ICT-257422

CHANGE

CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions

Specific Targeted Research Project

FP7 ICT Objective 1.1 The Network of the Future

D5.2: Architecture for Application Deployment

Due date of deliverable: 31 December 2012

Actual submission date: December 31, 2012

Start date of project	1 October 2010
Duration	36 months
Lead contractor for this deliverable	University Politehnica of Bucharest
Version	Final, December 31, 2012
Confidentiality status	Public

Abstract

CHANGE aims to bring flow processing to the masses. The project has delivered so far a scalable software platform for flow processing together with a set of primitives and a user interface to allow inter-platform, wide area flow processing. In this deliverable we discuss the list of applications selected in Deliverable 5.1 and the way they may be deployed using CHANGE . The report briefly analyzes each application and its motivation. The bulk of the discussion centers on how the CHANGE primitives can be used to support these applications with regard to functionality, security and scalability. We find that most applications benefit significantly from the scalability, security and redirection capabilities offered by CHANGE , while security applications benefit from invariants and their implementation. Finally, we discuss where CHANGE primitives are not enough to properly support certain applications, and discuss possible next steps.

Target Audience

Experts in the field of computer networking, the European Comission.

Impressum

Full project title	CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions
Title of the workpackage	WP5: Deployment and Applications
Editor	Costin Raiciu, University Politehnica of Bucharest (PUB)
Project Co-ordinator	Adam Kapovits, Eurescom
Technical Manager	Felipe Huici, NEC
Copyright notice	© 2012 Participants in project CHANGE

List of Authors

Authors	Felipe Huici (NEC), Mehdi Bezahaf (ULANC), Georgios Smaragdakis (TUB), Costin Raiciu (editor, PUB)
Participants	NEC, ULANC, PUB, TUB
Work-package	WP5: Deployment and Applications
Security	Public (PU)
Nature	R
Version	1.0
Total number of pages	28

Contents

List of Figures

List of Tables

1 Introduction

Flow processing is already taking over the Internet, with many transparent middleboxes being already deployed. CHANGE takes the next logical step giving users the ability to specify, reason about and instantiate complex in-network flow processing, while assuring providers that security rules are obeyed.

The basic building block of the CHANGE architecture is a scalable software platform that can run processing for many users simultaneously while ensuring isolation between the users. The platform is based on an OpenFlow switch that splits traffic among a number of commodity servers that do the actual processing of packets. The platform software supports dynamic creation and termination of software functionality, including the ability to scale processing resources to match the traffic volume.

The true power of CHANGE comes when a multitude of these platforms are deployed and collaborate to implement flow processing. The user interface is very simple: a few primitives are provided that cover a wide range of use cases. These primitives include the ability to originate (O), filter (F) or modify packets (M), as well the ability to change a packet's path via reroute (RR) or its destination (redirect, RD). End-hosts can express processing they want instantiated using these simple primitives. The processing is verified against a strict security policy that strives to ensure that CHANGE platforms cannot be used for malicious purposes.

An architecture is as useful as the applications it supports. This report describes architectural guidelines for the deployment of most of the applications selected in Task 5.1. The applications selected and discussed in this document are the remote firewall, inbound traffic engineering, shadow networks, content distribution networks, content-aware traffic engineering, and finally distributed intrusion detection systems.

We ask if the primitives selected do capture the required processing and analyze the desired locations of the flow processing platforms as well as the expected scalability of the network. We find that most applications' needs are met natively by the CHANGE architecture. In the next sections we discuss each application in turn, providing a brief overview of the application and the motivation behind it, as well as an implementation architecture for it. We conclude with a brief discussion of functionality missing in CHANGE that could be useful in deployments where there is greater trust between the platform and its users.

2 Applications

2.1 Firewall

Corporations and users alike run firewalls to protect against unwanted traffic. Running a firewall requires the ability to handle many rules and quickly change them when needed while being able to scale to traffic volume variations spanning several orders of magnitude.

While in principle it is easy to implement a firewall using commodity OSes (e.g. Linux), scaling such a firewall requires provisioning for the peak traffic volume (which can be potentially maliciously generated) with massive costs. In this light, outsourcing the firewall to a shared infrastructure becomes very sensible: in a shared infrastructure it is much cheaper to accommodate bursts of traffic relating to one user as long as user "spikes" are not aligned in time. such statistical multiplexing allows the infrastructure provider to charge customers only for what they use, and not for the provisioned capacity.

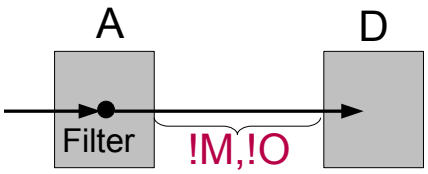

Figure 2.1: Using CHANGE to implement a remote firewall. Traffic is verified at a remote CHANGE platform and complying packets are then sent to the destination.

Using public clouds for offloading firewall functionality seems a very good proposition as long as the cloud is "close" to the real destination. With RTT being one of the major limiting factors for web performance, increasing the RTT is not an option for many sites [1]. Today's public clouds such as Amazon EC2 have deployed datacenters in a few vantage locations, typically one datacenter per continent. Using public clouds would therefore increase the RTT significantly, especially for local traffic that would need to travel to the remote cloud and back to its real destination.

CHANGE platforms are comparatively smaller than public clouds and more numerous, being deployed by ISPs and cloud providers alike. Users can select from the multitude of platforms the one that is closest to them and avoid the RTT increase. This scenario is shown in Figure 2.1: traffic is attracted to the CHANGE platform A where it is filtered. The complying traffic is then redirected to the destination D. We now discuss the requirements that this application puts on the CHANGE architecture:

- **Platform Selection**: The selection of the platform needs to take into account the distance from the destination to the platform. While CHANGE does not provide natively a discovery service, one possible solution would be to return to the user a set of platforms that are deemed close based on the user's location inferred with IP address geo-location. The user can then simply ping the platforms and select

[1] See Google's IETF efforts to cut down the number of RTTs by increasing the initial congestion window and reducing the three-way handshake.

the closest one. To attract traffic to the chosen platform DNS redirection will be used to resolve D's DNS name to A's IP address.

- **Primitives**: The firewall needs to attract traffic to platform A, Read (R) it A and then apply D's Filters (F). The traffic is then Redirected (RD) bidirectionally to the destination.

- **Redirection** can be implemented either with a full proxy (which hides the real source address from D), using Multipath TCP (if the traffic supports it) or by using Loose Source Routing inside a tunnel.

- **Invariants:** The traffic leaving platform A complies with D's security policy, but traffic arriving at D may not: if the edge between A and D crosses unknown middleboxes existing packets could be modified arbitrarily and new packets could be inserted. Note that traffic arriving directly from other hosts that try to bypass the firewall falls under the category of "originated" traffic. For the processing to be correct, the destination must impose an invariant on this edge, forbidding M and O primitives. Implementing the invariant could be a no-op if D resides in the same administrative domain as A (e.g. D is a customer of ISP A), or it could require a tunnel such as IPSEC AH if the edge A-D crosses an unknown network.

- **Security:** It is easy to see that the CHANGE security allow firewalls to be deployed. The only operation that needs to be checked is the redirection from A to D, but this is allowed since D requests it and proves it owns the destination address.

- **Scalability:** is mostly concerned with the ability to scale up and down the resources allocated to filtering on platform A, which is natively provided by the CHANGE platform. If the capabilities of the platform are exceeded than traffic could be split between platforms A and B via DNS load balancing.

2.2 Inbound Traffic Engineering

One of the problems faced today by the Internet is the ability of a destination site to load balance traffic destined to it via multiple links. Multi-homing is becoming the norm for big server farms, yet the routing system offers no easy solution for load balancing traffic across the different links. Typically these links would be to different ISPs to increase fault tolerance, but load balancing traffic across them is difficult.

If the same provider-independent address is advertised via both links, load-balancing is coarse grained as it will involve routing tweaks (in fact hacks) such as AS path prepending or AS path poisoning; in such setups one link is just used as a backup. If different addresses are advertised on the two links, load-balancing will be implemented by the destination site, and it can only use DNS which is also rather coarse grained. What's worse, DNS caching may prevent quick changing of load-balancing policy when one of the links fails.

CHANGE offers a more elegant solution to the inbound traffic engineering problem. Traffic is first attracted into a CHANGE platform nearby the origin via DNS redirection, much like in the CDN scenario. Then, fine

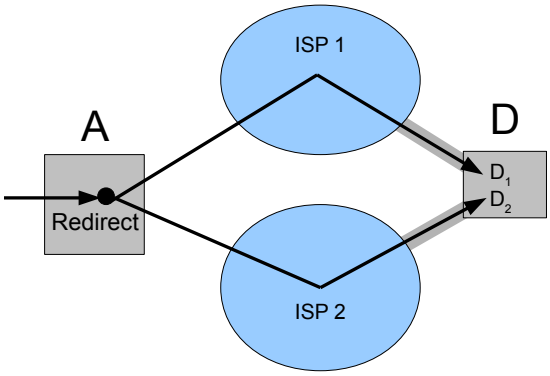

Figure 2.2: Using CHANGE to implement Inbound Traffic Engineering. Traffic is attracted to remote CHANGE platforms and then redirected using fine-grained, destination-defined policies.

grained policy can be used to split the traffic onto the links identified by provider-dependent addresses. This scenario is depicted in Figure 2.2.

To implement inbound traffic engineering we need the following tools from CHANGE :

- **Platform Selection** must take into account user location, as traffic engineering will be more effective if the destination can exert control over a bigger part of the end-to-end path. Geo-location will be used to select platforms nearby the client (in fact, close to its DNS resolver). Traffic Attraction will be implemented with DNS: D's DNS name will be resolved to the IP address of CHANGE platform closest to the client (e.g. A's IP address).

- **Primitives**: Once traffic reaches a CHANGE platform it will be load-balanced via the two paths corresponding to D's two IP addresses. In detail, a subset of the traffic will be Redirected them towards IP address D_1, and the rest to IP address D_2.

- **Redirection:** Redirection can be implemented either with a full proxy (which hides the real source address from D), using Multipath TCP (if the traffic supports it) or by using Loose Source Routing inside a tunnel.

- **Invariants:** The only invariant here is that all packets must be load balanced to D. This can be easily checked by parsing the configuration file and seeing if all possible traffic can reach the destination.

- **Security:** It is easy to see that the CHANGE security rules allow inbound traffic engineering. The only operation that needs to be checked is the redirection from A to D_1 or D_2, but this is allowed since D requests it and proves it owns the destination addresses.

- **Scalability:** is mostly concerned with the ability to scale up and down the resources allocated to any individual platform (e.g. A), which is natively provided by the CHANGE platform. If the capabilities of the whole platform are exceeded than traffic could be split between multiple nearby platforms (e.g. A and B) via DNS load balancing.

2.3 Shadow Networks

Running a large production network is a challenging task, with operators expected to provide stable, always-functioning networks. This collides with the reality that oftentimes software or equipment upgrades are necessary, both in order to ensure the network's availability but also to improve its functionality or capacity. Further, when things do go wrong, attempting to troubleshoot a network without adversely affecting its traffic presents additional challenges.

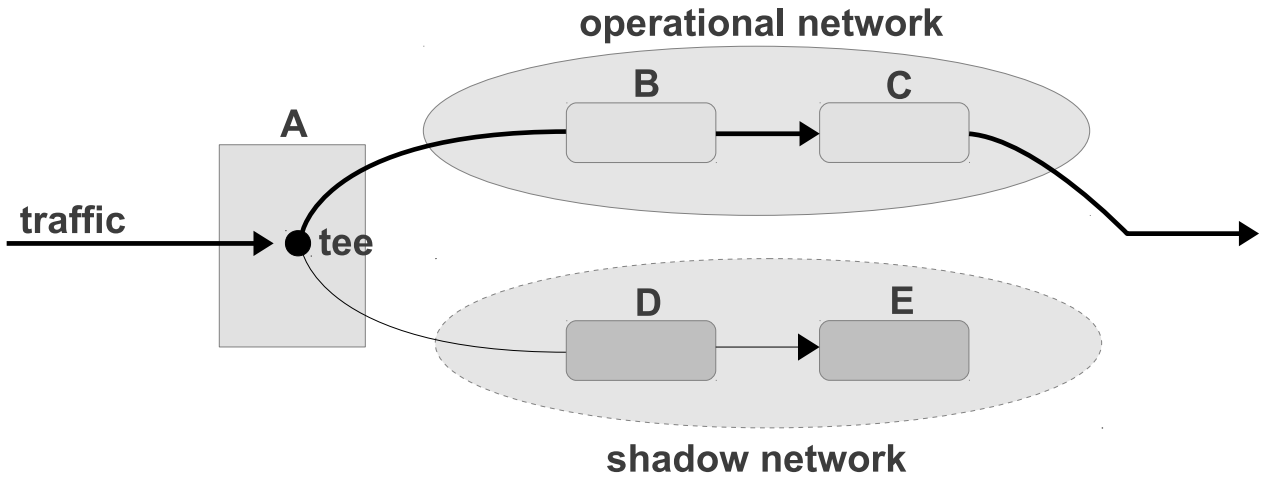

Figure 2.3: High-level depiction of a shadow network. Operational traffic is replicated onto this network to perform troubleshooting or to try out soon-to-be-deployed features.

In order to ease with troubleshooting and deployment of new features **without** affecting operational traffic, the CHANGE architecture and platforms can be used to direct a copy of the traffic (or a subset of it) to a separate network for further processing. This *shadow network* is shown in figure 2.3. In this case, CHANGE platform A takes care of replicating all or a subset of operational traffic and redirecting the copied packets to other platforms (D and E), in essence creating virtual shadow slices across these.

Once the replicated traffic arrives at the shadow network, a number of important actions are possible. For instance, a user of the CHANGE architecture, such as an operator, could instantiate monitoring in platform D and a DPI on platform E in order to troubleshoot a network problem or investigate suspect behavior. The advantage of the shadow network is that this can be done on real traffic without actually affecting the operational traffic nor putting additional load on equipment in the operational network. Being able to select a subset of the flows aids in being able to perform the troubleshooting without putting unrealistic demands on the processing platforms. Better still, shadow networks can be created and removed on-the-fly, as problems arise.

Another important use of shadow networks involves new deployments, technologies and upgrades. Adding new features to an operational network is often tricky, since operators are often weary of any unpredictable consequences the change might result in; the relatively slow deployment of "standard" features such as IPv6 is witness to this. A shadow network allows an operator to try out new technologies without risking unintended effects on client traffic. Once the new feature is thoroughly tested in the shadow network on the real traffic

(actually a copy of it), the new feature can be enabled on the operational network, or the shadow network "switched" with the operational one.

We now look into the various requirements that this application puts on the CHANGE architecture:

- **Platform Selection**: The selection of the platform in charge of replicating and redirecting traffic (platform A in figure 2.3) is likely to be performed manually, since it needs to be located where the operational traffic is flowing through (the operator would know where this point in his network would be). Alternatively, traffic could be directed towards an off-path platform (see section below on redirection). For the platforms performing the actual troubleshooting or that have the new features to be tested the selection can be automated. One criteria would be to find platforms that minimize delay between themselves and platform A.

- **Primitives**: A shadow net would use the Read (R), Originate (O) and ReRoute (RR) primitives. R would be optionally used by platform A in case it wants to replicate only a subset of traffic, while platforms D and E are likely to use it during the course of doing troubleshooting or new functionality deployment. RR is needed by A in order to send the traffic from itself onto the shadow net. Finally, O is used by platform A to actually replicate packets.

- **Invariants:** At a high level, this scenario would require that the Originate (O) and Modify (M) primitives do not take place on the path between platform A and platform D. Any such changes would mean that the traffic arriving at the shadow network would be different from that on the operational network, perhaps rendering the processing on platforms D and E ineffectual. These restrictions could probably be relaxed based on the specific use of the shadow net (e.g., if the troubleshooting involved looking at the IP-level 5-tuple, we wouldn't mind payload modification). In practice, shadow nets are likely to exist entirely within an operator's domain, and so the operator would know what sort of modification would exist (if any) between platforms A and D, and whether these would affect the effectiveness of the shadow net.

- **Redirection:** Since this scenario will likely exist entirely within an operator domain, there would be a number of ways to get the traffic to arrive at platform A (if it isn't already), including OSPF, MPLS tunnels, or even Openflow. Redirecting traffic from platform A to platform D could be done with a simple IP-in-IP tunnel.

- **Security:** Shadow networks operate on a copy of operational traffic and do not forward this copy outside of their own network. As a result, they do not introduce additional security problems. The CHANGE architecture just needs to ensure that platform A is allowed to originate and redirect packets to platform D. The only complication comes from the fact that normally, in the CHANGE architecture, destinations decide who's allowed to perform actions on traffic. In the shadow net scenario, this would be impractical, since platform A would have to get permission from a potentially large number of

destinations. In this case, we can override this mechanism and allow the Originate/ReRoute actions as a matter of policy.

- **Scalability:** Assuming that platform A is able to originate and redirect all the necessary traffic, the possible bottleneck in this scenario is the processing that platform D, for instance, would need to apply. To scale this out, it would be possible for platform A to load balance the traffic to several platforms within the shadow net; this would require additional tunnels and a processing module in the platform to do the actual load balancing. For instance, if using ClickOS, this would consist of a ClickOS virtual machine with several virtual interfaces, the output of each going to a different tunnel. On the processing end (e.g., platform D), it would be possible to split the incoming flows in an RSS-like fashion onto multiple virtual machines to do the actual troubleshooting or new feature deployment.

2.4 Content Distribution Networks

Today, a large fraction of Internet traffic is originated by Content Delivery Networks (CDNs). Furthermore, recent studies [8, 11, 16] show that a large fraction of Internet traffic is originated by a small number of Content Delivery Networks (CDNs). Major CDNs are popular media sites like YouTube and Netflix, One-Click Hosters (OCHs), e.g., Rapid- Share, commercial CDNs such as Akamai and Limelight, and content providers, e.g., Google, Yahoo!, and Microsoft. Gerber and Doverspike [8] report that a few CDNs account for more than half the traffic of a US-based tier-1 carrier. Poese et al. [16] report a similar observation from the traffic of a European tier-1 carrier. Labovitz et al. [11] infer that more than 10% of the total Internet inter-domain traffic originates from Google, and Akamai claims to deliver more than 20% of the total Internet Web traffic [15]. In North America, Netflix is responsible for around 30% of the traffic during peak hours [9] and utilizes multiple CDNs.

In each CDN there are three main components: (1) **Servers** that act as middleboxes for content delivery, (2) a **User Redirection** mechanism that is responsible to redirect the requests of users to an appropriate server, and (3) a **Content Replication** mechanism that replicate content in servers based on the demand for a given object.

Servers can be either front-end or back-end servers, or transparent proxies. The deployment of servers varies among CDNs. Some CDNs rely on large datacenters, while other deploy caches in strategic location in the Internet or deep inside eyeball networks. The user redirection can be (a) DNS-based, that is used by most of the commercial CDNs. the DNS request is forwarded to the DNS authoritative server and the user is assigned to a server based on its location, server availability, cost of delivery etc., (b) HTTP-based, that is used mainly by content producers that run their own servers, but it is quite slow and does not have high penetration; when a request reaches a server, if the server is busy it forwards the request to another server – also known as application-level anycast, and (c) BGP-based, that is used partially by some large CDNs where the ISP advertises the subnets along with weights to the locations of the servers via BGP feed. The users with lowest

weight are served by the server in the location where the lowest weight is announced, if the server is full is assigned by a server in another location. With regards to content replication, it is currently followed the pull-model, where a request is assigned to the appropriate server that should host the content, if the content is not available or the content lifetime has expired, then the servers asks neighboring servers to retrieve the content, if again the content is not available it fetches the content from a server higher in the server hierarchy or from the original server.

To cope with the increasing demand for content, large CDNs deploy massively distributed server infrastructures [12] to replicate content and make it accessible from different locations in the Internet [20, 4]. For example, Akamai operates more than 130 000 servers in more than 1 800 locations across nearly 1 000 networks [12, 15]. Google is reported to operate tens of data-centers and front-end server clusters worldwide [10, 19]. Google has also start deploying caches, under the program Google Global Cache (GGC) [2], inside the eyeball networks. Similar strategy has also been followed by Netflix that introduced the Open Connect program [1]. Microsoft has deployed its CDN infrastructure in 24 locations and Amazon maintains at least 5 large data-centers and caches in at least 21 locations. Limelight operates thousands of servers in more than 22 delivery centers and connects directly to more than 900 networks worldwide.

Implementing CDNs on top of CHANGE platforms is appealing because of many reasons. If many ISPs deploy CHANGE platforms, it becomes possible to reduce the RTT of the users connections by selecting nearby platforms. Second, since CHANGE platforms are a shared infrastructure the cost of using it will be far less than today's dedicated servers.

- **Platform Selection** must take into account user location. Geo-location will be used to select platforms nearby the client (in fact, close to its DNS resolver). Traffic Attraction will be implemented with DNS: D's DNS name will be resolved to the IP address of CHANGE platform closest to the client (e.g. A's IP address).

- **Primitives**: Once traffic reaches a CHANGE platform it will be directed to a processing module running a commodity OS and an instance of the CDN software. If the CDN has the content it will just reply to the client, otherwise the client request will be redirected to another server that could have the content. Redirection will happen in the CDN software, so there is no need for CHANGE to be involved in this step.

- **Security:** It is easy to see that the CHANGE security rules allow instantiating CDNs: the end-user connects to the CHANGE platform, thereby implicitly allowing any return traffic. The CDN software may also connect to other CHANGE platforms or origin servers to fetch content: this will be allowed explicitly by the platforms or the origin servers when the CDN is setup.

- **Scalability:** is mostly concerned with the ability to scale up and down the resources allocated to any individual platform (e.g. A), which is natively provided by the CHANGE platform. If the capabilities

of the whole platform are exceeded than traffic could be split between multiple nearby platforms (e.g. A and B) via DNS load balancing.

2.5 Content-Aware Traffic Engineering

The growth of demand for content and the resulting deployment of content delivery infrastructures pose new challenges to CDNs and to Internet Service Providers (ISPs). For CDNs, the cost of deploying and maintaining such a massive infrastructure has significantly increased during the last few years [17] and the price charged for delivering traffic to end-users has decreased due to the intense competition. CDNs also struggle to engineer and manage their infrastructures, replicate content based on end-user demand, and assign end-users to appropriate servers. The latter is challenging due to the mis-location of end-users [13, 3]. Furthermore, inferring the network conditions within an ISP without direct information from the network is difficult [15]. Moreover, due to highly distributed server deployment and adaptive end-user to server assignment, the traffic injected by CDNs is volatile. For example, if one of its server locations is overloaded, a CDN will re-assign end-users to other server locations, resulting in large traffic shifts in the ISP network within minutes.
We argue that the challenges that both CDNs and ISPs face separately can be turned into an opportunity if CDNs and ISPs are able to collaborate on the end-user to server assignment:

(i) on **small time scales** (minutes or even seconds)

(ii) by **fully utilizing the CDN server and network path diversity** that is currently available,

(iii) with **minimal overhead** in the current operation of CDNs and ISPs.

Today, no traffic engineering system supports all the above three requirements. Routing-based traffic engineering adjusts routing weights to adapt to traffic matrix changes. To avoid micro-loops during routing convergence, it is common practice to only adjust a small number of routing weights [6]. To limit the number of changes in routing weights, routing-based traffic engineering relies on traffic matrices computed over long time periods and offline estimation of the routing weights. Therefore, routing-based traffic engineering operates on time scales of hours, which can be too slow to react to rapid change of traffic demands. Only recently, link-weight tweaking [7] approaches were introduced to safely adapt weights on smaller time scales. The increasing challenges of cost reduction and end-user experience improvement that both CDNs and ISPs are confronted with, motivate us to propose a new tool in the traffic engineering landscape. We introduce Content-aware Traffic Engineering (CaTE). CaTE leverages the server location diversity offered by CDNs and, through this, enables adaptation to traffic demand shifts. In fact, CaTE relies on the observation that by selecting an appropriate server among those available to deliver the content, the path of the traffic in the network can be influenced in a desired way. Recent measurement studies [16] show that there is significant server and path diversity that is at large unexplored. Figure 2.4 illustrates the basic concept of CaTE. The content requested by the client is in principle available from three CDN servers (A, B, and C). However, the

client only connects to one of the network locations. Today, the decision of where the client connects to is solely done by the CDN and is partially based on measurements and/or inference of network information and end-user location. With CaTE the decision on end-user to server assignment can be enhanced by utilizing ISP recommendations.

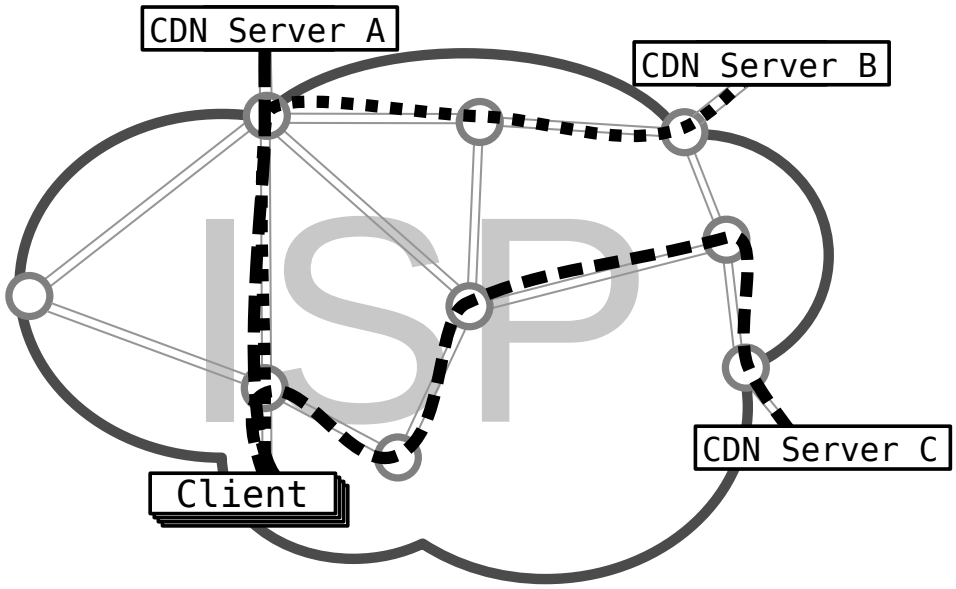

Figure 2.4: By choosing a CDN server for a client with the help of CaTE, traffic engineering goals and accurate end-user to server assignment become possible.

Formally, let **y** be the vector of traffic counts on links and **x** the vector of traffic counts in origin-destination (OD) flows in the ISP network. Then **y**=A**x**, where A is the routing matrix. $A_{ij} = 1$ if the OD flow i traverses link j and 0 otherwise. Traffic engineering is the process of adjusting A, given the OD flows **x**, so as to influence the link traffic **y** in a desirable way. In CaTE, we revisit traffic engineering by focusing on traffic demands rather than routing changes:

Definition 1: Content-aware Traffic Engineering (CaTE) is the process of adjusting the traffic demand vector **x**, given a routing matrix A, so as to change the link traffic **y**.

Only traffic for which server location diversity exists can be adjusted. Therefore, **x**=$\mathbf{x}_r$+$\mathbf{x}_s$ where $\mathbf{x}_r$ and $\mathbf{x}_s$ denote the content demands that can and can not be adjusted (single server location case) respectively. The degree of freedom in adjusting traffic highly depends on the diversity of locations from which the content can be obtained. We can rewrite the relation between traffic counts on links and traffic counts in flows as follows: **y**=$A(\mathbf{x}_s + \mathbf{x}_r)$. CaTE adjusts the traffic on each link of the network by adjusting the content demands $\mathbf{x}_r$: $\mathbf{y}_r$=$A\mathbf{x}_r$ to satisfy traffic engineering goals.

In CaTE the redirection of end-users to servers takes place on small time scales. Typical timescales for operation can be in the order of minutes, but in principle, it is possible to operate on even smaller time scales if applications or network conditions require, e.g., to react to flash crowds. It is also possible to operate on the time scales of the TTL value of a DNS query that is typically tens of seconds in large CDNs [23] or even per request. Notice that CaTE can be applied to CDNs that are operated by the ISP or to any other CDN, and

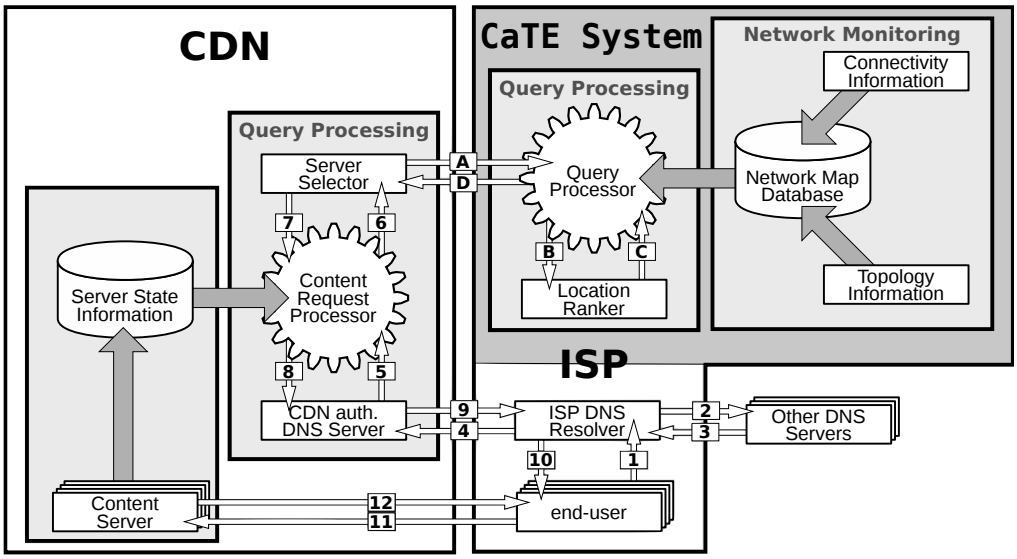

Figure 2.5: CaTE System architecture and flow of messages.

can apply to any type of redirection (DNS-, HTTP- or BGP-based).

Thanks to the online recommendations by ISP networks, CDNs gain the ability to better assign end-users to servers and better amortize the deployment and maintenance cost of their infrastructure. Network bottlenecks are also circumvented and thus the ISP operation is improved. Furthermore, the burden of measuring and inferring network topology, and the state of the network, both challenging problems, is removed from CDNs. Moreover, in [13, Sections 4 and 5] we show that the online CaTE decisions on the end-user to server assignment lead to optimal traffic assignment within the network under a number of different metrics. The advantage is that now the problem of assigning traffic to links reduces to a fractional solution (on the contrary, assigning routing weights to links is NP hard). In short, all involved parties, including the end-users, benefit from CaTE, creating a win-win situation for everyone.

The CaTE system is installed in an ISP and interacts with the existing CDN server selector. The main tasks of the CaTE system are to: (1) maintain an up-to-date annotated map of the ISP network and its properties, (2) produce preference rankings based on the paths between end-users and candidate servers, and (3) communicate with the CDN server selector to influence the assignment of end-user to servers. To this end, we propose an architecture that comprises a Network Monitoring component, a Query Processing component and a Communication Interface between an ISP and a CDN. For an overview of the architecture see Figure 2.5.

The network monitoring component gathers information about the topology and the state of the network from several sources to maintain an up-to-date view of the network. The network monitoring component consists of the following subcomponents: The Topology Information component gathers detailed information about the basic network topology, i.e., routers and links, as well as annotations such as link utilization, router load, and topological changes. An Interior Gateway Protocol (IGP) listener provides up-to-date link-state (i.e., IS-IS, OSPF) information. Information about routers and links is retrieved, thus, the network topology can be extracted. The nominal link delay, i.e., the latency on a link without queuing, can be found through the

link length and physical technology. The link utilization and other metrics can be retrieved via SNMP from the routers or an SNMP aggregator. The Connectivity Information component uses routing information to calculate the paths that traffic takes through the network. Finding the path of egress traffic can be done by using a Border Gateway Protocol (BGP) listener. Ingress points of traffic into the ISP network can be found by utilizing Netflow data. This allows for complete forward and reverse path mapping inside the ISP. Furthermore, the system can map customers as well as CDN infrastructures into the network map by finding the routers that announce the address space associated with them. In total, this allows for a complete path map between any two points in the ISP network. Finally, our system has access to an uplink database that provides information about the connectivity statistics of end-users. The Network Map Database component processes the information collected by the Topology and Connectivity Information components to build an annotated network map of the ISP network tailored towards fast lookup on path properties. It uses a layer of indirection to keep the more volatile information learned from BGP separate from the slower changing topological information. This allows address space to be quickly reassigned without any re-processing of routing or path information. It also enables pre-calculation of path properties for all paths that yields a constant database lookup complexity independent of path length and network architecture. If topology changes, e.g., IGP weights change or a link fails, the Topology Information component immediately updates the database which only recalculates the properties of the affected paths. Having ISP-centric information ready for fast access in a database ensures timely responses and high query throughput.

The Query Processing component receives a description of a request for content from the CDN, which specifies the end-user making the request and a list of candidate CDN servers. It then uses information from the Network Map Database and a selected ranking function to rank the candidate servers. This component consists of the following subcomponents: The Query Processor receives the query from the CDN. First, the query processor maps each source-destination (server to end user) pair to a path in the network. In most cases, the end-user is seen as the ISP DNS resolver, unless both ISP and CDN support the client IP eDNS extension [5]. Once the path is found, the properties of the path are retrieved. Next, the pairs are run individually through the location ranker subcomponent (see below) to get a preference value. Finally, the list is sorted by preference values, the values are stripped from the list, and it is sent back to the CDN. The Location Ranker component computes the preference value for individual source-destination pairs based on the source-destination path properties and an appropriate function. Which function to use depends on (a) CDN deployment and load, (b) CDN preferences and (c) the optimization goal of the ISP. The preference value for each source-destination pair is then handed back to the Query Processor. Multiple such optimization functions being defined upon the collaboration agreement between a CDN and an ISP, and subsequently selected individually in each ranking request. For example, a function might be the minimization of end-user and server delay. In Section 4 we evaluate CaTE with multiple ranking functions for different optimization goals.

When a CDN receives a content request, the Server Selector needs to choose a content server to fulfill this

request. We propose that the server selector sends the list of eligible content servers along with the source of the query and an optimization goal to the ISPs CaTE system to obtain additional guidance about the underlying network. If the guidance is at granularity of a single DNS request, we propose a DNS-like protocol using UDP to prevent extra overhead for connection management. If the granularity is at a coarser level, i.e. seconds or even minutes, we rely on TCP. Any other communication based on BGP (e.g., in the case of Netflix Open Connect is also possible).

Deploying CaTE using CHANGE will just use the general-purpose computing resources provided by the platform, as well its native ability to scale to different loads.

- **Platform Selection** will be performed manually by the ISP.
- **Primitives**: CaTE does not use any inter-platform primitives offered by CHANGE . All the app needs is to receive the traffic from the CDN, and to reply to that traffic; this is trivially supported by destination based forwarding.
- **Trust and security:** CHANGE does not natively support the integration with the routing system since this is not a commonplace requirement for most considered applications while requiring a great deal of trust in the application. CaTE is a special case where the ISP is the one running the functionality so the problem of trust disappears. The same considerations apply for security too.
- **Scalability:** to cope with higher request volumes CaTE can use the platform's scaling abilities. Once the capabilities of a platform are exceeded the ISP needs to replicate the CaTE software and load balance the CDN requests to the two platforms. This could be implemented by performing load-balancing in a CHANGE platform, as in our Inbound Traffic Engineering scenario.

2.6 Distributed Intrusion Detection Systems

In this section, we discuss the CHANGE architecture and platforms in the context of a single type of application—namely an intrusion detections system (IDS) that monitors network activities for malicious activities or policy violations and produces reports for the system administrator. Deploying IDS in the network involves complex processing including decoding packet data, aggregating distinct packets into streams and inspecting them according to a specific rule set.

In the context of CHANGE architecture, and in order to properly deploy DIDS systems, the CHANGE platforms should be able to communicate between them, and select the right flow to process if there is enough resource in the current platform.

We propose a distributed implementation where every CHANGE platform makes the decision whether it should pick-up and process a flow or not based only on locally available information, and it also dynamically reacts when a flow turns out to be too heavy (i.e. overloads the resources processing the flows).

We first introduce the basic Stateless approach, then extend it to use additional information about the network.

2.6.1 Stateless flow selection

In this basic approach we assume that the platforms do not maintain any additional information about the state of the network and of other platforms than what they would normally. However, they have to keep track of the flows they are processing, for this purpose we assume that they maintain a flow-table that identifies the flows they have decided to process, while also keep track of other ongoing flows that might be processed by another downstream node. This is important, in order to make sure that platforms only pick up new flows and not the ones that have been present for a while and might already be processed by another node downstream. However, this later constraint might be relieved in order to avoid the case where flows are crossing the network without being processed even though there is enough capacity for it.

This can happen if there was not enough capacity on the flows path to process it when it arrived and none of the nodes have picked it up, but shortly after, some of the resources got freed. There is another factor that has to be considered before letting platforms pick up flows that have been present for some time. That is, if the functionality(s) applied to the flows maintain states about them, this state, depending on the functionality, might have to be moved to the new processing node, which requires some coordination between nodes. Furthermore, we also have to make sure that a flow is not picked up for processing by more than a single node, thus, we assume that the CHANGE platform are tagging the packets using a single bit in the packet header, for example, either in the DiffServ (a.k.a. ToS) or the IP-id field. Hence, a platform only considers flows that are untagged for processing.

The decision whether a newly arrived flow should be picked up for processing is solely based on the available processing capacity at the platform. That is, when a flow arrives it is picked up by the platform for processing unless the platform is overloaded or is very near to its saturation point. The research community has proposed several solutions to predict the weight of arriving flows [18, 14] , which might be used to make a more realistic decision about whether there is enough resource to process the flow at the given node or not. Hence, we assume that we have an approximated weight of the flows upon their arrival.

However, in the case a flow turns out to be too heavy and pushes the platform above its capacity, the latter might decide to stop processing the flow and also stops tagging it, so that one of the next-hop nodes on the flow's path can pick it up instantly.

Moving the processing of a flow from one platform to the other might pose several problems. If the state pertaining to the flows is lost there may be a brief period of time where a flow will not be subject to IDS processing, opening a window for potential vulnerability. Here we can use the state migration support offered by CHANGE to avoid possible attacks.

2.6.2 Path-Aware flow selection

The Stateless approach has the advantage that it does not need any regular update about the state of the network or the other platform in the network, but this also limits the extent to which it is able to make good decisions about which flows it should pick in order to get the highest possible number of flows processed

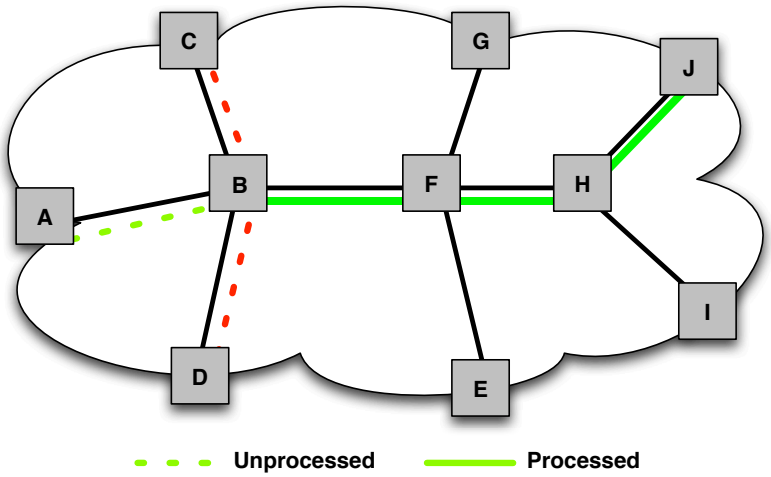

Figure 2.6: Shortcomings of the Stateless protocol

in the whole network. For example, most platforms in a network are part of multiple ingress-egress paths. Therefore, flows not picked up by a given platform have a higher chance of being picked up by a downstream platform. The Stateless approach described above unfortunately ignores this fact, and simply picks up flows based on order of arrival. This could lead to some flows leaving the network without having been processed, while there is spare capacity in other parts of the network where some other contenting flows could have been processed.

Figure 2.6 illustrates this shortcoming of the Stateless protocol through a simple example with two flows, assuming that the green flow arrived just before the red one did, and that both ingress and egress platforms for the red flow are saturated and cannot accommodate new flows for processing.

The information required to support such "path-aware flow selection" is generally available in the routers' routing information base (RIB) and we propose the following extension to the flow selection algorithm: we define a threshold value T and if the utilization of the flow processing resources at a platform has reached this threshold, the platform does not pick up all the flows anymore, but instead, taking into consideration the number of processing nodes left on the flow's path, it chooses a flow with a probability inversely proportional to the length of the downstream path (including the current platform itself). For example, if there are four other platform left on the path the flow will be picked up with a probability of 20%, while if there is only one other platform left this probability is going to be 50%, and if there is no other platform left, the current platform will pick the flow up, unless of course its flow processing resources are already overloaded. The intuition behind this approach is, that when the resources at a node become scarce, it is going to prefer flows that have less chance to be processed by another before they exit the network (i.e. cross fewer platform) over flows the encounter more nodes downstream. Figure 2.7 illustrates the expected behavior of the Path-Aware flow selection protocol.

The following pseudocode briefly summarizes the protocol:

if $Utilization \geq Threshold$ **then**

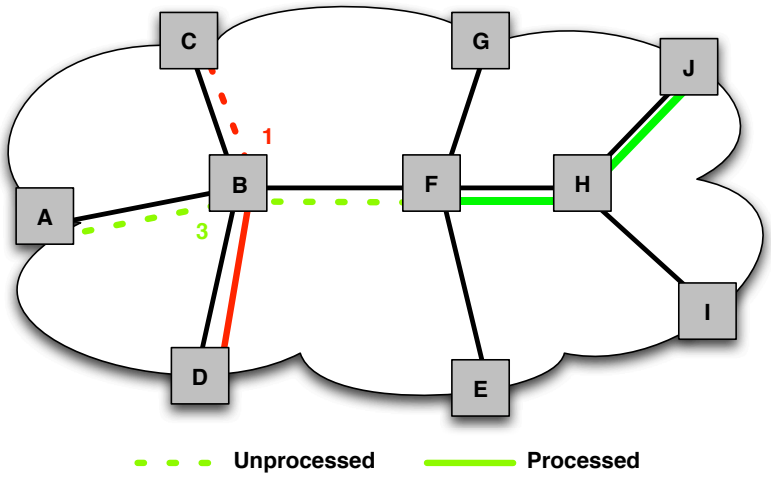

Figure 2.7: The Path-Aware protocol gives priority to the flow with fewer platforms left on its path

$P \leftarrow \frac{1}{PlatformLeft+1}$
if $random.uniform(P)$ **then**
ProcessFlow()
end if
end if

2.6.3 Capacity-Aware flow selection

Even though the Path-Aware approach should pick flows in a more efficient way due to its more global knowledge compared to that of the Stateless approach, it suffers from the fact that remaining path length is a very crude estimator of available processing capacity. Indeed, because platforms are part of multiple ingress-egress paths, load on a platform simply cannot be estimated remotely.

For example, it can easily be that there are three saturated nodes left downstream of one path while the single node left on another path is hardly utilized at all. See Figure 2.8 for an example scenario, where the orange bars represent the utilization of the resources at every node.

To address this problem, we propose a third approach, which we call the Capacity-Aware approach, that uses control packets to gather information about the available capacity on each path. Instead of collecting the available capacity per platform, which would require lots of signaling, we only collect the aggregate of the available capacity. This is carried out in two stages in the following manner: Every ingress platform periodically sends a COLLECT packet down on every path to every egress platform; the packet payload carries an `available capacity`, initialized to zero. As these control packets follow the path to their destined egress platform, the platforms they encounter on they way add their available capacity to the capacity recorded in the packet. At the same time, the platforms also record the value they have added to the packet. When such a control packet arrives at an egress platform, the packet is sent back to the ingress platform, which turns the packet into an INFORM packet (e.g. changes a bit in the packet header) and sends it down the same path again. This time when the packet arrives at a platform, the latter subtracts the value it had added to

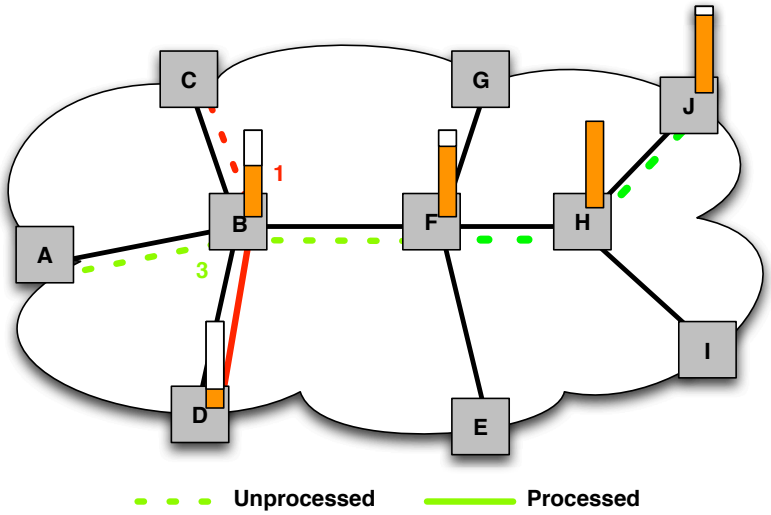

Figure 2.8: Shortcomings of the Path-Aware protocol

the COLLECT packet previously from the value stored in the INFORM packet and also stores the difference, which represents the total available capacity left on the path between the given platform and the egress one. The accuracy of this method largely depends on the frequency of the updates and thus on the number of COLLECT/INFORM packets on a single path. In order to increase the accuracy of the method, a path could have multiple COLLECT/INFORM packets circling on it, starting each with some delay after the other at the ingress platform. In this case, every packet has to have a unique identifier and every platform has to store multiple values, one for every COLLECT/INFORM packet. We believe that the overhead of forwarding these packets in the network is negligible, however, calculating the available capacity of a platform very often might induce perceivable overhead. Hence, we recommend that one should carefully decide about the number of these packets being active at a time. One should consider the number of metrics used to describe the resource utilization and the cost of calculating the capacity left on the platform.

When a new flow arrives at a platform, it calculates the average available capacity on the path downstream, using the previously collected information about the capacities and the knowledge about the number of platforms left on the path (i.e. the total available capacity on the downstream path is divided by the number of platforms left on the path). If this value is larger than its own available processing capacity then it lets the flow carry on without processing it, otherwise it picks the flow up and processes it. The following pseudocode briefly summarizes the protocol:

if $Utilization \geq Threshold$ **then**
  $C_L = \frac{TotalCapacityLeft}{PlatformLeft}$
  if $C_{MINE} > C_L$ **then**
    ProcessFlow()
  end if
end if

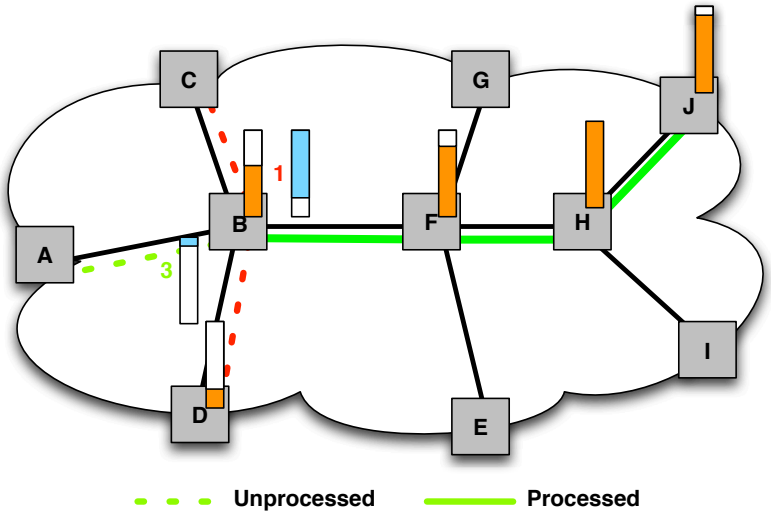

Figure 2.9: The Capacity-Aware protocol gives priority to the flow with less capacity left on its path

Figure 2.9 demonstrates how this mechanism is expected to overcome the shortcomings of the Path-Aware protocol unaware of the capacity left on the path. In this figure the blue bars at the platform represent the volume of capacity left on the path (collected previously through the protocol described above).

Despite the simplicity of collecting only the aggregate capacity and calculating the average left-over capacity on the path, the method provides sufficient information for the platforms to decide whether there is any platform left on the path that has more processing capacity left than themselves (i.e. if the average available capacity downstream is more than what the given platform has, then there is at least one platform downstream that has more free capacity and we should let that node pick the flow up).

The more often the capacity measurement part of the protocol is run, the better the capacity-aware protocol will load balance flow processing between the platforms.

2.6.4 Implementation

There are two ways of implementing the Distributed IDS using CHANGE . The first option is to deploy the IDS within one ISP's network, which is the typical setup used today. In this context CHANGE can help provision the resources for processing and solve scaling issues by dynamically sizing platform resources as well as load balancing (i.e. rerouting) flows through multiple platforms based on load. In this context traffic is not rerouted; all that's needed is a way to make traffic go through CHANGE platforms, and MPLS or OpenFlow could be used for this purpose.

The algorithms we have described can be easily implemented as ClickOS configurations and deployed using the CHANGE configuration files. The signaling required between the platforms in the Path-Aware and Capacity-Aware protocols will be implemented using the Signaling architecture described in Deliverable 4.3.

An alternative and perhaps more interesting setup is to deploy the distributed IDS system across multiple administrative domains. In this context the destinations can setup many platforms attracting traffic from nearby customers, similar to the Inbound Traffic Engineering application. After attraction, traffic needs to be

explicitly ReDirected to the next platform towards the destination. We briefly discuss the requirements of the distributed setup below:

- **Platform Selection** must take into account user location; ideally the CHANGE platforms should be as close to the original end-to-end path as possible, or they should improve the performance compared to the original path. Geo-location could be used to select ingress platform nearby the client.
- **Primitives**: Once traffic reaches a CHANGE platform it will be processed locally according to the algorithms described above, and the ReDirected towards the next platform en-route to the destination. Note that, in effect, the user is implementing some sort of "overlay" routing to steer traffic through the different IDS platforms. In this context, the algorithms we have described can be enhanced to take advantage of the added degree of liberty introduced by flexible routing.
- **Redirection:** Since the algorithms need to setup certain bits in the IP header (e.g. diffserv), we need a tunneling solution to ensure these bits travel unchanged to the destination. In this context it is easiest to implement redirection with Loose Source Routing.
- **Invariants:** Traffic traveling between CHANGE platforms processing the same flow must be immutable, so Modify and Originate invariants will be specified for the inter-platform hops.
- **Security:** The redirections are allowed explicitly by the participating platforms and by the destinations (which request processing in the first place).
- **Scalability:** overlay routing (e.g. redirection) can be used to load balance traffic across the different platforms.

3 Conclusions

In this document we have discussed specific requirements of a number of interesting applications in relation to the CHANGE architecture, finding that all the selected applications will benefit from using CHANGE , albeit with different levels of support.

The document provides a solid basis for the further development, deployment and testing of CHANGE applications. It also provides feedback to the platform and architecture work, validating the choices made in the project so far.

According to our analysis, traffic redirection is by far the most used primitive in the CHANGE architecture. Invariants are useful especially for security applications such as firewalls and intrusion detection systems. Scalability is an issue for all applications; CHANGE helps with its scalable platform design and the ability to load balance traffic to multiple platforms via DNS - discussed next.

A common requirement across many applications is the ability to perform traffic attraction to one of multiple platforms based on the end-user's location. This functionality is not supported natively by CHANGE , but can and will be implemented easily as an add-on service. Dynamic DNS redirection can be implemented as an always-on processing in (a subset of) CHANGE platforms using the same IP address. BGP anycast will load-balance client requests to the nearest platform. Each platform needs to be configured with the address of the authoritative platform for every resolved domain name. The authoritative platform will be configured with the list of eligible platforms for that domain. When a DNS request arrives, the list of platforms will be retrieved from the authoritative platform and the results cached locally for future queries. Next, a geo-location database will be use to map both the user's and the platforms' IP addresses to geographical locations. Finally, geographical distance will be used to rank platforms according to their distance to the user, and the nearest platform will be returned.

CHANGE support is lacking for certain features of content-aware traffic engineering, including the ability to tap into the routing information base of the ISP and the routing updates. Giving access to such information requires a great deal of trust on behalf of the ISP and only works if the application is run by the ISP or close collaborators. In CHANGE we target wide-area flow processing for the masses, in this context the required trust does not exist.

In conclusion, the architectural and implementation choices we have made in this project are appropriate to properly support the selected applications. In ongoing and future work we will implement a subset of these applications using CHANGE and test them in our wide-area test-bed.

Bibliography

[1] Announcing the Netflix Open Connect Network. http://blog.netflix.com/2012/06/announcing-netflix-open-connect-network.html.

[2] GoogleCache. http://ggcadmin.google.com/ggc.

[3] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS Resolvers in the Wild. In *ACM IMC*, 2010.

[4] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Web Content Cartography. In *ACM IMC*, 2011.

[5] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden. Client IP Information in DNS Requests. IETF draft, work in progress, draft-vandergaast-edns-suspend-ip-01.txt, 2011.

[6] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE J. Sel. Areas in Commun.*, 2002.

[7] P. Francois, M. Shand, and O. Bonaventure. Disruption Free Topology Reconfiguration in OSPF Networks. In *INFOCOM*, 2007.

[8] A. Gerber and R. Doverspike. Traffic Types and Growth in Backbone Networks. In *OFC/NFOEC*, 2011.

[9] Sandvine Inc. Global broadband phenomena, Feb 2011.

[10] R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving Beyond End-to-end Path Information to Optimize CDN Performance. In *ACM IMC*, 2009.

[11] C. Labovitz, S. Lekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet Inter-Domain Traffic. In *ACM SIGCOMM*, 2010.

[12] T. Leighton. Improving Performance on the Internet. *CACM*, 2009.

[13] Z. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *Usenix ATC*, 2002.

[14] Mikkel Thorup Nick Duffield, Carsten Lund. Properties and Prediction of Flow Statistics from Sampled Packet Stream. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, Marseille, France, November 2002.

[15] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network. *SIGOPS Rev.*, 44, 2010.

[16] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann. Improving Content Delivery using Provider-Aided Distance Information. In *ACM IMC*, 2010.

[17] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the Electric Bill for Internet-scale Systems. In *ACM SIGCOMM*, 2009.

[18] Augustin Soule, Kav Salamatian, Nina Taft, Richard Emilion, and Konstantina Papagiannaki. Flow classification by histograms: or how to go on safari in the internet. In *In SIGMETRICS'04/Performance'04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 49–60, 2004.

[19] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. Answering What-if Deployment and Configuration Questions with Wise. In *ACM SIGCOMM*, 2009.

[20] S. Triukose, Z. Al-Qudah, and M. Rabinovich. Content Delivery Networks: Protection or Threat? In *ESORICS*, 2009.