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

D2.2 CHANGE Architecture Specification: Early Draft

Due date of deliverable: July 1, 2011

Actual submission date: December 9, 2011

Start date of project	1 October 2010
Duration	36 months
Lead contractor for this deliverable	University Politehnica of Bucharest (PUB)
Version	Revised, December 9, 2011
Confidentiality status	Public

Abstract

The Internet was built on the end-to-end principle, whereby the network limits itself to forwarding packets, and any higher-layer processing and applications reside on end hosts. This model has the great virtue of enabling new applications to be deployed with only the hosts needing to know about them.

However, the introduction of middleboxes (e.g., NATs, firewalls, etc) in the network has meant that the original Internet architecture has not described the real Internet for around fifteen years now, with studies showing that at least over a third of paths perform some form of L4+ functionality.

It therefore seems futile to attempt to fight this trend; it is too entrenched at this point and there are just too many reasons why operators choose to deploy these middleboxes in their networks. The only reasonable strategy seems to embrace middleboxes, and to provide a unifying architecture in which they have roles that we can reason about and functionality that we can communicate with from the end systems. To help realise this vision, we present the CHANGE architecture.

This document discusses the considerations and primitives behind the CHANGE architecture. The CHANGE architecture is based around the notion of flow processing platforms deployed throughout the network that can perform not only the type of processing done by current middleboxes, but those that might be envisioned in the future. We discuss these platforms in detail, including their requirements and the set of basic processing primitives that flow processing may be constructed from. We describe the platform discovery mechanisms required, and an authorization framework for dictating who can request processing. Finally, we describe how a number of motivating scenarios may be realised with the CHANGE architecture and provide a review of the current state of the art.

Target Audience

Technical experts, researchers, students in the field of networking

Impressum

Full project title	CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions
WP2 Scenarios, Requirements and Architecture	D2.2 CHANGE Architecture Specification: Early Draft
Editor	Costin Raiciu, PUB
Project Co-ordinator	Adam Kapovits, Eurescom
Technical Manager	Felipe Huici, NEC
Copyright notice	© 2011 Participants in project CHANGE

Executive Summary

The great virtue of the Internet architecture has been to enable the deployment of an astonishing range of applications and services; based on a relatively simple organisation and a set of common abstractions. However, research shows that the "traditional" Internet architecture model is disappearing. As discussed in Section 1, this is because the original end-to-end transparent Internet is being replaced by one where there are segments of transparent IP connectivity interconnected by nodes that perform L4+ processing on flows.

In short application logic - in the form of middleboxes, such as NATs, firewalls, DPIs and a multitude of other applications now lives in the middle of the network at the behest of network operators. There are two broad consequences to this situation. First and foremost, the lack of transparency makes it difficult to reason about the emergent behaviour of traffic traversing the Internet. The complexity and uncertainty that results from this hinders the deployment of new applications and services; already existing applications such Skype have to depend on a myriad of techniques just to provide an end-to-end service. Unchecked, this situation threatens to curtail the ease of extending and evolving network functionality since network providers will be unwilling to deploy untested mechanisms on operational networks. Second, the trend is already entrenched; for all the opaqueness middleboxes induce, they also provide functionality that makes network operators deem them as indispensable.

This report describes our initial attempt at bringing some order to current state of affairs, by providing a reasoned approach to incorporate middleboxes and their functionality into general Internet architecture. We introduce a novel architecture based around the notion of *flow processing platforms*. We define flow processing to be the type of processing done currently by middleboxes and those that might be envisioned in the future.

We start by presenting the architecture requirements on a flow processing platform, listing what would be needed to realise a coherent flow processing architecture for today's Internet. We also include motivating scenarios to test both the applicability and flexibility of our approach. Based on the requirements listed, we define the foundations of a flow processing platform; starting from the basic definition of a flow, its labelling and identification, while highlighting the advantages, limitations and practical considerations of our proposal.

To present an attractive alternative to the current state of affairs, a new flow processing architecture must not raise a new set of vulnerabilities that will require in ad-hoc patching in the future. For this reason, we describe the security considerations associated with flow processing, including flow ownership, authentication and instantiation. We show the weaknesses induced by each, and discuss how they may be tackled. Further, we decompose the actual flow processing operation to a set of widely applicable primitives, showing both how they may be used to define instances of processing and the security considerations associated with them.

Because flow processing platforms are most likely to be owned and operated by multiple organisations, we discuss the issues related to auditing, billing, platform discovery and flow attraction. In each case, we present our proposals, again highlighting their advantages, limitations and practical considerations.

Finally, we map our architecture definition to the motivating scenarios, and show how they may be realised based on the architecture primitives presented.

List of Authors

Authors	Mohamed Ahmed, Mehdi Bezahaf, Olivier Bonaventure, Gregory Detal, Mark Handley, Felipe Huici, Laurent Mathy, Costin Raiciu, Octavian Rinciog
Participants	NEC, UCL-BE, UCL-UK, ULANC, PUB
Work-package	WP2. Scenarios, Requirements and Architecture
Security	PUBLIC (PU)
Nature	R
Version	1.0
Total number of pages	50

Contents

Executive Summary 4

List of Authors 6

List of Figures 9

List of Tables 10

1 Introduction 11

2 Related Work 13

2.1 History of network architecture . . . 13

2.1.1 Circuit and Flow Switching . . . 13

2.1.2 Programmable Networks . . . 14

2.1.3 Service-oriented Architectures . . . 15

2.2 Survey of Recent Research . . . 17

3 Change Architecture Overview 20

3.1 Architecture Requirements . . . 22

3.2 Architecture Overview . . . 23

3.3 Motivating Scenarios . . . 25

4 CHANGE Architecture Building Blocks 26

4.1 Flow Naming and Identification . . . 26

4.2 Flow Ownership . . . 28

4.2.1 Once You Forward the Traffic, It's Yours . . . 29

4.2.2 Authenticating Flow Ownership . . . 30

4.3 Basic Flow Processing Primitives . . . 31

4.3.1 Flow Processing Principles . . . 32

4.3.2 On-Path Platforms Should Not Be Allowed to Instantiate Upstream Processing . . . 35

4.4 Auditing and Billing . . . 36

4.5 Platform Discovery . . . 37

4.6 Drawing Traffic into a CHANGE Platform . . . 39

4.7 Remaining Open Questions . . . 41

5 Implementing the Motivating Scenarios 43

5.1 Deploying New Transport Protocols . . . 43

5.2 Inbound Traffic Engineering 44
5.3 DDos Filtering 45
5.4 Monitoring 45

6 Conclusions **46**

References **46**

List of Figures

3.1 CHANGE Internet Architecture 24

4.1 An end-to-end flow in the CHANGE architecture 29
4.2 Being On-Path Enables Nodes To Re-direct Traffic Arbitrarily 30
4.3 Rewriting the Destination Address Has Security Implications 33
4.4 Instantiating Tunnels Can Lead to Routing Cycles and Loss of Reachability 34
4.5 CHANGE Platform Discovery 37
4.6 CHANGE Traffic Attraction Mechanisms 40

5.1 Stub AS Perfoming Inbound Traffic Engineering with CHANGE 44

List of Tables

4.1 Processing Primitives and Authentication 32

1 Introduction

The basic architecture of the Internet has been embodied for thirty years in a set of protocols and design documents that indicate requirements for Internet hosts and routers. The architecture is a layered one, with diverse layer 2 networks interconnected by IP routers providing an end-to-end datagram service model at layer 3. Above this, at layer 4, sits TCP, providing reliable congestion-controlled in-order delivery of data to applications at the layers above. The concept is simple; routers look at the headers in IP packets, decrement the TTL and forward the packet on to the next hop if they are able, or drop the packet otherwise. TCP then functions end-to-end, with only the end-systems understanding its semantics.

The great virtue of this model is that it is possible to reason about the emergent behaviour of all the concatenated components, at least if they perform according to spec. In particular, new applications and transport protocols can be introduced, and only the end-hosts need to know about them. It is this great generality that has been the Internet's main advantage, allowing a wide range of new applications to be deployed that were unforeseeable when TCP and IP were being designed. It can be argued that the Internet doesn't do any single task terribly well, but it does everything well enough. And in economic terms, 'well enough' is what actually matters. The problem though is that the Internet doesn't really do everything well enough.

Unfortunately the original Internet architecture has not described the real Internet for around fifteen years now. Network Address Translators and firewalls were the first commonly deployed middleboxes that placed layer 4 (or higher) functionality in the middle of the network, not just at the endpoints. More recently it has become common to employ deep-packet inspection to look within packets and perform rate-limiting of "less important" traffic. Transparent web proxies and application accelerators improve performance by employing L4+ knowledge. Intrusion detection systems reassemble flows and snoop everything, as does lawful intercept equipment. And traffic normalizers try to remove ambiguous corner cases that might represent threats or attempts to bypass detection equipment.

There are two key problems that this plethora of enhancements to the original architecture bring. First, they embed the limitations of current protocols and applications within the network, making it very hard to deploy anything new without jumping through hoops to make it look to the network like existing traffic. Second, they make it extremely difficult to reason about what precisely will happen to a packet sent across the Internet, which makes the network fragile and hard to debug.

In a previous EU project, several of the Change partners attempted to standardize extensions to TCP to perform encryption and to support multipath operation. It became clear that no-one really knows what will happen to a TCP packet sent across the Internet if it strays from common practice in any way. In collaboration with Michio Honda from Keio University, we set out to probe the Internet to test its response to a wide variety of possible extended TCP syntax and semantics. Our study probed 142 access networks in 24 countries (14 of them in the EU) ranging from cellular providers to WiFi hotspots, residential DSL and cable to university networks, plus a number of corporate networks. By carefully crafting TCP segments between a client on

each network and our server, we can probe corner case behaviour. As anecdotal evidence indicated that many middleboxes do not treat all ports equally, we repeated the tests using port 80 (HTTP), port 443 (HTTPS) and a port with no special meaning. The results are somewhat eye-opening.

- New TCP options are removed from traffic on 6% of paths. However, on port 80 this rises to 14% of paths.
- TCP sequence numbers are rewritten on 10% of paths. On port 80, 18% of paths rewrite sequence numbers. Of these on port 80, two-thirds implement some form of TCP proxy which splits the end-to-end path into two or more distinct TCP path segments, and acknowledge data before it reaches the final end point. The remainder, and most on other ports, appear to rewrite sequence numbers just to improve initial sequence number randomization.
- 5% of paths will fail if holes are left in the TCP sequence space. On port 80, the number is 15%.
- If an ACK is sent for data that has not yet been sent, 25% of paths will either drop it or "correct" it. For traffic on port 80, this rises to 33% of paths.

Thus, over a third of the paths are keeping state and performing L4+ functionality. This does not include NAT or basic firewall functionality, which is in addition to the figures above but too ubiquitous to be noteworthy these days. Our study also cannot measure rate-limiting triggered by DPI, lawful intercept, or any form of L4+ processing at the server end of the connection on typical Internet paths. We expect that functionality such as intrusion detection systems and server load balancers are commonplace in modern datacenters, but cannot measure it with the methodology used in this study. The full results are available in[28].

It is clear then that the original end-to-end transparent Internet architecture is disappearing, to be replaced by one where there are segments of transparent IP connectivity interconnected by nodes that perform L4+ processing on flows. It is also clear that traffic receives significantly different processing depending on the transport port.

It seems futile to attempt to fight this trend - it is too entrenched at this point. IPv6 may help with NATs, but is unlikely to help with all the other reasons network operators install these middleboxes. The only reasonable strategy seems to be to attempt to embrace middleboxes and the concept of flow processing, but to then attempt to provide a unifying architecture in which they have roles that we can reason about and functionality we can communicate with from the end systems. In particular we would like to answer the question *is it possible to provide an architectural framework in which middleboxes could actually enhance our ability to deploy new applications?*.

This document presents a first draft of the architectural framework to be constructed in the Change project. It is at this stage very much a work-in-progress, and will continue to be revised in the light of further developments of both the platform and the applications that will use the platforms.

2 Related Work

Despite all research efforts done in networking, and the enormous range and popularity of the applications that are running over the Internet, the core architecture of the deployed Internet infrastructure has not changed greatly in the last 15 years.

Many application-level solutions have appeared to answer the deployment barrier, but these solutions are network and operator policy agnostic, will therefore tend to "hurt" the network if used in large-scale. As highlighted, one of our purposes is to enhance the Internet architecture in a bottom-up manner, adding flexibility and new processing primitives in a way that provides benefits to early adopters, while ultimately being guided by the much greater benefits to be had from the coordinated services.

So what is the current network architecture and how can we bypass existing issues to obtain more flexible solution? We begin this chapter by briefly highlighting history of network architecture, and then provide a brief survey of more recent trends in research, pointing out how CHANGE stands out.

2.1 History of network architecture

In this section we present a brief history of network architecture, discussing three broad architecture classes and their relationship to CHANGE , these are; circuit and flow switching, programmable networks, and service-oriented architectures.

2.1.1 Circuit and Flow Switching

The Internet architecture as we know it today has been based on both the datagram model and the end-to-end principle. The datagram model assumes that all packets are treated independently in the network while the end-to-end principle puts intelligence only in the end-systems. These two assumptions have been challenged during the last two decades.

The first approach to support Quality of Service in the Internet, Integrated Services (Intserv) [9], included layer-4 flows in the network and required QoS state in the routers. Unfortunately, using layer-4 flows made the integrated services architecture unscalable and it is still not deployed today in the global Internet. Differentiated Services (Diffserv) [8] appeared later as a more scalable alternative to support QoS. Compared to integrated services, Diffserv introduces less state in the network as it deals only with highly aggregated flows. Diffserv is the basis for several uses of QoS in today's IP networks. Multi-Protocol Label Switching (MPLS) [38] appeared initially as a solution to better integrate layer-2 techniques such as Asynchronous Transfer Mode (ATM) and FrameRelay into IP networks. However, it quickly evolved as a new layer-2.5 technique with its own header below IP. MPLS is widely used by ISPs to provide services such as VPNs, VoIP or IPTV. It serves as the basic infrastructure to allow an operator to support multiple services on top of a single network. Furthermore, GMPLS [7] allows optical networks to be integrated in the IP architecture in a coherent manner. MPLS also deals with flows and adds state to the routers. Compared to DiffServ and integrated services, MPLS is slightly more flexible in the definition of the flows that it supports since several

types of Forwarding Equivalence Classes can be defined. However, most deployments define a flow, either as all the packets that need to exit a network through an egress router, or as all the packets exchanged between an ingress and an egress router. From a QoS viewpoint, the main benefit of MPLS compared to Diffserv is the ability to forward flows along chosen paths to meet traffic engineering constraints.

In parallel with the evolution of support for QoS, we have more recently seen growth in the use of middleboxes in IP networks to provide services that were not considered when the architecture was designed. These include deep-packet-inspection, shaping, firewalls and denial-of-service protection and new services are created. These services need to be able to process particular flows at particular locations in the network. Compared to existing flow-based techniques, an important benefit of CHANGE is that it will support a flexible definition of the flows to be processed. A second benefit is that it will be possible to instantiate some processing, whatever its complexity, inside the network on flow processing platforms. These flow processing platforms will either be on the path followed by the flows to be processed or the network configuration will be changed to route flows via these platforms. More recently, the OpenFlow [35] concept has been gaining momentum. OpenFlow switches are programmable via a so-called flow table that can be configured by adding entries. These entries aggregate packets into flows by matching on a number of L2, L3 and L4 fields and then specify the switch port to which the flow should be sent. In this way, OpenFlow provides a basic L2-L4 flow classification mechanism. CHANGE will use OpenFlow switches as a fast classifier in its flow processing platforms, but by also including more programmable hardware we will provide more advanced processing capabilities.

2.1.2 Programmable Networks

Active and programmable networks [43, 44] were two packet processing models proposed in the mid-1990s. They shared with CHANGE the laudable aim of enabling more flexible in-network processing. The major differences between the three lies in the processing model and implementation approach used. In active networks, data packets carried code to be executed in routers, while in the programmable network model the code could be pre-loaded in the routers. In both these proposals, the implementation model was that this code was typically executed on either custom router hardware or on commodity PC hardware adjunct to the router. As both computational models used packets as the processing unit and hardware speed was severely limited compared to todays capabilities, performance was a critically limiting scalability factor. Also, because the approach was packet-centric, the deployment models often required that most, if not all, routers in the network (and sometimes the network stack in end-hosts) had to be upgraded to achieve desired benefits. As a result, neither active nor programmable networks gained wide acceptance and both failed to see any deployment.

In contrast, CHANGE proposes the use of flexible flow definitions as processing units. This means that only the traffic of interest is handled by the programmable flow processing platform. The only operation that takes place at the packet granularity is classification, where CHANGE exploits, and greatly benefits from, high-speed, cheap and commoditized hardware solutions (e.g., TCAMs in routers and OpenFlow switches). On the

processing front, CHANGE also benefits from high-performance multi-processor, multi-core systems. Implementation of the CHANGE processing model can be supported either by off-the-shelf high-performance custom components (e.g. network processors or tile processors) or even by current high-performance commodity server hardware (with their multi-core CPUs and high speed system interconnects). Finally, CHANGE assumes that flow processing platforms will be deployed only where needed, exploiting virtualization to improve sharing of the platform and isolation of processing.

The major advantage of such a deployment model is that all the assumptions and characteristics that made the existing Internet so successful are retained, while the notion of flows and processing are only actuated at processing points. For all these reasons, we therefore believe that CHANGE is poised to succeed where previous attempts at in-network processing failed, and see an incremental, but eventually wide, deployment. More recently, router programmability has seen a resurgence of interest.

Projects like Vrouter [18] or RouteBricks [2] exploit software router platforms to provide customized fast paths. Here, the unit of customization is the router itself so there is no attempt at providing processing differentiation on a per-flow or per-packet basis. When used in conjunction with virtualization techniques, these approaches allow the sharing of a common hardware box or cluster between several independent software routers, each running a custom network stack and belonging to a different virtual network. Super-Charging PlanetLab [41] has the same goal but uses network processors to implement the fast paths of the virtual routers. CHANGE builds upon this work, and will support flow-centric processing within the same hardware context as these programmable routers.

2.1.3 Service-oriented Architectures

Until recently, innovation in the Internet was mostly technological, fuelled by the ever increasing need for speed, growth and reach. This era saw rapid increase in network bandwidth through the advent of high-speed switch fabrics, optical networking and fast IP lookup, to name just a few. Access to the network was improved by "last mile" technologies such as ADSL or wireless access. As a result, innovation was mainly the realm of equipment manufacturers. However, changes to the architecture itself have met much resistance. For instance, in the mid 1990s there was a great deal of interest in IP Multicast as an enhancement to the basic Internet architecture that would support multi-point distribution. In practise, although almost all Internet routers now support multicast, it is rarely enabled inter-domain. In recent years with the rise of IPTV, multicast has once again become popular with consumer ISPs that wish to distribute video, but it tends to be limited to the edge ISPs. There are two main reasons for this. First, there is a chicken-and-egg problem: application writers cannot assume that multicast exists, so don't add multicast support; ISPs can't see any application demand, so don't enable multicast. Second, IP multicast is complex and difficult to manage, except in limited deployments. There are real costs for deploying multicast, even though the hardware supports it. The effect of this failure to deploy multicast has been that application writers have implemented application-level workarounds. For example, the BBC developed iPlayer, which performed

peer-to-peer distribution of high-quality video of all the BBC's programs. While the application worked well, the effect on British ISPs was predictable; they were overwhelmed with additional traffic taking paths which were suboptimal. In response ISPs started to implement deep-packet inspection and traffic shaping for iPlayer traffic.

It is just this sort of response that CHANGE hopes to avert. In-network flow processing platforms have the potential to serve as traffic fan-out points, but to do so in a manner that satisfies both the ISPs need to manage their network and the application's need to push the same content to many customers. It is unlikely that it is worthwhile deploying dedicated infrastructure for such a task (after all there is still a chicken-and-egg problem), but CHANGE allows this to be done in flexible flow-processing infrastructure installed for other purposes, and thus the up-front costs are small. In addition, CHANGE allows this fan-out to be done outside of the core IP forwarding path, avoiding the risk associated with deploying new services in critical infrastructure. The same arguments apply to other fundamental architectural changes: CHANGE lowers the costs and barriers to entry, while enabling deployment as layer-3 flow overlays atop the current IP network.

The transparency of the Internet has greatly encouraged innovation in the end-hosts and facilitated the deployment of successively more complex network-agnostic applications and services. This accounts for the second area of recent rapid innovations, namely the application layer (e.g. overlays, peer-to-peer, multimedia, content distribution networks, games, VoIP [27], IPTV, etc). The world-wide web has been particularly conducive to the introduction of many new applications (e.g. YouTube, Facebook, Google Maps, etc.) through the adoption of development frameworks such as Web2.0 or AJAX.

In fact, we have seen several trends at the application layer. First, applications are more content-rich than in the past. The importance of audio and video in the global Internet is growing beyond the intra-ISP deployments to support VoIP and IPTV. Content is often sourced from multiple servers that reside in the same data center or even in multiple data centers. These data centers are now a key part of the Internet and source more and more content. Data centers can belong to a single company (for example, Google), but there is a growing trend towards data centers such as Amazon EC2, Microsoft Azure and OVH that dynamically rent processing to external users. Their infrastructure is mainly composed of Ethernet switches and x86 servers. Most network specialists expect such data centers to serve an increasing fraction of the content in the future. Another trend is mashup applications, built by composing several applications that are hosted by different providers. This allows developers to enrich their application with information coming from several providers; applications built on top of Google Maps and Facebook are popular examples. CHANGE will allow operators and application developers to make better use of these data centers by pushing flow-processing capabilities within the data center itself. However, service innovation has often occurred at the application layer not by choice, but as a way to bypass the stumbling block to changes that the instrumental TCP/IP network layer represents. Of course, while introducing new concepts at the application layer simplifies deployment, the solutions are often far from ideal and can exhibit redundancy and sub-optimality resulting from competing objectives.

Cisco's Unified Computing [14] initiative aims to provide an enhanced service-oriented architecture for data centers. By combining switching, server hardware, storage and virtualization in the same device and adding a unified management framework, Cisco's unified computing aims to reduce costs and complexity for data centers, consolidating and integrating components that were traditionally dissociated. Further, it facilitates the provisioning and deployment of new services. Nevertheless, this integration preserves the existing networking model of servers as end-hosts. With CHANGE we go much further and push support for processing not only within the data centers, but also within the network by building on-path flow processing, realized as a composition of processing pipelines, which would otherwise be made up of several specialized boxes. In other words, CHANGE provides a novel way for building complex networks and thus enabling Internet innovation by enabling the integration and convergence of the components functionality.

A network flow-processing platform such as proposed by CHANGE therefore appears to be a good match to meet the varied demands for evolution of the Internet. Indeed, the flexibility of the CHANGE platform is poised to play a crucial role in answering calls for network customization for the future Internet (through the run-time composition of flow-processing modules) to green networking (through the use of virtualization and associated module migration for better power control). There is no doubt that the advent of future network applications will create yet unforeseen communication needs and requirements on the network. We believe the future Internet Architecture must be sufficiently agile to seamlessly accommodate such future requirements. By proposing a paradigm shift towards network processing as a software architecture, CHANGE will provide an agile and flexible base that will allow network providers to quickly meet new communication requirement. In fact, CHANGE will enable the deployment of inherently flexible service delivery infrastructures, significantly simplifying long term network provisioning and planning and thus helping reduce ISPs costs while increasing their competitiveness.

2.2 Survey of Recent Research

Many researchers have recently observed the shortcomings of the current Internet and tried to address them. There is a vast body of research on how to fix the different perceived problems with the current Internet architecture, such as [21, 42, 25, 17, 30, 24, 1, 5, 37, 4, 16, 10, 11, 40] to name just a few. The problems attacked in these works are numerous; we enumerate here the most prominent ones:

- Making the Internet more robust [16] and/or more transparent by including middle-boxes in the Internet architecture [25, 42].
- Supporting seamless mobility, multi-homing, fixed identities whether at the network level [17, 36] or at the content level [24, 30, 4].
- Optimizing access to content [24, 30] by treating content as a network entity.
- Supporting accountability at different levels [1, 10].

- Security - resilience to DDos attacks [1, 5, 37, 10, 46].
- Multicast [37, 6, 29].
- Anycast [24, 30, 37, 22, 39].
- Sharing the internet capacity [10, 40].

Although the problems are diverse, the basic techniques used to solve them seem to revolve around a relatively small toolset:

- Indirection [17, 16, 36, 22, 39].
- On-path and off-path signalling [17, 25, 42, 27].
- Locator/ID split [24], typically with a flat ID space [30, 17, 42], or even without locators [11].
- Name based routing [24, 30, 11].

Despite this huge research effort few of these techniques have been deployed and even fewer are currently used. The problem is twofold; first, for many problems there is currently no clear business case at Internet scale (e.g., multicast, anycast, or name based routing). For the pressing problems, such as security or capacity sharing, point solutions have been deployed and applied as patches in individual operator networks, or indeed at application layer, hence there is no pressing need for a global, optimal solution. The main effect of this myopic approach to problem solving is Internet ossification, as we have also pointed out elsewhere.

Further, for the vast majority of network layer solutions, the biggest barrier has been bootstrapping deployment, as there is no incentive to deploy unless early adopters get a benefit. Early adopters bear most of the costs, so they must be able to capitalise on the benefits; if benefits only come from a critical mass of deployment, early adopters gain nothing. These deployment hurdles exist not only for clean-state approaches, that require changing all (or a significant fraction of) routers [24, 30, 17, 42, 1, 37], but also solutions that require modest changes at the IP layer [10].

Application layer solutions such as I3, OASIS, Overcast, ALM, or RON are relatively easy to deploy, but face other hurdles. By definition these services are network and operator agnostic, and therefore suboptimal in many respects. For instance they do not (and indeed cannot) consider operator policy, or network topology when deciding how to provide a service. Compared to their network layer counterparts application layer solutions end up using suboptimal paths (e.g. i3, RON, OASIS, etc.), and/or transferring more data (e.g. ALM). If used extensively these solutions will effectively fight the network, and the network will respond as it did with peer-to-peer traffic: downgrade service by embedding even more application knowledge in the network.

In contrast, CHANGE has a clear deployment path, but it also inherits most of the desirable properties of network-based solutions. Deployment of a single CHANGE platform should require not much more intervention that deploying a DPI box today. Once such a platform is installed, it can be used within the operators

network in many ways, and can even be rented out to customers for custom processing. Hence, deploying such a platform can not only simplify the management and the evolution of the operational network, but also provide additional income (with a business model similar to cloud computing). Once enough individual platforms are deployed, interconnection benefits become obvious, and we expect there will be a strong push for unified standards and platform cooperation. The second key difference is that CHANGE does not preach running every packet through a flow-processing platform: we envision most of the added functionality will only be needed for a comparatively small number of flows, leaving the bulk of the network to do what its best at: moving packets. For instance, DDoS defence requires filtering at a few vantage points in any given DDoS attack, at any given moment in time; this is in contrast with most other proposals [5]. Not only does this ensure the whole platform scales, but it also limits the initial investment needed to get things going. Finally, CHANGE includes as basic primitives both indirection (virtual nets) and on-path processing (signalling). We believe these two mechanisms are fundamental enough to power solutions to most of the aforementioned problems. CHANGE itself does not set out to solve all the above problems; instead, it extends the Internet architecture just enough that proper solutions can be deployed when needed.

3 Change Architecture Overview

Whether we would like it or not, there is no returning to the original end-to-end transparent Internet architecture. Quite simply, there are just too many reasons why processing of data flows needs to be performed within the network, not just in the end-systems. To enable innovation, we need to play to the strengths of both packet-switching and flow processing, rather than dogmatically camping on one side or the other. We acknowledge the fundamental advantages offered by packet-switching, but believe that the architecture is missing the primitives to introduce and reason about flow-processing at selected key points in the network. We assert that flow processing most not be provided in the form of in situ hacks (as is the current state of affairs) but in a way that makes it a first class object in the network. Done right, as aside from consolidating the current state affairs, we believe it will allow operators and application writers to reason about the emergent behaviour of the end-to-end path through such a networks - an arduous task in today's Internet.

The deployment of a general purpose flow-processing architecture is what is required to break the innovation log-jam that has been developing over the last fifteen years. This is the overall goal of the project. Before delving deeper, we must first answer the question; *what exactly is a flow processing?*.

Flow processing is any manipulation of packets where the service given to or the operations applied to a set packets is differentiated based on their implicit membership of a labelled flow. Flows can be of different granularity; a single TCP connection might comprise a flow, but equally all the traffic between two sites can comprise an aggregate flow, or even all the Internet telephony traffic traversing a router. The concept of a flow is therefore generic, it simply defines a relationship between a set of packets. But what all flow processing has in common is that it requires the maintenance of some measure of flow state in the network.

The processing that might be applied to flows is very varied. At one extreme, flow processing might involve providing low-latency forwarding to traffic in a flow. At the other extreme, it might involve the reassembly of the contents of a packet stream and parsing of application-level content to perform network intrusion detection, or explicit authentication to a site firewall to allow subsequent packets of a flow to proceed.

To enable innovation, an architecture must be flexible, it must be possible to support a wide range of flow processing. It must allow for the rapid innovation and deployment of new flow processing primitives so that flow processors are not locked into the applications of today. In essence, this means that anything more than simple packet forwarding should be a software function, so as to allow for quick deployment. Contrast this with the current Internet, where flow processing is almost always performed in special purpose boxes sold by vendors to solve a specific problem.

In recent years several trends have come together to transform this general vision and bring it to practical reality. First, general-purpose x86 server hardware has become cheap and powerful enough for packet processing at rates of up to 20Gbit/s. The combination of PCI-Express, Gigabit or 10-Gigabit Ethernet supporting virtual queueing, CPUs with many cores, and high-bandwidth NUMA systems architectures, has resulted in low-costs systems that have been optimized for high-performance network processing on general purpose

servers. Such machines are equally comfortable at performing network flow processing, and having been optimized for virtualization, individual machines are able perform flow processing on behalf of more than one organization.

Secondly, high-performance Ethernet switch chipsets have become a low-cost commodity item. Such chipsets have flow-processing capabilities; typically they are able to perform matching and forwarding based on arbitrary combinations of packet header fields, as well as supporting multipath forwarding; useful for hash-based load-balancing across multiple output ports. Current commodity chipsets can support flow tables containing tens of thousands of flows. OpenFlow [35] takes advantage of such chipsets and provides a common API to control flow processing in these switches.

Combining OpenFlow-style switches with clusters of commodity servers allows powerful and scalable platforms to be built [23]. The switch provides the first level of classification, and balances traffic directly across virtual queues on the servers, allowing traffic to be directed to specific CPU cores for more sophisticated processing. Such a hardware platform provides a very powerful, scalable and flexible base on which to build a flow processing system. An enabling goal of CHANGE is to build upon preliminary work on these platforms, with the goal of realising flow processing systems that can take full advantage of the flexibility inherent in such a hardware platform.

Flow processing platforms offer the potential to realised several distinct advantages:

- The capacity to scale up processing by merely adding more inexpensive servers
- The capacity to scale down processing by concentrating load on a few boxes at quiet times to save power consumption
- The capacity to roll out new flow processing functionality at short notice to handle unexpected problems, or take advantage of unexpected opportunities, with only software reconfiguration required.
- The capacity to support a wide range of functionality thanks to relying on general-purpose hardware and operating systems
- The capacity to dynamically shift processing between flow processing servers
- The capacity to concurrently run different kinds of processing on different sets of flows while providing high performance and fairness guarantees

Running isolated CHANGE platforms brings many benefits to the deploying entity: it will be easy to update processing functionality and to increase processing capacity on demand. However, this is only a small improvement with regards to both the transparency and the rapid evolution of the Internet. The real benefits come when the CHANGE platforms cooperate to ensure end-to-end flow processing, at which point users will be able to reason about expected performance of their instantiations, detect bottlenecks, and scale out processing, all in a principled and transparent manner.

3.1 Architecture Requirements

The Internet is a playing field where different stakeholders fight to impose their requirements. End-hosts wanting to run new applications would like the Internet to return to the days of ubiquitous reachability (that allows peer-to-peer operation, for instance) and a "dumb" network that treats all packets equally. The network operators, on the other hand, must enforce security via firewalls, deal with the depleting IPv4 address space by deploying NATs, and use DPI to differentiate traffic to impose some notion of "fairness" among users. These tussles in cyberspace will not be resolved any time soon, they are built in to the fabric of the Internet [15]. Therefore technology must not take sides, and solutions should allow for the tussle to play out in different ways. Currently the balance of power is tilted towards to the middle; ISPs deploy functionality that makes it harder and harder for the endpoints to deploy new applications, and applications get more complex in order to jump the hurdles.

In this context Skype provides a telling example: it does almost everything possible to get through NATs and firewalls, including tunnelling voice traffic over TCP and using application-level proxies to allow conversations between two hosts behind NATs. Anecdotally, a big part of Skype's success stems from the very fact that it works most of the time where no other similar app achieves quite the same results.

While Skype does work, it is far from optimal: voip traffic between two users in the same city can sometimes end up re-routed through different countries, inflating end-to-end RTT and decreasing user satisfaction. This state of affairs is suboptimal for the operators too: they have to pay for intra-domain traffic that should have never left their network.

In fact, this is a symptom of a bigger problem, and Skype is not an isolated case: iPlayer and BitTorrent are examples of other applications whose traffic can take suboptimal paths that hurt the operators' economic interests. The real problem is that the operators' policy is unknown to the applications, and the applications' desires are unknown to the operators. The result is an escalating war where DPIs are deployed and P2P traffic is throttled, then the apps encrypt the traffic to avoid DPI, and so on.

To avoid all apps having to include complex functionality just to ensure basic functionality such as reachability, the network must be able to provide, on request, standard functionality that makes it possible for endpoints to communicate even if they are behind NATs. One useful functionality is the the ability to encapsulate and decapsulate packets in the network. Further, the users must be able to tell the network what they want and the network should help them implement that processing, rather than placing all the processing at endpoints - this will allow us to avoid the suboptimal routing we saw earlier, and hopefully the resulting race to-the-bottom.

All of these can be accomplished within a flow processing architecture, as proposed by the CHANGE project. The original Internet architecture lacked the concept of a flow, except as realized at the end systems. The end-to-end principle provides a good argument against putting functionality in the core of the network, but taken to its extreme it means than a network operator cannot see if the network is actually functioning. A network can be moving very large numbers of packets but still getting no useful work done, as in the case of a DDoS

attack. Although packets are the basic forwarding entity, it is flows that perform work, and so it should be unsurprising that network operators insert equipment into the network to manage and enhance flows.
Retrofitting a flow-processing abstraction to the Internet architecture is difficult to do in a clean way. With hindsight, the original placement of addresses in IP and ports in TCP and UDP might have been a mistake, as it places at least one component of flow identification in L4, where it was not intended to be used to make forwarding decisions. Nevertheless, we assert that it can be done. The main pieces are [1]:

- A general-purpose scalable flow processing platform on which L4+ middlebox functionality can be implemented in software and updated as apps change.
- A categorization of flow processing behavior into a handful of classes, so we can reason about the behaviour of concatenations of middleboxes without needing to understand the precise functionality of each.
- A way to identify who can request processing.
- A way to name flows.
- A way for end-systems to discover platforms close to them or to the paths that their flows take.
- Attraction of flows to platforms that will process them.

The requirements above are functional, saying what CHANGE should incorporate to be able to fulfil its tasks. Given these, how should we choose from the wide range of implementation possibilities that exist? Our goals are to create an architecture that is feasible to deploy and use in practice and is appealing to both operators and end-users. In the next section we discuss a few high-level requirements that steer CHANGE towards these goals.

3.2 Architecture Overview

The high level CHANGE architecture is presented in Figure 3.1. An end-to-end path comprises segments performing conventional IP destination-based forwarding, interconnected via Flowstream platforms that can optionally provide L4 processing on demand. The platforms that process a flow can be on-path, as shown for flow A, or can be off the original destination-based path, as shown for flow B. Typically tunnelling is used to move traffic off the original destination-based path.
A flow processing architecture that allows every host to instantiate arbitrary processing in the network will make the Internet even harder to secure and reason about. What properties make a flow processing architecture feasible to deploy and enticing to users at the same time?

(i) *Deployability*. Flow processing must be easily deployed and compatible with the current Internet. Technical solutions should be compatible with and easily deployed in the current Internet; anything

[1] These components follow directly from the functional architecture requirements spelled out in Deliverable 2.1

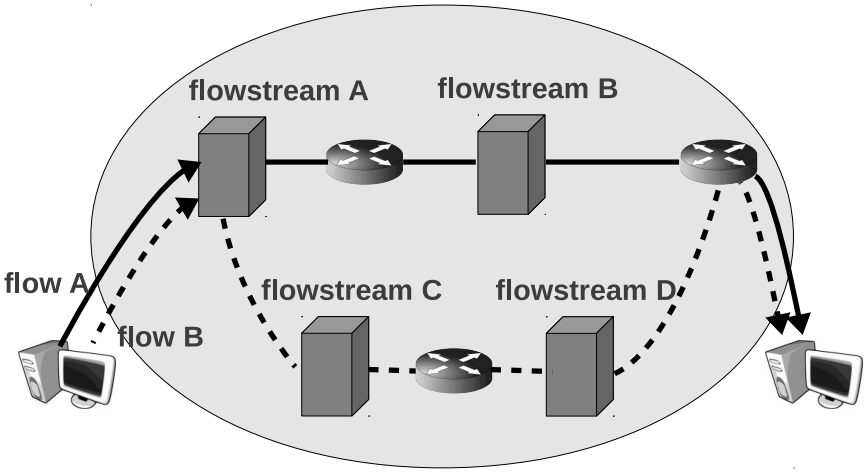

Figure 3.1: CHANGE Internet Architecture

short of this is considered a show-stopper. Given the size and importance of the Internet, *revolutionary* changes are unlikely to succeed - our best bet is *evolution*. Indeed, the recent flurry of clean-slate Internet architecture research, surveyed in section 2.2, has had little, if any, impact on the Internet we use everyday.

(ii) *Security.* Deploying the architecture should enhance rather than reduce security in the Internet - a low bar by any measure. At the very least, a new architecture should provide protection against DDoS attacks and it should provide accountability: the ability to name the hosts that are the sources or destinations of traffic, the ability to detect modifications to the packets, and so forth.

(iii) *Correctness.* Flow processing should not result in forwarding loops and loss of reachability and it should comply with the policies of the operators and of the end-hosts.

(iv) *Flexiblity.* New applications should be enabled by this architecture, as long as the other goals are obbeyed.

Flow processing is a broad term to describe the arbitrary operations on packets judged to be members of a labelled flow. A starting point of the CHANGE architecture is defining what a flow actually is, and how endpoints and platforms can refer to the same packets in a flexible and performant manner.

Rather than using higher level definitions (e.g. HTTP traffic to Youtube, or a certain SIP call) CHANGE simply uses **bitmasks to specify flows**. If packets seen by a platform match that bitmask, they are deemed to be part of the flow. This definition is both flexible in that it can describe flows at different granularities and it is easy to implement in hardware using TCAM memory (OpenFlow provides similar capabilities).

To be able to reason about the composition of flow processing primitives we analyze a set of basic operations applied to packets from both a security and functionality perspective. Combinations of these operations support arbitrarily complex functionality; read operations, filtering operations, the ability to overwrite IP source or destination addresses, the ability to generate new packets, and finally the ability to change arbitrary parts of the packet payload (layer 4 and up). We discuss them in detail in Section 4.3.

A central part of the architecture is the ability to authenticate the platforms and endpoints requesting architecture. CHANGE leverage existing solutions such as DNSSEC reverse lookup and RPKI to provide the desired functionality. A discussion on the various methods used to authenticate platforms and their implications is provided in section 4.2.2.

Endpoints must find platforms for processing. Techniques such as BGP anycast or the DNS can be used for some purposes, but do not cover all the use cases we would like for CHANGE platform discovery. These are described in Section 4.5. Once processing is instantiated, traffic must be attracted to processing platforms that are not on-path. Our preferred solution out of the possible alternative is to use DNS for cooperating traffic and BGP FlowSpec for uncooperating traffic; these are briefly described in Section 4.6.

3.3 Motivating Scenarios

To understand how Change would work in the current Internet we select a few scenarios and discuss them in greater detail:

- *Deploying New Transport Protocols.* Our first scenario looks at how instantiating path segments (e.g. partial tunnels) for flows can help deploy transport protocols in an efficient way. This use-case is instrumental in empowering the users to evolve the network (i.e. deploy new applications).
- *Inbound Traffic Engineering.* This is an example application useful for ISPs that is very difficult to implement today with fine granularity. It shows that CHANGE equally useful to both end-users and ISPs.
- *Distributed Denial of Service Attacks.* DDoS attacks have been a common occurrence for many years, and there no signs of them going away anytime soon.
- *Monitoring* helps users understand what is going on with their traffic, and helps them both debug the network and their flow processing, and to make the right choices in enabling the desired functionality.

4 CHANGE Architecture Building Blocks

The CHANGE project proposes to enhance the Internet Architecture with the introduction of *flow processing extensions*, whereby flow processing platforms, scattered throughout the Internet, perform flow processing on behalf users. We envision that the greatest benefits and flexibility of our proposal will come when non-local users can request remote functionality to be performed on their flows.

4.1 Flow Naming and Identification

In raising the topic of flow naming and identification we are posing the question; what is a *"flow"*? To provide adequate answers, we need techniques that allow both users and platforms to uniquely identify the packets belonging to a single flow, both for labelling and processing.

At the highest level of abstraction, flow identification is equivalent to defining a Boolean function, such that given a flow definition and an arbitrary packet, it returns $TRUE$ iff the given packet belongs to the named flow and $FALSE$ otherwise.

In principle, there are two basic requirements for a flow naming solution; *flexibility* and *performance*. The flexibility clause is best expressed when we look at how a flow definition may be used. For example, a flow definition may be required to capture a single TCP connection, or all UDP traffic destined to an IP address, or ICMP traffic originating from a given source. From these examples it becomes clear that the range of possibilities is clearly vast, and as such an adequate flow naming solution must be flexible enough to capture all these flows.

Given that platforms will capture the flows of interest to them by applying filters to the transient traffic they see, it is obvious that this fundamental task must be carried out efficiently. To realise this, flow naming must be amenable to a fast, line-speed implementation. This performance requirement is somewhat elusive, though hardware advances will push the envelope for what processing capacity, it is not adequate to simply rely on "better hardware in the future", instead the basic definition presented must be implicitly suited to high-performance processing.

Conventional wisdom holds that maintaining and accessing per-flow state for all Internet traffic is in practice unfeasible. The reasons for this are mostly economic; while it is possible to build routers with large and fast flow memory in practice these are prohibitively expensive. Though DRAM-based solutions are cheaper, they are also slower. To allow fast identification of a flow, we require that a flow-processing platform be stateless; it does not maintain additional per flow state. This means the Boolean function identifying a filter must also be memory-less so as not have any side effects.

Traditionally, flows are defined using the 5-tuple syntax of `(IP_SRC, PORT_SRC, IP_DST, PORT_DST, PROTO)`, while the corresponding Boolean function implements a checks for the exact matching of all the five fields. The most evident disadvantage of the 5-tuple syntax is that it cannot scale up and down the protocol stack. For example, if a flow consists of all packets to one destination, there is not a single

function that matches all the desired packets.

A more scalable approach to the problem is to make the flow identification process protocol agnostic. By this we mean replacing the implicit IP header dependence of the traditional 5-tuple syntax with an easily generalisable *bitmask*. Such that when the bitmask is applied to a given packet, if the mask matches/captures the packet, the packet is labelled as belonging to the flow that defines the bitmask.

Such a bitmask could be used to address any bit from an IP Packet, including payload. For example, given the first byte of 4 packets as: `0110 0110`, `0111 1010`, `1010 0110` and `1110 0001` and the bitmask: `*1*0 ****` - whereby $*$ defines a wild card, it is clear that AND'ing the bitmask will only match packets 1 and 4.

Because of the generality of this approach, it is trivial to craft bitmasks that can support basic flow identification tasks such as:

- Identifying a TCP connection - specify the 5-tuple
- Identifying ICMP traffic originating from a given host- match the ICMP protocol number and the source address.
- Identifying the packets going to one destination host: we only match the IP_DST field.
- Identifying the packets destined for a group of hosts in a subnet: we simply modify the mask, in order to contain all the hosts from that group. (e.g. match 128.16.6.*)

Matching bitmasks has the advantage that is easy to implement in hardware using TCAM memory - which is becoming cheaper and increasingly commonplace with the recent advent of OpenFlow switches. While our flow definition is naturally applicable to packet headers and allows for expressive selection of flows at layers two, three and four, be they host-to-host, many-to-one or even many-to-many. Further, bitmasks can be used to select application level flows where port numbers are sufficient for identification.

Though immensely flexible, there exists a whole class of application-level flow definitions (characterised by the requirement that flow state information be kept) that our solution does not capture. For instance, selecting flows that contain a certain worm signature or HTTP requests for a given Youtube video or a the packets associated with a given FTP session or all the sub-flows belonging to a given multipath TCP connection.

In principle some of this functionality can be achieved by applying the bitmasks on packet payloads. For instance, if the destination IP is that of Youtube and the payload is `'GET /video.avi'` we could redirect the traffic to a local cache. However, there are implicit limitations with such an approach. For example:

(i) Packets belonging to a flow may get redirected while others don't - in the Youtube example, the request segment is redirected but the SYN exchange or subsequent packets are not. This is not what we would like to happen: we would like the whole flow to be redirected, or at least the part of flow after the request.

(ii) Middleboxes could split/re-segment packets, which would make payload bitmasks fail. In the Youtube example, if the resource name is split across two segments the bitmask will fail to match the segments.

(iii) Interesting patterns could start at different offsets within the payload (e.g. a worm signature).

The first problem can be fixed if there is some callback for a matched packet (e.g. redirect to rule owner), whereby a new rule would be inserted when a match occurs. For example in order to capture the Youtube cache, a new 5-tuple rule would be inserted to match all subsequent packets associated with the flow.
The second and third problems are more difficult to solve, because they require the flow processing platforms to reconstruct the TCP payload before it applies the matching. While the matching itself needs to be more complex to support varying offsets; for example, full blown regular expressions will be required for searching worm signatures. The commonality between the listed examples is that they require per flow state to function and as such will not scale to large numbers of flows.
As discussed, though limited, our bitmask based solution (applied to packet headers) provides the highest level of generality without instantiating per-flow state. Applications that do require per-flow state can still be supported, by using a two-stage approach to first identify and then handle their traffic. Whereby, first the bitmasks are used to select a slice of the flows (i.e. flows going to a certain destination, or having a certain destination port, etc.) and to redirect them to a local processing module. Second, the processing modules keep per flow state as required, and implement the desired processing functionality, for instance detecting flows that carry a given worm signature. Once detected, these flows will be either redirected for further processing (c.f. FlowStream), tunnelled to the destination, dropped, etc.
Finally, while we consider a packet-level bitmask to uniquely define flows, there are currently limitations due to the OpenFlow architecture. OpenFlow currently accepts only bitmasks based on source IP and destination IP addresses, and exact matches on other L2 and L3 header fields, as well as TCP ports. It is expected that OpenFlow will in the future support bitmaks for arbitrary bits in the header.

4.2 Flow Ownership

The CHANGE architecture envisions high-performing and flexible flow processing, however, if all users are allowed to freely instantiate processing or retrieve statistics for all other flows in the Internet it can be easily be misused. It is evident that if the actions users are able carry out are not controlled in some way, the benefits to be gained will be outweighed by the risks raised. To realise useful flow processing that is suitable for deployment, we must define the methods to reason about and limit the actions users are able to carry out. In effect we seek to provide the means to reason about who may request flow processing including flow statistics. For any point-to-point message exchange it could be the source, the destination, or the network boxes that see the traffic. (We can generalize this definition for many-to-one and one-to-many traffic patterns by limiting each user to process packets coming to it).
There is an interesting parallel here to the roles of entities in the routing system today: traffic (or flow)

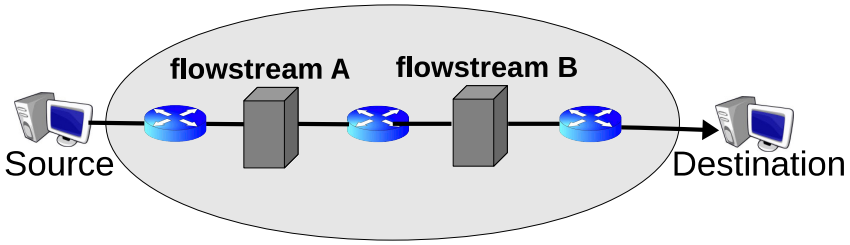

Figure 4.1: An end-to-end flow in the CHANGE architecture

ownership is centred on the destination; hosts (or networks) can choose to announce or withdraw routes to their prefixes depending on whether they wish to reachable or unreachable. Multi-homed sites can choose one or a subset of uplinks on which to announce their addresses, intermediary routers rank routes to the same prefix according to their own preference, in order to influence the paths of the traffic that passes through them. While the source should be able to control the path its traffic takes, in practice it has the least control over its packets, because extensions such as Loose Source Routing are not widely supported [19].

It is obvious that the end-hosts or any network element "owning" a flow (i.e. observing the flow's packets going through) can, even today, apply any processing it wishes to those packets. Today such processing includes: filtering, dropping, redirection, content modification, application-type inference, and so on.

To address this problem, the question we need to ask is; out of the entities that own a flow, which are allowed to request processing to be performed *on their behalf* from other platforms? The answer is not straightforward, as it also depends on what type of processing is in question. Traffic monitoring is essentially a read-only operation, and bears less security implications than, say, rewriting packet contents. To understand the differences between these mechanisms, we discuss authentication in the context of a few primitive types in section 4.3.

4.2.1 Once You Forward the Traffic, It's Yours

Particular properties of the current Internet architecture constrain the possible solutions to flow and entity authentication. It is easiest to see this with the example shown in Figure 4.1. First, assume that only the `Source`, as identified by the IP source address in the packet header, is allowed to make processing requests. Even if we assume that the source can cryptographically prove that it owns the address it refers to, any on-path platform can network-address translate or even tunnel packets it wishes to have processed, by pretending to be the source. In our example, flowstream `A` can NAT the flow and request processing on flowstream `B`, pretending it is the source of the traffic. In the current Internet this is perfectly acceptable, and there is no way to prohibit such behaviour. The implication is that *any* on-path entity has full downstream control of the flows it forwards if source authentication is allowed.

Now assume that only the destination (`DST`) of a flow is allowed to request *any* flow processing functionality. As illustrated in Figure 4.2, consider an on-path platform `A` that sees the traffic to destination and wishes to have it processed downstream by another platform `B`. Platform `A` can apply the following strategy to convince `B` to do processing on its behalf: First `A` create a tunnel from its self to `B`. Then, `A` takes each packet destined

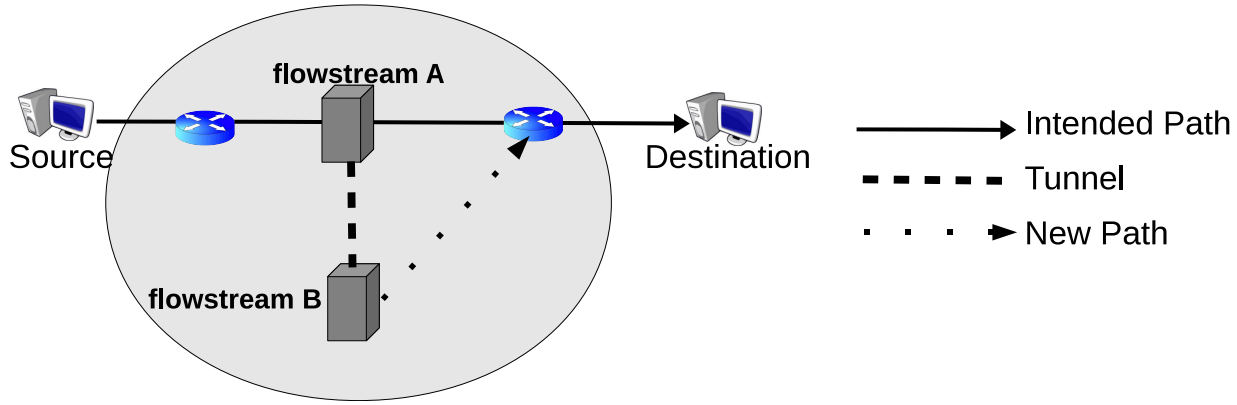

Figure 4.2: Being On-Path Enables Nodes To Re-direct Traffic Arbitrarily

to DST, rewrites the destination address to its own address and sends it through the tunnel to B, and finally at B, it decapsulates the traffic.

Since B (which was previously off-path) now sees traffic exiting the tunnel is labelled to be destined to A, it will consequently allow A to instantiate new processing on the flow. A can then rewrite the destination address of the packets back to DST, redirecting the traffic back to its original destination.

Downstream Flow Processing Property: if either the source or the destination of a flow are allowed to request in-network processing of it, *any* on-path network element can request processing from other *downstream* platforms.

Note that the *downstream* distinction is very important; it says that a platform must be able to *see* the traffic before it can request processing for it. In general, this effect seems benign: a platform that already has full control of the traffic can exert its control downstream.

If the traffic going through a platform is encrypted (e.g. an IPSEC tunnel), can a platform misuse flow processing to read traffic contents downstream? The answer is no: the platform can control downstream processing up to the tunnel exit point, which is the current destination of the flow. The platform has no control after the traffic is decapsulated.

4.2.2 Authenticating Flow Ownership

An on-path platform will regularly receive flow processing requests from third parties. Accepting all requests creates obvious security risks. Platforms may allow processing to be instantiated by traffic sources, destinations and possibly on-path platforms.

Authenticating the traffic source and destination is relatively straightforward; in principle, we can check that the requesting entity is the rightful owner of the source or destination address of the IP packets. If the request involves a prefix rather than a single IP, authentication could be performed at the prefix level.

There are two practical ways of implementing such authentication in today's IPv4 world; leveraging DNSSEC reverse look-ups or using the newly proposed RPKI infrastructure. The basic idea in both of them is that IP address owners will have a public/private key pair which they can use to authenticate themselves. The public

key will be delivered with X.509 certificates authenticated either by the DNSSEC or RPKI infrastructure. We discuss these and other alternative solutions in greater detail in Deliverable 4.1. Because of the way address allocation works, both RPKI and DNSSEC support authentication at the prefix level.

Extending such authentication all the way down to IP address level is a matter of deploying RPKI/DNSSEC infrastructure at every ISP; the resulting host-based certificates could be deployed via new DHCP options (as proposed in D4.1). Alternatively, an ISP may just sign processing requests from hosts with its prefix's secret key. The downside is that this allows L2 impersonation attacks within the ISP network, for instance ARP spoofing in 802.3 networks.

In IPv6, authenticating individual traffic sources or destinations is easier as it does not require PKI. The 128 bit IPv6 address is split in two equal parts: the network part of the address that is used by the routing system and the host part. Hosts can freely choose the host part of the address with Stateless Address Autoconfiguration. This allows hosts to create addresses that are self-certifying; a given host may create a public/private key pair, and use the first 64 bits of a cryptographic hash of the public key as the lower 64 bits of the address. The host can then prove to CHANGE platforms it owns the address by using its secret key. This is the concept used by Cryptographically Generated Addresses (CGAs) [3]. CGA's do not allow the ISP to authenticate itself, to support it, RPKI is also needed.

Authenticating *on-path* platforms focuses on checking whether or not a requesting platform is downstream to the traffic in question. As mentioned, upstream platforms can always request processing from downstream platforms if source or destination requests are allowed. Further, if on-path platforms are allowed to request processing, either the source or the destination will be allowed too.

The basic solution to address this problem is to insert authentication messages in the data path and check that the requesting platforms sees those messages. Secure solutions must have two basic properties:

(i) Authentication messages must be routed on exactly the same path as the traffic to be processed. This implies that in-band signalling is desirable.

(ii) Authentication should be continuous. Authenticating periodically is not satisfactory since attackers can be "active" for short periods of time and then move off-path, while the attack is in place.

Together, these two requirements limit the applicability of on-path platform authentication. A more detailed discussion of on-path authentication is provided in Deliverable 4.1.

4.3 Basic Flow Processing Primitives

A handful of primitives applied to packets can be used to capture most of the functionality needed to perform flow processing. These primitives are:

(i) Read primitives (including monitoring flows, collecting statistics and etc.).

(ii) Filter traffic.

Primitive	Authentication Condition	Access	New Values
Read	Traffic source or destination	Packet contents and visible flow processing	N/A
Filter	Traffic source or destination	N/A	N/A
Change IP source	Traffic source or destination	IP source	Own new source address
Change IP destination	Traffic source or destination	IP Destination	Any
Generate Packet	Traffic source or destination	Full	Own new source address
Change Payload	Traffic source or destination	Full	Any

Table 4.1: Processing Primitives and Authentication

(iii) Change IP source

(iv) Change IP destination

(v) Generate new packet

(vi) Change payload (for simplicity we consider payload to be anything after the IP header)

Though simple, these primitives can be combined in arbitrarily complex ways to enable powerful flow processing. We can see this flexibility by listing possible applications/functionalities and highlighted how they may be implemented based on the primitives.

Monitoring is based on read primitives (and applied mostly on packet headers) coupled with a method of returning the statistics to the requesting platform. The latter requires the ability to generate new packet using flow information. A Tee operation, on the other hand, requires a full read of the packet and generates a new identical packet with a different destination address.

Defending against denial of service attacks requires the ability to instantiate remote filtering at upstream platforms. Network address translation requires changing at least the IP source for forward packets and the IP destination for incoming packets. Packet scrubbing involves changing the payload and/or dropping.

An explicit packet proxy requires the ability to change the destination address on the forward path, and the ability to change the source address on the return path. Tunnelling requires rewriting both the packet source and destination as well as the payload at the tunnel entry and exit points.

Using combinations of the primitives listed we can implement almost all of the scenarios presented by CHANGE in Deliverable 2.1. Different primitives are useful for different functionalities and have different security implications; picking a single one-fits-all authorization decision (e.g. destination only) will result in a platform that is either overly restrictive or insecure.

4.3.1 Flow Processing Principles

In listing the properties and requirements on different primitives, we are seeking to answer the question; Who is allowed to instantiate processing using these primitives? Both traffic sources and destinations gain value by instantiating the primitives described above on their traffic. On-path platforms may also gain value, but it is fundamentally difficult to prove that a platform is on-path for a traffic aggregate; we discuss why this is

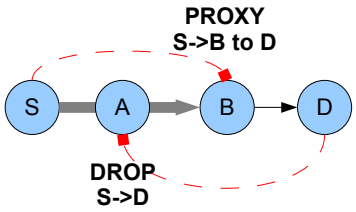

Figure 4.3: Rewriting the Destination Address Has Security Implications

the case in the next section, providing our reasoning for forbiding on-path platforms from request upstream processing (one such example would be platform B requesting processing from platform A in Figure 4.1).
Read primitives are intended to allow access to packet statistics as well as information about instantiated flow processing. One interesting question is whether the source should be able to see the processing instantiated by the destination. Transparency dictates that both endpoints should be able to find out about processing while security argues the opposite: for example, in the case of destination-initiated filtering against DDoS attacks, the source should be kept in the dark as to whether its packets are dropped or redirected. The technology we design should not argue one way or the other, but instead support both outcomes.

Principle 1. In-network flow processing should allow requesting endpoints to specify whether a processing function is visible to the other endpoints.

This principle raises the following question: once an entity is allowed to change the destination or source IP address, what new values are acceptable? Allowing arbitrary source addresses to be written is insecure, as it allows in-network spoofing. This leads to our second principle:

Principle 2. In-network flow processing should be allowed to change the packet's source address only when the requesting entity or the platform can prove it *owns* the new address (or has been authorized by the owner).

Changing the destination address inside the network is very useful as it allows instantiating proxies and tunnels: explicit proxies (such as squid) receive connections from the users, and then parse the HTTP headers to connect to the destination server; in effect, the destination address of the traffic is rewritten by the CHANGE platform, at the request of the traffic source.
Changing the destination address can also me misused: consider the attacker S shown in Figure 4.3 wishing to send traffic to destination D. D has instantiated filters on platform A to drop traffic originating from S. If S proxies the traffic via platform B, it can effectively bypass D's filter.
Repeatedly changing destination addresses, either via tunnels or just by rewriting the destination address, can easily lead to routing loops and the associated loss of reachability (due to expiring TTL values).
Figure 4.4.a illustrates a simple example, whereby the destination instructs platform B to tunnel traffic des-

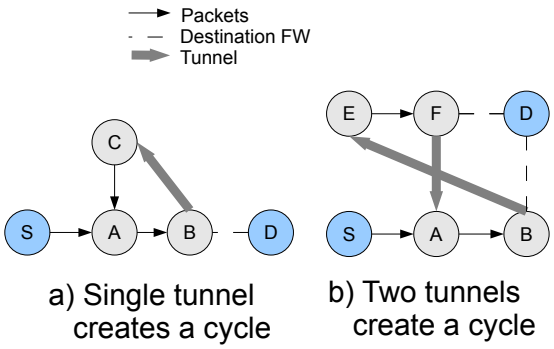

Figure 4.4: Instantiating Tunnels Can Lead to Routing Cycles and Loss of Reachability

tined to it to platform C. At C the traffic resumes destination based forwarding, but C's default route to D passes through B, and the packets loop. This behaviour is easy to avoid if B and C compare the AS paths of the routes they use to reach D, and C discovers that B is on-path.

Security aside, traffic sources and destinations acting without coordination can easily cause themselves harm. Take the simple topology shown in Figure 4.4.b: there the source sets up the B to E tunnel for traffic from S to D, and the destination sets up the F to A tunnel for the same traffic. The effect is a routing loop that no single host or set of tunnelling platforms can diagnose on its own.

Principle 3. Changing the destination address of a packet should be preceded by checks that ensure the new path is loop free.

How might we enforce this principle in practice? In Figure 4.4.b, if S and D choose to keep their processing invisible to one another, neither endpoint knows the full set of processing being instantiated on their flows, and as such cannot detect the cause of the routing problem. However, in such a case $\widehat{D}$ is probably already dropping the traffic since it implies it does not trust S. The more common case is when both S and D trust each other: in this case they will make processing visible, and each can correctly diagnose the problem.

The solution above is reactive, as it waits for things to go bad before it finds the problems. The same problems could also be fixed pro-actively: first, if the source and destination use visible primitives they can query the path of their flow to understand existing processing before instantiating new primitives. Such techniques may work for simple issues like loop detection, but will not work when the underlying network deviates from destination-based forwarding (for instance, if invisible processing moves the traffic in a different direction).

Should we ban unauthorized destination-address rewriting? This straightforward alternative mitigates both the security issues and simplifies the detection of reachability problems related to arbitrary destination address rewriting. All we need is to request that the entity wanting to rewrite the destination address should either own it or should be delegated to do so by the owner (this mirrors our Principle 3).

In Figure 4.3 this requirement would imply that S should ask permission from $\hat{D}$ if it wants to proxy traffic to D via B. If D grants the permission, it gives A a delegation certificate which B will use to authorize the proxy. If enforced, this would also mean that on-path proxies cannot instantiate arbitrary downstream processing with primitives reserved to the destination. Reconsider the attack shown in Figure 4.2: A can request B to tunnel, but it cannot redirect the traffic coming out of the tunnel to DST; it is forced to route it back to one of its own addresses. In other words, downstream flow processing can be effectively forbidden from certain primitives if only the destination is authorized to use them.

We've just reviewed a number of strong reasons to use delegated authorization, but the final choice is far from obvious. Delegated authorization also introduces a few restrictions when the same primitives are used for legitimate purposes.

Consider a source, such as a web server, wanting to create two distinct paths to a given destination; the host may run Multipath TCP [20] on top of these paths to get increased robustness and throughput for its flows. A strategy would be to choose the default path provided by BGP as the first path, and to proxy traffic via a way-point to create the second path. The trouble is that sources cannot instantiate proxied connections or even tunnels without the destination agreeing.

In the early days of deployment, few ISPs and fewer hosts will support flow processing. In this case there is no way that the source can get permission to proxy its traffic. The same restrictions apply to tunnels, another tool that is instrumental to the evolvability of the network (see our examples on using path segments to deploy new applications and protocols in Section 3.3). Such restrictions will limit the incentives to deploy CHANGE , reducing the competitive advantage of early adopters.

The jury is still out on which is the best choice for CHANGE : should destination address rewriting need authorization or not? The rest of this document assumes that destination authorization is not needed to rewrite addresses, but we are still investigating the security-functionality trade-off provided by these alternatives.

4.3.2 On-Path Platforms Should Not Be Allowed to Instantiate Upstream Processing

Safely authenticating on-path platforms is quite difficult as it requires in-band, continuous solutions; on-path challenge-response protocols run at specific points in time are not enough to guard against on-path attackers that move off-path after processing is started, while filtering precludes in-band authentication altogether. Such circumstances, raise the question; should we allow on-path platforms to instantiate processing on upstream platforms for *any* of the listed primitives?

The primitives that seem to pose the least security threats are in the read category, and seem to be the best candidate.[1] It turns out that even in this case in-band solutions are either insecure or difficult to implement.

To understand why this is the case, we can think about two possible strategies for implementing them. In both cases, a platform is requesting upstream processing. The first strategy is to use on-path challenge-

[1] Full packet capture (a read operation) along with transfer of the information to the remote site is equivalent to a tee, and thus has further security implications. However, for the remainder of this discussion, let's focus our attention on requesting packet-level statistics (such as counts, drops, rates).

response when monitoring is started. The first step here is for the user to provide an IP bitmask specifying the source/and or the destination of the traffic to be monitored, and challenge-response would need to happen on each individual IP that matches this bitmap. However, checking one or a few individual flows is not enough to defend against attackers that see a subset of traffic. Hence, a majority of traffic needs to be included in the challenge-response phase; platforms that only pass a subset of the challenge response test would not be allowed to monitor traffic. This last solution breaks down if there is genuine packet loss; the verification platform cannot reliably check whether packets are lost or routed elsewhere, away from the requesting platform. Similar problems appear with in-band monitoring, we could address this in a number of ways, for instance by add a total packet count to each prefix but this would allow an on-path both to estimate the total traffic volume going through the monitoring platform, it would however be undesirable since such information is considered sensitive by ISPs.

Principle 4. On-path platforms should not be allowed to instantiate upstream processing.

This principle has implications for instantiating certain classes of flow processing. To enable DDoS protection the destination must be able to instantiate filters upstream, yet it may be swamped with traffic and may not be able to react quickly enough. On-path platforms close to the destination can detect the attack, but are not allowed to directly install filters upstream as they are not the destination of the traffic. What is needed is a way to delegate authority from the destination to on-path platforms:

Principle 5. The platform must provide a way to delegate authorization from traffic endpoints to the on-path platforms.

There are straightforward ways of implementing delegation: the endpoint can sign a delegation certificate that specifies the delegated platform's IP address, the amount of time the delegation is valid for and the types of processing primitives allowed. When making a flow processing request, the platform would authenticate its own address and present the delegation certificate. To ensure correctness, platforms and endpoints must be loosely time synchronized.

4.4 Auditing and Billing

Processing requests are accompanied by a cryptographic proof that the requesting entity either owns the subject flow or has been delegated by the flow owners to instantiate processing. Platforms maintain logs of all such requests, so that they may be consulted offline for detecting configuration problems, security auditing and billing.

Billing is seemingly orthogonal to the architecture, but it does depend on the deployment model. The deployment model further influences how the CHANGE architecture is instantiated, and what technical solutions are most appropriate.

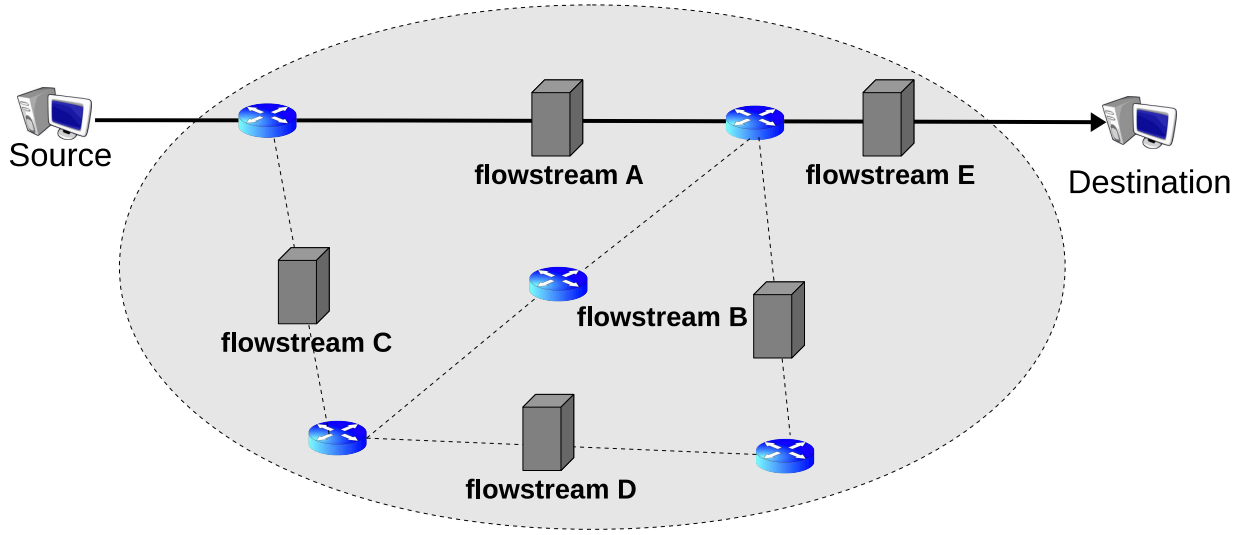

Figure 4.5: CHANGE Platform Discovery

We foresee two basic deployment models; in the first, a single entity for instance a Content Distribution Network, deploys a global infrastructure that offers flow processing functionality. In this case flow processing users get account with these CDNs, use an interface to make flow processing requests and pay the CDN for the services they use. In the CDN deployment authentication problems for on-path platforms are reduced, as there is an implicit trust relationship between all platforms operated by the same provider. However, flow processing is significantly more powerful then existing CDNs. Any one CDN deployment will likely be limited in the functionality it offers.

The second model is that of the Internet where a multitude of Internet Service Provides collectively provide global services. This is the deployment model we think is likely in the long term. The billing for such deployments is significantly more complex than the current cash flow in the Internet, with money flowing up the AS hierarchy. Solutions such as anonymous e-cash [12] seem more appropriate in this context.

4.5 Platform Discovery

If the CHANGE vision is successful, it would mean that multitudes of platforms will be deployed globally as shown in Figure 4.5. This however raises the obvious question of how do flow owners find the appropriate platform on which to run their desired flow processing functionality, and what requirements are needed for a good solution.

There are two main requirements; first, the platform must have the processing functionality required by the flow owner, and it should be willing and have the resource to participate in the processing. Out of this feasible set, the platform should be selected subject to the constraints and metrics defined by the user, for example minimising the overall processing cost.

Today we expect cost will equal end-to-end delay; delay has become the single most important factor affecting user experience, as exemplified by the efforts of the Web providers to shorten the paths [31] and to reduce the number of RTTs requires to download an average web object [13]. That is why we will use delay as our base

metric for discovering platforms. Arbitrary constraints can be implemented on top the delay metric; once nearby platforms are discovered, the user can query them to discover their various capabilities (e.g. required bandwidth or CPU availability); this information enables the user choose the most appropriate platform for their needs.

We have considerable flexibility in designing solutions to meet these two goals, and the end solution will also depend on the deployment type chosen. In a CDN-like deployment where is a known set of platforms and full trust between them it is simple to create a database of supported functionalities and possibly of available resources. In such a case, users will send requests to instantiate processing via an external, opaque API, and replies will include the identity of the platform.

In a federated deployment, resource availability is sensitive information, therefore maintaining a reliable database is unfeasible, which limits the applicability of a centrally accessible API. In this case, distributed solutions that better reflect the trust relationships between the platform owners, as well as being scalable are best suited.

Regardless of the deployment model, platform discovery needs to provide answers to the following questions posed by the flow processing customers:

(i) What is the closest platform to me? This might be used by `Destination` in Figure 4.5 to locate platform `E` and instantiate an intrusion-detection system on all its traffic.

(ii) What is the closest platform to a given IP? A host could use this functionality to instantiate filtering close to traffic sources with the purpose of defending against DDoS attacks. For instance, the `Destination` could use platform `A` or `C` to filter traffic from the `Source`.

(iii) What are the k closest platforms to a given IP? A generalization of the two questions above, this would allow requesting platforms to select the CHANGE platforms that can support the desired functionality that instant in time.

(iv) What is the platform closest to an end-to-end path? If we wanted to monitor a TCP flow, what platforms should we use?

The most obvious solution is to have a database of addresses of all CHANGE platforms (e.g. in DNS) and the requesting host should use active measurement to choose the appropriate platform. This approach has high costs for each platform discovery, and does not support locating platforms close to another IP.

An alternative solution leverages Internet routing by using BGP anycast. With this, each platform will have a common IP `C`, advertised via BGP anycast, and its own unique IP. When a flow-processing client wants to find a platform would create a TCP connection to IP `C` and a known port; the packets will be routed to the closest platform as determined by BGP. This solution gives the most accurate results, but does not directly support finding the k-closest platforms, or the platforms close to one IP. Using multiple IPs for CHANGE platforms solves the first problem.

To find platforms close to given IPs we need to be able to estimate latencies between two entities. There are a number solutions proposed in the research literature that can be applied:

(i) The King approach [26] makes the assumption that each host is close topologically to its authoritative DNS server. Thus, we can measure end-to-end delay of two hosts by measuring the delay between their authoritative DNS servers. This solution is attractive because it uses the existing DNS infrastructure, but its results are as close to reality as the assumption it relies on. With all the CDN deployments out there and DNS-based server load balancing, it is unclear whether this assumption is true today, 10 years after its proposal.

(ii) Virtual Coordinate Systems such as GNP [34] or Meridian [45] actively measure delays between sets of participating nodes, creating a cartesian or multi-dimensional space where Internet hosts are placed. The promise is that with a few initial measurements, and after the system converges, these coordinates can be used to directly estimate delays between hosts without further active measurement. The main drawback of Virtual Coordinate Systems are that they are not that accurate (for instance, it is difficult to deal with triangle inequality violations [32]).

None of the existing solutions fully meet our requirements; we are currently investigating a hybrid solution that can achieve both the accuracy of BGP anycast and the flexibility of Virtual Coordinate Systems solutions. Finally, there exists a simple approach to solve an instance of (iv) above: the source can find on-path platforms to the destination. The source can use traceroute to the destination to find the addresses of the intermediary routers, and will then attempt to connect to each IP on a specific port. If the connection is successful, in the final step the platform authenticates itself to the user.

4.6 Drawing Traffic into a CHANGE Platform

Within the CHANGE architecture, actual flow processing may be carried out by one or several cooperating platforms. However, when several platforms cooperate to perform processing, they may not always be on the same path. For example, if forwarding tables are left untouched, a flow may not traverse any of the intended processing platform from its sources to its destinations. This in effect means that in order to guarantee that traffic traverses the intended processing platforms, we must define the mechanisms to attract it there. Further, given the practical limitations of clean slate approaches, so as to be widely applicable, we must as much as possible provide mechanisms that function by leveraging existing and deployed technologies.

Figure 4.6 shows the high level solution used by CHANGE to attract flows into a platform located outside the initial AS path. Our proposed traffic attraction mechanism is composed of two disjoint parts; an *attraction mechanism* and an *delivery mechanism*. The attraction mechanism redirect flows from a position inside the Internet (for example a router in an Autonomous System) towards a designated platform. The delivery mechanism redirects packets from the platform back to their rightful destination.

We list the following set of properties that we require from a good solution to traffic redirection.

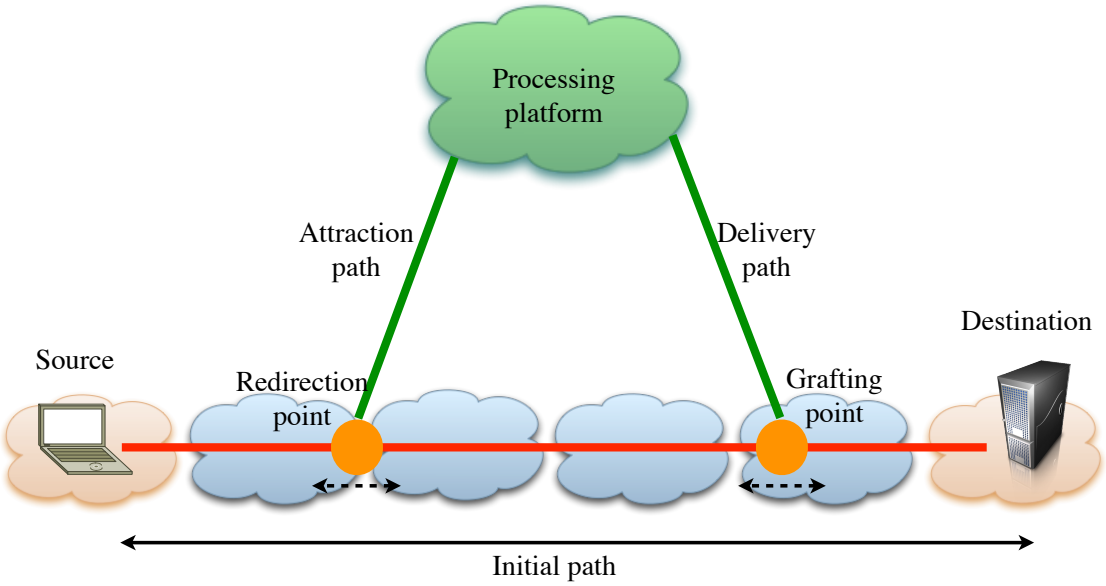

Figure 4.6: CHANGE Traffic Attraction Mechanisms

Flow granularity Ideally, we would like to support flow attraction based on our bitmask-based flow definition (see Sec. 4.1) but none of the existing solutions gives us this type of flexibility. Adapting them would require significant effort.

Manageability Establishing and diagnosing failures relating to traffic attraction and delivery should be easy to do in practice.

Convergence The activation, de-activation, and rerouting of flows should not cause data plane or control plane disruptions.

Possible attraction and delivery mechanisms are explored in more detail in deliverable D4.1. For both attraction and delivery direct tunnelling provides the simplest solution. Because tunnelling may be used to ensure the end-point of the tunnel is located downstream of the attraction point, it does not create loops in the traffic path.

We list the following set of approaches as possible solutions for attraction and delivery:

DNS-based redirection: DNS is the most obvious solution to use in order to redirect the flows directly from their sources to the CHANGE platforms. This solution allows attraction only based on the destination address. Attraction can also be made on a per-service basis if an IP address is associated with a service. The solution requires that the CHANGE platform control the name mapping of the rightful destination; when attraction is requested, the address of the processing platform is given as the answer to the DNS request for the destination's name. As the destination address in the packets does not correspond to the original destination IP address, NATing must be in order to send a flow, after it has been processed, from the platform onto the destination. A solution based on a one-to-one NAT is detailed in deliverable D4.1.

BGP announcements: This solution uses classic BGP announcements to attract a given IP prefix. It can be decomposed into two parts: attraction intra-AS or inter-AS. For intra-AS, we assume that the AS has a processing platform and that all flows for the destination prefix will be re-routed through this platform; BGP can be easily used to perform this. For the inter-AS solution, BGP can also be used, except we need to limit the scope of BGP announcements and only attract the flows for the destination in a controlled manner, i.e., in chosen neighboring ASes.

Flow Specification: *Flow Specification* (FlowSpec) [33] allows us to easily deploy matching rules on packets and apply a specific action to them. FlowSpec supports matching on different fields in the packet header (from layer 3 to layer 4), providing great flexibility in terms of flow granularity compared to the two other solutions. FlowSpec requires that the redirection is made using a tunneling solution. When a router matches a flow, it forwards it using a tunnel towards the platform. This differs from the two other solutions, where native forwarding is used to draw the packets to the platform.

Each solution has pros and cons. With regard to flow granularity, the FlowSpec solution enables to be more precise in our description of a flow than it is possible with the DNS or BGP solution. In fact, FlowSpec is the only solution that allows us to attract based on the source address. We detail the rest of the comparison of the three solutions in D4.1. DNS is simplest to use, but does not work uncooperating traffic (e.g. DDoS traffic).

4.7 Remaining Open Questions

We have described solutions for all the main components needed to deploy the CHANGE flow processing architecture in today's Internet. Further, we have set architectural principles to guide how these building blocks should be used to create a network that is extensible, secure and transparent.

We are still working on the glue between these building blocks, and there are remaining open questions that need answering to fully define CHANGE . Below is a list of the remaining open questions which will be addressed by the CHANGE partners:

Inter-platform communication Platforms and end-hosts alike will need to communicate to setup processing, to retrieve statistics and reason about end-to-end semantics. Which signalling protocol should we use for these purposes? Should we use existing solutions such as SOAP, or do we need fundamentally different functionality?

Instantiating processing primitives We've assumed so far that hosts (or interested parties) discover platforms and directly communicate with them to setup processing. This is a rather simplistic view: in many cases these requests will have to be authorized by the ISP which is the owner of the IP prefix. If the ISP is running a NAT it effectively needs to proxy these requests using the new flow identification (NATed source address). What protocols can we use to implement this functionality?

Instantiating multi-party primitives Many primitives require more than one participating platform: for

instance setting up a tunnel in the middle of the network requires that both the tunnel entry and exit be configured properly? Should the host contact these platforms separately, or should it only contact the tunnel entry point, and this should contact the exit point? What is the correct order of operations? Which existing protocols can we use (for instance GMPLS, or IPSEC)? Do we need new protocols?

Delegation In a few cases an entity needs third-party authorization to use certian flow processing primitives. What protocol shall it use for this purpose?

APIs Inter-platform flow processing needs to speak to individual CHANGE platforms to setup processing. For this we need standardized APIs and means of discovering platform capabilities.

5 Implementing the Motivating Scenarios

As discussed in Section 3.3, to understand how the CHANGE architecture can function in the current Internet this chapter presents four motivating sample deployment scenarios.

5.1 Deploying New Transport Protocols

The Internet has ossified, the net effect is that we are "locked in" an IP world. Deploying new transport protocols over UDP is possible (if not completely efficient), but UDP has a much harder time getting through the various middleboxes than TCP. Should we tunnel new transport protocols over TCP then? This would be nothing short of disaster, as the new transport protocol will inherit TCP's built-in functions such as reliable and in-order delivery, together with their associated problems such as head-of-line blocking.

What is needed is a way to create short TCP tunnels for the parts of the network that only allow TCP to pass through, and a way to "glue" these with segments that only do UDP encapsulation.

CHANGE hosts (source or destination nodes) may discover on-path platforms and setup tunnels between themselves to "hide" traffic from misbehaving middleboxes by creating an encrypted tunnel to ferry the traffic between intended targets. To instantiate the tunnel, the initiating host may contact the platform target of interest and setup the "payment" information. The receiving host does not need to authenticate the host to setup the tunnel; it will need to authenticate it if further processing is required, for instance network address translation. Note that implementations of this scenario need to take into account existing on-path NATs.

An issue of concern for tunnelled traffic is the question of how the source address is defined after the traffic exists the tunnel? The problem here is that the source address of traffic may be used to define its flow id, for example as is the case for the 5-tuple flow id structure. To address this point, the source host may choose to change the source address of the traffic exiting the tunnel, as if it were running a NATing at the tunnel exit point. This is possible because the source host has full control over the source address of the traffic that entered the tunnel, it is in a position to know the previous identifier information including the original source address. Such a set up would also force the return traffic to enter the tunnel, and can enable hosts to have a private (non-routable) IP address.

A second case is when the source host wishes to setup a tunnel for a path segment in the middle of the network. In this case, it needs to discover and contact the tunnel entry point and authenticate itself as the source of the traffic. The tunnel entry point then contacts the tunnel exit point and creates the tunnel. In this case, we are dependent on destination-based routing to direct the traffic to the tunnel entry point. If we want to ensure that traffic always goes through the tunnel, we have to either attract destination traffic to the platform or the source has to rewrite the destination address of the traffic to be the platform (preferred). Now the tunnel exit point must behave like a proxy, rewriting the destination address. If we also want the reverse traffic always cross the tunnel, NATing is again required at the tunnel exit point.

In practice we expect that endpoints will instantiate path segments when needed for the "access" part of the

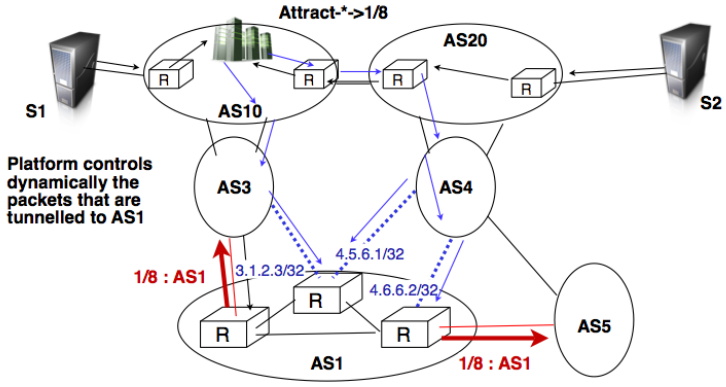

Figure 5.1: Stub AS Perfoming Inbound Traffic Engineering with CHANGE

path, to bypass deployed middleboxes, as the core is currently just doing plain packet forwarding. Complexity for this step is reasonable, as in the worst case one network address translation per path segment needs to be setup. When the path being created is in the core, though, things are more complicated, as hosts may need to instantiate both a NAT and a proxy for both directions of traffic or to otherwise draw traffic into processing platforms, for instance via BGP.

5.2 Inbound Traffic Engineering

In this scenario, we consider a stub Autonomous System (such as a cable or DSL provider) that is multi-homed to different upstream providers and wishes to balance its traffic over all the links. A concrete example is AS 1 in Figure 5.1 that has 5 uplinks to upstream ASes. Currently ASes advertise different IP address blocks over their uplinks, but this is coarse-grained and increases the size of the global routing tables. In IPv6, only the network part of the address can be advertised, and this technique is effectively unusable.

With CHANGE , AS 1 can choose to advertise its prefix only on two of its uplinks, and it sets up tunnels from a CHANGE platform in AS10 to use its three links. Within AS 10, the CHANGE platform will attract all the traffic destined to 1/8 by using FlowSpec. AS 1 can fully control traffic reaching it via AS 10 using processing on the CHANGE platform: it can choose to receive it using destination-based forwarding or via one of the three tunnels.

AS 1 owns both 1/8 and the three tunnel exit points; it can use its RPKI certificates to get authorization for processing. The configuration will be more effective in load-balancing when the amount of traffic crossing AS 10 (the platform) bound for AS 1 is large. If this is not the case, the host can try to get more traffic into AS 10 or use another platform upstream of AS5.

Knowing the topology helps speed up the process of instantiating effective load-balancing. AS 1 can discover nearby platforms and check their AS numbers, and use information on the AS level topology to instantiate processing at the right place in network. However topology information is sensitive and not easily disclosed by ASes; if no topology information is available the stub AS can still use trial-and-error to look for an appropriate load-balancing configuration.

5.3 DDos Filtering

Instantiating DDoS filtering is a simple process with CHANGE . The destination can use its RPKI secret key to authenticate itself as the traffic owner(or will have its own provider sign its requests using the prefix's secret key).

If AS-level topology information is available, the destination needs to discover and instantiate processing at its closest CHANGE platforms (directly connected/upstream ASes), selectively dropping the malicious traffic. If the traffic volume is too much for the current set of platforms to handle, filtering must be pushed further upstream. The destination will delegate platforms to act on its behalf and request further filtering, quenching the traffic volume.

5.4 Monitoring

Today's Internet harbours a myriad of middleboxes that independently perform useful functionality. The behavior of existing protocols like STUN/ICE is difficult enough to predict on their own, but when combined with other middleboxes that change, drop, scrub packets, re-segment or pro-actively acknowledge TCP traffic, understanding network behaviour and finding the causes of problems is a daunting task.

CHANGE can help here with its "read-only" primitives that allow traffic owners to examine their flows at different vantage points in the network. Traffic sources and destinations can instantiate (preferably on-path) processing and directly compute packet loss rates for path segments, detect packet changes (e.g. dropped TCP options), and so forth.

6 Conclusions

Flow processing is the way to break the innovation log jam affecting the Internet today. It provides powerful tools to end-points and operators alike to evolve the network in a transparent and principled way. Security is paramount in flow processing; surely a deployed but insecure flow processing platform will make the Internet worse than it is today.

In this deliverable we have shown how the CHANGE vision can be realised in practice. We have addressed basic questions such as answering "what a flow is" while slowly building a coherent architecture from a number of fundamental building blocks.

One of the main contributions of our work is teasing apart the authentication questions underlying the security of the platform and selecting practical solutions to implement them today. Our current preferred solution is the newly deployed RPKI infrastructure.

Security, correctness or practicality constraints limit the ways flow processing can be performed. We have reasoned about and described a set of design principles that prescribe what should and should not be allowed in a CHANGE platform. These principles dictate that only the traffic source and destination or parties delegated by them are allowed to instantiate processing. Further, rewriting source addresses in the network is limited to the owner of the source addresses.

An interesting open question is whether rewriting destination addresses should be allowed without the explicit authorization of the new destination. Security implies authorization, while flexibility and deployability seem to imply the opposite. Further work is needed to understand the best solution in this security-flexibility tradeoff.

CHANGE entities need to discover the right platform for flow processing and must be able to redirect traffic to them. We have analyzed existing solutions in this space and selected the most promising ones. All of the techniques we use are either already deployed and in active use, or are currently being deployed. This serves to reassure us that CHANGE is deployable in the current Internet.

There are a number of interesting open questions that we need to answer before the CHANGE architecture is fully defined. These questions mostly related to how we "glue" together these building blocks to realise the end-to-end flow processing goal.

Bibliography

[1] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (aip). In *Proc. ACM SIGCOMM*, Seattle, WA, USA, August 2008.

[2] K. Argyraki, A. Baset, B-G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, and S. Nedveschi. Can software routers scale? In *Proc. PRESTO*, Seattle, WA, USA, August 2008.

[3] T. Aura. *Cryptographically Generated Addresses (CGA), RFC 3972*. IETF, mar 2005.

[4] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the internet. In *Proc. ACM SIGCOMM*, Portland, OR, USA, August 2004.

[5] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. Off by default! In *Proc. ACM Workshop on Hot Topics in Networks (Hotnets)*, Maryland, MD, USA, November 2005.

[6] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, USA, August 2002.

[7] L. Berger. *Generalized Multi-Protocol Label Switching (GMPLS) Signalling Functional Description, RFC 3471*. IETF, January 2003.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services, RFC 2475*. IETF, dec 1998.

[9] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview, RFC 1633*. IETF, jun 1994.

[10] Bob Briscoe and Alessandro Salvatori. Policing congestion response in an internetwork using re-feedback. In *Proc. ACM SIGCOMM*, Philadelphia, PA, USA, August 2005.

[11] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, Ion Stoica, and Scott Shenker. Rofl: routing on flat labels. In *Proc. ACM SIGCOMM*, Pisa, Italy, September 2006.

[12] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, CRYPTO '88, pages 319–327, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[13] J. Chu, Nandita Dukipatti, Y. Cheng, and M. Mathis. *Increasing TCP's Initial Window, Internet Draft*. IETF, april 2011.

[14] Cisco. Cisco, unifed computing, http://www.cisco.com/en/US/netsol/ns944/index.html.

[15] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 347–356, New York, NY, USA, 2002. ACM.

[16] M. Frans Kaashoek Robert Morris David G. Andersen, Hari Balakrishnan. Resilient overlay networks. In *Proc. ACM Symposium on Operating Systems Principles SOSP*, Banff, AL, Canada, October 2001.

[17] L. Eggert and M. Liebsch. *Host Identity Protocol (HIP) Rendezvous Mechanisms, draft-eggert-hip-rendezvous*. IETF, July 2004.

[18] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards high-performance virtual routers on commodity hardware. In *Proc. CoNEXT*, Madrid, Spain, December 2008.

[19] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala. *Source Demand Routing: Packet Format and Forwarding Specification (Version 1), RFC 1940*. IETF, may 1996.

[20] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses, Internet Draft*. IETF, jul 2011.

[21] P. Francis and R. Gummadi. Ipnl: A nat-extended internet architecture. In *Proc. ACM SIGCOMM*, San Diego, CA, USA, August 2001.

[22] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazires. Oasis: anycast for any service. In *USENIX OSDI*, San Jose, CA, USA, May 2006.

[23] Adam Greenhalgh, Felipe Huici, Mickael Hoerdt, Panagiotis Papadimitriou, Mark Handley, and Laurent Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Comput. Commun. Rev.*, 39:20–26, March 2009.

[24] M. Gritter and D. Cheriton. An architecture for content routing support in the internet. In *USENIX OSDI*, San Francisco, CA, USA, March 2001.

[25] S. Guha and P. Francis. An end-middle-end approach to connection establishment. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

[26] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. *SIGCOMM Comput. Commun. Rev.*, 32:11–11, July 2002.

[27] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. *SIP: Session Initiation Protocol, RFC 2543*. IETF, March 1999.

[28] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Submitted to ACM IMC 2011*, 2011.

[29] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole. Overcast: reliable multicasting with on overlay network. In *USENIX OSDI*, San Diego, CA, USA, October 2000.

[30] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

[31] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 75–86, New York, NY, USA, 2010. ACM.

[32] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality variations in the internet. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 177–183, New York, NY, USA, 2009. ACM.

[33] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson. *Dissemination of Flow Specification Rules, RFC 5575*. IETF, August 2009.

[34] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *In INFOCOM*, pages 170–179, 2001.

[35] OpenFlow. The openflow switching, http://www.openflowswitch.org/.

[36] C. Perkins. *IP Mobility Support for IPv4, RFC 3344*. IETF, August 2002.

[37] L. Popa, I. Stoica, and S. Ratnasamy. Rule-based forwarding (rbf): Improving internet's flexibility and security. In *Proc. ACM Workshop on Hot Topics in Networks (Hotnets)*, New York, NY, USA, November 2009.

[38] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture, RFC 3031*. IETF, January 2001.

[39] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, April 2004.

[40] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Transactions on Networking*, 11(1):33–46, February 2003.

[41] Jon Turner, Patrick Crowley, John Dehart, Amy Freestone, and Fred Kuhns. Supercharging planetlab a high performance, multi-application, overlay network platform. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

[42] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *USENIX OSDI*, San Francisco, CA, USA, December 2004.

[43] D. Wetherall, U. Legedza, and J. Guttag. Introducing new internet services: Why and how. *IEEE Network Magazine*, 12(1):12–19, August 1998.

[44] D. Wetherall, U. Legedza, and J. Guttag. K. clavert and s. bhattacharjee and e. zegura and j. sterbenz. *IEEE Communications Magazine*, 36(10):72–78, October 1998.

[45] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: a lightweight network location service without virtual coordinates. *SIGCOMM Comput. Commun. Rev.*, 35:85–96, August 2005.

[46] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2004.