ICT-MobileSummit 2008 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2008
ISBN: 978-1-905824-08-3

RTFM: Publish/Subscribe Internetworking Architecture

Mikko Särelä[1], Teemu Rinta-aho[1], Sasu Tarkoma[2]
[1]*Nomadiclab, Ericsson Research, Hirsalantie 11, Jorvas, 02420, Finland*
Tel: +358 44 2992116, Fax: + 358 9 2993535
Email: mikko.sarela@nomadiclab.com, teemu.rinta-aho@nomadiclab.com
[2]*Helsinki University of Technology, Computer Science and Engineering Dept. and Helsinki Institute for Information Technology HIIT,*
Tel: +358 50 3841517, Fax: +358 9 451 5014, Email: sasu.tarkoma@tkk.fi

Abstract: We present a packet switched inter-networking paradigm different from the existing ones in that we argue for publish/subscribe (pub/sub) based networking architecture. The key idea of the paradigm is to distribute control over data reception from network endpoints to the network itself, and combine this with the way routing and forwarding is performed. In this paper, we outline the argumentation for the new paradigm and discuss its possibilities and challenges. We demonstrate the concept by presenting a prototype implementation and highlighting some lessons learned from local-area pub/sub networking.

Keywords: Internetworking, publish/subscribe, network architecture

1. Introduction

Over a decade ago, the network protocol stack consisted of the physical layer, the MAC layer, the internetworking layer, the transport layer, and the application layer. Today, the protocol stack has a more complex nature - there are several sub-layers within the network layer, a number of other protocols within the stack, and a number of layers on top of the transport layer. The evolution of the Internet architecture has resulted in numerous new protocols. Typically these new protocols have been specific to a class of applications, such as real-time transport; however, some have been very general in nature, such as the Session Initiation Protocol (SIP) useful for many application classes.

We live in a world with little or no interoperability between different applications or application classes. The network architecture delivers any given packet, achieving interoperability between different networking technologies, but does not help when the problem is achieving interoperability between different applications.

Event-based computing and publish/subscribe are vital ingredients for future services and applications. The event paradigm allows asynchronous and decoupled many-to-many communication, and typically supports data-centric information dissemination. Taking into account the asynchronous nature of many current applications, especially in the mobile environment, motivates us to consider a new kind of networking stack inspired by the publish/subscribe (pub/sub) paradigm.

The physical qualities of a link, be it Ethernet, Wi-Fi, or some other, give control to the sender. As an example, anybody connected to an Ethernet link may send to it regardless of what others do. The Internet Protocol broadens this concept of a local 'physical' link to the whole network. It provides a full mesh of unicast links between all hosts that reside in the Internet (we consider middleboxes, e.g. firewalls, NATs, etc. to be a violation of the architectural design of the Internet). Thus, each host may not only send packets to its

immediate neighbours but to any hosts in the internetwork. This full mesh nature of the Internet Protocol is well known to be one cause for the distributed denial-of-service problem [15,16].

In this paper, we present a new pub/sub based internetworking architecture called RTFM, (for Rendezvous, Topology, Forwarding, and physical Media architecture). It changes the control from sender to the receiver in the networking and internetworking layer. This is accomplished by changing the network architecture from push to pull, i.e. having the network deliver data to those who subscribe to it rather than to those parts of the network the sender wants it to be sent to.

The paper is structured as follows: Section 2 provides necessary background and surveys existing work. Section 3 presents the architecture, and Section 4 the implementation. Finally, we present the conclusions in Section 5.

2. Background

Distributed publish/subscribe has become an active research in recent years [6]. A pub/sub router is a component that connects, and decouples, the publishers and subscribers of data, and mediates packets between them.

The Siena system can be considered to be a classic example of a distributed content-based routing system that was implemented in the application layer. Siena is envisaged to integrate at the network service level, coexisting for example with TCP/IP instead of working above the network level. This would eliminate an extra protocol layer, and provide greater efficiency in routing and forwarding. The risk in using Siena as a network service is that content-based routing is computationally more expensive than explicit-address or subject-based routing [2].

A number of networking architectures have been proposed recently that support data-centric operation. Many of these systems build on flat labels, such as ROFL [10] and DONA [1]. ROFL is a network layer protocol for flat label based routing and forwarding. DONA introduces flat label based anycast support on a higher layer and includes a model for inter-domain policies.

Pub/sub functionality is also present in today's telecommunications systems. The SIP event package enables a SIP client to subscribe to the desired events and receive notifications when the expected event occurs [7].

3. RTFM Architecture

All previous pub/sub proposals are either overlays on top of the existing IP protocol, by using IP multicast or creating an application level multicast topology on top of IP. The problem with those approaches is that they don't solve the problems of IP and the Internet architecture such as DDoS attacks and the hugely changed environment (e.g. mobility) and used applications (e.g. web browsing, P2P file sharing). We, however, take a clean slate approach and try to think everything from data-centric receiver-controlled viewpoint instead of the end-point centric sender-controlled viewpoint of the existing networks.

In this part, we will describe the essential parts of a pub/sub network. How a host can join a network, publish data, and subscribe to publications, the requirements for the pub/sub routers (sprouters), how to find a route to a publication in case of a subscribe, and what happens when a host decides to publish.

To accomplish this, a pub/sub network gives each host only two network primitives: *publish* and *subscribe*. When a node publishes data, no data transfer actually takes place (the rendezvous system is informed of its existence). Only when a node subscribes to a named piece of data, the network finds the publication and creates a delivery path from the publisher to the subscriber.

Publications have two identifiers: a private identifier and a public identifier. The architecture doesn't restrict how the identifiers are formed, but it is a good idea for the identifiers to be cryptographically bound together. The idea is that the private identifier is known only to the publisher(s) of the publication, and the public identifier is used to subscribe the publication. The private identifier can be created by e.g. hashing the content of the publication data or metadata and a signature created with the private key of the publisher. The public identifier can be e.g. a one-way hash of the private identifier. Having these two identifiers per publication enables a) the publisher to prove it "owns" the publication b) the subscriber to check the validity of a received publication.

Obtaining an identifier calls for a directory service, where long lived publications may have entries which map human readable names into the corresponding public identifiers. This is similar to DNS, which maps domain names into IP addresses. A notable difference to the current architecture is that the directory service lists names and IDs of *data* objects and not endpoints or locations of data. It is the job of the rendezvous system to match the subscription to a publication, and this can happen in more than a single network location.

The network architecture is composed of three modules: Rendezvous, Topology, and Forwarding (and the physical Media). Everything in the network, starting from network attachment, and including how the above mentioned modules are built up, is done using these primitives. The following discussion describes the system in a steady state situation.

3.1 Forwarding

Forwarding is used to actually deliver data from one location to another. It is based on label switching, i.e. each packet has a label (or a stack of labels) and a forwarding table as shown in Table 1. A label is a bit string in the packet that is used by the routers to make forwarding decisions, and it can be the same as the publication identifier. Port is a local numbering (internal to the host) of the different next hops that the packet can be forwarded to, including the wired and wireless network interfaces towards the next hop sprouter, and other software modules (such as applications) internal to the host in question.

Labels can be stacked as shown in Table 1. The first row shows a packet labelled X, which will be forwarded on port 1 and a label A is appended to it. The second row shows the branching of a multicast tree Y. The last row shows a packet from which the label Z is popped from the stack and the packet forwarded to port 3. '*' denotes any label. Port 3 can for example be a loopback, i.e. returning the packet to be processed by the forwarding table after Z has been popped from the label stack.

Table 1: Forwarding table

Incoming label	outgoing port(s)	outgoing label(s).
X	1	AX
Y	1 2	Y Y
Z*	3	*

The system may, thus, deliver a given packet labelled Y in one location to multiple destinations. We call the route that the packet travels from the insertion of a label to the destinations a delivery tree (and chose against using the term multicast tree, because of its IP architecture connotations).

3.2 Topology

The main purpose of the topology module is to create and maintain delivery trees used for forwarding traffic. It acts both proactively and reactively. It proactively creates delivery trees that may be needed and reactively constructs new trees, when existing delivery trees do not provide the necessary connectivity, paths break down, or the combined multicast tree is "too" suboptimal.

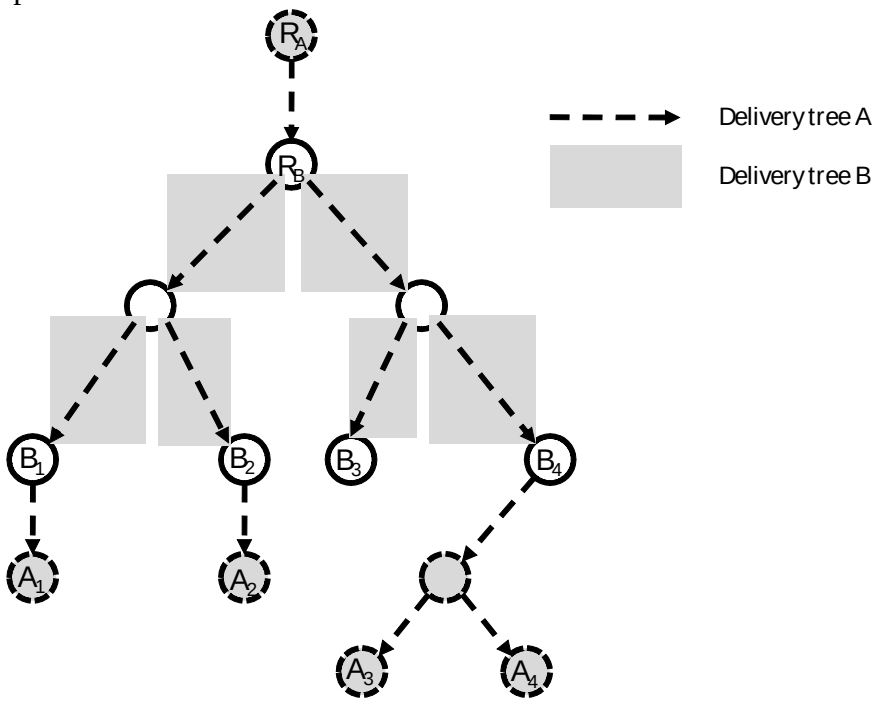

Figure 1: Combined delivery trees

The delivery trees are constructed both via individual lower layer links and by combining existing "lower layer" delivery trees. An example is illustrated in Figure 1 above, where delivery tree B is being used by delivery tree A to forward data from the tree's root node R_A to the leaves $A_1..A_4$. It can be noted that the multicast tree B leaf node B_3 also receives data sent in multicast tree A. This example presents a suboptimal solution as B_3 receives data that might not be of interest to it. This could be optimized by creating a new delivery tree C that B_3 doesn't subscribe to. However, creating new delivery trees requires new entries in the forwarding tables, and it may be more optimal resource use to use an existing suboptimal delivery tree than always creating a new optimal one.

3.3 Rendezvous

When a node subscribes to a publication, the network must first find a copy of it. The rendezvous system is a distributed structure that provides this service. It can be e.g. a Distributed Hash Table, or a semi-hierarchy, such as Data Oriented Network Architecture (DONA) [1]. It uses the distributed structure to route to a copy of the wanted data and gathers enough information on the way for the topology system to identify the delivery trees needed to forward the actual data to the subscriber.

4. Implementation

We have implemented an RTFM prototype. It is running on Linux and it is implemented completely in userspace. Everything above the link layer is implemented "from scratch", i.e. no existing (IP) protocols are being used, except mechanisms to send and receive raw Ethernet frames. The prototype consists of a library implementing the pub/sub API, a pub/sub daemon process and some test applications written in C and Java (see Figure 2).

The daemon is using libpcap and libnet to receive and send Ethernet frames. However, the source and destination fields of the Ethernet frame are ignored, and all sent frames use the broadcast address as the destination. Ethernet was chosen simply because of the existing hardware and software base. The prototype would run over any broadcast links. The goal of this first implementation has been mainly to learn the new way of thinking and it is currently completely unoptimized, being able to utilize roughly 25% of the Ethernet capacity.

4.1 Architecture

The internal architecture of a node itself is also following the pub/sub paradigm. The components of the protocol stack are not in a stack in the traditional sense, but they use a blackboard approach to access the publications. The blackboard in this case is a common directory which holds the stored publications. The directory can be a memory file system to prevent delays introduced by frequent disk accesses. In a sense, having several "managers" and caching data in an application independent manner inside the networking "stack" has similarities to the Haggle network architecture [14].

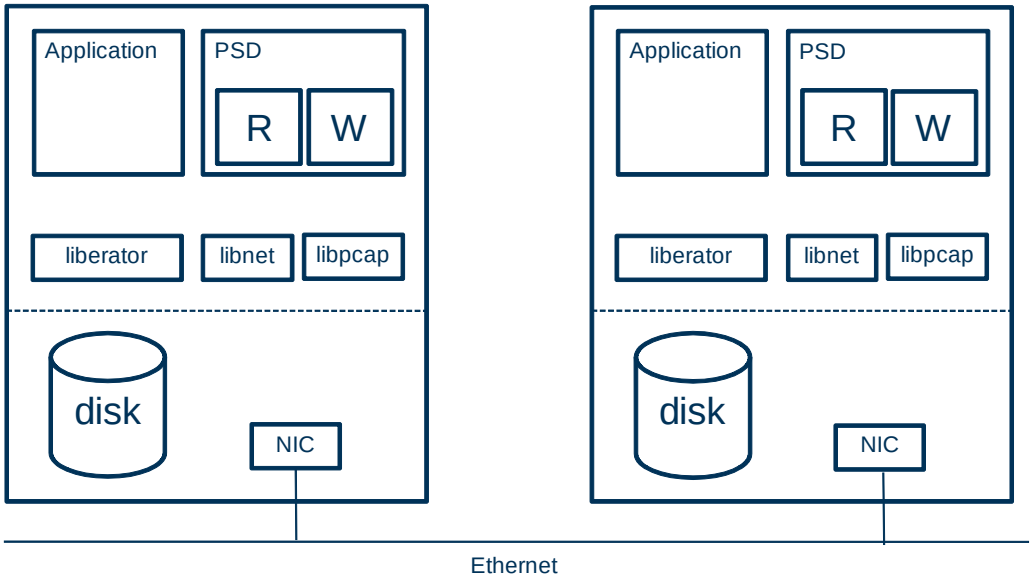

Figure 2: Prototype Architecture

4.2 Operation

When an application publishes a file, it calls a library function *create*. It creates two files per publication: one for metadata and one for the data coming from the application. It maps these files to the memory space of the calling process and the application fills in the metadata and data. The private and public identifiers are assigned by the system, and the application doesn't need to worry about them. Currently the private identifier is just a random number, and the public identifier is a hash of the former. Once the application is finished with the publication, a call to the *publish* function will bring the new publication visible to other local applications and the pub/sub daemon. Depending on the metadata, a special publication of type *publication list* may be updated. The publication list is periodically broadcast on the Ethernet link so that other nodes are aware of the available publications. This is a minimalistic rendezvous implementation.

When an application calls the *subscribe* function with the ID of the subscribed publication, the library either maps the cached copy of the publication (if it is already locally available) and returns a pointer to the caller process or, it creates a new publication

of type *subscription* into the directory. The pub/sub daemon will notice this subscription and broadcasts it to the Ethernet link(s).

When receiving a publication from the network, the daemon checks whether it is a subscription or subscribed data. If it is a subscription, it will broadcast the publication back if the publication is stored locally. If the received publication is data, it will be stored in the directory and it becomes available for the applications.

In a *sprouter* node (pub/sub router), the daemon is run with the sprouter flag set. In this case, it will also process subscriptions sent by other nodes. If the subscribed data is already cached locally at the sprouter, it will be delivered to the subscriber. Otherwise the sprouter subscribes to the data on other links and after the data has been received, it forwards the data to the original subscriber. It will also forward the broadcasted publication lists on other links, so that all nodes in the connected networks are aware of the available publications in the whole network. The nodes, however, are not aware of the presence of the sprouter; it can be seen as a combination of a transparent proxy and a switch.

The actual forwarding of a large publication may require fragmentation. An Ethernet frame can hold up to 1540 bytes of data. Our pub/sub header currently occupies 32 bytes (1 byte for publication type and 31 bytes for ID). In the current prototype the fragmentation is solved by introducing a new type of publication: *fragment list*. A fragment list publishes the IDs of publication fragments (see Figure 3). When a subscriber receives a fragment list, it needs to subscribe to all fragments. After receiving all fragments, the daemon on the receiving side can create a copy of the original publication from the fragments. There is a simple re-subscribe timer to recover from lost fragments and a simple rate limiting timer to prevent from causing too many collisions on the Ethernet. While this solution might be feasible for static files, it doesn't work with stream type connections. There's work ongoing to create a single and more efficient mechanism to support forwarding of files of any size and stream type of connections.

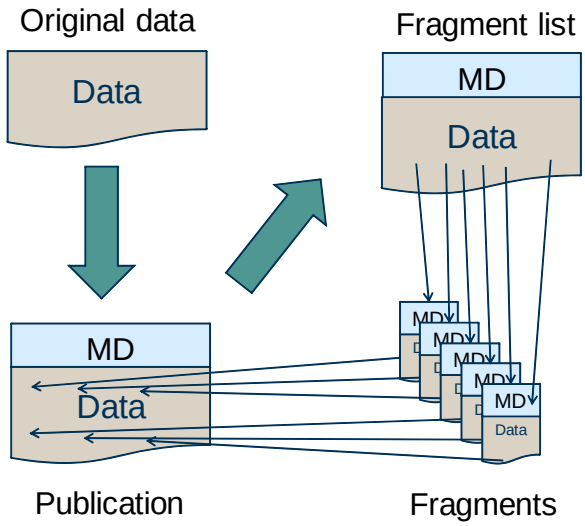

Figure 3: Fragment list

4.3 First Test Results

The prototype has been run with the following setup: two Ethernet links which are connected by a sprouter. On Link 1 there are two subscribers and on Link 2 there is one publisher (see Figure 4). All nodes are using a picture sharing application which can be used to publish and subscribe e.g. JPEG files. Currently we can publish and subscribe to a number of files and the sprouter is supporting links with different MTUs by creating separate fragment lists on each link. We also have caching function on the sprouter: once

the file has been published on Link 2 it will be forwarded to the subscribers on Link 1 from the sprouter cache – even if the original copy is lost at the publisher. Even though the current implementation is suboptimal in resource usage, it proves that supporting one of the most common type of applications is possible without identifying the senders or receivers on any network layer.

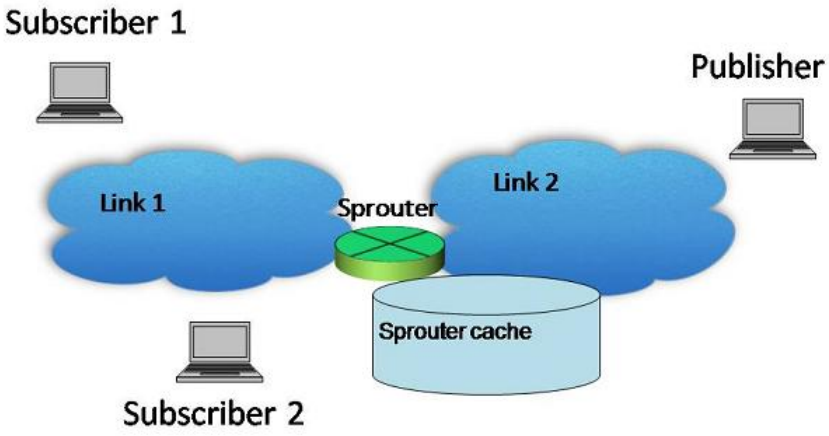

Figure 4: Prototype network

5 Conclusions

In this paper, we have considered a pub/sub inspired networking architecture on Layer 3 in the OSI stack, and presented the first prototype implementation of pub/sub based networking. The aim of this work is to move beyond overlay networks, and to be able to address many networking issues in a fundamental way, for example issues pertaining to denial-of-service attacks, unwanted traffic, and multicast.

The pub/sub paradigm turns tables on unwanted traffic and unicast forwarding, because it places more control to the subscriber and supports multicast by nature. Although pub/sub has become an active research area in networking, at the time of writing, we are not aware of any existing pub/sub research prototypes built directly on top of link layer.

In our work on the RTFM architecture and prototype implementation, we found that it is possible to create a network architecture without explicitly naming hosts, or end-points. Instead, the RTFM architecture gives names to publications and distributes the forwarding information to the network. A data object, such as a video, can be a publication. Similarly, a delivery tree can also be considered a special kind of publication, one which is used for delivering other kinds of data from one topological location to another – it is this mutable vehicle of delivery that is named rather than the end points. We leave it for future research to devise efficient means of representing a 'stream' of publications for a network element in a concise manner, while still keeping each packet a separate publication.

The key parts of the architecture are routing, forwarding, and rendezvous. The data naming system is fundamental to the approach, and we propose to utilize self-certified flat labels as the basic data naming scheme. One aspect of the pub/sub architecture, including the data-centric naming, is its ability to redefine the network topology in terms of subscribed (and announced) data. This has profound implications for intermediaries, network management, mobility, and end-to-end communications.

This paper highlights some of the key drivers for this new technology, and identifies a number of research questions. Specifically, the scalability of pub/sub networking to Internet scale is an open issue. Early experiments with our implementation demonstrate that the basic concepts are realistic for local area networks. However, the scalability of rendezvous

and topology systems is still an open question, especially as modelling inter-AS policies is not easy, and it can be expected that any new global network architecture proposal has to meet at least the current requirements for inter-AS operation. The penultimate benchmark is the BGP protocol, which has been observed to have scalability and security limitations. One motive in developing a new internetworking architecture is to address these challenges, and the cost of building and maintaining networks.

We postulate that recent results in compact routing might be useful also for alleviating pub/sub routing algorithm scalability issues in the scale of the Internet. We also observe that label-based pub/sub maps well to different label-switching technologies, such as MPLS [12] and GMPLS [13], and thus is a good match for future all-optical core networks [11]. Pub/sub appears to be also useful for mobile access networks due to its multicast and broadcast friendly nature.

In addition to wide-area scalability and inter-AS policy modelling, open issues include data caching, programming API, and network management. Moreover, the implications for different stakeholders and network economics need to be analyzed in detail. Indeed, our future work focuses on both technological and economical aspects of pub/sub Future Internet networking.

References

[1] Teemu Koponen et al. A Data-Oriented Network Architecture. ACM SIGCOMM, 2007.

[2] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In Proceedings of ACM SIGCOMM 2003, pages 163–174, Karlsruhe, Germany, Aug. 2003.

[3] D. R. Cheriton and M. Gritter. TRIAD: A New Next-Generation Internet Architecture. http://www-dsg.stanford.edu/triad/, July2000.

[4] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the Addressing Architecture. ACM SIGCOMM Computer Communication Review, pages 313–321, 2003.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. ACM Comput. Surv., 35(2):114–131, 2003.

[7] H. Schulzrinne and E. Wedlund. Application-layer mobility using sip. SIGMOBILE Mob. Comput. Commun. Rev., 4(3):47–57, 2000.

[8] Dima Krioukov, kc claffy, Kevin Fall, and Arthur Brady. On Compact Routing for the Internet. ACM SIGCOMM Computer Communication Review (CCR), Vol. 37, No. 3, 2007.

[9] A. Myers, T.S.E. Ng, H. Zhang Rethinking the Service Model: Scaling Ethernet to a Million Nodes, Proceedings of the ACM workshop HotNets-III, 2004.

[10] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. ACM SIGCOMM, August 2006.

[11] R. van Caenegem, et al. The Design of an All-Optical Packet Switching Network. IEEE Communications Magazine, November 2007, Vol. 45, No. 11.

[12] E. Rosen, A. Viswanathan, R. Callon. RFC 3031: Multiprotocol Label Switching Architecture. IETF RFC, January 2001.

[13] A. Banerjee, J. Drake, J.P. Lang, B. Turner, K. Kompella and Y. Rekhter. Generalized multiprotocol label switching: an overview of routing and management enhancements. IEEE Communications Magazine, 2001. 39(1): p. 144-50.

[14] J. Su, J. Scott, P. Hui, E. Upton, M. How Lim, C. Diot, J. Crowcroft, A. Goel, E. de Lara. Haggle: Clean-slate networking for mobile devices. University of Cambridge Computer Laboratory. UCAM-CL-TR-680. ISSN 1476-2986

[15] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default. Proceedings of the ACM workshop Hotnets-IV, 2005.

[16] M. Handley and A. Greenhalgh. Steps towards a DoS-resistant Internet architecture. Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture. 49-56, 2004.