.

The RAE sends 6 types of messages that are summarized in Table 5.

4.2 Consideration on RAE deployment

This section provides considerations for deployment of RAEs. The routing awareness process performed by RAEs is similar to the current inter-domain routing performed by BGP speakers in the Internet. However, there are some differences which have direct implications in the deployment. In the current Internet, inter-domain routing and forwarding are coupled in the Border Routers so that they are in charge of the exchange of inter-domain routes as well as the packet forwarding. However, in COMET the routing and forwarding processes are decoupled so

send NLRI	Response	Expected CME behavior
(RESET)	version number	Remove all entries from the Path Storage
(KEEP_ALIVE)	version number	Reset the keep_alive timer
(INSERT_PROVISIONING_INFORMATION) • source (prefix or AS number) • destination (prefix or AS number) • CoS_id (name: PR, BTBE, BE) • QoS parameters (vector of float values)	version number	Add provisioning information to the Path Storage (Type 1 data).
(REMOVE_PROVISIONING_INFORMATION) • source (prefix or AS number) • destination (prefix or AS number) • CoS	version number	Remove provisioning information from the Path Storage (Type 1 data).

(INSERT_PATH_INFORMATION)
• prefix (IP address and length of prefix)
• CoS_id(name: PR, BTBE, BE)
• edge_identifier (AS number)
• Vector of:
  ○ as_path (list of AS numbers)
  ○ Metric {…} - IPTD, IPLR, and BW for a single content consumption.
	version number	Add path information to the Path Storage (Type 2 data).

(REMOVE_PATH_INFORMATION) • prefix (vector of bytes + length of prefix) • CoS (name: PR, BTBE, BE) • edge (AS number)	version number	Remove path information from Path Storage (Type 2 data).

Table 5: Messages exchanged between RAE and CME

that RAEs are in charge of the routing awareness process (exchange of NLRIs), while CAFEs are in charge of forwarding. Moreover, since the routing awareness protocol expresses next hop in terms of next AS and not in terms of next router, only one inter-domain speaker is needed (or two in order to provide resilience) for the exchange of routing information, whereas in the current Internet all Border Routers need to participate in the exchange of NLRI with eBGP[1]. RAEs are expected to interact with:

- Other RAEs in peering domains to exchange routing information (including QoS path characteristics and multiple paths) with other RAEs in COMET peering domains.
- CMEs/CRMEs in the same domain, in order to propagate the previous information.

Regarding deployment of RAEs, it can be different depending on the specific approach for content resolution. In the decoupled approach, the decision process is performed by the client's CME, which must be aware of any route changes. Therefore, route changes (updates and withdrawals)

[1] In the current Internet, it is not strictly necessary that BRs speak eBGP. Instead, RRs or BGP routers different from BRs can speak eBGP with other domains. This is done by configuring "ebgp-multihop" field accordingly. However, this is not a common practice since it makes the configuration difficult and can cause instabilities.

must be propagated to all CMEs (see Figure 6). On the other hand, not all domains need to deploy RAEs.

In the coupled approach, where content resolution is performed on a hop by hop basis, the decision process is performed in each CRME along the resolution path based on the information propagated during the content publication. In practice, this means that the decision is taken by upper tiers. Therefore, route changes should not reach all CRMEs, but only those in upper tiers (see Figure 7). On the other hand, due to the hop-by-hop nature of content resolution, it seems appropriate a priori that all domains deploy RAEs.

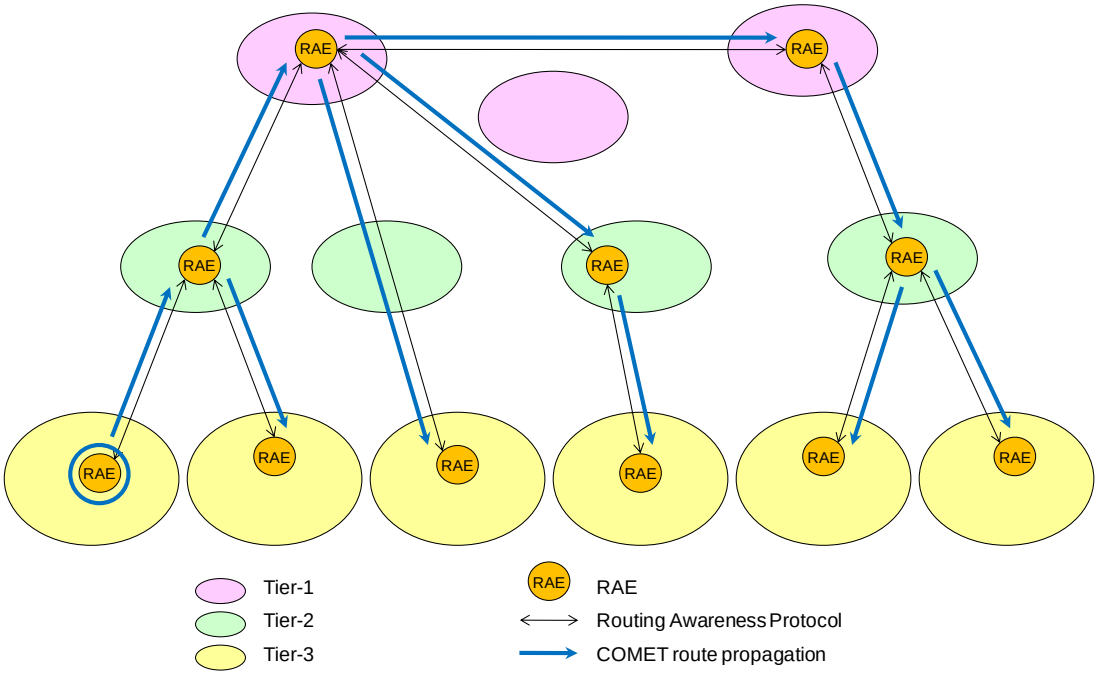


Figure 6: Deployment of RAEs in the decoupled approach

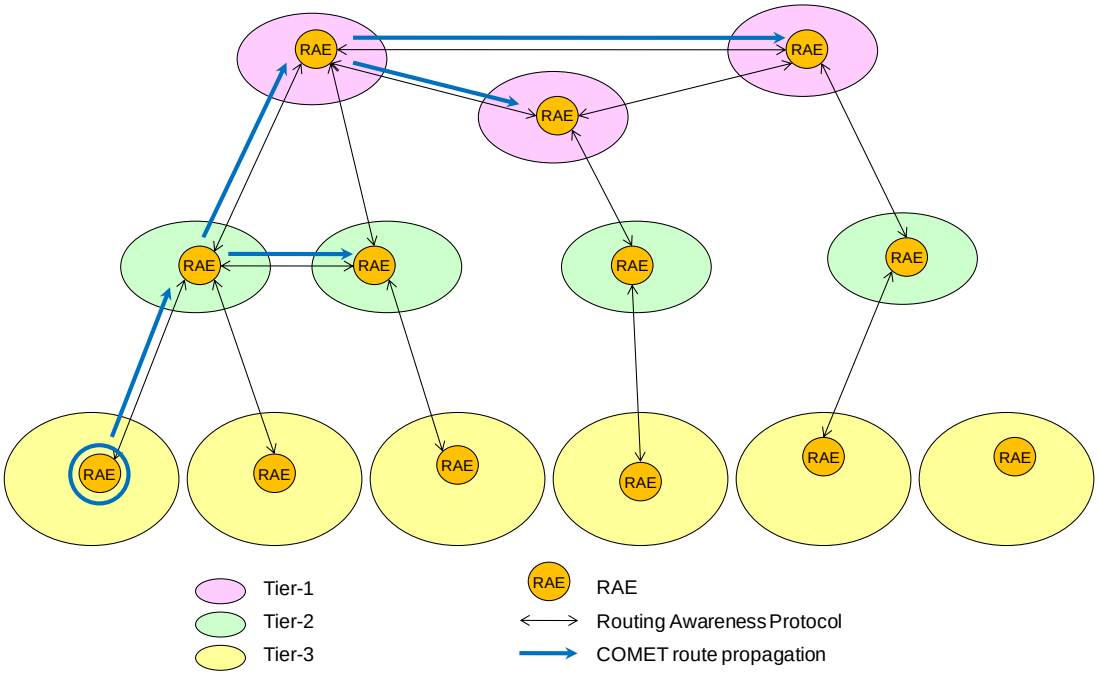


Figure 7: Deployment of RAEs in the coupled approach

5 Stateless Content-aware Forwarding

The stateless content delivery assumes that CAFEs maintain only the neighbourhood (local) information, i.e., how to forward packet to the peering CAFEs. All information about content delivery path is stored in a COMET header attached to the original packet containing content payload. The COMET header includes information of the end-to-end path, which allows CAFEs to forward packets along the selected path. The COMET header is attached and removed by edge CAFEs located close to the content server and the client, respectively. The path selected during resolution process is configured in the edge CAFE during the path configuration process [1].

The stateless content delivery follows the source routing principle at the domain level, which allows for flexible selection of content delivery path for each content request. Furthermore, the proposed approach is technology-agnostic in the sense that it allows each domain to use different packet forwarding technologies between peering CAFEs. In our approach, the end-to-end path is made up by a chain of content delivery path segments between neighbouring CAFEs.

Figure 8 presents the concept of stateless content delivery process. In this example, we assume a simple network consisting of a client domain, a server domain and two transit domains. The content delivery path selected during content resolution process goes through domains C-B-A, which is different than the IP routing path.

In order to deliver content through path C-B-A, the edge CAFE located in domain C intercepts the data packets generated from the content server and embeds the COMET header into them. It includes the list of forwarding keys. Each forwarding key determines the next CAFE as well as the required encapsulation to forward packet to the next CAFE. Note that the forwarding key has only local meaning within a given CAFE. Consecutively, each CAFE on the content delivery path draws successive forwarding key from the COMET header and encapsulates the packet following specific forwarding technology used between CAFEs. In our example, the forwarding key "2" enables forwarding of data packets from the edge CAFE located in domain C towards the CAFE located in domain B. Then, this CAFE uses next forwarding key, "3", to deliver packets to the edge CAFE located in domain A. Finally, the last CAFE on the list of forwarding keys removes the COMET header and sends data packet directly to destination.

The proposed approach allows flexibly using any packet forwarding technology and applying specific packet processing within CAFE. In particular each domain may define a set of locally specified classes of service and use specific forwarding keys to instruct CAFE how assign packets into them.

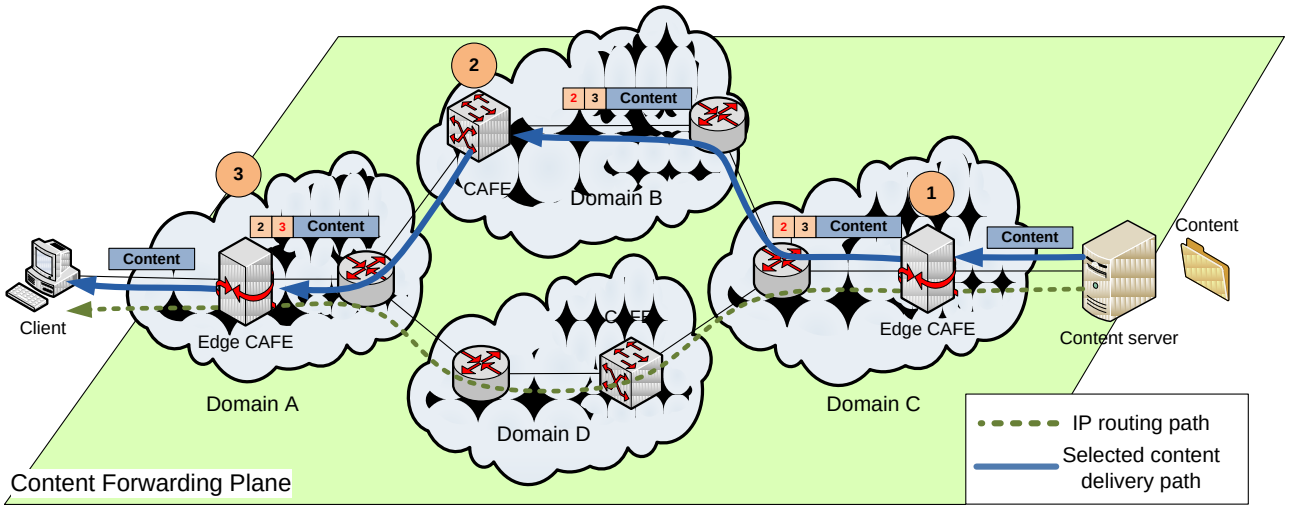


Figure 8: The concept of stateless content forwarding

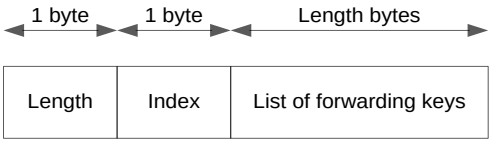


Figure 9: Structure of the COMET header

Although content forwarding is based on the stateless principle, the edge CAFE located at the server side has to classify packets belonging to given content flow and assign them appropriate values of the COMET header. This information is provided by the CAFE configuration component during the path configuration process.

Below we present the specification of COMET header and packet processing performed by CAFEs.

5.1 COMET header

The stateless approach uses the COMET header to carry information about content delivery path. The structure of COMET header is presented in Figure 9.

The COMET header covers following fields:

- Length (1 byte) indicates the length of the forwarding keys list. Allowed values: {0, ..., 255}. Note that length of list field is correlated with the length of content delivery paths. Typically, we need two CAFEs to cross a domain, so we need 2 bytes per domain. Thus, we need about 6-10 bytes for typical inter-domain paths.
- Index (1 byte) indicates the active position of in the Key field. Allowed values: {0, ..., Length-1}.
- List of forwarding keys (exactly Length bytes) includes forwarding keys of content delivery path.

Figure 10 presents mapping of COMET header into 4 basic forwarding technologies that are: IP over Ethernet, IP over VLAN Ethernet, IP over GRE Ethernet and IP over MPLS Ethernet.

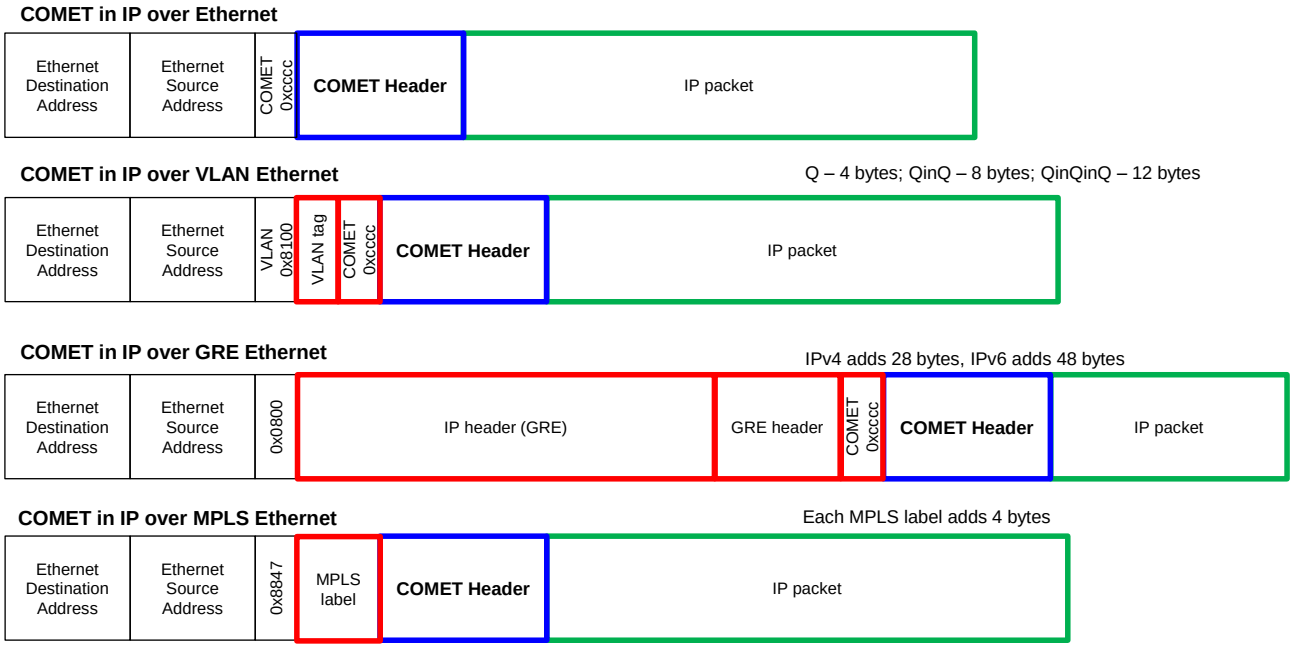


Figure 10: Mapping of COMET header into different forwarding technologies

In case of COMET in IP over (VLAN) Ethernet, the processing of COMET header requires definition of new value of *Ether_type*, which invokes processing of COMET header. In our

prototype we used value 0xcccc, which is not yet assigned by IANA. This value should also be added after VLAN tag. In case of COMET over GRE Ethernet, the COMET header is added as payload for GRE (Generic Routing Encapsulation) packet. We should also use COMET specific value of *Ether_type*. Otherwise, the sink of the GRE tunnel will assume it is plain IP packet. In case of COMET in IP over MPLS Ethernet, the COMET header is added as payload for MPLS (Multi-Protocol Label Switching) frame. The MPLS node, which closes the MPLS tunnel, must be aware that payload is encapsulated in COMET header.

Note that the COMET header will slightly increase the payload of Ethernet frames over the 1500 bytes. This could be a problem for domains that do not accept the jumbo frames or at least baby jumbo frames. In such a case, we need to reduce the MTU (Maximum Transmission Unit) at the server side.

Figure 11 presents the example of COMET header processing in a CAFE. Each CAFE consumes 1 byte of the key from COMET header, i.e., {129}. It uses this fragment of a key as index to lookup forwarding table, where forwarding behaviour is described, e.g., the output interface and a set of parameters required for encapsulation. Finally, the *Index* field must be increased by 1 to point out on the next key.

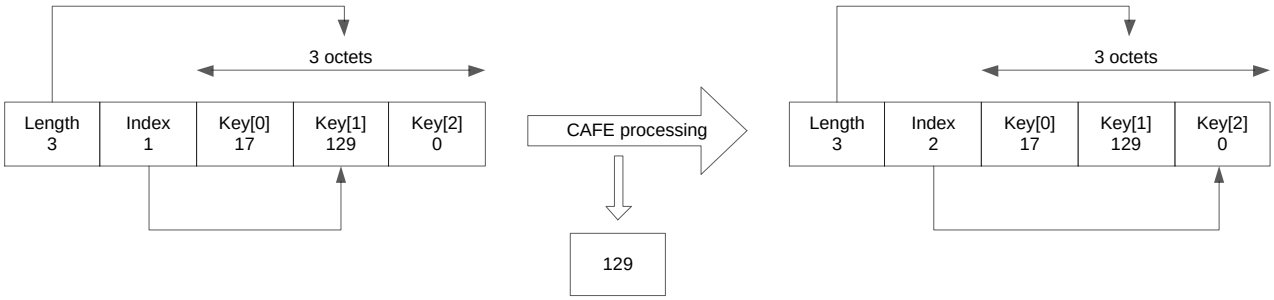


Figure 11: COMET header processing

5.2 Packet processing

Each CAFE has the forwarding table To *(interface, key)→(forwarding rule)*. It is used to forward packets to appropriate output interface using specific forwarding technology. The forwarding is based on the key value included in the COMET header. The details of this mapping are as follows:

- *Interface* identifies the link that the packet arrived from.
- *Key* is local identifier (typically one byte), which indexes forwarding table in CAFE. Key determines forwarding rule, which should be applied to forward packet to next CAFE. Note, that all *keys* in the CAFE should form a proper prefix tree for given *interface*, i.e., a *key* cannot be a prefix of any other *key*. The CAFE should use one of 2 rules for mapping: 1) longest matching prefix over key or 2) constant length matching over key.

Interface	Key	Forwarding rule
#1	{0xa1, 0xbb}	Send packets via interface #9 using MPLS label 16
#9	{0xaa}	Send packets via interface #1 using Ethernet frame with destination address a1:01:02:ff:01:d3

Table 6: Exemplary forwarding table To

- *Forwarding rule* describes "how to forward packet". It depends on forwarding technology used between CAFEs. For each CAFE, forwarding rule ensures that packets reach next CAFE in proper way.
  Note, that information included in forwarding To should be provided by the Domain Management System as a result of the domain provisioning and CAFE configuration. In our prototype, each CAFE will read this information from configuration file.

Upon receiving packet with COMET header, the CAFE performs following steps:

1. Register the input interface *if_in*.
2. If *Index* field is greater than *Length* field, drop the frame.
3. If *Index* is equal to *Length* field, remove the COMET header and send it as IP packet.
4. Use *if_in* and *key* pointed by *Index* to find forwarding rule in table To.
5. Increase *Index* to point out on next forwarding key
6. Process the frame accordingly with forwarding rule and send the packet to appropriate output interface.

The edge CAFE performs additional functions related to data packet interception and mapping them to the particular content delivery path. The request for installation of the classification rule requires following parameters:

- *Multi-field filter* with the following fields:
  - IP protocol number: 4 or 6 (mandatory).
  - Source address, IPv4 or IPv6 (mandatory).
  - Destination address, IPv4 or IPv6 (mandatory).
  - Protocol (IPv4) or next header (IPv6): 1 to 255 (mandatory).
  - Source port number: 0 to 65535 (optional). Value 0 indicates wildcard matching (any port number matches).
  - Destination port number: 0 to 65535 (optional). Value 0 indicates wildcard matching (any port number matches).
- *Key* for COMET header. Up to 255 octets.
- *Timeout* which limits the validity of the interception rule. Integer indicating number of seconds.
- Optionally, other information, e.g., session identifier, traffic profile, etc. This information is used for traffic policing, multicast, etc.

Intercepted and classified packets are prefixed with COMET header containing provided forwarding *Key*. Each intercepted packet refreshed the validity of the classification rule. If the rule validity is not refreshed for *Timeout* duration, interception rule would be removed. Optionally, CAFE may report removal of the classification rule to CME.

Optional features

The CAFEs may not use the whole key from COMET delivery header. This field can be used to propagate other information related with this session, e.g., session identifier assigned by client-side CME. This session identifier may be used for identification of local multicast tree identifier, masquerading, etc.

The CAFE also measure the traffic carried by content delivery paths. In particular, it measures number of bytes and packets transferred through given content delivery path identified by forwarding keys. This information would be provided to COMET Server and Network Monitoring (SNME) entity, to support decision maker and path configuration process.

6 Stateful Content-aware Forwarding

The state-based content delivery process is carried out in a hop-by-hop[2] fashion following states installed within the content-aware forwarding entities (CAFEs) by the content resolution and mediation entities (CRMEs) during the content resolution process described in D3.2 [1]. Every domain along the content resolution path will therefore typically contain two states relating to each content delivery session, one in the ingress CAFE and one in the egress CAFE. In order to avoid keeping a user waiting after a content request, content is delivered immediately after the resolution process reaches either the content server or a point *en route* where the content is already cached. However, the process of content delivery initiates within each domain a separate background process of route optimisation, which aims to improve the route taken by the content from the content server to the client over the basic customer-provider route used in the resolution phase (see D3.2, §4.5) [1].

In the following sections, the content delivery path configuration step is described, followed by the specification of both the content delivery and route optimisation processes.

6.1 Content Delivery Entities

The primary entity involved in content delivery is the content-aware forwarding entity (CAFE). CAFEs are able to natively process content packets according to their IDs.

Figure 12 shows the internal architecture of a CAFE, which consists of the following three logical components:

Content State Table – stores state information relating to content streams that the CAFE is currently handling. The basic table structure is given in the next section.

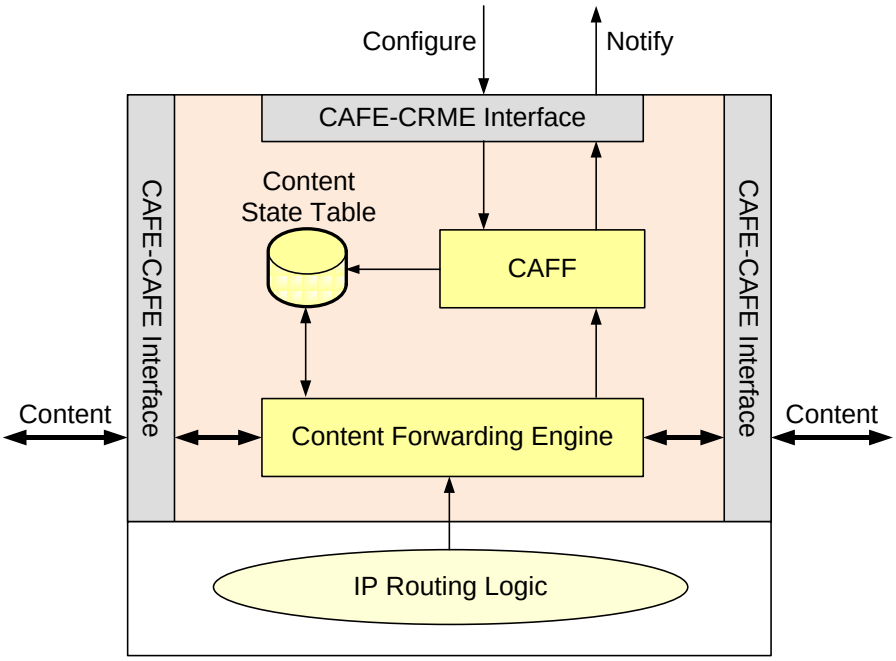


Figure 12: Internal CAFE architecture

[2] 'Hop' here refers to CAFE-level hops.

Content Forwarding Engine – forwards content it receives from uphill CAFEs to downhill CAFEs, in accordance with the states contained in the Content State Table.

Content-Aware Forwarding Function (CAFF) – logically interfaces the CAFE with its local CRME to allow for installation of states and to send notifications of new content flows.

Communication between CAFE components and external entities is carried out over two main interface types. The CRME-CAFE interface connects the CAFE with its local CRME to allow it to send Notify messages and receive Configure messages. The CAFE-CAFE/Router interface connects the CAFE to its neighbouring CAFEs, both within its domain and within neighbouring domains.

6.2 Content Delivery Path Configuration during Content Resolution

According to our design, content delivery paths are enforced in a receiver-driven multicast manner that needs state maintenance based on content identifiers. As described in D3.2 [1], content consumption requests from clients are resolved through a sequence of content resolution and mediation entities (CRMEs) residing in individual domains according to either the business relationships between ISPs (in *wildcard* and *filtering* modes) or the BGP reachability information on the scoped source prefix (in *scoping* mode). In both cases, once a CRME has passed the content consumption request to its next hop counterpart in the neighbouring domain, its content mediation function (CMF) configures the local content aware forwarding entities (CAFEs) that will be involved in the delivery of content flows back from the potential server. Specifically, as shown in Figure 13, once a CRME receives a content consumption request from its counterpart in the previous hop domain and forwards it towards the next-hop CRME, it sends a `Configure` message to both the local egress and ingress border CAFEs[3] connecting the two neighbouring domains, triggering them to install the appropriate content state within their content state tables[4]. In case of failed content resolution, the content states installed in CAFEs can be either left to eventually time-out, or explicitly torn down by the CRME upon return of an error message. It is worth mentioning

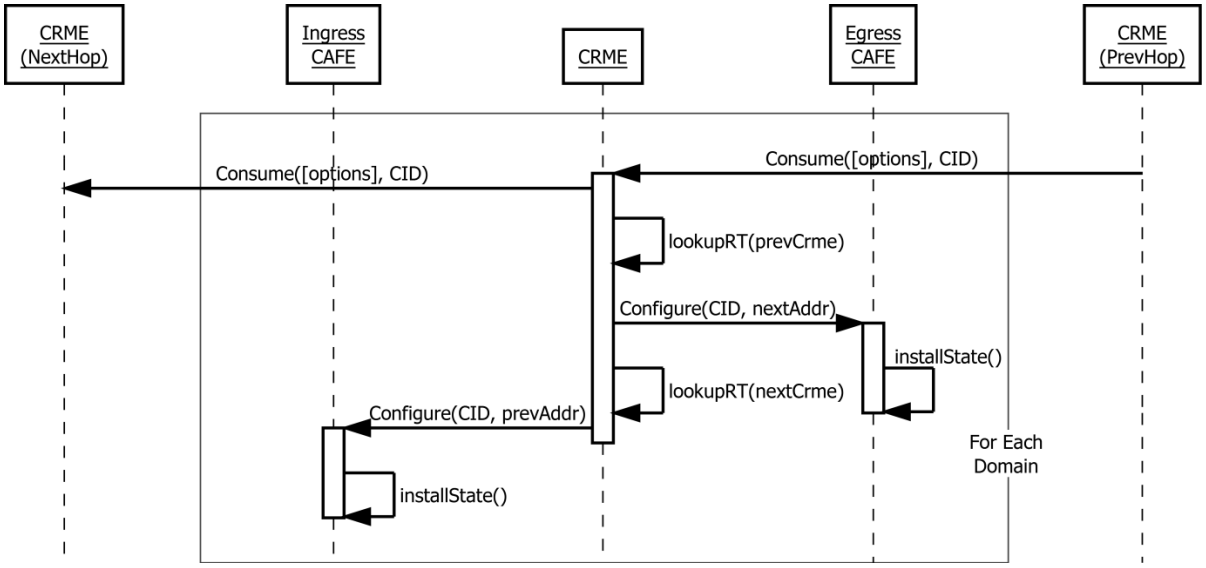


Figure 13: CAFE content state installation process in each domain

[3] The determination of ingress/egress CAFEs for each content consumption request is based purely on the BGP reachability information [12].

[4] Within each domain, the communication between the non-physically connected ingress and egress CAFEs can be achieved either by establishing intra-domain tunnels that traverse non-content-aware core IP routers, or natively through the content-centric network routing protocols.

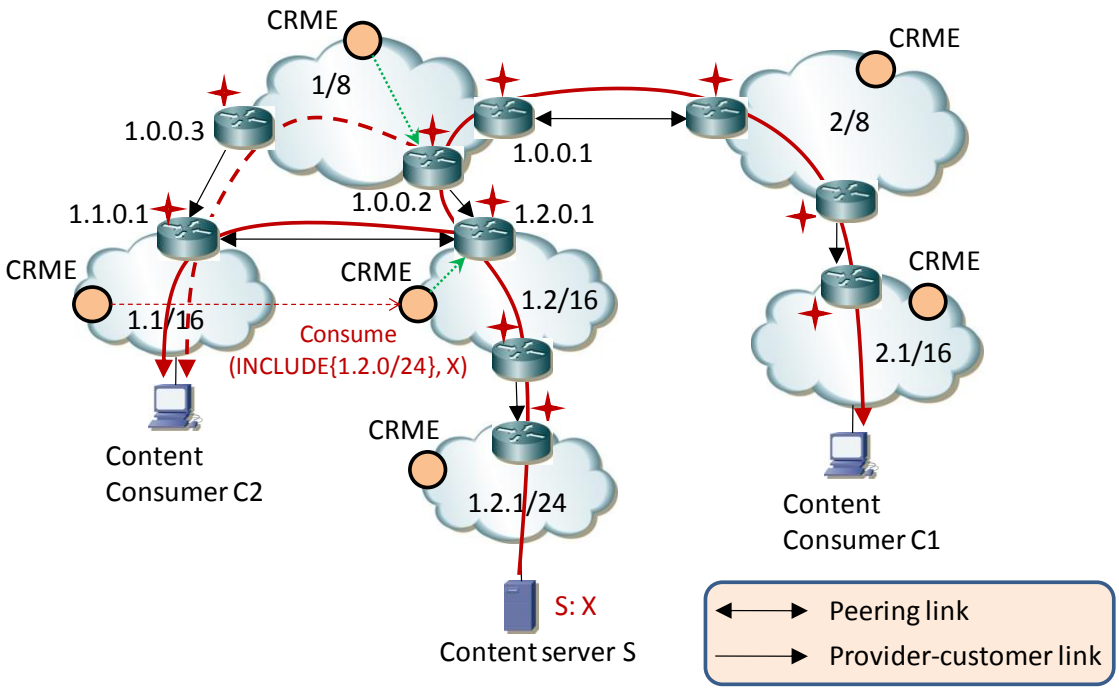


Figure 14: Multicast-based content delivery process

here that CRMEs do not directly constitute the content delivery paths, therefore the configuration interaction between the CRME (specifically the content mediation function, CMF) and local ingress/egress CAFEs is necessary.

To illustrate the content path configuration process carried out during content resolution, consider the example scenario shown in Figure 14. In this example, content client **C1** (attached to domain 2.1/16) is currently consuming a live streaming content **X** from server **S** (attached to domain 1.2.1/24). The content delivery path traverses a sequence of intermediate domains, and each of the corresponding ingress/egress CAFEs contains a content state (represented by a star) relating to that content delivery path. As previously mentioned, such content states are configured by the local CRMEs (the CMF function) during the content resolution phase. Now, content client **C2** (attached to domain 1.1/16) issues a content consumption request for the same content to its local CRME. Upon receiving the content consumption request, the local CRME forwards it to its provider counterpart in domain 1/8 (uphill), as it is not aware of the content source location. It also sends a `Configure` message to ingress CAFE 1.1.0.1 to trigger it to install the corresponding content state pointing towards **C2**. Meanwhile, since the CRME in domain 1/8 knows (from the ingress CAFE notification – see §6.4) that content flow **X** is being injected into the local network via the originally configured ingress CAFE 1.0.0.2, it updates its *outgoing next-hop CAFE list* by adding a new egress 1.0.0.3 leading towards content client **C2**. As a result, a new branch is established from CAFE 1.0.0.2 which is responsible for delivering the content back to the new client **C2** (shown by the dashed line), but without any further content resolution process.

According to our design, the basic forwarding state format maintained at each CAFE is shown in Figure 15. The *content ID* entry specifies the identifier of a specific piece of content being delivered, which is similar to the concept of an IP multicast address. The *previous-hop CAFE address* entry indicates the previous-hop CAFE from which the content was forwarded. This can be either an

Content ID	Previous-hop CAFE address	List of next-hop CAFE addresses

Figure 15: Content state table structure within each CAFE

ingress CAFE belonging to the same domain (e.g. in Figure 14 the *previous-hop CAFE address* of 1.0.0.3 for *content ID*, X is 1.0.0.2, which is effectively the ingress CAFE in the network 1/8 for X), or an egress CAFE belonging to the previous hop domain (e.g. in the same figure, the *previous-hop CAFE address* of 1.0.0.2 for *content ID*, X is 1.2.0.1 which is the egress CAFE in the network 1.2/16 for X). The *previous-hop CAFE address* entry is required here to ensure that content is received from the expected neighbouring CAFE, in a manner similar to the reverse path forwarding (RPF) [13] in IP multicast. Finally, the entry, *List of next-hop CAFE addresses* indicates the sequence of next hop CAFEs to which the local CAFE should send the content. For instance, for content **X**, the list of next-hop CAFEs maintained by 1.0.0.2 include 1.0.0.1 and 1.0.0.3 leading towards specific downstream content consumers. A next-hop CAFE can be either a (egress) CAFE within the same domain or a (ingress) CAFE belonging to the next-hop domain.

6.3 Content forwarding

Once states have been installed in all CAFEs along the content delivery path, the content server, or any cache *en route* will forward the requested content to the content client. Figure 16 shows the content delivery process that takes place after the resolution phase. Once the content consumption request reaches the content server or any intermediate node that contains the requested content in its cache, the content is forwarded to the ingress CAFE of the next-hop domain towards the client. This CAFE will then read the ID of the content it has received and look up in its content state table the next egress CAFE hop(s) within its domain to which to forward the content. When the egress CAFE(s) have received the content, it will carry out the same procedure as the ingress CAFE and forward the content to the ingress CAFE(s) of the neighbouring domain(s). Meanwhile, the ingress CAFE will also look up in its active session table to check if it has informed its local CRME that it has received the content that was requested. This is primarily done for the purpose of route optimisation, which is described in the following section. The ingress and egress CAFEs in each domain will carry out the same process, until the content reaches the client.

Since the content forwarding mechanism is state-based, content is forwarded in a similar style to that of standard IP multicast, but only at the level of individual CAFEs at the edge of each domain. However, despite this similarity, it is important to note a key difference: in IP multicast it is individual routers that are responsible for sending a group join request towards the targeted source based on their own knowledge about the location of the content source (or rendezvous point), typically following the default IP paths. In our case, individual CAFEs are not responsible for forwarding the "join request" which is effectively the content consumption request. This is instead done in the CMP in a much more flexible manner, whereby CRMEs can forward requests according to specific rules or policies such as business relationship between ISPs, local ISP policies as well as network conditions. As such there is a vertical interaction between a CRME (in the CMP) and its

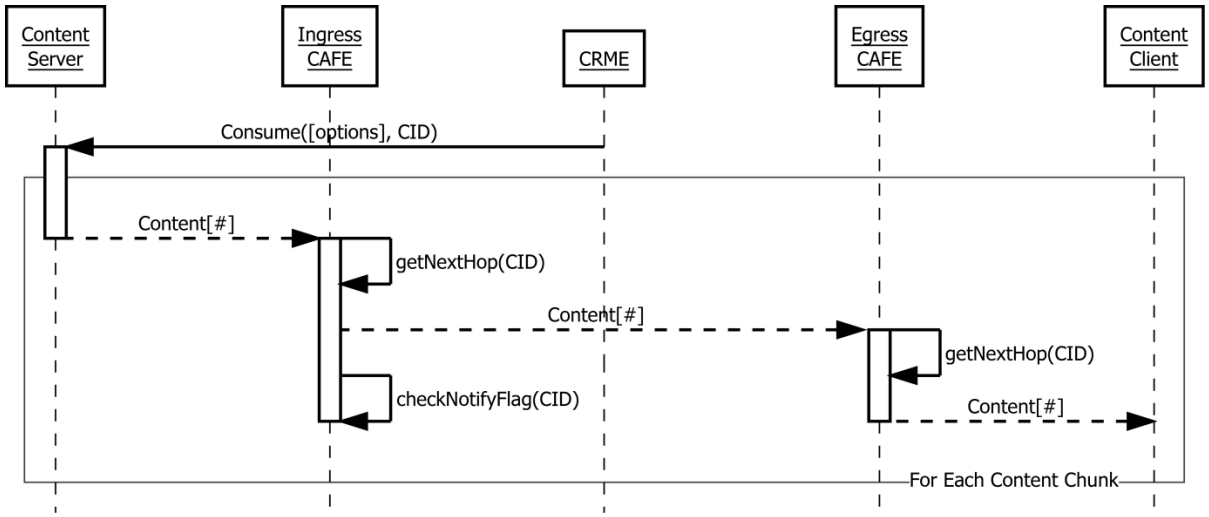


Figure 16: Content delivery process.

local CAFEs (in the CFP) for installing content states.

It is worth mentioning that some additional features will be further developed during the rest period of the project. These will mainly include:

- Interactions between CAFEs at the network edge and legacy IP routers at the core. Although using tunnelling is an obvious option, we will also explore more native approaches without necessarily involving content packets encapsulation.
- Quality of Service (QoS) support –how content flows requiring different levels of treatments can be gracefully delivered with different paths capabilities and how this will have an impact on the content delivery mechanisms in the CFP.

6.4 Inter-domain Content Delivery Route Optimisation

The proposed content delivery operation is also supported by a route optimisation technique that allows paths to be switched from *provider* routes to potential *peering* routes, or from a longer content delivery path with large number of AS hops to a shorter one with less hops. A route optimisation process is initiated by a CRME once it receives a `Notify` message from an ingress CAFE and is able to find a more optimal route towards the content server. Figure 17 shows the basic route optimisation process. For every content chunk that arrives at an ingress CAFE, it will check from its active session table whether or not it needs to notify its local CRME about the arrival of a new content stream into the domain. If it does, it will send a `Notify` message to the CRME indicating the Content ID, the address of the content server, and the address of the egress CAFE of the previous domain-hop. The CRME will in turn add an entry to its `ActiveSessions` table which it maintains for all Content flows it its domain. It will then look up in its BGP routing table to check if the next-hop prefix towards the content server contained in the routing table differs from the address of the prefix of the previous-hop CAFE. If so, the CRME will send a scoped `Consume` message to the next-hop CRME along the route-optimised (RO) path, and the resolution process will continue as before. The original CRME will also update its content ID entry in its content table with the route-optimised next-hop, so that future requests for the same content will follow the optimised path during the resolution phase.

To illustrate the route optimisation process in greater detail, consider again the example shown in Figure 14. Once the CRME (specifically, its CMF) in domain 1.1/16 has noticed that the content flow with source address belonging to prefix 1.2.1/24 has been injected into the local domain via ingress CAFE 1.1.0.1 via the provider route, and it also knows from the local BGP routing information that

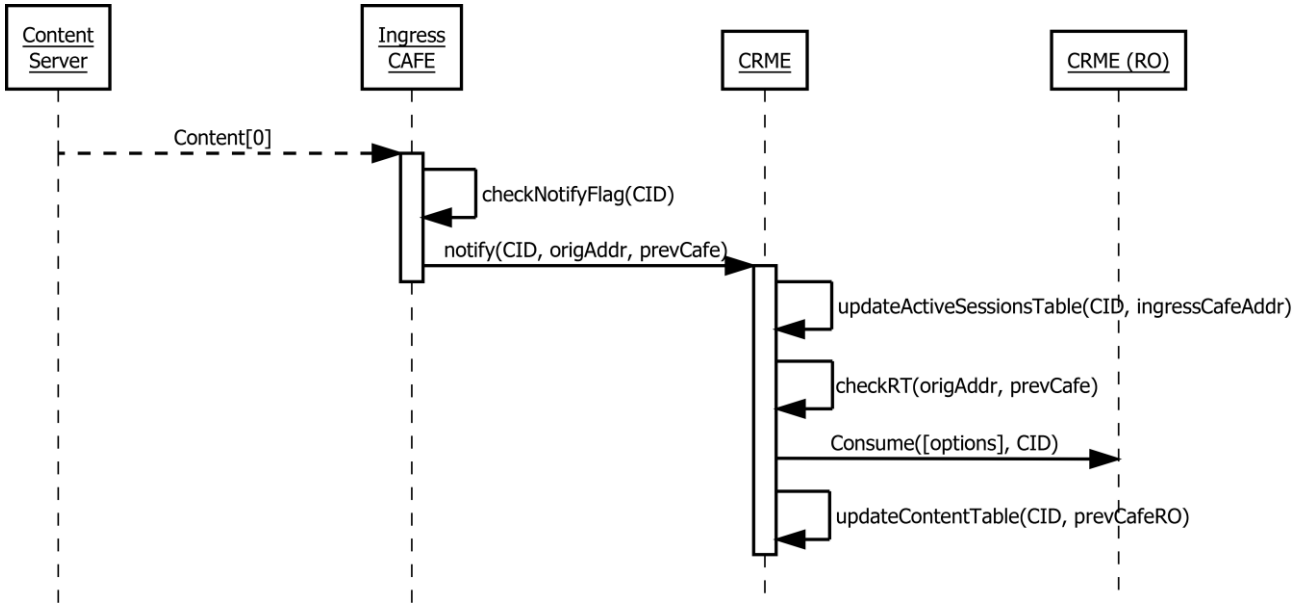


Figure 17: Route optimisation process

there exists a peering route towards the content source, it then issues a new *scoping-based* content consumption request: `Consume(INCLUDE{1.2.1/24}, X)` and sends it to the CRME in domain 1.2/16 in the peering route towards the source. Upon receiving the request, the CRME in 1.2/16 will update the local CAFE 1.2.0.1 by adding a new outgoing next-hop CAFE 1.1.0.1. As a result, a new branch via the peering route is established towards content client **C2**. Once the ingress CAFE 1.1.0.1 has received the content via the interface connecting to 1.2.0.1, it will prune the old branch via the provider route (the dash line). The purpose of such content delivery path optimisation across domains is to effectively reduce content traffic within top-tier ISP networks and also possibly reduce the content delivery cost for customer domains. Of course, this operation is not necessary if a CRME is allowed to send content consumption requests to its peering counterparts (in addition to the provider direction) during the resolution phase. However, such an option will incur unnecessarily higher communication overhead in disseminating content consumption requests, especially when the peering route does not lead to any source that holds the requested content.

7 Content Caching

Naming network objects instead of end-host machines gives the opportunity to identify packets that travel in the network from the sender to the requesting user. This case was supported in a number of recent studies, such as [3][14][15]. Building on the concept of *named contents* and given that the network transfers *named objects* instead of *data containers* (*i.e.*, IP packets), the concept of buffering in intermediate routers is transformed to *in-network caching* [3]. That is, if packets/objects are identified by name they can be cached *in the network* and forwarded to subsequent users interested in the same content.

We depart from the Content-Centric Networking (CCN) in-network caching scheme used by Van Jacobson *et al.* [3], in which content is wastefully cached at all nodes along a content delivery path, and propose two novel schemes to support more efficient content caching. The first scheme, presented in Section 7.1, is a probabilistic in-network caching scheme (ProbCache), which approximates the caching capacity of a path and then caches contents probabilistically along the path in order to: i) leave space for other flows on the same path to cache their contents, and ii) fairly multiplex contents in caches along the path from the server to the client. The second scheme, presented in Section 7.2, is a centrality-based caching scheme, where content is cached only at specific high centrality node(s) along the content delivery path. We show that such a scheme outperforms the standard ubiquitous caching (i.e., where content is cached at each and every node it traverses along the delivery path).

7.1 Probabilistic In-Network Caching

7.1.1 Background and Motivation

The lifetime of a packet according to the in-network caching scheme, introduced mainly in [3], can be briefly summarised as follows. Upon a content request, indicated by an *Interest* packet, the corresponding server sends the requested content towards the end-user. The routers along the path from the server to the user are equipped with caches (or Content Stores) and hence, the content is cached in *every* router on its way to the requesting user. If subsequently requested, the packet is retrieved from the closest cache and is sent to the requesting user, instead of having to retrieve it from the origin server. The specifics of the router structure as well as the details on content packet naming and routing can be found in [3]. We refer the reader there for further details.

The concept of *in-network caching* can be supported by any *content naming* and *location-independent routing* scheme, such as [14][16]. Having said this, in this section, we assume that a naming scheme and therefore, a routing protocol for Interests and Content Packets as well, is already in place and we focus instead on *the properties of in-network caching, independently of the rest of the architecture*. We borrow some terms from [3], such as the *Interest* packets that are generated to request and consume Data or *Content* packets, but we explicitly note that our design is not necessarily applicable to the CCN architecture proposed in [3] only, but to a wider range of ICN architectures.

In particular, we, somehow intuitively, observe that caching *every* content in *every* router the content goes through, inline with [3], inherently causes huge *caching redundancy*. However, given that the ICN research area is still in its infancy, it is not yet clear whether caching is going to find its way into deployment in every router, or in a few routers within a domain only [17][18]. We note that our study herein is still applicable to any *in-network caching* deployment scenario. We use the terms ‘router’ and ‘cache’ interchangeably to refer to the cache-enabled routers in the network and not necessarily to every router as such.

Departing from the above observation, we define a small system of caches and assess the impact of caching in every router along the path from the server to the client; we envision the ideal caching policy, based on: *i)* the caching capability of the system, and *ii)* the population in terms of distinct content objects; we design a *probabilistic caching algorithm* for distributed content placement in a system of in-network caches (Section 7.1.2). Our goal is to reduce *caching redundancy*, which

further means reduce the number of *server hits* and therefore, improve the overall performance of the system, in terms of available resource exploitation. Our results indicate that indeed there is a lot of space for improvement in the design of in-network caching, given correct content multiplexing rules are in place (Section 7.1.3).

Our study differs from past research on caching in two main aspects: first, past research focuses on overlay approaches for web-caching [19][20], or the replacement policies of single caches and the corresponding *single-cache system* properties [21][22]. In these cases, network dynamics, in terms of content multiplexing, are not really affected. Instead, in-network caching of packets, or chunks instead of whole files (*e.g.*, [23]), raise issues of fair resource allocation and efficient flow multiplexing [24]. Second, *hierarchical caching*, e.g., [25][26], which has looked into content replication in a string of caches, has mainly assumed some form of cooperation and hierarchical responsibility allocation between (a maximum of three) caching levels. Instead, in the case of in-network caching, the scene is flatter and all caches share equal caching responsibilities. In this study, we focus on a distributed, in-network caching environment.

We assume a small system of caches to monitor the behaviour of the *Cache Everything - Cache Everywhere* (CE^2) scheme . Our goal in this Section is to identify the drawbacks of CE^2 and set the foundations for the design of more efficient in-network caching algorithms. Our system is depicted in Figure 18.

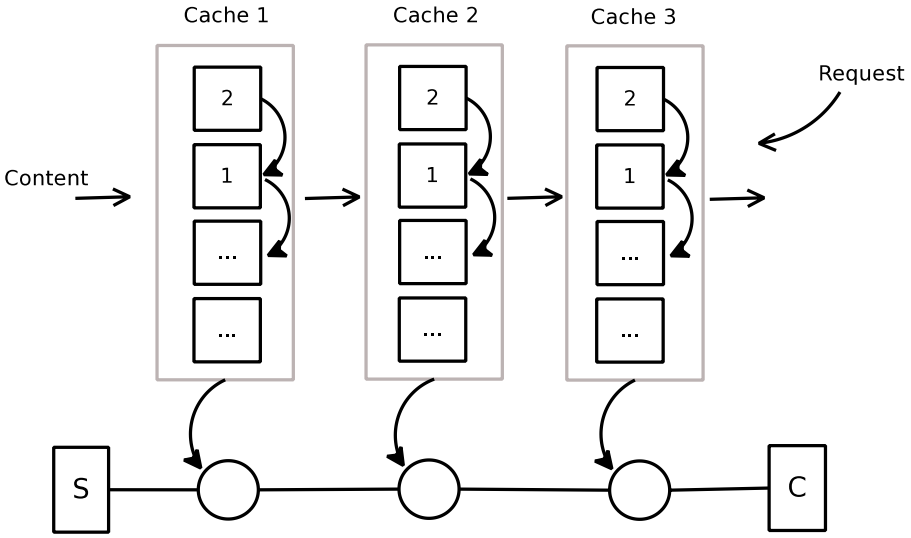
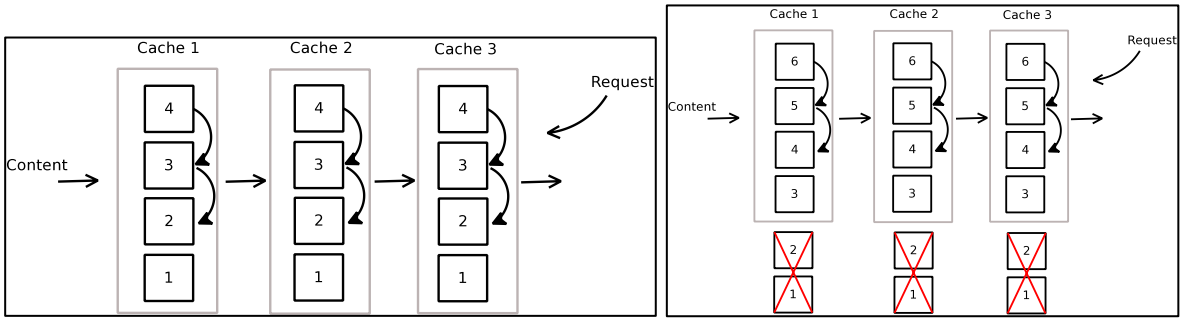


Figure 18: Controlled Caching System

According to the CE^2 scheme, as contents flow through the system, they get cached in every router they traverse, each time pushing the oldest content out of the cache. Hence, assuming three caches of four slots each and a system-wide population of four contents in total, stored in server S, then after four requests, one for each of the four contents, the status of the system will be the one shown in Figure 18. That is, every cache will have a copy of the whole set of requested contents.

The capacity of our system in this exercise is 12 contents split over three caches. Therefore, the four contents fit exactly three times in our system of caches, maximising the chances of a cache hit in the next round of requests. However, if we assume six different contents as the population of our system, then although the whole content population can fit twice into the system of caches, we observe in Figure 19 that the CE^2 fails to accommodate even a single copy of the set of contents. Instead, after six requests, the two oldest contents will be pushed out of all caches and the four newer ones will be cached in all three caches (Figure 19). Caching redundancy is clear in this example and reduces the chances of a cache hit in the next round of requests by 33% (*i.e.*, two out of six contents have to be retrieved from the main server).



(a) Content Population=4 (b) Content Population=6]

Figure 19: CE^2 Caching Policy

An example of the ideal content placement is shown in Figure 20. We have studied and modelled the behaviour of CE^2 in [3] and argue that *in-network cache management has to take into account the approximate capacity of the system of caches and the estimated population of contents that pass through the system per unit time in order to: 1) cache the corresponding number of copies for each content that travels through the system, and 2) cache content in a fair manner, depending on the resources available in the path*. We attack the problem of cache placement from a probabilistic point of view. We deal with issue number 1) above in Section 7.1.2.1 and with issue number 2) in Section 7.1.2.2.

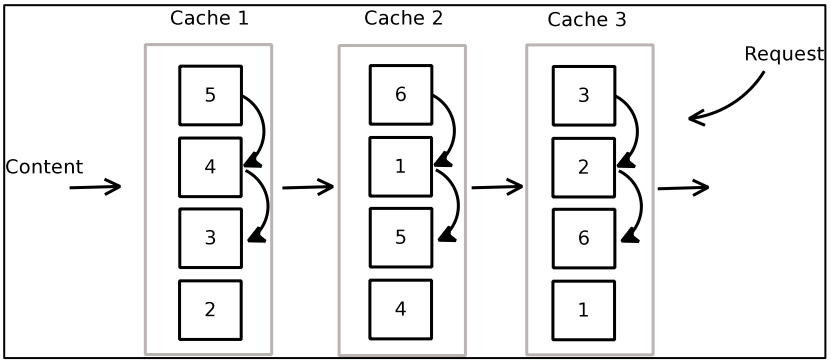


Figure 20: Ideal Caching

7.1.2 Building ProbCache

We approach the problem of content distribution within a system of caches from the *system capacity* point of view. In particular, *each router calculates the distance from the server and the end-user respectively, based on which it indirectly approximates the number of copies of this content that the system can accommodate* (Section 7.1.2.1). Based on this indication and on *the distance from the requesting user, each router probabilistically caches contents as they travel along the path* (Section 7.1.2.2). Our Model Notation is given in Table 7.

Symbol	Meaning
n_c	Number of Caches on the *path*
N	Number of Caching Slots in Cache i
R_c	Content Catalog/Population in the *Path*
TSI, x	Time Since Inception (Header field - Interest Packet): Hop-Distance from Client, Value range: 1 to n_c
TSB, y	Time Since Birth (Header field - Content Packet): Hop-Distance from Server, Value range: 1 to n_c

Table 7: Model Notation

We base our design on the following four facts.

1. The path from the Server to the Client (see Figure 18) comprises of n_c caches, where router r_i has N_i available slots. Therefore, the *caching capacity* of our system of caches is $\sum_{i=1}^{n_c} N_i$.
2. The Server hosts R_c contents[5]. Therefore, the *caching capability* of an n_c-long path is $\sum_{i=1}^{n_c} N_i/R_c$.
3. Content and Interest packets follow the same route. This is the assumption used in [3], but a fair assumption in general, given the name-based, location-independent routing assumed in ICNs.
4. Similarly to the TTL field included in IP packets, our design pre-requisites that an ICN packet header includes the following two fields (Figure 21): *i) Time Since Inception* (TSI), which denotes the number of hops that the Interest packet has travelled on its way from the client to the server, and *ii) Time Since Birth* (TSB), which denotes the number of hops the content has travelled from the server. Routers along the path increase the TSI value of each Interest packet and the TSB value for each Content packet by one[6]. Content packets carry the TSI value seen by the Interest packet on their way back to the user.

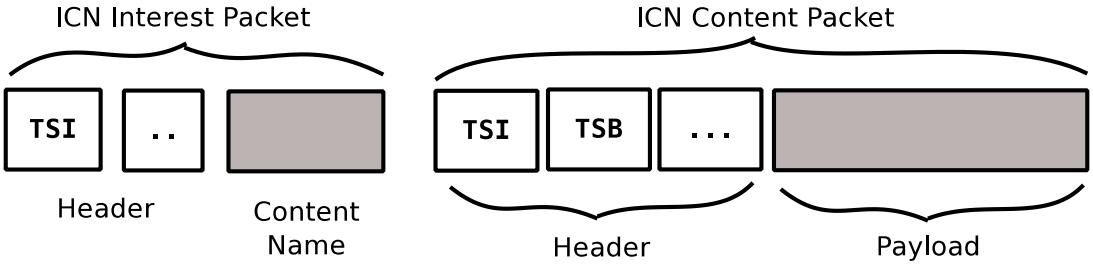


Figure 21: CN Packet Layout

7.1.2.1 Estimating the Caching Capability of a Path

We assume the topology of Figure 22. Each router along the path, upon reception of a content packet, calculates the *caching capability of a path*, or in other words, the *number of times this packet should be cached in the system*, which is reflected in the $TimesIn$ factor. Consider the two users shown in Figure 22, five and four hops away from the server, respectively. The total cache capacity of the system is $\sum_{i=1}^{x} N_i$, where $x_1 = 5$ for $Request_1$ and $x_2 = 4$ for $Request_2$.

The calculation of $TimesIn$ takes place as follows:

$$TimesIn = \frac{\sum_{i=1}^{x-(y-1)} N_i}{R_c} \qquad \text{Equation 1}$$

where x is the *Time Since Inception* (TSI) value and y is the *Time Since Birth* (TSB) that the router sees in the header of the packet (Figure 21). For example, content packets traveling through router r_2 to fulfil $Request_1$ in Figure 22 will have $TSI = 5$ and $TSB = 2$, while contents for $Request_2$, $TSI = 4$ and $TSB = 2$. The sum in Equation 1 considers the subtract of TSI minus TSB (or $x - y$) to account for the *remaining* number of caches, instead of the total number of caches from the server to the client. This is because router r_2 does not know whether the packet was retrieved from a previous router (r_1, in this case), or from the original server.

[5] This can be translated to the already seen as well as the anticipated traffic that this router serves. We discuss this issue further later on.

[6] In case of a cache hit, the TSI and TSB values are treated as if the cache is the origin server, that is, the TSI value of the content packet is replaced by that of the *Interest* packet, while the TSB is set to 1.

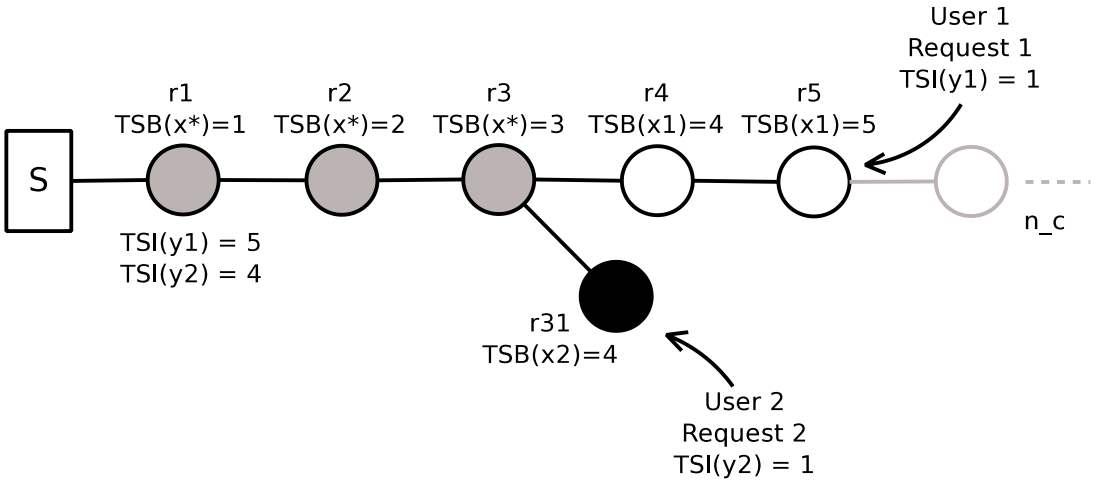


Figure 22: Design Topology

7.1.2.2 Weight-based Caching

We argue that distributed fair caching demands that each user takes into account other users sharing the same path. Hence, to decide *where* to cache the number of copies that *TimesIn* indicated, we use the *Cache Weight* factor:

$$CacheWeight = \frac{x}{y} \qquad \text{Equation 2}$$

where y is the TSB value of the packet header and x is the TSI value. We note that the TSI value is fixed during the content packet's journey from the Server to the Client, while the TSB value is increasing for each router the packet traverses, hence $CacheWeight \rightarrow 1$ as the packet gets closer to its destination.

7.1.2.3 ProbCache: Probabilistic In-Network Caching

The *ProbCache* algorithm proposed herein is the product of the two factors described above. In particular, upon receiving a content packet, a router performs the following calculation to decide whether it is going to keep a copy of it in its cache, or will just forward it further to the next router on the path towards the Client.

$$ProbCache = \underbrace{\frac{\sum_{i=1}^{x-(y-1)} N_i}{R_c}}_{TimesIn} \times \underbrace{\frac{y}{x}}_{CacheWeight} \qquad \text{Equation 3}$$

In terms of implementation, we normalise the product of Equation 3 and decide whether to cache or not based on a random number generator function.

One may argue that the *TimesIn* factor inherently favours contents that travel from further away, over contents from Clients closer to the Server. However, we consider that this is only due to factors external to our design, *i.e.*, the network topology, and we attempt to counter-balance the unfairness that this may cause by the *Cache Weight* factor: the fraction of TSB over TSI, namely the *Cache Weight* factor, increases as the content packet gets closer to the user. In turn, this increases the probability of the content being cached closer to its destination.

This way, we achieve *fair cache multiplexing* between requests that travel to different destinations in terms of path length. That is, for example, the client behind $Request_1$ in Figure 22, has to share the caching space available on this path equally with all other clients using the same, or a part of

the same path. Therefore, contents for User 1 should be cached inversely proportionally to its distance from the server. For the content packet of $Request_1$ this means that it has higher chances of being cached in routers $r_1 = 4$ or $r_1 = 5$, in order to leave caches $r = 1 - 3$ for clients travelling shorter paths to cache their contents. This counterbalances the slightly unfair behaviour of the $TimesIn$ factor described before and is in accordance to our previous findings [27] that contents tend to be cached for longer towards the edge of the network.

To calculate the $TimesIn$ factor each router has to make the following two assumptions:

1. N_i value: Given that the scene is very blurry still as to what amount of memory each router will have, or if backbone routers, for instance, will have bigger caches than edge-network routers, we make the following simplistic assumption. *Each router assumes that all other routers on the path have the same amount of cache as it has got*. Even in a random-size (*i.e.*, not standardised) cache deployment scenario, this assumption serves our purposes well. That is, a router with a big cache, compared to the caches along the path, will have higher probability of caching more contents, while a router with a small cache will experience the opposite effect (Equation 1), which is a desirable property. Therefore, in terms of content multiplexing this simplifying assumption alleviates the effect of unknown cache sizes.
2. R_c value: This value represents the number of different contents that a given n_c-long *path* has to accommodate. However, in real terms this is impossible to approximate. Therefore, in our implementation *each router considers that R_c equals the amount of traffic that it delivers (in terms of Content packets) per unit time*. The more the requests a router receives, the less the probability that it will cache an incoming packet (Eq.1), which is also in agreement with [27]. This is another desirable system property that achieves efficient content distribution along a path, as we show later in the results section.

7.1.3 Performance Evaluation

We test our algorithm in a custom-built simulator, where we use *Least Recently Used* (LRU) caches as the default option. To the best of our knowledge, this is the first study to question the efficiency of *universal*, or *ubiquitous* caching proposed in [3] and therefore, we implement and evaluate the performance of *ProbCache* against CE^2. Given that the ultimate goal of the *ProbCache* algorithm is to manage caching resources more efficiently, by reducing cache redundancy, the straightforward metric of interest is the *reduction of Server Hits*.

The gain from serving user requests from intermediate caches instead of travelling to the origin server depends on the number of hops that the request travels before it eventually finds what it is looking for. Clearly, as the number of hops increases, the overall gain decreases. To measure this gain, we also monitor and present the *Hop Reduction Ratio*.

We use the topologies shown in Figure 23 and Figure 24 and present two scenaria: the first one is a simplistic setup over the String Topology to illustrate the operational details of *ProbCache*. In the second scenario, we observe the performance of the algorithms over a more realistic 6-Level Tree Topology. In both topologies, *Node 1* is the server node.

Figure 23: 10-Hop String Topology

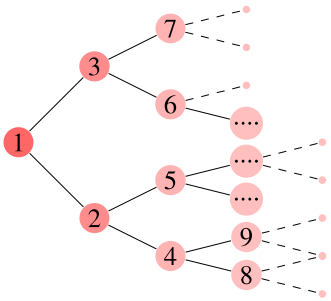


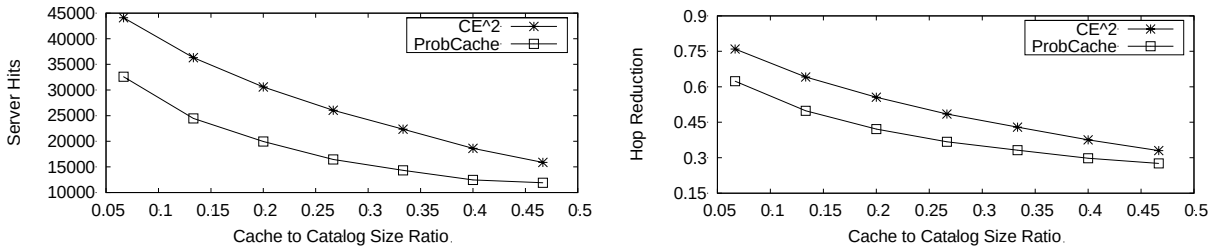
Figure 24: 6-Level Tree Topology

7.1.3.1 Scenario 1: Efficient Distributed Content Placement

We use the String Topology of Figure 23. Clients are connected to all nodes along the path, with the longest path from the server being ten hops (*i.e.*, clients connected to node 11). The goal of this scenario is to show how requests coming from users connected at different points along the path are multiplexed in terms of cache resource allocation.

We generate 60,000 requests in total; individual contents are requested according to a Zipf distribution, where the value of the exponent characterising the distribution, a, is 0.9 [35]; the number of distinct contents that the server hosts (that is, the content catalog) is 300 and requests arrive according to a Poisson distribution. In Figure 25 (a), we plot the number of server hits for the whole experiment and repeat the experiment for different *Storage to Traffic* ratios per router. We achieve this by increasing the cache capacity, N, of intermediate routers. In terms of server hits, we observe a difference of 15-20%, on average, and the difference can go up to 26.6% for very small caches, always in favour of *ProbCache*.

In terms of *Hop Reduction* gains, we observe in Figure 25 (b) that the performance gain is in the order of 5-12%. Although this may sound as a marginal gain, we argue that in a topology where the longest path is ten hops, which is the case in our experiment, but very frequently the case in Internet topologies as well, this translates to a hop-reduction of one to two hops. Considering that requests may come from as close as one or two hops away from the Server (*i.e.*, requests from users attached to nodes 2 or 3 in Figure 23), we consider this to be a substantial performance gain and continue to explore the operational details of *ProbCache* that lead to this result.



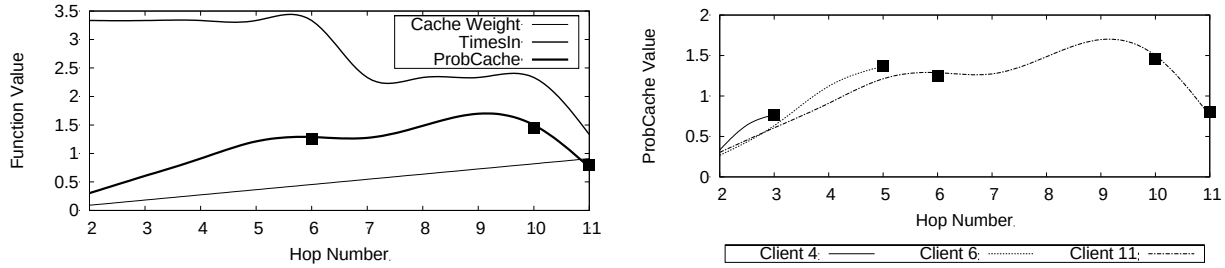
(a) Server Hit Saving (b) Hop Reduction Ratio

Figure 25: General Metrics for Scenario 1

We randomly choose a request from a user connected to the last router of the string topology (r_{11}), and plot the two factors of *ProbCache*, as well as their product, namely the value of the *ProbCache* function itself, as the request is travelling from r_2 to r_{11}. The result is shown in Figure 26. We see that initially the value of the *TimesIn* factor is large, but as the request is getting closer to its destination its value declines. This follows precisely our modelling goal in designing *TimesIn*, which is to approximate the number of copies of a specific content that the path can accommodate.

On the other hand, the $CacheWeight$ factor increases as the content packet gets closer to the user (or in other words as the TSB value moves towards the TSI one).

The dots in Figure 26 (a) represent the routers, where the content packet is cached. We monitor the next two content requests, which come from users connected to routers 4 and 5, respectively. In Figure 26 (b), we plot the *ProbCache* function of these two requests and replot the *ProbCache* of Figure 26 (a). The dots represent the router where the packet is eventually cached. We observe that the further away the request comes from, the further away the packet gets cached. This leaves space for shorter flows (in terms of hop-length) to cache their packets without having to evict other packets from the caches. We argue that this behaviour is indeed desirable and achieves fair content multiplexing in path-caches and results in better exploitation of the available cache capacity. We further elaborate on this behaviour in our next simulation scenario.



(a) $ProbCache$ Value Decomposition (b) $ProbCache$ for three different clients

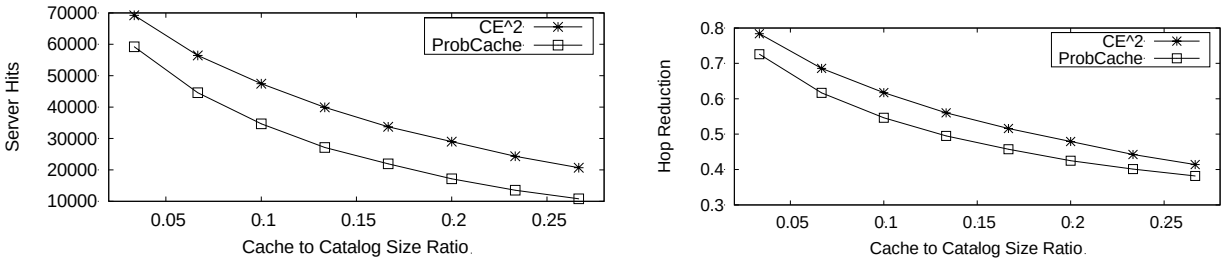
Figure 26: $ProbCache$ Function Analysis

7.1.3.2 Scenario 2: Cache Capacity Management

In our last simulation scenario, we use the 6-level tree topology shown in Figure 24, which consists of 127 nodes in total. Node 1 is the server node, which hosts a total of 500 distinct contents, while we configure requests to come from the last two levels of the tree, *i.e.*, Nodes 32 to 127, a total of 95 nodes. We consider this to be a more realistic setup, as we assume that individual users are not connected to backbone network routers, although we report that results do not differ massively in either case. This time, we run the simulation for longer, generating a total of 100,000 requests, to allow enough time for the system to reach a steady-state. The exponent of the Zipf distribution of requests is now set to 1,2 to represent more realistic reflection of Internet traffic.

In Figure 27 (a), we present the performance difference of *ProbCache* against CE^2 in terms of Server Hits, while in Figure 27 (b) we present the performance difference in terms of Hop Reduction. The Server Hits performance difference is slightly smaller than in the previous scenario and balances around a reduction of approximately 12-15% (12,000-15,000 Server Hits less for *ProbCache*).

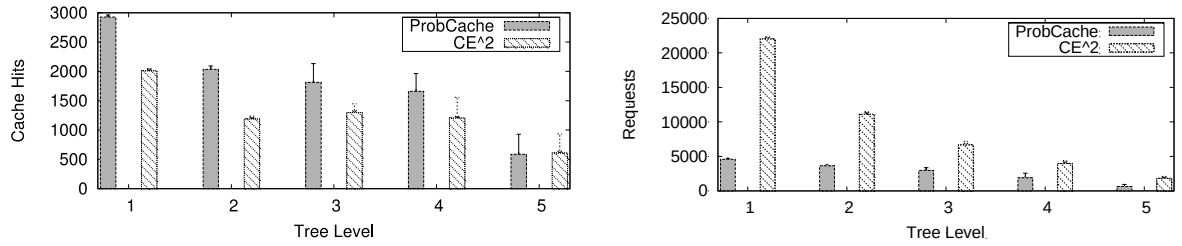
The Hop Reduction difference is smaller too in this scenario (roughly 8-10%), due to the fact that in this topology the longest possible path is six hops, compared to the previous longer string topology. However, considering the operational properties of CE^2, which inherently caches content as close to the end-user as possible, we consider that even this small performance difference unveils better exploitation of network (storage in this case) resources. To prove this claim further, we plot in Figure 28 the average number of: *i)* Cache Hits, in Figure 28 (a); *ii)* Received Requests, in Figure 28 (b) and; *iii)* Cache Drops, in Figure 28 (c), per each level of the tree topology, where the smaller the tree level, the closer is this level to the server.



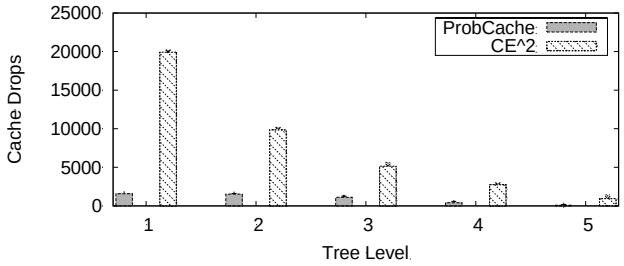
(a) Server Hit Saving (b) Hop Reduction Ratio

Figure 27: General Metrics of Scenario 2

We observe disproportionate differences between the number of Cache Hits for the different levels of the tree topology (Figure 28 (a)) and the corresponding numbers of requests they receive (Figure 28 (b)). We argue that this difference in Cache Hits and Received Requests owes to the efficient resource management of the *ProbCache* algorithm, which results in contents staying in the cache for longer. To prove our claims, we plot in Figure 28 (c) the average number of Drops per tree-level. The huge difference in terms of drops validates our claim that although the number of cache hits *per se* between the two algorithms is not extremely big (although the 15-33% difference is still a substantial figure), the difference in terms of traffic flow in the network is striking.



(a) Average Cache Hits (b) Average Received Requests



(c) Average Cache Drops

Figure 28: Cache Behaviour Monitoring per Tree-Level

We argue that these are the properties of *in-network caching* that the research community needs to investigate further, in order to make the most out of this inherent capability of ICNs.

7.1.4 Related Work

Work in the field of ICNs has spread across several directions recently, from service-oriented extensions to the CCN architecture [28] and caching overlays [29][30], to new information-centric architectures [31] and pub/sub approaches to ICNs [32] and further to feasibility studies from a hardware perspective [17][18], the implications of in-network caching to the overall energy consumption of the network [33] and the fairness of an in-network caching-enabled Internet [34].

A number of recent studies focused specifically on the properties of a network with in-network caches: for example, in [35] the authors provide a comprehensive performance evaluation of in-network caching taking into account several parameters, such as the content request distribution,

the content catalog size, the size of caches and the cache replacement policies. They conclude that content popularity is (by far) the most important parameter of all. In [36], the authors investigate service differentiation techniques for information-centric networks. Among other findings, they conclude that as content popularity increases, so does the number of cache hits at leaf routers, which is in line with our previous modelling study in [27]. Other modelling studies of the ICN caching property include [37] and [38].

7.2 Centrality-based In-network Caching

In this section, we detail a centrality-based in-network caching mechanism that seeks to improve the overall content delivery performance in COMET. We compare our approach with Van Jacobson's CCN approach [3].

We first introduce the background of the in-network caching building block in the context of ICN. Next, we detail a solution scheme that can exploit specific features of a network topology.

7.2.1 Background and Motivation

The ICN concept has gained wide attention in both the research community and the industry. It breaks the current end-to-end resource sharing model. In ICN, content names are decoupled from their host addresses, effectively separating the role of identifier and locator as opposed to the current IP addresses which are used for both purposes. Naming content directly enables the exploitation of in-network caching since the content can now be accessed in an application-independent manner. In fact, one of the main building blocks in ICN that makes it an attractive choice for next generation Internet is the pervasive in-network caching capability.

In the literature, all ICN-related proposals by default include this capability. The most explicit of them is the Networking Named Content (NNC) proposal by Van Jacobson [3] where it was proposed that content chunk be cached in each and every router it traverses along the content delivery path with each router applying the least recently used (LRU) cache eviction policy. Such caching strategy ensures quick diffusion of content across network and in the area of caching in general, seems to be the default strategy (i.e., caching in all intermediate nodes on the delivery path).

We argue that such caching strategy is unnecessarily costly and sub-optimal and study an alternative in-network caching strategy for enhancing the overall content delivery performance in COMET. In the caching-related literature (not specific to ICN), some authors have already questioned this conservative "cache-all" strategy [39][40][41]. We propose a caching algorithm based on the concept of centrality derived from the area of social network analysis [42] where only selected nodes in the content delivery path cache the content chunk with the rationale that some nodes have higher probability of getting a cache hit compared to others and by strategically caching the content at "better" nodes, we can decrease the cache eviction rate and increase the overall cache hit performance.

7.2.2 Centrality-based Caching Scheme

The current host-centric Internet requires all content requests to be resolved to the location of host servers before content can be delivered while in ICNs (e.g., [3], [43]), content are labelled and identified by their own content names without needing to know the host machine. Thus, in ICNs, a content request can be satisfied by any matching content regardless of its location (i.e., a cached content can serve a request). In [3], this feature of ICNs is exploited by caching every content chunk that passes a router with the assumption that routers are equipped with (large) cache stores. With this approach, content would disperse into cache store across a network (or domains) in a fast way and the cache stores would update frequently following the LRU policy. For the rest of the section, we refer this as "NNC-LRU" as the benchmark for performance comparison.

In caching problems, there are two main aspects of investigation. First is the question on *where* to cache and second is *how* to cache. The first question relates to the location of caching (e.g., specific

surrogate server, networked caches etc.) while the second question relates to how cached content is managed (e.g., how to replace a cached content when the cache is full). In this section, we focus on the first question.

We propose a new caching algorithm based on the concept of betweenness centrality [42] which measures the number of times a specific node occurs on the shortest paths between all pairs of nodes in a network topology. The idea is that if a node lies in the path of many shortest paths, then it is more likely to get a cache hit.

For a topology G=(V, E) with V vertices and E edges, the betweenness centrality, $C_B(v)$ of node v is computed as follows.

$$\text{betweenness centrality: } C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of shortest paths from s to t that pass through a node v.

Each node in a topology thus has its own betweenness centrality value. When a content client initiates a content delivery, the initiation message (e.g., `Consume`) will record the node with the highest centrality value. When the content is delivered, it will be cached at the recorded node along the delivery path. In case of more than one node has the same highest centrality value, all of them will cache the content.

The same LRU cache eviction policy is assumed at each node. Hereafter, this caching strategy is referred to as "Betweenness-LRU". `Consume` requests for different content are generated based on Zipf-distribution as follows:

$$\sum_{k=1}^{N} \frac{C}{k^{\alpha}} = 1$$

where N is the total population of content unit in the network (i.e., the unit can be the whole of a content, a content chunk or a content packet), k represents the k^{th} most popular content unit and the possibility for a request for the k^{th} content unit is $\frac{C}{k^{\alpha}}$ with α being the popularity factor. The sum of the possibility of all content equals one.

We show a preliminary comparison of the two caching strategies in a 7-node string topology where all content are hosted at one end of the topology while content requests can originate from any of the other nodes. The content request arrival process for content unit m, $1 \leq m \leq M$, follows the Poisson process with mean rate, $\lambda = \sum_{k=1}^{N} \lambda_k$.

We measure the performance by the cumulative ratio on hops saved from each request with in-network caching i.e.,

$$\text{Hop reduction ratio, } \beta = \frac{\sum_{m=1}^{M} h}{\sum_{m=1}^{M} H}$$

where H is the number of hops from the content consumer to the content server which hosts the requested content and h is the number of hops from the content consumer to the first hop where the requested content is cached. If no matching cache is found along the path to the server, then $h = H$. In a non-caching system, then $\beta = 1$. Better performance (i.e., higher hop reduction) is signified by smaller ratio.

The comparison result is shown in Figure 29 which shows our *Betweenness-LRU* performs better than the *NNC-LRU* algorithm.

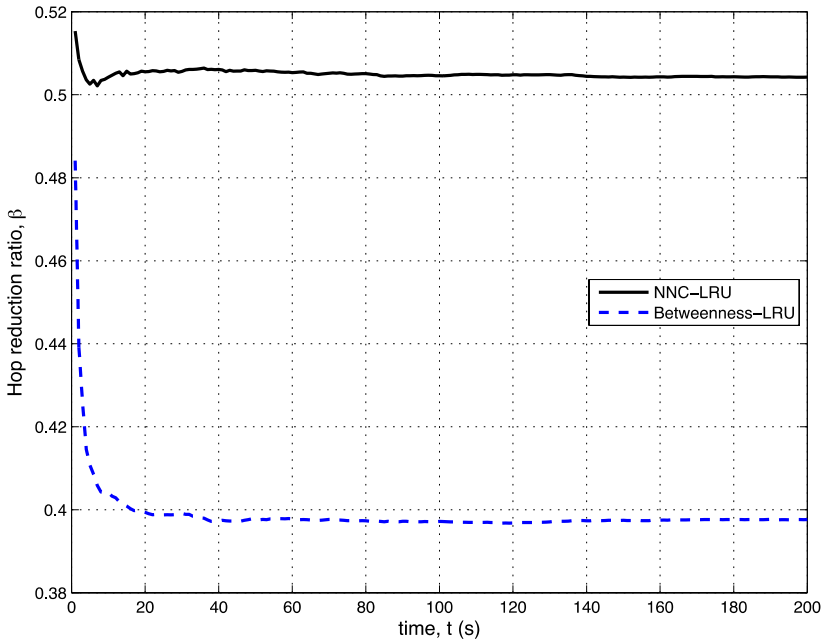


Figure 29: Comparison of NNC-LRU and Betweenness-LRU in string topology

From this simulation, we also see that the number of content request reaching the server (hereafter referred to as server hit) is reduced. In the NNC case, from the total of 998993 requests, 398846 hit the server; resulting in a server hit reduction ratio, $\gamma = 0.3992$ while with our algorithm, we achieve $\gamma = \frac{247052}{999369} = 0.2472$.

7.2.3 Scalable Realisation of Betweenness Caching Scheme

In the real world, it is usually not practical to assume each node capable of computing its betweenness centrality since it requires the knowledge of the shortest paths between all pairs of nodes. Such computation is not scalable. We then further develop an approximation of our caching strategy based on the ego network betweenness concept [43].

Ego network consists of a central node together with the nodes (one hop neighbors) they are connected to and all the links among those nodes. The advantage of ego network is the ease for central node to collect data from the neighbors compared to collecting the data of whole network and it is simple to calculate the betweenness centrality of the central node within its ego network. This type of centrality is called ego network betweenness centrality. Although ego network betweenness only reflects the importance of a node within its ego network, its scalability and ease of implementation makes it a good alternative for betweenness in large networks. The caching algorithm using ego network betweenness centrality and LRU cache eviction policy is referred to *Ego-Betweenness-LRU* hereafter.

To study the system in a more realistic conditions, we replace the simplistic string topology with a scale-free topology following the Barabasi-Albert (B-A) power law model [44] which builds a graph taking into account the preferential attachment property of the Internet topology and resulting in highly skewed degree distribution. It is also interesting to note that it has been found that the betweenness distribution of such graph similarly follows power law [45]. Figure 30 shows an example of a topology generated based on the B-A model with 100 nodes.

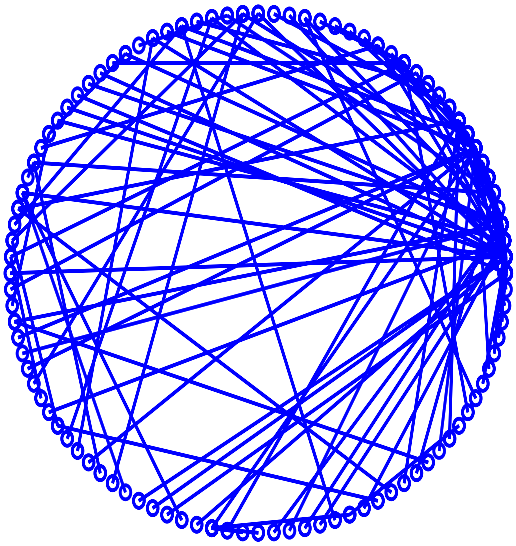


Figure 30: An example of B-A graph with 100 nodes

We show in Figure 31 the hop reduction ratio of the three caching strategies (i.e., *NNC-LRU*, *Betweenness-LRU* and *Ego-Betweenness-LRU*) with B-A graph. First of all, we see that the performance of both our centrality-based caching scheme performs better than NNC. However, we note that at the beginning of the simulation, *NNC-LRU* performs better (i.e., the ratio drops more rapidly). This is due to the fact that we have a "cold start" to the system where all cache stores are empty in all nodes. With NNC caching in all intermediate nodes, it manages to disperse content more rapidly than our strategies. However, after this initial short "cold start" period where cache stores are populated with some content, our schemes stably performs better than NNC.

Also, note that the curves for both *Betweenness-LRU* and *Ego-Betweenness-LRU* overlap. This shows that the more scalable approximation of the betweenness metric using the ego-centric concept achieve similar performance.

7.2.4 Conclusions

We proposed a caching strategy based on the concept of betweenness centrality (i.e., *Betweenness-LRU*) and an approximation of it (*Ego-Betweenness-LRU*) for scalable and distributed realization and compare their performance against Van Jacobson's proposal [3] (*NNC-LRU*). Although *NNC-LRU* has the highest content dispersion speed, its caching gain is worse than our two approaches. Our simulation results on *Betweenness-LRU* suggest that it has good hop reduction ratio but its non-scalability and complexity restricts the implementation in large scenarios. Our results show that *Ego-Betweenness-LRU* approximates closely the *Betweenness-LRU* and thus present itself as the more practical candidate for real networks deployment.

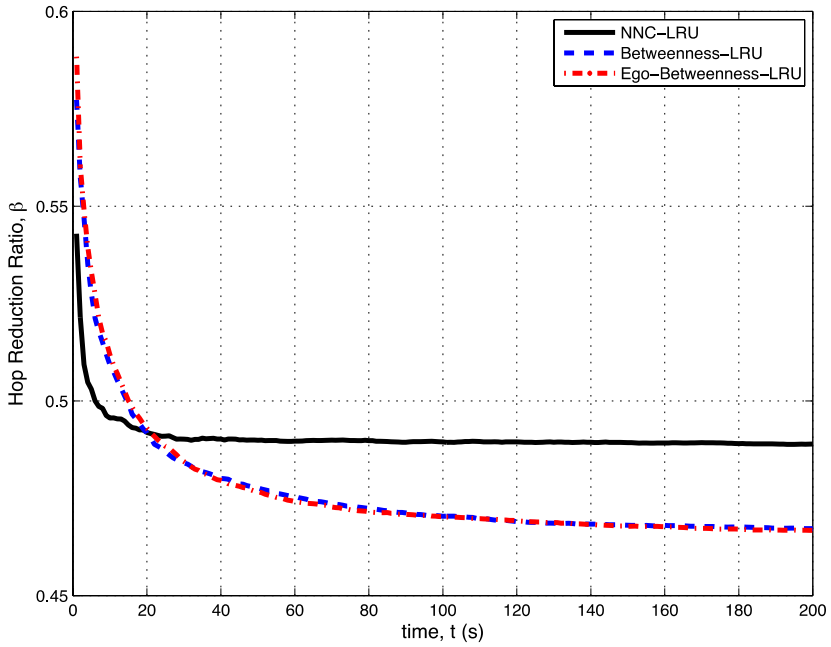


Figure 31: Performance comparison of NNC-LRU, Betweenness-LRU and Ego-Betweenness-LRU in a power law topology

8 Content-centric Mobility Support

In general, ICNs have an inherent ability to support faster handover than current host-centric mobility protocols. This is due to the fact that ICNs are better able to 'heal' changed paths as a result of mobility, without necessarily having to explicitly inform the end-host or a fixed agent within the network, since the network is inherently content-aware. Nevertheless, there exists the potential to further improve handover performance in ICNs to ensure that handover latency is minimised as much as possible. Whilst previous works on mobility support for ICNs have been addressed from a system level [39], we propose to tackle the problem mainly at the link level, at the least reliable part of the network: the air interface.

This section lays the foundation for mobility support in COMET by first specifying architectural extensions to the normal CAFE to facilitate such support. Following this, we turn to the mechanisms that are being proposed to support content-aware mobility at the link level. Finally, system-level mobility support is specified for both the intra- and inter-domain cases.

8.1 Content-Aware Mobility Support Entities

Mobility in Comet is supported through the use of mobility-aware CAFEs (mCAFEs) which are placed at the edge of the access network, and are equipped with functionality to track the motion of the users it is serving as well as the content they are accessing. Figure 32 shows the proposed internal architecture of an mCAFE, which consists of a number of logical components in addition to those inherited from standard CAFEs:

User Management Function (UMF) – As users roam around within the coverage area of a mobile network, they send to the mCAFE (either periodically or aperiodically) channel quality information (CQI), as well as information about other nearby mCAFEs that it has detected. This information is processed and stored by the UMF, and can be used by it to infer the general direction of the mobile user, and predict users' likely future locations.

Radio Network Management Function (RNMF) – mCAFEs that neighbour each other may share information about its current load condition. This information is processed and stored by the RNMF. The RNMF may also be pre-programmed with information about the mCAFE's own coverage area, as well as that of its geographical neighbours. However, it may instead *infer* the coverage area dynamically, using information provided to it by the UMF as users hand over between cells. In such case, the RNMF will have the function of running the inference algorithms.

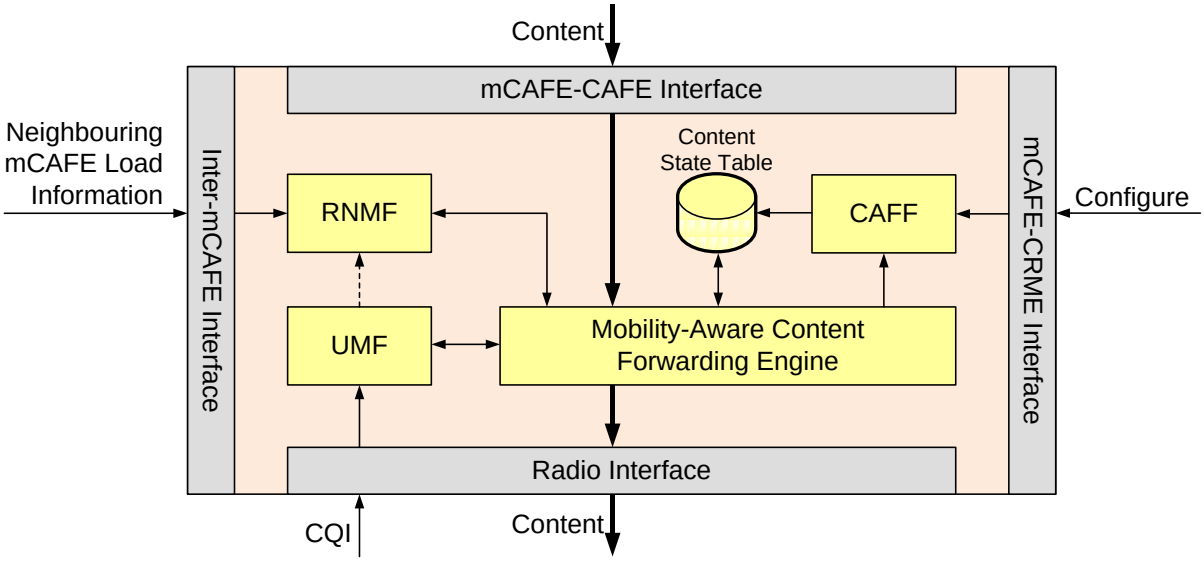


Figure 32: Internal mCAFE architecture

Mobility-Aware Content Forwarding Engine – This component uses information stored within the UMF, RNMF and content state table to make intelligent scheduling decisions that aim to sustain user satisfaction, particularly during handover when the user is most susceptible to temporary disconnection from the network.

The mCAFE also inherits the CAFF and Content State Table components from the standard CAFE. The function of these components are as described in section 6.1, except that the CAFF does not send Notify messages to its local CRME, as the mCAFE always acts as an egress point.

Communication between mCAFEs and external entities occurs over four main interfaces. The CRME-mCAFE interface connects mCAFEs with its local CRME, to allow it to be configured with content-states in response to a `Consume` message sent to it by a Content Consumer (**CC**). The mCAFE-CAFE interface connects is a purely content-plane interface through which content from the previous-hop CAFE flows towards the CCs. This information is then scheduled to the users by the mobility-aware content forwarding engine and sent across through the air interface to the CCs. Finally, the inter-mCAFE interface is that through which the mCAFE communicates with its neighbouring mCAFEs, exchanging load information between each other.

8.2 Mobility- and Content-Aware Scheduling

Scheduling across the air interface in mobile networks today is typically carried out based on the broad class of application being consumed (e.g. video, voice, etc.) without regard to the particular QoS specifications that this specific session requires. Since users close to a base station typically receive higher throughput compared to users located further away from it, proportionally fair scheduling algorithms [51] try to compensate for this by increasing the resources allocated to cell-edge users according to a given proportion. Altering this proportion alters also the balance between the overall information throughput of the cell, and the degree of user 'fairness'.

The concept of fairness referred to in existing work on scheduling assumes a rather static view of the quality of service requirements of users in relation to the content they are consuming, and the device being used to consume it. For example, mobile users accessing video content are usually scheduled resources in such a way that results in all those users receiving the same *average* information throughput, regardless of both their terminal's screen resolution and their mobility characteristics. However, it is reasonable to assume that the resource requirements of a video user are *directly proportional* to the resolution of the terminal's screen, therefore users of large-resolution devices should expect to have a higher amount of resources provisioned to them than to those of small-resolution devices. Likewise, mobile users who are about to perform a layer-2 or layer-3 handover should typically receive greater priority than other mobile users, so as to ensure that they have buffered sufficient content to carry out the handover without interruption to their session. This perspective on the concept of fairness essentially redefines its definition from one based on the *amount of* r*adio resources* a user is consuming to one based on the *amount of content* it is consuming.

As an example of some of the content-aware mobility support mechanisms that are being proposed, consider the example shown in Figure 33, in which Content Consumers, **C1** and **C2**, are receiving content from Content Server, **S**. As **C1** moves towards the edge of the coverage of mCAFE, m1 and into the coverage, m2, the mobility-aware content forwarding engine of **m1** will increase the scheduling priority of that user to ensure that its streaming session does not get interrupted while is it handing over between mCAFEs. If m1 has information that **m2** is currently experiencing high load, ,m1 will give even higher scheduling priority to **C1** so that **m2** does not have to begin scheduling content to **C1** immediately once it has handed over, but can instead opportunistically wait until **C2** has moved closer towards the centre of m2's coverage.

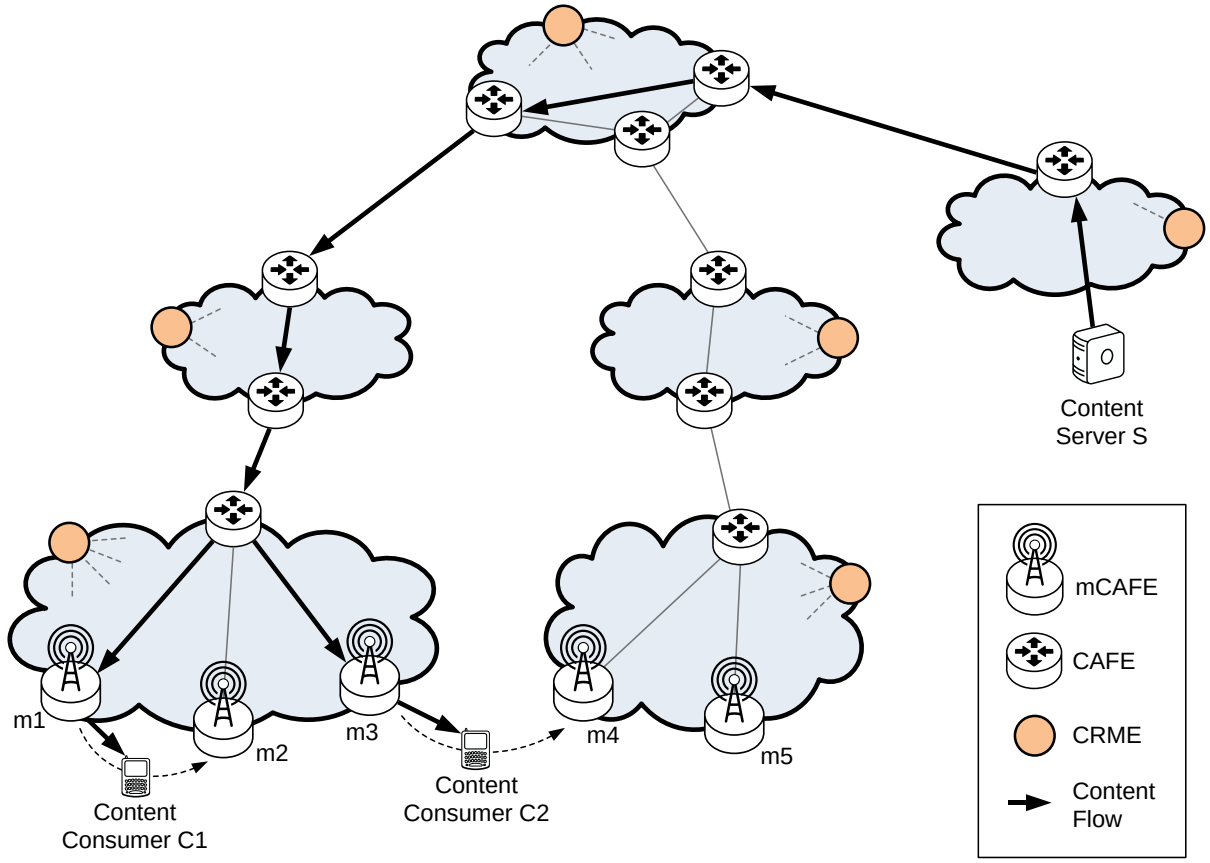


Figure 33: Example COMET handover scenario

Again based on Figure 33, Content Consumer, **C2**, is moving between mCAFEs, **m3** and **m4**, which belong to different administrative domains. Since **m3** is aware that **C2** will be performing an inter-domain handover, it will increase the scheduling priority perhaps more than that given to **C1**, since it knows that the handover process will take longer due to the need to carry out content resolution in the new network. However, **m3** will also be aware of the amount of video left to be streamed to the user, and so it may decide to increase the scheduling priority of the user to such a degree that allows it to finish buffering the rest of the video before reaching the edge of its coverage. This will avoid the need to perform content resolution for only the remaining fraction of the video.

8.3 Handover Mechanisms

Besides the mobility- and content-aware scheduling mechanisms designed to support mobile COMET users, various handover mechanisms are required to maintain session continuity as users move between mCAFEs. Such handover mechanisms can be categorised into *intra*-domain and *inter*-domain handovers.

Figure 34 shows the sequence diagram for intra-domain handover. In this scenario, a content client (CC) is moving between two mCAFEs belonging to the same domain, and all handover signalling is contained within the same domain to which **CC** is connected. Initially, **CC** is receiving content from its serving (srv) mCAFE. However, as it moves towards the edge of this mCAFE's coverage, it will select another mCAFE to which to connect based on the mCAFE Advertisement it receives. **CC** will then send a `Consume` message to the local CRME, indicating the ID of the content it is accessing, and information about the mCAFE to which it was last connected. This will cause the CRME to configure in the normal way the ingress CAFE, and the target (tgt) mCAFE which it will have inferred from the `Consume` message. However, it will also send a `Configure` message to remove the state in **mCAFE$_{srv}$** (if no other CCs are being wireless multicasted the same content)

and trigger it to send to **mCAFE$_{tgt}$** the CC's context information, which will contain information about the current status of the user's session, and potentially, information about the user's handover history to assist in optimising scheduling decisions.

The *inter*-domain handover scenario, shown in Figure 35, is slightly more complex than the intra-domain case. The main difference lies in the need for the user context information (containing information such as handover history and the status of the user's content session) to be transferred across domains. Therefore, when the CC issues a `Consume` message to the target CRME, providing it with information about the ID of the content it is accessing, together with its previous mCAFE, its new (target) CRME will send a Context Information Request message to its old (serving) CRME. In turn, the serving CRME will send a `Configure` message to **mCAFE$_{srv}$** to tear down the content state associated with the CC (again, provided that no ongoing consumption for the same content is present), and trigger it to send the context information of the CC to **mCAFE$_{tgt}$**. Thereafter, the interactions between entities are the same as for the intra-domain case.

Since the request for context information across domains may lead to extended handover latency, it may also be possible to support a more proactive handover approach, whereby the CC's serving domain is able to pre-empt the domain to which the CC is likely to hand over. This will not only allow the user context information to be transferred to the new domain in advance, but it will also allow the serving CRME to inform (publish) to the target domain information about the content being served to the CC. Thus, content paths can be resolved along optimal routes from the outset, potentially avoiding routes through more congested networks.

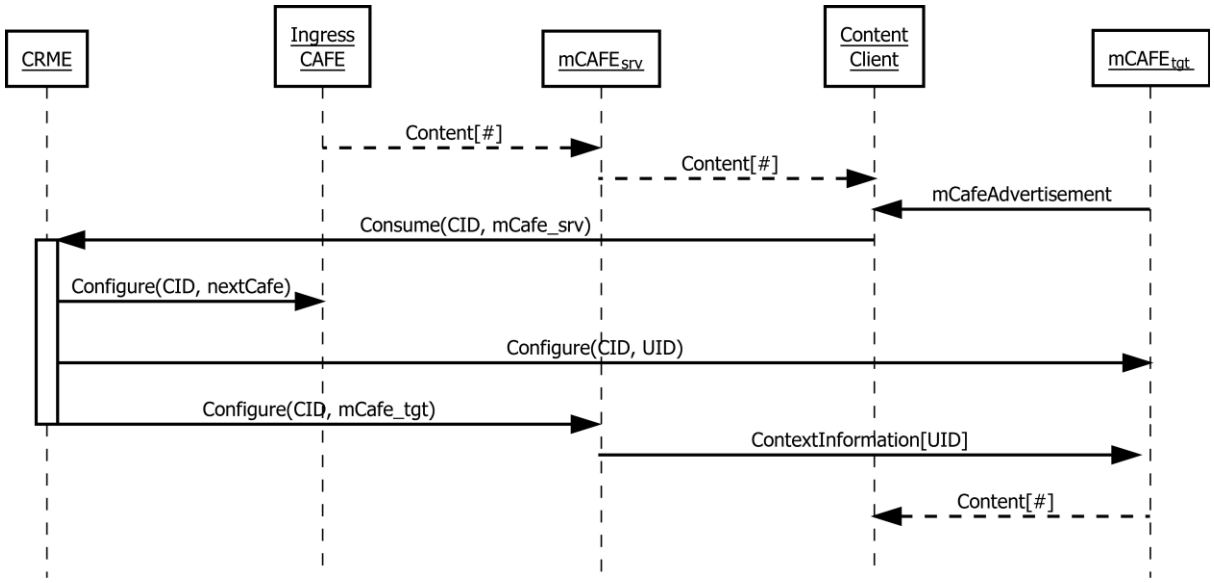


Figure 34: Intra-domain handover signalling

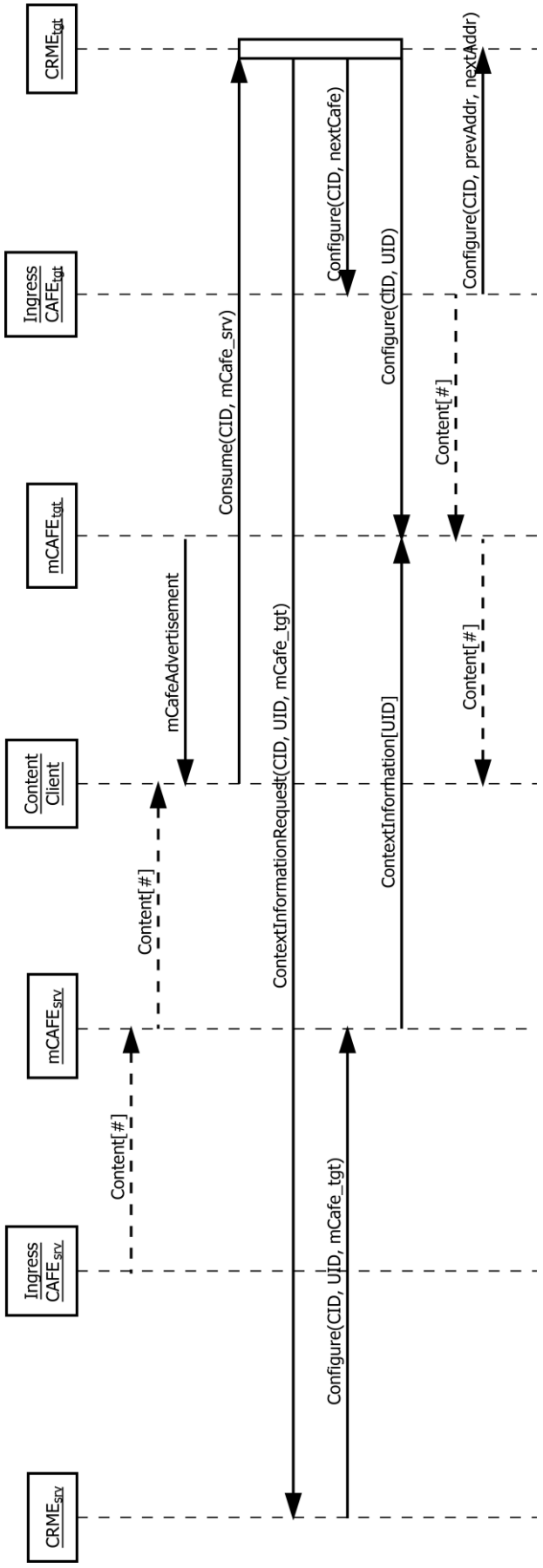


Figure 35: Inter-domain handover signalling

9 Congestion-aware Content Traffic Load-balancing

In Section 6 of COMET deliverable D3.2 [2], we presented a number of strategies for choosing the best server-path tuple to achieve the best possible performance. One of the main metrics we used in D3.2 was the "shortest path" in terms of the number of AS hops the request (and therefore, the content) has to travel. We considered that the shortest path is most probably going to be the one that will provide better performance. However, the potential use of hops in the server selection process raises the question of whether the number of IP routers, for intra-domain traffic, or the number of AS domains, for inter-domain traffic, provides a reliable indicator as to the quality of service a given path is expected to offer.

In order to ascertain both the current nature and the evolution of path hops for Internet traffic, preliminary studies were conducted on non-anonymised traces of inter-domain traffic provided by WIDE, the Japanese academic network providing connectivity for all national academic and research facilities. We are using this data set to get further insight as to whether choosing a path based on the number of hops is the best choice.

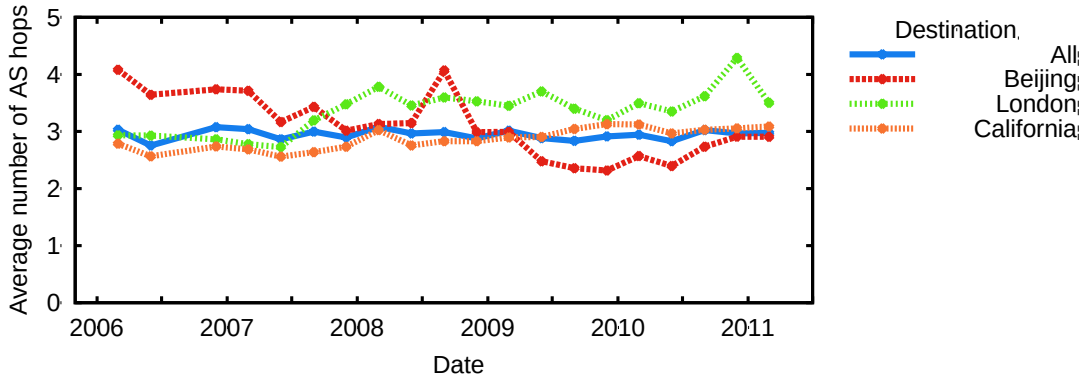


Figure 36: Average number of AS hops to remote host

Figure 36 shows the average number of AS hops traversed by flows over the past five years for traffic leaving the WIDE network. Additionally, we further refine the results for three different regional destinations (Beijing, London and California) to verify the suitability of the number of hops as a metric. By weighting the average by flow we introduce some bias towards destinations with greater number of flows, which we believe reflect the connections that are mostly likely to gain from content-centric architectures.

At the inter-domain level, the average AS path length has remained remarkably consistent over the past five years, in spite of an ever-changing underlying physical network. The number of AS hops also bears little relation to the distance travelled, or the delay experienced as can be verified in Figure 37. While Beijing is closer to Tokyo than California, existing infrastructure provides lower delays towards the latter. Conversely, despite being further both geographically and topologically, a content server located in London could provide a similar delay to a content server located in Beijing over many periods.

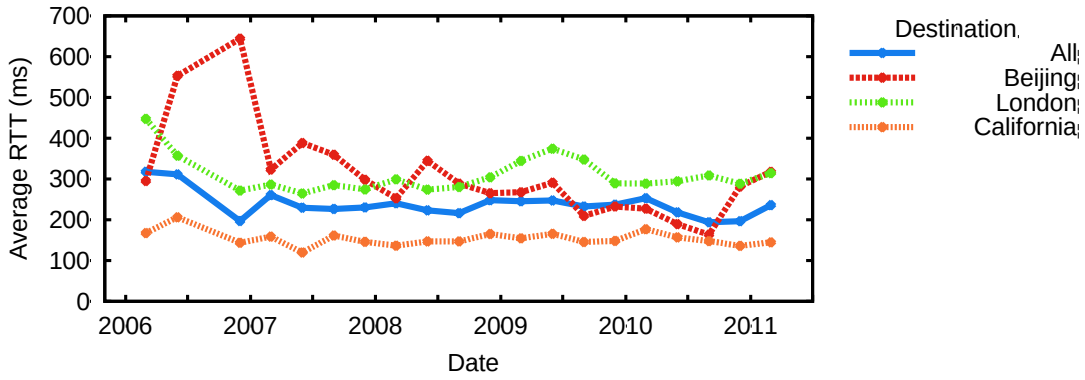


Figure 37: Average RTT to remote host

These preliminary results suggest that by conflating both policy and a measure of performance, AS hops provide an adequate indicator of neither. For inter-domain traffic, where intrinsic knowledge of the end-to-end path is unavailable, optimizing content delivery requires a holistic approach that takes into account not only network characteristics, but also feedback from end-systems on the perceived path quality. The following section describes the design and evaluation of PREFLEX, a novel traffic engineering solution that balances traffic in order to minimize loss. Hence, our observation based on the above set of results (and other extensive sets of experiments that we do not present here) suggest that a "short path" may not always be the best path as for the quality it is going to provide. We therefore, design PREFLEX, a novel traffic engineering solution that balances traffic in order to minimize loss. We argue that minimising loss is an important system property that has the potential to improve overall performance. The following section describes the design and evaluation of PREFLEX. Our initial evaluation results show that indeed choosing a server that sits at the end of a longer path may sometimes be more beneficial in terms of overall network performance.

9.1 PREFLEX overview

PREFLEX (Path RE-Feedback with Loss EXposure) [52] is an architecture which redraws the boundaries between transport and network layers in an attempt to make both aware of each other's actions. PREFLEX relies on three components:

1. Hosts provide the network with an estimate of loss using LEX (Loss EXposure).
2. Edge networks provide hosts with information on which outgoing path to use through PREF (Path RE-Feedback). In COMET terms, this functionality is assigned to the CAFE of the edge-domain to which the user is connected. Since CAFEs are situated at the edge of domains, the information provided to users by PREF need not travel far, resulting in significantly reduced traffic overhead.
3. A mechanism which assigns "flowlets" [53] to paths in order to balance loss.

The latter is the focus of this section. In the following sections we will briefly review the first two components.

9.1.1 Loss Exposure

Loss exposure is a mechanism by which hosts signal transport semantics at the network layer by marking the IP header, in a similar fashion to re-ECN [54]. This is initially signalled to the closest CAFE that the user is connected to. If a host does not support LEX, all traffic is unmarked and will appear to the network as legacy traffic. For LEX-capable traffic, there are three possible codepoints:

Codepoint	Meaning
Not-LECT	Not Loss Exposure Capable Transport
LECT	Loss Exposure Capable Transport
LEx	Loss Experienced
FNE	Feedback Not Established

Table 8: LEX markings

The FNE codepoint is set by the host when feedback on the path has not yet been established, and therefore there is no congestion control loop. This applies to the first packet of a flow, such as TCP SYN packets, or the first packet after a long idle time, such as keep-alives. An important distinction is that FNE marks the beginning of a “flowlet", rather than a flow. We define flowlets as a burst of packets which the sender does not wish to be reordered. As such, it is the end-hosts decision on how and when to divide traffic into flowlets. While a flow is a transport association between two endpoints, a flowlet is a sequence of packets which is expected to follow the same path. As an example, an IMAP connection may remain open several hours, but is composed of several distinct flowlets each time a client requests data from the server. Operating on flowlets rather than flows allows networks to balance traffic at a finer granularity and in a transport agnostic manner.

The remaining two codepoints, LECT and LEx in Table 8, are used to signal path loss. By default, traffic is marked as being Loss Exposure capable, and for every loss experienced by a host, the ensuing packet is marked with the Loss Experienced codepoint. In reliable transport protocols such as TCP this corresponds to marking every retransmitted packet accordingly, but can also be applied to non-reliable transport protocols with some form of feedback on loss.

9.1.2 Path Re-Feedback

With LEX, hosts provide information on both path loss and flowlet start at the network layer. We now focus on how edge networks specifically, which are naturally multi-homed, can reflect path diversity back to hosts.

In PREF (Path RE-Feedback) the network selects a preferred outgoing path for each incoming FNE packet (Figure 38). Upon receiving the first packet of a flowlet, the host performs a reverse lookup on the source address (step 1) and selects a path according to the perceived performance (2). The corresponding CAFE then associates the chosen path identifier to the packet and forwards the packet toward the host (3).

The treatment of outbound traffic by PREFLEX is illustrated on the rightmost figure. The host, having received indication of the preferred path, tags all traffic deemed sensitive to reordering with the given path identifier. As the PREFLEX-aware CAFE receives this marked traffic it updates statistics associated to path, aggregating loss for each destination prefix (4). It then discards the path identifier (5) and forwards traffic along the appropriate path (6).

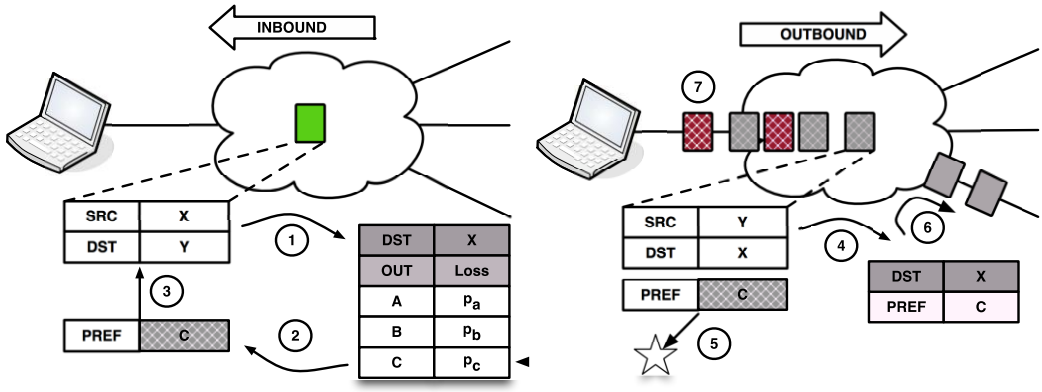


Figure 38: PREFLEX architecture

The resulting architecture provides many benefits. The balancing is both transport agnostic and allows flow state to be kept to a minimum, requiring policing at the edges only if congestion accountability is required. Additionally, path selection is receiver driven, aligning the stakeholder who can decide when to issue FNE packets with the stakeholder who benefits the most. The key to path selection is the information provided by the sender (point 7), which allows the network (i.e., the CAFE to which the user is connected) to calculate the percentage of path loss. We assume for the entirety of this section that each flow follows a single path and that a host does not override the path selected by its domain's CAFE, although neither is strictly necessary.

9.2 Balancing congestion

While PREFLEX provides an architecture within which transport and network layers exchange relevant information, it does not yet define how domains, networks or CAFEs select outgoing paths. In this section we present a solution for balancing traffic according to expected congestion, which according to our measurement study above has the potential to work in most cases.

A simplistic means of balancing traffic across links would be to equally split flows amongst paths, that is

$$f_i = \frac{1}{N}$$

where f_i is the probability of assigning a flow to path i of N available paths. This approach mimics equalisation. An alternative approach might be to balance traffic according to the proportion of traffic it carried over a previous interval.

$$f_i = \frac{T_i}{T}$$

where T_i is the traffic in bytes over path i and $T = \sum_j T_i$. This approach tends to be conservative, as it prefers paths which obtain higher throughput.

To illustrate the limitations of both approaches in isolation, we present a simple example which will be further expanded in Section 9.3. Consider a network balancing traffic to a given prefix between two paths with equal bottleneck capacity $L_1 = L_2 = 120\,Mbps$. For the duration of the experiment, a source sends traffic according to the same distribution. At $t = 300s$, cross-traffic is generated towards client C through L_1. At $t = 600s$, a greater quantity cross-traffic is generated towards client C through L_2. The results for both equalisation and conservative strategies are shown in Figure 39.

We first investigate the effect of equalisation, which is necessary to ensure all paths are continually probed. Equalisation alone, however, leads to inefficient use of the network if there is a mismatch in path capacity, as is clearly visible as congestion arises on either link. Such behaviour arises in traditional traffic engineering, which resorts to equalising traffic weighted by a local link's capacity. Where a bottleneck is remote and distinct however, such behaviour will lead traffic across all links to be roughly bound by the capacity of the slowest path, as can be seen between $t = 300s$ and $t = 600s$ where throughput over the first path is dragged down by congestion on a second path.

On the other hand, a conservative approach is more successful in saturating both links, but does so without correctly balancing loss. More worryingly, a pure conservative approach demonstrates the wrong dynamics, as highlighted between $t = 250s$ and $t = 300s$. Despite being saturated, the balancer continues to push more traffic towards path 1 while the second path remains under-utilized, dropping its throughput further.

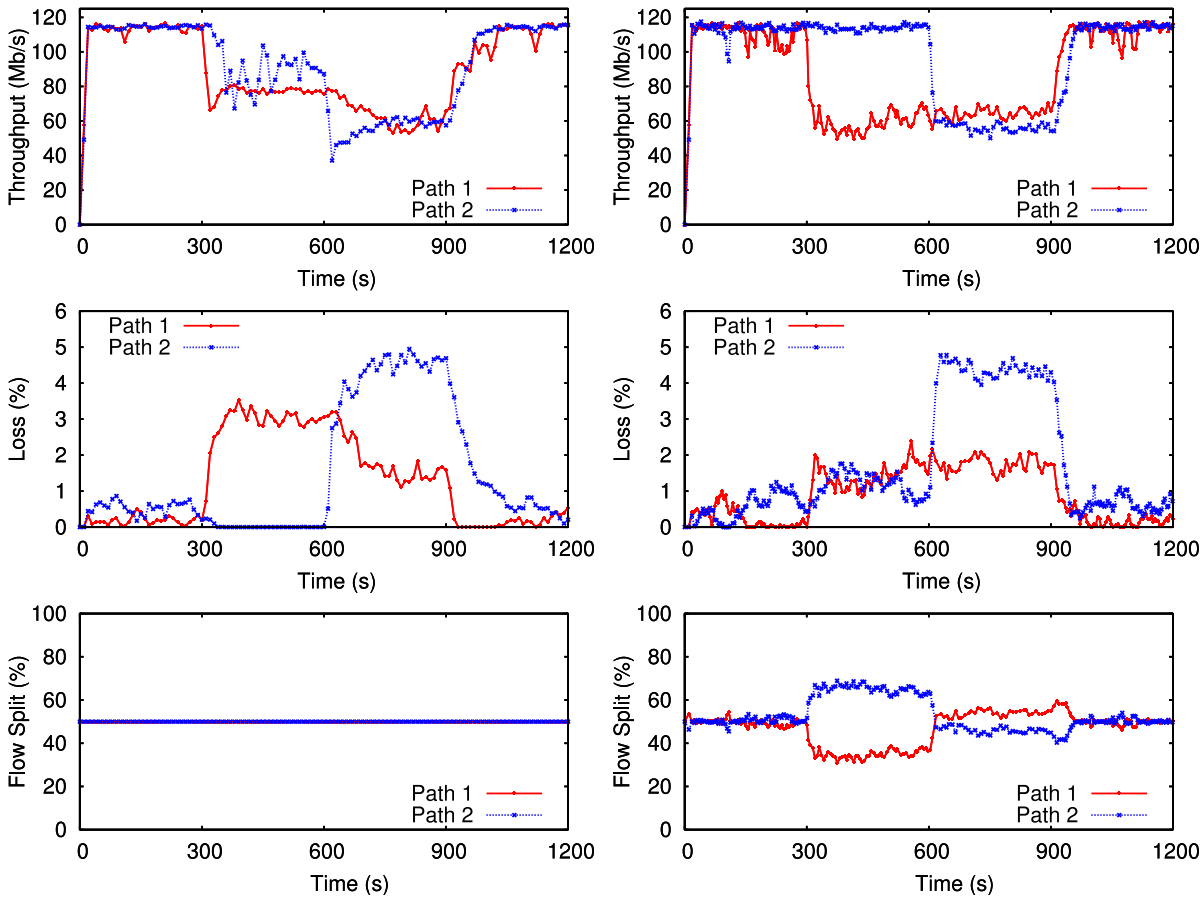


Figure 39: Equalisation (left) and conservative (right) traffic balancing strategies

A third strategy is to balance flows according to expected path loss, according to the equation derived in Appendix 15.

$$f_i = \frac{T_i^2}{R_i \sum_j T_j^2 / R_j}$$

where R_i is the total number of bytes retransmitted over path i. This loss-driven approach is shown in Figure 40. Predictably for small values of loss the loss-driven approach overreacts and flaps between either path. While the net effect of these oscillations does not result in significant losses, a conservative approach lends itself more naturally to situation where loss is too small to provide a reliable indicator on path quality.

An ideal balancer would use all three modes where appropriate. Equalisation is required for no prior knowledge of network conditions. Conservative mode is appropriate where no loss is experienced. Once paths become saturated, the loss-driven mode can balance congestion appropriately. The final equation as derived in appendix 15.1 uses all three and balances between modes according to the variance of loss. The results of the final, balanced solution is shown in Figure 40.

9.3 Evaluation

We evaluate PREFLEX through simulation in ns-3 [55]. Since PREFLEX balances traffic using loss rather than load, there is a need to emulate the end-to-end behaviour of traffic. This proves more challenging than analysis of existing traffic engineering proposals which typically only focus on adjusting load, since we wish to verify the impact of PREFLEX on end-user metrics.

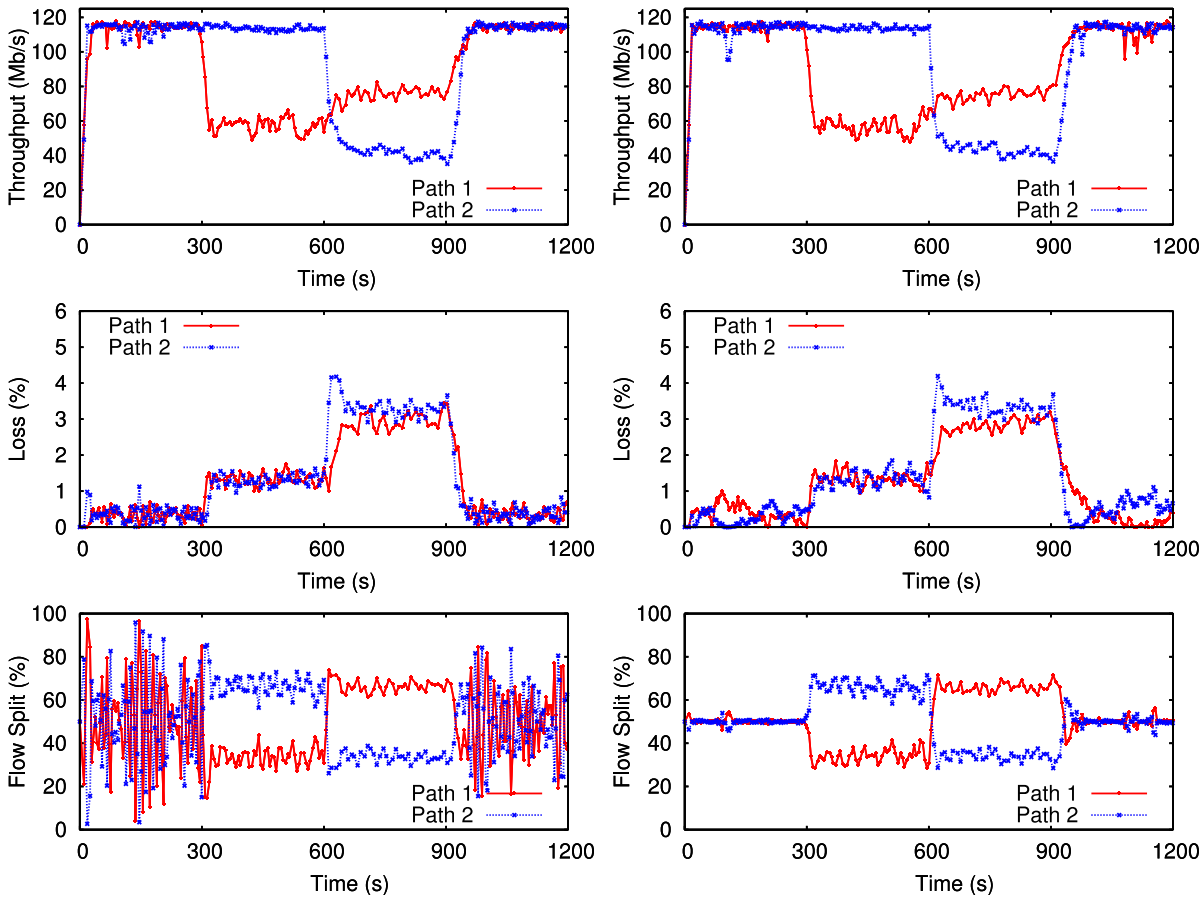


Figure 40: Loss-driven (left) and balanced (right) traffic balancing

9.3.1 Methodology

For all simulations we will use the topology displayed in Figure 41. The topology links a client domain C to a server domain S through N paths with equal bottlenecks L_i, and total bandwidth $B = \sum L_i$. While a domain is represented as a single entity in Figure 41, each domain is composed of a traffic generator connected to a router. Client C generates G simultaneous HTTP-like requests (or "gets") from S according to a specified distribution, described at the end of this section. As traffic flows from S to C, the router within S is responsible for balancing traffic over all available paths.

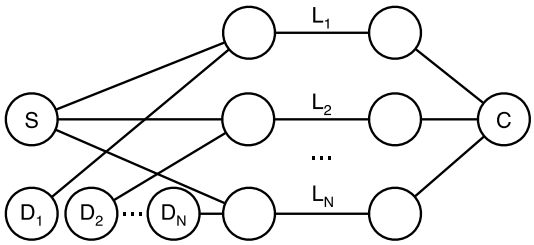


Figure 41: Simulation topology

Across simulations, as the number of paths increases, total bandwidth B and the number of simultaneous requests G is fixed. In this manner we wish to analyze how PREFLEX balances traffic as the granularity with which it can split traffic becomes coarser.

Since we are interested in evaluating how PREFLEX shifts traffic in response to loss, we introduce additional "dummy" servers D_i, which are connected to C through a single path. We partition the total simulation time T into $N + 2$ intervals starting on s_i, in which s_0 and s_{N+1} have no traffic to D_i.

Starting at time s_i, client C generates g_i requests to D_i according to the same distribution as used to server S. All requests to D_i end at time s_{N+1}. Equation 4 sets the start time s_i for requests to D_i as a function of total simulation time T and number of paths N. Likewise, Equation 5 sets the number of simultaneous requests g_i to D_i as a function of G, the total number of requests to S, and N.

$$s_i = T\frac{i}{N+2} \qquad \text{Equation 4}$$

$$\Theta_i = \frac{\frac{1}{N+i-1}}{\sum \frac{1}{N+1-i}}, \qquad g_i = G\Theta_i \qquad \text{Equation 5}$$

Figure 42 illustrates the number of simultaneous gets from C to D_i for $N = 2$ (used in the example shown in Figure 41) and $N = 4$. Generating cross-traffic in this manner serves two purposes. First, $\sum g_i = G$, so independently of the number of concurrent paths, the maximum load in the system is $2G$. However, as the number of paths increases, the fluctuation in load for each path becomes smaller, and so we will stress the sensitivity with which PREFLEX balances traffic. Secondly, the number of requests for each D_i over time is the same. Over timescale T, equalisation appears to be an acceptable strategy, however within each interval we will show it performs poorly achieve consistent behaviour. This is a fundamental limitation of offline traffic engineering, which is calculated over very long timescales and is unable to adapt as traffic routinely shifts.

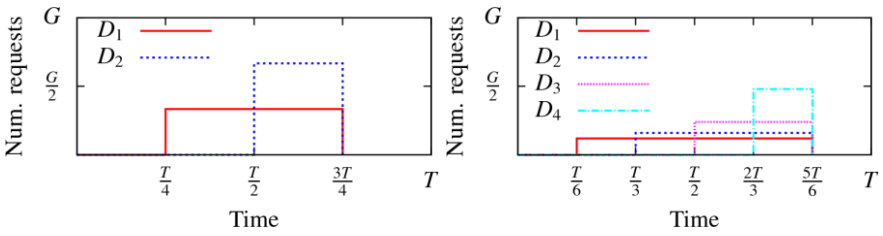


Figure 42: Number of requests from C to cross traffic servers D_i for $N = 2$ (left) and $N = 4$ (right)

We now specify the settings common to all simulations, including those previously shown in Section 9.2. Total simulation time T is set to 1200 seconds, while total bandwidth B is fixed at 240 Mbps. The number of requests G sent from C to S is set to 240. Upon completing, a request is respawned after an idle period following an exponential distribution with a $15s$ mean. Transfer size follows a Weibull distribution with an average value of 2 MB. These values attempt to reflect traffic to a single prefix with a file size that mimics the small but bursty nature of web traffic, which does not lend itself to being balanced by the end-host. PREFLEX is configured with $\beta_E = 0.05$, μ_{min} and $\delta = 0.005$.

9.3.2 Varying bottleneck distribution

We start by examining the case where all bottlenecks share the same bandwidth, $L_i = B/N$, and compare PREFLEX to equalisation, which mimics traffic engineering techniques based on hashing flow tuples and assigning them to a path. The goodput, calculated as the total data transferred to client C by flows completed within T, is shown for both equalisation and PREFLEX methods in Figure 41. While both saturate most available bandwidth, equalisation leads to disproportionate distribution of goodput amongst competing traffic. As loss is not equalised over all paths, the amount of goodput achieved by servers D_i differs despite demand being similar.

Equalisation, even when weighted according to local link capacity, is often prone to remote bottlenecks. We investigate the effect of differing bottlenecks by repeating previous simulations with the same total bandwidth B, but with L_i set proportionally to B in a similar manner to Equation 5, that is $L_i = \theta_i B$.

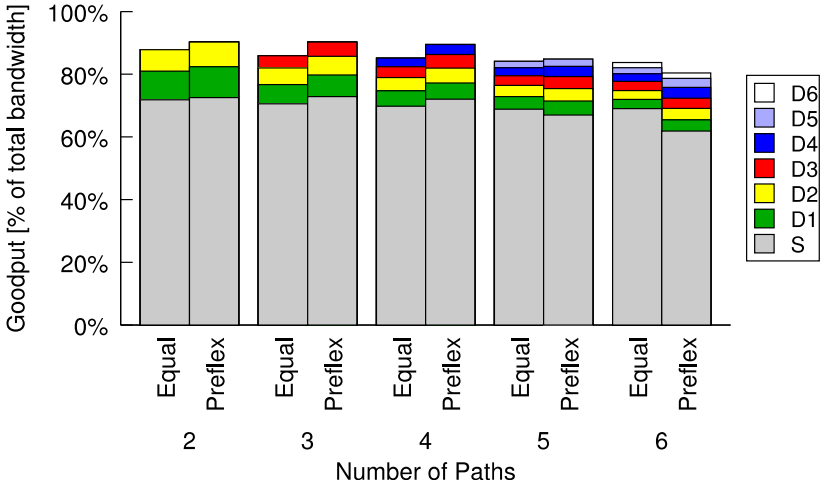


Figure 43: Goodput relative to B achieved by each server for equal capacity links

Figure 43 shows the goodput as a proportion of total link bandwidth for the case where all links have equal bandwidth. We vary the number of links N, and for each case compare equalisation (as illustrated in Figure 39 and PREFLEX as the balancing methods used. The bulk of goodput originates from server S, which is the only domain to be connected to all links. If traffic is correctly balanced, we expect to see servers D_{1-N} generate the same amount of goodput.

In this scenario, equalisation can be seen as the optimal static TE solution, yet both approaches bear similar performance. With no knowledge of topology, link bandwidth or expected traffic matrices, PREFLEX is able to adequately mimic the performance of the static TE solution for the case where such an approach is best suited.

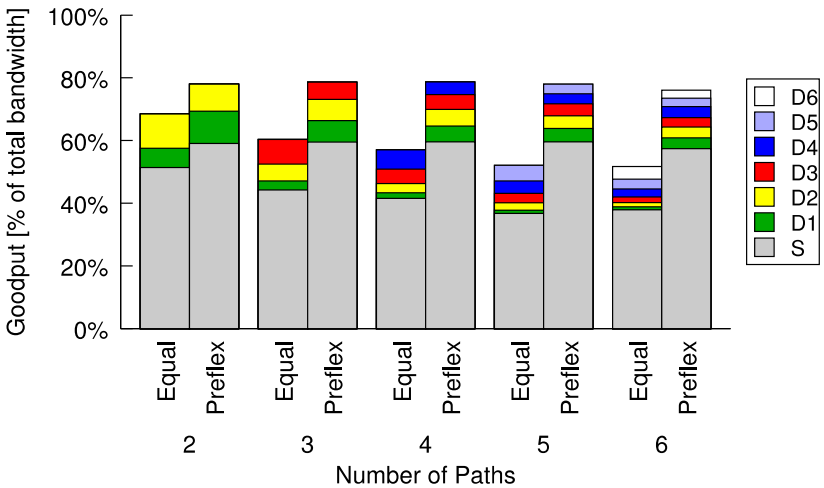


Figure 44: Goodput relative to B achieved by each server for different capacity links

Where bottleneck bandwidth is unequal however equalisation proves inadequate. Once again comparing goodput (Figure 44) we highlight two significant shortcomings of equalisation which PREFLEX overcomes. Firstly, goodput for S drops as N increases. Unable to realise it is

overloading a path, equalisation is reduced to sending traffic over each link at approximately the same rate as the most congested link. In contrast, PREFLEX detects congestion and adapts accordingly. Secondly, the incorrect distribution of traffic due to equalisation in S distorts the goodput of others servers. While in PREFLEX goodput from D_{1-N} is perfectly balanced, with equalisation traffic crossing the most congested links are directly affected by another domain's inability to distribute its traffic appropriately. It may seem unfair to judge equalisation for cases where there is a mismatch in link capacity, however this mismatch between link weight and path capacity arises regularly as operators continue to adjust traffic engineering according to local conditions, with little thought spared for the impact this may have further downstream.

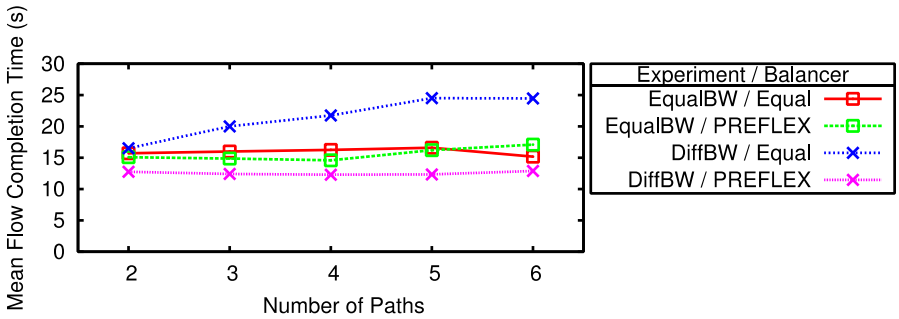


Figure 45: Mean average flow completion time for equal and differing bottleneck links

This impact is in turn perceived by users, who experience longer flow completion times, as shown in Figure 45. In the equal bandwidth case the flow completion time is similar for both balancers. Where bandwidth differs however, PREFLEX outperforms equalisation and maintains a stable performance when balancing over all six paths. This shows that the algorithm scales well as the number of available paths increases.

10 Summary and Conclusions

This document has described the final specifications of protocols and algorithms for the Content Delivery System of the COMET architecture. It starts from the definition and description of the general building blocks of the content delivery framework, including the COMET Class of Services and Routing awareness. Based on these building blocks, two distinct content delivery mechanisms are specified, namely stateless and stateful approaches. Towards the end, a number of advanced techniques for content delivery are introduced based on the proposed content delivery framework. Content caching aims to enhance content delivery efficiency and performance by means of caching popular content items at intermediate CAFEs along the delivery path, and mobility awareness caters for supporting flexible content forwarding towards mobile consumers.

Based on the design efforts and the implementation activities, we conclude on the following.

- The **COMET CoSs** are mapped into appropriate intra-domain network CoSs offered by particular domains on the end-to-end path. We defined three COMET CoSs that are: (1) Premium CoS, dedicated for delivering the content with QoS guarantees; (2) Better than Best Effort CoS (BTBE) delivers content over paths controlled by COMET traffic engineering algorithms. While BTBE does not introduce any kind of guarantees, it provides relative differentiation of the service; and (3) Best Effort CoS (BE), which transfers the content using IP routing paths in the network. Such CoS definition is able to support flexibly a wide range of content delivery service requirements from heterogeneous applications.

- **Routing awareness** offers necessary support for inter-domain content delivery through an offline process. Compared to the standard BGP protocol, the proposed routing awareness process is able to support advanced features that enables significantly enhanced content delivery functionalities at the Internet scale: (1) multi-criteria (QoS) routing, which allows RAE to build routing paths taking into account the requirements of COMET CoSs and (2) multipath routing as RAE builds a set of routes going towards a given destination prefix. On the other hand, implementation issues of realising routing awareness entity (RAE) within individual domains and its interaction with conventional BGP routing is still yet to be investigated in the rest of the project.

- In this document, two distinct content forwarding mechanisms have been proposed. The **stateless content delivery** approach assumes that CAFEs maintain only the neighbourhood (local) information, i.e., how to forward packet to the peering CAFEs. All information about selected content delivery path is stored in a COMET header attached to the original packet containing content payload. In such a case, the end-to-end content delivery paths are determined by the head node of the content delivery chain. On the other hand, the **stateful content delivery** approach follows the alternative philosophy of maintaining content states at CAFEs at the edge of individual domains. Such a feature shares some commonality with traditional multicast paradigms. The establishment of the content delivery path is coupled with the original content resolution process through the configuration interaction between each CRME and its local CAFEs in the same domain. Further routing optimisation is also enabled once the content source has been identified.

  An interesting direction for future research is the comparison of the distinct features between the two approaches at the top level. First of all, the main innovation of stateless content delivery comes from the source routing principle applied at the domain level, which allows for flexible selection of content delivery path for each content request. Also it does not require content states to be maintained at intermediate CAFEs in the end-to-end delivery chain. Certainly the trade-off is the necessity of embedding explicit path information (rule) inside content packets as additional overhead. On the other hand, the stateful approach needs necessary state maintenance at intermediate CAFEs. However, the good news is that it is able to intrinsically support point-to-multipoint content delivery

across domains, while such a feature has not been well supported by the current IP multicast paradigm. From this point of view, the stateful approach is more suitable for supporting the delivery of *popular content items* in the Internet.

- In order to support enhanced content delivery performance, e.g. reduced delay and bandwidth consumption, **content caching** has been also investigated as an advanced feature in the COMET system. In WP4, two distinct new caching schemes have been investigated from different angles. The *ProbCache* scheme caches contents *probabilistically* and most importantly approximates the capacity of the path and keeps the corresponding number of copies in the network. This way, it effectively distributes contents as they travel through the network in a fair manner among participating flows. On the other hand, our second scheme is a *centrality-based* in-network caching scheme. Our study on both string and scale-free topologies suggests that caching content only at specific important nodes (by the betweenness centrality measure) can effectively achieve better hop reduction ratio while maintaining similar server hit reduction ratio as compared to the ubiquitous solutions (e.g., the CCN approach proposed in [3]). We plan to continue the development and evaluation of the two approaches for content caching, and we also aim to derive useful policies for applying different caching schemes in specific application environments for content delivery.

- We have considered **mobility support** in the COMET content delivery system. Up until the time of writing, we have developed some initial technical ideas on how to achieve seamless content distribution for mobile consumers. We have specified a mobility support framework for COMET at both the system level in the form of intra- and inter-domain handover mechanisms, and at the link level in the form of air-interface scheduling mechanisms that take into account the mobility of the user and the content they are accessing. The main novelty of our work lies in the scheduling aspect since, to the best of our knowledge, no research has considered combining both mobility- and content-awareness in this regard. We plan to continue on this research direction in the rest of the project and obtain comprehensive results for performance evaluations.

- Last but not least, considering the content traffic dynamicity within each autonomous domain, we have also developed a new scheme called **PREFLEX** which uses content packet marking to estimate and balance loss across multiple paths. It requires no per-flow state or significant changes at routers, it is computationally simple to implement, and does not cause packet reordering. Thanks to such an adaptive content traffic control by edge network elements (CAFEs), optimised traffic engineering performance can be achieved for cost-efficient content delivery in dynamic environments.

11 References

[1] COMET Deliverable, "D4.1: Interim Specification of Mechanisms, Protocols and Algorithms for Enhanced Network Platforms", February 14th, 2011.

[2] COMET Deliverable, "D3.2: Final Specification of Mechanisms, Protocols and Algorithms for the Content Mediation System", November 15th, 2011.

[3] V. Jacobson, D. K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, Rebecca L. Braynard, "Networking Named Content," *ACM CoNEXT 09,* 2009, pp.1-12.

[4] IEEE Std. 802.1Q-2005, Virtual Bridged Local Area Networks, 19 May 2006.

[5] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)" Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 4271, January 2006.

[6] J. Uttaro et al., "Best Practices for Advertisement of Multiple Paths in BGP", Internet Engineering Task Force (IETF), Internet Draft,draft-ietf-idr-add-paths-guidelines-00.txt, November 2010

[7] J. Babiarz, K. Chang, F. Baker, "Configuration guidelines for DiffServ service classes", Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 4594, 2006

[8] K. Chan, J. Babiarz, F. Baker, "Aggregation of Diffserv Service Classes", Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 5127, 2008

[9] X. Masip-Bruin et al. "The EuQoS System: A solution for QoS Routing in Heterogeneous Networks". IEEE Communications Magazine, vol . 45 no 2, February 2007, pp. 96-103.

[10] G. Cheng, N. Ansari, "On selecting the cost function for source routing", Computer Communications, vol. 29, issue 17, 2006, Elsavier, pp.3602-3608.

[11] P. Khadivi, S. Samavi, and T. D. Todd, "Multi-constraint QoS routing using a new single mixed metrics", *J. Netw. Comput. Appl., vol.* 31, no. 4, November 2008, pp. 656-676.

[12] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 4271, January 2006.

[13] F. Baker, and P. Savola, "Ingress Filtering for Multihomed Networks," Internet Engineering Task Force (IETF) Request for Comments (RFC), RFC 3704, March 2004.

[14] T. Koponen, A. Ermolinskiy, M. Chawla, K. Hyun Kim, I. Stoica, B. Chun, and S. Shenker, "A Data-Oriented (and beyond) Network Architecture," in SIGCOMM, vol. 37, no. 4, pp. 181–192, 2007.

[15] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," *in Proc.* ACM SIGCOMM workshop on Information-centric networking, pp. 1–6, 2011

[16] W. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. Garcia de Blas, J. Ramon Salguero, L. Liang, S. Spirou, A. Beben and E. Hadjioannou, "CURLING: Content-ubiquitous resolution and delivery infrastructure for next-generation services," IEEE Communications Magazine, vol. 49, no. 3, pp. 112–120, 2011.

[17] D. Perino and M. Varvello, "A reality check for content centric networking," *in Proc.* ACM SIGCOMM workshop on Information-centric networking, 2011, pp. 44–49.

[18] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," *in Proc.* Re-Architecting the Internet Workshop, ReARCH '10, vol. 9, pp. 1–6.

[19] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in Proc. INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 126–134, 1999.

[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," IEEE/ACM Trans. Netw., vol. 8, pp. 281–293, June 2000.

[21] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in Proc. ACM SIGMETRICS conference on Measurement and modeling of computer systems, SIGMETRICS '90, pp. 143–152, 1990.

[22] P. R. Jelenkovic, A. Radovanovic, and M. S. Squillante, "Critical sizing of LRU caches with dependent requests," Journal of Applied Probability, vol. 43, no. 4, pp. 1013–1027, 2006.

[23] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," *in Proc. INFOCOM 2009, IEEE, pp. 2631*–2635, April 2009.

[24] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," *in Proc.* INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1416–1424, 2001.

[25] N. Fujita, Y. Ishikawa, A. Iwata, and R. Izmailov, "Coarse-grain replica management strategies for dynamic replication of web contents," Comput. Netw., vol. 45, no. 1, pp. 19-34, 2004.

[26] S. Srikantaiah, E. Kultursay, T. Zhang, M. Kandemir, M. J. Irwin, and Y. Xie "Morph cache: A reconfigurable adaptive multi-level cache hierarchy," High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, pp. 231–242, Feb 2011.

[27] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and Evaluation of CCN-Caching Trees," *in Proc.* IFIP NETWORKING, pp. 78–91, May 2011.

[28] S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, "SoCCeR: Services over content-centric routing," *in Proc.* ACM SIGCOMM workshop on Information-centric networking, 2011, pp. 62–67.

[29] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking," Computer Networks, to appear, 2011.

[30] V. Sourlas, G. S. Paschos, P. Mannersalo, P. Flegkas, and L. Tassiulas, "Modeling the dynamics of caching in content-based publish/subscribe systems," in ACM SAC, 2011, pp. 478–485.

[31] A. Detti and et al., "CONET: A content centric inter-networking architecture," *in Proc.* ACM SIGCOMM workshop on Information-centric networking, 2011, pp. 50–55.

[32] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. K. Ramakrishnan, "COPPS: An efficient content oriented publish/subscribe system," in ACM/IEEE ANCS, 2011, pp. 99–110.

[33] U. Lee, I. Rimac, and V. Hilt, "Greening the internet with content-centric networking," in Proc. Energy-Efficient Computing and Networking, 1st International Conference on, eEnergy '10, ACM, p. 179–182, 2010.

[34] M. Tortelli and et al., "A fairness analysis of content centric networks," *in Proc.* IFIP Int. Conf. Network of the Future, NoF, Paris, France, Nov. 2011.

[35] D. Rossi and G. Rossini, "Caching performance of content centric-networks under multi-path routing," Technical Report, 2011.

[36] G. Carofiglio, V. Gehlen, and D. Perino, "Experimental evaluation of storage management in content-centric networking," in IEEE ICC, 2011.

[37] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in IEEE INFOCOM, 2010.

[38] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in Proc. ACM SIGCOMM workshop on Information-centric networking, 2011, pp. 26–31.

[39] H. Che, Y. Tung, Z. Wang, "Hierarchical web caching systems: modelling, design and experimental results," IEEE Journal on Selected Areas of Communications, vol. 20, no. 7, pp. 1305–1314, Sept 2002.

[40] T. M. Wong, J. Wilkes, "My cache or yours? Making storage more exclusive," *in Proc.* USENIX Annual Technical Conference, pp. 161-175, Monterey, CA, 2002.

[41] N. Laoutaris, H. Che and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," Performance Evaluation, vol. 63, no. 7, pp. 609–634, July 2006.

[42] L. R. Izquierdo, Robert A. Hanneman, "Introduction to the Formal Analysis of Social Networks Using Mathematica", University of California, Riverside.

[43] M. Everett, and S. P. Borgatti, "Ego network betweenness," Social Networks, vol. 27, no. 1, pp. 31–38, Jan. 2005.

[44] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[45] H. Wang, J. M. Hernandez and P. Van Mieghem, "Betweenness centrality in a weighted network," Physical Review E 77, 046105, 2008.

[46] T. Koponen, M. Chawla, B-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker and I. Stoica, —A Data-oriented (and Beyond) Network Architecture, in Proc. ACM SIGCOMM '07, Kyoto, Japan, Aug. 2007.

[47] M. D'Ambrosio et. al., D-6.2 Second NetInf architecture description, 4WARD project deliverable. Available from: http://www.4ward-project.eu/index.php?s=Deliverables

[48] Y. Huang, and H. Garcia-Molina, "Publish/subscribe in a mobile environment," Wireless Networks, vol. 10, no. 6, pp. 643–652, November 2004.

[49] N. Fotiou, D. Trossen, and G. Polyzos, "Illustrating a Publish-Subscribe Internet Architecture," Technical Report TR10-002, PSIRP, 2010 [online]. Available: http://mm.aueb.gr/technicalreports/2010_TR10-002_PSIRP.pdf

[50] J. Lee, D. Kim, M. Jang, and B. Lee, "Proxy-based mobility management scheme in mobile content centric networking (CCN) environments," Consumer Electronics (ICCE), 2011 IEEE International Conference on, pp. 595–596, January 2011.

[51] H. J. Kushner, and P. A. Whiting, "Convergence of proportional-fair sharing algorithms under general conditions," *Wireless Communications, IEEE Transactions on*, vol. 3, no. 4, pp. 1250–1259, July 2004.

[52] Araújo, J.T., Rio, M., Pavlou, G.: A mutualistic resource pooling architecture. Third Workshop on Re-Architecting the Internet (Re-Arch), Philadelphia (2010)

[53] Sinha, S., Kandula, S., Katabi, D.: Harnessing TCP's burstiness with flowlet switching. 3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets) (2004)

[54] Briscoe, B., Jacquet, A., Cairano-Gilfedder, C.D., Salvatori, A., Soppera, A., Koyabe, M.: Policing congestion response in an internetwork using re-feedback. ACM SIGCOMM Computer Communication Review 35(4) (2005) 288

[55] Network Simulator 3. http://www.nsnam.org

12 Abbreviations

AS	Autonomous System
BE	Best effort service
BGP	Border Gateway Protocol
BTBE	Better-than-best-effort service
CAFE	Content-Aware Forwarding Entity
CC	Content Client
CCN	Content-Centric Network
CE²	Cache Everywhere, Cache Everything
COMET	COntent Mediator architecture for content-aware nETworks
CoS	Class of Service
CQI	Channel Quality Information
CRME	Content Resolution and Mediation Entity
ECN	Explicit Congestion Notification
FNE	Feedback Not Established
GRE	Generic Routing Encapsulation
ICN	Information-Centric Network
IP	Internet Protocol
IPLR	IP Packet Loss Ratio
IPTD	IP Packet Transfer Delay
KRT	Known Routes Table
LECT	Loss Exposure Capable Transport
LEx	Loss Experienced
LEX	Loss Exposure
LRU	Least Recently Used
mCAFE	Enhanced CAFE
NNC	Networking Named Content
PR	Premium service
PREF	Path RE-Feedback
PREFLEX	Path RE-Feedback with Loss Exposure
ProbCache	Probabilistic In-network Caching
PRT	Preferred Routes Table
QoS	Quality of Service
RAE	Routing Awareness Entity
RNMF	Radio Network Management Function
SNME	Server and Network Monitoring Entity
STREP	Specific Targeted Research Project

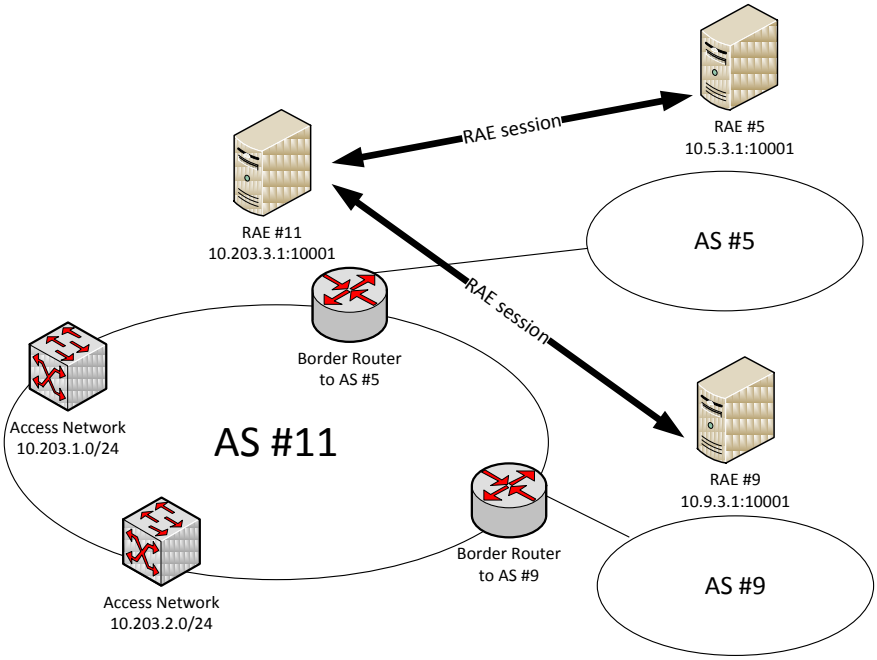
TCP	Transport Control Protocol
UMF	User Management Function
VLAN	Virtual Local Area Network

13 Acknowledgements

This deliverable was made possible due to the large and open help of the WP4 team of the COMET team within this STREP. Many thanks to all of them.

14 Appendix A: Exemplary configuration file for RAE

Configuration for RAE is stored in binary format. In order to prepare binary configuration file one can use a script prepared in the Python language. Below we show how to prepare this script file for an exemplary domain depicted on following figure.



14.1 Script example

The script should be located in the main RAE folder, e.g., /opt/comet/rae. Otherwise, one should adapt the `sys.path.append()` method call.

```
#!/usr/bin/python
import sys
sys.path.append("build/resources")
import config_pb2
cm = config_pb2.ConfigurationMessage()

#Limit for the number of paths maintained for a single destination prefix.
cm.limit_of_preferred_paths = 3

#Configure information about local AS
cm.local_as_number = 11
cm.local_ip_address = "10.203.3.1"
cm.local_port_number = 10001

#Limit for the intensity of the UPDATE messages exchanged between RAEs
cm.minimum_update_interval = 15

#Provide the listening port of the CME
cm.cme_ip_address = "10.203.3.1"
cm.cme_port_number = 9090
#Create peering to AS #5
peer = cm.peer_table.add()
peer.remote_as_number = 5
```

```
peer.remote_ip_address = "10.5.3.1"
peer.remote_port_number = 10001


#Set the TTL value that will allow to send IP packets to RAE in AS #5. It should
be the minimum #value allowing for correct operations.

peer.ttl_value = 3

#Describe QoS parameters on the link towards the AS #5 (2 classes).

c = peer.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
c = peer.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 5e-6
c.metric.maximum_delay = 0.005


#Create peering to AS #9

peer = cm.peer_table.add()
peer.remote_as_number = 5
peer.remote_ip_address = "10.9.3.1"
peer.remote_port_number = 10001


#Set the TTL value that will allow to send IP packets to RAE in AS #9. It should
be the minimum #value allowing for correct operations.

peer.ttl_value = 4

#Describe QoS parameters on the link towards the AS #9 (2 classes).

c = peer.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 2e-6
c.metric.maximum_delay = 0.002
c = peer.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 4e-6
c.metric.maximum_delay = 0.010


#Create prefixes for access networks: 10.203.1.0/24 and 10.203.2.0/24.

prefix = cm.prefix_table.add()
prefix.ip_address = "10.203.1.0"
prefix.prefix_length = 24
prefix = cm.prefix_table.add()
prefix.ip_address = "10.203.2.0"
prefix.prefix_length = 24


#Provide provisioning information about the intra-domain part: from prefix
10.203.1.0/24 to #10.203.2.0/24 prefix (2 classes).

prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.1.0"
prov.source.prefix.prefix_length = 24
prov.sink.prefix.ip_address = "10.203.2.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
```

```
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from prefix 10.203.1.0/24 to #border router leading to AS #5 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.1.0"
prov.source.prefix.prefix_length = 24
prov.sink.as_number = 5
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from prefix 10.203.1.0/24 to #border router leading to AS #9 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.1.0"
prov.source.prefix.prefix_length = 24
prov.sink.as_number = 9
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from prefix 10.203.2.0/24 to #10.203.1.0/24 prefix (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.2.0"
prov.source.prefix.prefix_length = 24
prov.sink.prefix.ip_address = "10.203.1.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 2e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 2e-6
c.metric.maximum_delay = 0.002
```

#Provide provisioning information about the intra-domain part: from prefix 10.203.2.0/24 to #border router leading to AS #5 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.2.0"
prov.source.prefix.prefix_length = 24
prov.sink.as_number = 5
c = prov.class_table.add()
```

```
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from prefix 10.203.2.0/24 to #border router leading to AS #9 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.prefix.ip_address = "10.203.2.0"
prov.source.prefix.prefix_length = 24
prov.sink.as_number = 9
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from border router connecting AS #5 to prefix 10.203.1.0/24 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.as_number = 5
prov.sink.prefix.ip_address = "10.203.1.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from border router connecting AS #5 to prefix 10.203.2.0/24 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.as_number = 5
prov.sink.prefix.ip_address = "10.203.2.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

#Provide provisioning information about the intra-domain part: from border router connecting AS #5 to border router leading to AS #9 (2 classes).

```
prov = cm.provisioning_table.add()
prov.source.as_number = 5
```

```
prov.sink.as_number = 9
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

```
#Provide provisioning information about the intra-domain part: from border router connecting AS #9 to prefix 10.203.1.0/24 (2 classes).
```

```
prov = cm.provisioning_table.add()
prov.source.as_number = 9
prov.sink.prefix.ip_address = "10.203.1.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

```
#Provide provisioning information about the intra-domain part: from border router connecting AS #9 to prefix 10.203.2.0/24 (2 classes).
```

```
prov = cm.provisioning_table.add()
prov.source.as_number = 9
prov.sink.prefix.ip_address = "10.203.2.0"
prov.sink.prefix.prefix_length = 24
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

```
#Provide provisioning information about the intra-domain part: from border router connecting AS #9 to border router leading to AS #5 (2 classes).
```

```
prov = cm.provisioning_table.add()
prov.source.as_number = 9
prov.sink.as_number = 5
c = prov.class_table.add()
c.cos_id = 1
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.0005
c = prov.class_table.add()
c.cos_id = 2
c.metric.loss_ratio = 1e-6
c.metric.maximum_delay = 0.001
```

15 Appendix B: Flow Balancing according to Expected Path Loss

15.1 Loss balancing

Consider LEX-capable traffic that is explicitly marked as either being retransmitted or non-retransmitted from a single origin prefix to a single destination prefix, with a number of possible paths and within a single time period. Let N be the number of paths (numbered $1, \dots, N$). Let T_i be the number of bytes sent down path i for the previous time period. Let R_i be the number of marked as retransmissions down path i for the previous time period. Let $R = \sum_i R_i$ and $T = \sum_i T_i$. While R_i does not strictly represent the number of lost bytes, the ratio of R_i/T_i should be a good approximation of the loss rate within period i. A starting assumption is that it is desirable to equalise the proportion of lost bytes on all interfaces – that is make R_i/T_i equal for all i – and by doing so loss is balanced.

The loss rate $\rho_i = R_i/T_i$ is an unknown function of T_i and B_i the bandwidth of the link. It is, however, likely that the loss rate is increasing (or at least non-decreasing) with T_i. Similarly, the loss rate is decreasing (or at least non-increasing) with the unknown B_i. Assume then that whatever the true function, for a small region around the current values of T_i and B_i then it is locally linear $\rho_i = k_i T_i/B_i$ where k_i is an unknown constant. Substituting gives

$$\frac{B_i}{k_i} = \frac{T_i^2}{R_i} \qquad \text{Equation 6}$$

Now consider the next time period. Use the dash notation for the same quantities in the next time period. In typical time periods $E[T'] = E[T]$ (since, for example, if the next time period on average had more traffic, the overall amount of traffic would be growing – while this is true in the year-on-year setting, this effect is negligible in the time scale discussed here). Now $\rho_i' = R_i'/T_i' = kT_i'/B_i'$. Choose the T_i' to make all $R_i'/T_i' = C$ (hence all equal) where C is some unknown constant. Therefore $T_i' = CB_i'/k_i'$ but if this is still near our locally linear region then $B_i'/k_i' = B_i/k_i$ and B_i/k_i is determined by Equation 6, hence the predicted distribution is $T_i' = CT_i^2/R_i$. Now it is necessary to calculate C by summing over i. $T = C/\sum_i T_i^2/R_i$ and hence $C = T/\sum_i T_i^2/R_i$. This gives the final answer:

$$T_i' = \frac{TT_i^2}{R_i \sum_j (\frac{T_j^2}{R_j})} \qquad \text{Equation 7}$$

There is, of course, a problem when $R_i = 0$. Because of the assumption that the traffic is assigned in inverse proportion to loss rate then a branch with no loss would naturally have all the traffic assigned to it. To correct this, we increment R_i for each path by one maximum segment size (MSS).

Because only local linearity is assumed then large adjustments of the traffic split are likely to cause problems. It is also useful to send a small amount of probe traffic down each route even if the loss rate is high. In the absence of other information (for example when there is no loss) it is useful to assign traffic according to the current traffic throughput split. Therefore there are three tendencies to be accounted for, the loss-equalisation tendency from Equation 7, the conservative tendency to keep the traffic split the same as the throughput split in the previous period and the equalisation tendency to keep the traffic equally split on all interfaces. Call the traffic split assigned to path i by each of these schemes $T(E)_i$, $T(C)_i$ and $T(L)_i$ where E, C and L stand for “equal", “conservative" and “loss-driven".

Then our final distribution of traffic across all links is:

$$T_i = \beta_E T(E)_i + \beta_C T(C)_i + \beta_L T(L)_i \,,$$

where the $\beta_{\blacksquare}$ are user set parameters in (0,1) such that $\beta_E + \beta_C + \beta_L = 1$. Now $T'(C)_i = T_i$ and $T'(E)_i = T/N$ where N is the number of interfaces. This gives the equation for the desired flow split f_i', which represents the probability with which path i will be assigned to a new flow:

$$f_i' = \frac{T_i'}{T} = \beta_E \cdot \frac{1}{N} + \beta_c \cdot \frac{T_i}{T} + \beta_L \cdot \frac{T_i^2}{R_i \sum_j \left(\frac{T_j^2}{R_j}\right)}$$

Equation 8: Balanced mode equation

15.2 Balancing between conservative and loss-driven modes

We wish to tune PREFLEX to balance between conservative and loss-driven modes for differing regimes of loss. Equalisation is required in some measure as every path must attract some traffic if it is not to fall out of use. If this is not the case, a path with relatively high loss will never be probed to determine whether it has improved. The remaining two components must be adjusted to be able to respond adequately to loss while not being overly sensitive to statistically insignificant fluctuations in loss. We define γ to replace both β_C and β_L in Equation 8 and adjust between conservative and loss-driven modes:

$$f_i' = \beta_E \cdot \frac{1}{N} + (1 - \beta_E)\left(\gamma \cdot \frac{T_i}{T} + (1 - \gamma) \cdot \frac{T_i^2}{R_i \sum_j \left(\frac{T_j^2}{R_j}\right)}\right)$$

As a result, β_E may now vary between [0,1]. A simple but effective value for γ is to define a minimum average loss μ_{min} below which we do not react to loss, and react increasingly as the average loss μ becomes more significant:

$$\gamma = \begin{cases} \frac{\mu_{min}}{\mu}, if\ \mu > \mu_{min} \\ 1, if\ \mu \leq \mu_{min} \end{cases}$$

This completes the PREFLEX balancer, as shown in Figure 40. The evolution of γ for the example shown in Figure 40 is shown in Figure 46 (left) The value of μ_{min} was set to 0.005.

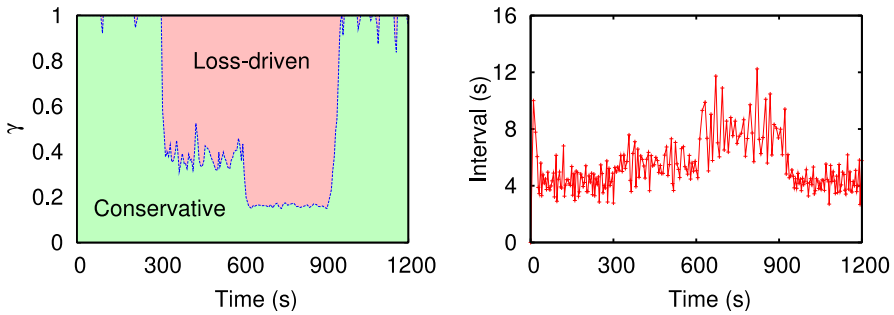


Figure 46: Parameters γ (left) and μ for simulation shown in Figure 40