Department of Communications and Networking

BloomCasting for Publish/Subscribe Networks

Mikko Särelä

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 49/2011

BloomCasting for Publish/Subscribe Networks

Mikko Särelä

Doctoral dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the School of Electrical Engineering for public examination and debate in Auditorium S4 at the Aalto University School of Electrical Engineering (Espoo, Finland) on the 14th of June 2011 at 3 o'clock.

Aalto University
School of Electrical Engineering
Department of Communications and Networking
Networking Technology

Supervisor
Professor Jörg Ott

Instructors
Dr. Pekka Nikander
Dr. Göran Schultz

Preliminary examiners
Dr Colin Perkins, University of Glasgow, United Kingdom
Dr. Kevin Fall, Intel Corporation, USA

Opponent
Professor Jon Crowcroft, University of Cambridge, United Kingdom

Aalto University publication series
DOCTORAL DISSERTATIONS 49/2011

ISBN 978-952-60-4149-0 (pdf)
ISBN 978-952-60-4148-3 (printed)
ISSN-L 1799-4934
ISSN 1799-4942 (pdf)
ISSN 1799-4934 (printed)

Aalto Print
Helsinki 2011

Finland

The dissertation can be read at http://lib.tkk.fi/Diss/

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Mikko Särelä

Name of the doctoral dissertation
BloomCasting for Publish/Subscribe Networks

Publisher School of Electrical Engineering

Unit Department of Communications and Networking

Series Aalto University publication series DOCTORAL DISSERTATIONS 49/2011

Field of research Networking Technology

Manuscript submitted 14 October 2010 **Manuscript revised** 17 May 2011

Date of the defence 14 June 2011 **Language** English

☐ **Monograph** ☒ **Article dissertation (summary + original articles)**

Abstract

Publish/subscribe has been proposed as a way of addressing information as the primary named entity in the network. In this thesis, we develop and explore a network architecture based on publish/subscribe primitives, based on our work on PSIRP project. Our work is divided into two areas: rendezvous and Bloomcasting, i.e.~fast Bloom filter-based forwarding architecture for source-specific multicast. Taken together these are combined as a publish/subscribe architecture, where publisher and subscriber matching is done by the rendezvous and Bloom filter-based forwarding fabric is used for multicasting the published content.

Our work on the inter-domain rendezvous shows that a combination of policy routing at edges and an overlay based on hierarchical distributed hash tables can overcome problems related to incremental deployment while keeping the stretch of queries small and that it can solve some policy related problems that arise from using distributed hash tables in inter-domain setting.

Bloom filters can cause false positives. We show that false positives can cause network anomalies, when Bloom filters are used for packet forwarding. We found three such anomalies: packet storms, packet loops, and flow duplication. They can severely disrupt the network infrastructure and be used for denial-of-service attacks against the network or target services. These security and reliability problems can be solved by using the combination three techniques. Cryptographically computed edge pair-labels ensure that an attacker cannot construct Bloom filter-based path identifiers for chosen path. Varying the Bloom filter parameters locally at each router prevents packet storms and using bit permutations on the Bloom filter locally at each router prevent accidental and malicious loops and flow duplications.

Keywords Publish-subscribe, Internetworking, Multicast, Bloom filters

ISBN (printed) 978-952-60-4148-3 **ISBN (pdf)** 978-952-60-4149-0

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Espoo **Location of printing** Helsinki **Year** 2011

Pages 171 **The dissertation can be read at** http://lib.tkk.fi/Diss/

Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä
Mikko Särelä

Väitöskirjan nimi
Bloom-suodattimiin pohjautuva viestintä julkaisija/tilaaja verkoissa

Julkaisija Sähkötekniikan korkeakoulu

Yksikkö Tietoliikenne- ja tietoverkkotekniikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 49/2011

Tutkimusala Tietoverkkotekniikka

Käsikirjoituksen pvm 14.10.2010 **Korjatun käsikirjoituksen pvm** 17.05.2011

Väitöspäivä 14.06.2011 **Kieli** Englanti

☐ **Monografia** ☒ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Yksi Internetin puutteista on se, ettei ole mitään kaikille sovelluksille yhteistä tapaa nimetä informaatiota. Julkaisija/tilaaja-malli on yksi ehdotus, jolla Internet-arkkitehtuuria voisi muuttaa tämän puutteen korvaamiseksi. Väitöskirjassani kehitän julkaisija/tilaaja-malliin pohjautuvan verkkoarkkitehtuurin, joka pohjautuu työlleni PSRIP-projektissa. Arkkitehtuuri koostuu kohtaamisjärjestelmästä, joka yhdistää julkaisijat ja tilaajat, ja Bloom-suodattimiin pohjautuvasta monen vastaanottajan viestintäkanavasta, jolla julkaistu sisältö toimitetaan tilaajille.

Internetin kattavalla kohtaamisjärjestelmällä on korkeat vaatimukset. Tutkin kahta erilaista menetelmää: paikallisiin reitityspolitiikoihin pohjautuvaa järjestelmää ja toinen hajautettuihin hajautustauluihin pohjautuvaa järjestelmää. Ensimmäisen haasteena on skaalautuvuus erityisesti silloin, kun kaikki Internetin verkot eivät osallistu järjestelmän ylläpitoon. Jälkimmäinen on ongelmallinen, sillä siihen pohjautuvat järjestelmät eivät voi taata, mitä reittiä julkaisu ja tilaus -viestit kulkevat järjestelmässä. Näin viesti saattaa kulkea myös julkaisijan tai tilaajan kilpailijan verkon kautta. Ehdotan väitöskirjassani menetelmää, joka yhdistää reunoilla politiikkaan pohjautuvan julkaisu/tilaaja reitityksen ja verkon keskellä yhdistää nämä erilliset saarekkeet hierarkista hajautettua hajautustaulua hyödyntäen.

Julkaisujen toimittamiseen tilaajille käytän Bloom-suodattimiin pohjautuvaa järjestelmää. Osoitan väitöskirjassani, että Bloom-suodattimien käyttö pakettien reitittämiseen voi aiheuttaa verkossa merkittäviä vikatilanteita, esimerkiksi pakettiräjähdyksen, silmukan, tai samaan vuohon kuuluvien pakettien moninkertaistumisen. Nämä ongelmat aiheuttavat verkolle turvallisuus- ja luotettavuusongelmia, jotka voidaan ratkaista kolmen tekniikan yhdistelmällä. Ensinnäkin, Bloom-suodattimiin laitettavat polun osia merkitsevät nimet lasketaan kryptografiaa hyödyntäen, ettei hyökkääjä kykene laskemaan Bloom-suodatinta haluamalleen polulle ilman verkon apua. Toisekseen, reitittimet määrittävät Bloom suodatinparametrit paikallisesti siten, ettei pakkettiräjähdyksiä tapahdu. Kolmannekseen, kukin reititin uudelleen järjestelee Bloom-suodattimen bitit varmistaen, ettei suodatin ole enää sama, jos paketti kulkee esimerkiksi silmukan läpi ja palaa samalle takaisin samalle reitittimelle.

Avainsanat Publish-subscribe, Internetworking, Multicast, Bloom filters

ISBN (painettu) 978-952-60-4148-3 **ISBN (pdf)** 978-952-60-4149-0

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Espoo **Painopaikka** Helsinki **Vuosi** 2011

Sivumäärä 171 **Luettavissa verkossa osoitteessa** http://lib.tkk.fi/Diss/

Preface

When I had journeyed half of our life's way,
I found myself within a shadowed forest,
for I had lost the path that does not stray.
Dante Alighieri, Divine Comedy

This thesis has been a long project of learning and maturing as a researcher, an employee, and a person. Many people have crossed my path during these years. While it is impossible to thank everyone individually, your involvement has changed me for the better and for that you have a place in my heart.

In the beginning of 2010 I started the final push to finish my thesis. Through the year, there was a race between my thesis and our baby boy who was born in the fall 2010. I managed to finish the manuscript for pre-examination mere hours before our journey to the hospital. Rea, I thank you for all the support you have given me during these years.

In 2002, as a young man, I was hired as a research assistant by professor Hannu Kari. You provided me freedom to learn, not only about my research subject, but also about philosophy of science, and many other things.

I would also like to express my gratitude to Pekka Nikander whose vision and guidance gave my research a direction and purpose. With your help I have learned a bit of both and am ready to find my own way.

My professor Jörg Ott, you have given me both space to work on my own and your valuable time and expertise. You have given me great guidance, feedback, and support when I have needed them. I have enjoyed working together with you.

I wish to thank Göran Schultz for mentoring and instructing me. You always had words of wisdom when future was hidden in shadows.

I am deeply grateful to my pre-examiners Dr. Kevin Fall and Dr. Colin

Perkins and to the countless anonymous reviewers who helped make the articles published in this thesis much better and taught me a thing or two about doing science as well.

I learned a lot in the scientist club discussions organized by Ronja Addams-Moring and in the study groups we had with Teemu Koponen, Jukka Ylitalo, and Pasi Eronen and in the private discussions with Harri HaanpŁŁ. I also wish to express my gratitude to my co-authors, Teemu Rinta-aho, Sasu Tarkoma, Dirk Trossen, Karen Sollins, Jarno Rajahalme, Pekka Nikander, Kari Visala, Janne Riihijärvi, Christian Esteve, Petri Jokela, Jukka Ylitalo, Tuomas Aura, András Zahemszky, and Jörg Ott. It has been a lot of fun working with you all.

The funding for this thesis has come from Ericsson, Aalto University, TEKES funded FI-SHOK project, and EU 7th framework funded PSIRP and PURSUIT projects.

Helsinki, May 19, 2011,

Mikko Särelä

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Mikko Särelä, Teemu Rinta-aho, Sasu Tarkoma. RTFM: Publish/subscribe internetworking architecture. In *ICT Mobile Summit*, Accepted as a poster, published on the full paper track, Stockholm, Sweden, June 2008.

II Dirk Trossen, Mikko Särelä and Karen Sollins. Arguments for an Information-Centric Internetworking Architecture. *ACM SIGCOMM Computer Communication Review*, Vol. 40, Issue 2, April 2010.

III Jarno Rajahalme, Mikko Särelä, Pekka Nikander, and Sasu Tarkoma. Incentive-Compatible Caching and Peering in Data-Oriented Networks. In *ACM ReArch'08 - Re-Architecting the Internet (ReArch'08)*, Madrid, Spain, December 2008.

IV Jarno Rajahalme, Mikko Särelä, Kari Visala, Janne Riihijärvi. On Name-Based Inter-Domain Routing. *Elsevier Computer Networks*, Volume 55, Issue 4, Pages 975-986, Special Issue on Architectures and Protocols for the Future Internet, March 2011.

V Christian Esteve Rothenberg, Petri Jokela, Pekka Nikander, Mikko Särelä, and Jukka Ylitalo. Denial-of-Service Forwarding with In-Packet Bloom Filters. In *European Conference on Computer Network Defence (EC2ND)*, Milan, Italy, Month 2009.

VI Mikko Särelä, Jörg Ott, Jukka Ylitalo. Fast inter-domain mobility with in-packet Bloom filters. In *The 5th ACM International Workshop on Mobility in the Evolving Internet Architecture (Mobiarch)*, Chicago, Il, USA, September 2010.

VII Mikko Särelä, Christian Esteve Rothenberg, Tuomas Aura, András Zahemszky, Pekka Nikander, Jörg Ott. Forwarding Anomalies in Bloom Filter Based Multicast. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)*, Shanghai, China, April 2011.

VIII Mikko Särelä, Christian Esteve Rothenberg, András Zahemszky, Pekka Nikander, Jörg Ott. BloomCast: Security in Bloom filter based multicast. In *15th Nordic Conference in Secure IT Systems (Nordsec)*, Espoo, Finland, October 2010.

Author's Contribution

Publication I: "RTFM: Publish/subscribe internetworking architecture"

RTFM: Publish/subscribe internetworking architecture is a result of an early experimentation with the PSIRP conceptual architecture. I was responsible for the design of RTFM together with Teemu Rinta-aho.

Publication II: "Arguments for an Information-Centric Internetworking Architecture"

I was a member of the *Arguments for an Information-Centric Internetworking Architecture* team. Much of the work was done collaboratively between the authors. I contributed especially to the architectural parts and the tussle arguments.

Publication III: "Incentive-Compatible Caching and Peering in Data-Oriented Networks"

The *Incentive-Compatible Caching and Peering in Data-Oriented Networks* was born out of my discussions with Jarno Rajahalme during the spring 2009 special course on Future Internet at TKK that I organized together with Pekka Nikander. I participated in analyzing the effects of Data-Oriented Networking for internet topology and routing.

Publication IV: "On Name-Based Inter-Domain Routing"

I was a co-designer of the *On Name-Based Inter-Domain Routing* and contributed especially in the problems related to inter-domain rendezvous.

Publication V: "Denial-of-Service Forwarding with In-Packet Bloom Filters"

I was a member of the *Denial-of-Service Forwarding with In-Packet Bloom Filters* design team. The design and security evaluation was done collaboratively within the team. My contributions are especially on the design and security evaluation.

Publication VI: "Fast inter-domain mobility with in-packet Bloom filters"

I was the designer of the *Fast inter-domain mobility with in-packet Bloom filters*. I was responsible for the protocol design and evaluation of the protocol.

Publication VII: "Forwarding Anomalies in Bloom Filter Based Multicast"

I was the main author and the architect for the *Forwarding Anomalies in Bloom Filter Based Multicast* work. I was responsible for the design and the theoretical analysis of the provided solutions. The empirical evaluation of the work was jointly done with other authors.

Publication VIII: "BloomCast: Security in Bloom filter based multicast"

I was the architect of the *Security in Bloom filter based multicast*. I was responsible for the design of the architecture and the security evaluation of the solutions proposed in the paper.

1. Introduction

The Internet architecture has been a tremendous success. During the past decades it has grown from a small research network into a global critical communications infrastructure with billions of users. Its success has come with rapid changes in the user base and use patterns.

While the original Internet was used by a tightly knit community with computers too large to move, the current Internet is characterized by large number of (often mobile) stakeholders with adverse and even hostile interests. These changes have made mobility [78] and denial-of-service attacks [85] some of the most studied problems in the current Internet architecture.

in 1977 Kleinrock and Kamoun showed that hierarchical naming was superior to source routing and flat naming in terms of packet header size and required routing state [67]. In 1982 Saltzer [118] defined four types of objects in networks that could be named as destinations of packets: services and users, network attachment points, nodes, and paths. The Internet architecture converged to using network attachment points as the only named destination in the network. This was useful, since network attachment points could be named in a way that enabled them to be aggregated.

Each network attachment point (e.g. network interface card) was named with a unique 32-bit IP address[1]. The address itself was originally composed of two parts. The network identifier was either 8, 16, or 24 bits long for class A, B, and C network respectively and was used by routers to route packets to the corresponding network. The host identifier (24, 16, or 8 bits long respectively) was then used at the destination to route packets to the interface that the IP address pointed to. The possibility for aggregation and the form of an IP address was crucial, since it made it

[1]Later on the introduction of NATs made this assumption invalid.

possible to build routers that could perform route lookups fast for a vast number of hosts later on.

The original IP addressing was organized as a two layer hierarchy [102]. The first part was either 8, 16, or 24 bits long and denoted the network address, while the rest was used to denote the host within the network. When networks grew, subnetting was developed [88, 89]. It brought another layer of hierarchy by allowing networks to divide the host part of the address into a local subnet part and a host part.

Later, as the Internet grew the division of the two parts was relaxed with classless inter-domain routing (CIDR) [47], so that the network part could be of any length. This enabled more efficient use of the IP address space and made it possible for the Internet to continue its growth.

There was also a need for human readable host names. This need and the growth of the early network saw the development of the HOSTS.TXT file [33, 74]. It was centrally maintained and distributed to all hosts in the Internet. Later on, as the Internet grew, Domain name system (DNS) [86, 87] was designed to improve the scalability of the naming system. However, IP addresses still remained the only routable names in the Internet architecture. Domain names needed to be resolved into one (or more) network attachment points (i.e. IP addresses), which could then be used for sending packets to the host in destination.

The early Internet saw the development of two transport protocols, transmission control protocol TCP [103] and user datagram protocol UDP [101], and the BSD 4.4 socket API [79] that became a de facto standard. The port numbers were intended as a way of denoting services and together with the IP addresses, they became the primary naming mechanism of destinations for applications.

This application layer binding of connections to IP addresses became problematic when mobility and multihoming became common. One avenue of proposed solutions suggested that hosts should have a name, either by reusing the IP address space and giving hosts a stable home address as in Mobile IP [99] or using a separate cryptographically based host identity [91].

1.1 Information networking and publish/subscribe

While the Internet was designed primarily for accessing computing facilities remotely, the patterns of communications have changed. In 2006 Van Jacobson gave his talk [60], “A New Way to Look at Networking". The main claim of the talk was that use of networks was changing from the idea of interconnecting hosts to interconnecting information - and that the network architecture should change with it. To back up his claims, he stated that “the overwhelming use ($> 99\%$ by most measurements) of today’s networks is for a machine to acquire named chunks of data (like web pages or email messages)". The quantity of services focusing on information, such as World Wide Web, peer to peer networks, and RSS feeds shows that there is indeed a shift in the network focus.

However, there is no Internet wide naming system across application platforms that would enable communication parties to directly communicate about information. Instead, applications are responsible for naming and locating the information by themselves.

Publish/subscribe has been proposed as a general solution to the problems of locating, requesting, and receiving information. Publish/subscribe systems can be divided into topic based, content based, and type based publish/subscribe. In this thesis our focus is on building a network architecture that can better support scalable topic based publish/subscribe. Traditional publish/subscribe systems implemented as overlays over the current Internet suffer from scalability problems. [40]

Topic based publish/subscribe is based on the notion of keywords to which nodes can publish or subscribe to. The publish/subscribe system then delivers documents published with a certain keyword to all subscribers for that keywork. Multicast can be considered as one way of providing such publish/subscribe communications. A keyword can be interpreted as a group, subscribe as joining to a group and publishing as sending a message to a group. Unfortunately, existing multicast architectures suffer from scalability problems. These scalability problems can be divided into scalability on the forwarding plane and scalability on the rendezvous, i.e. locating sources and receivers.

1.2 Multicast

IP multicast, proposed in the early 90s by Deering et al. [32, 31], was designed as a new general Internet layer service. In IP multicast, a new named entity, group address, was introduced to the network layer. Senders could choose a multicast group address they would send to and receivers could choose the group address they would listen to. The network would, thus, match the senders and the receivers in a general many-to-many architecture. For the many-to-many model, the group address was topologically independent. Later on, source-specific multicast [18] was designed, making a group name dependent on the source IP address in addition to the group address.

However, such multicast architectures placed the group maintenance at the routers. This requires routers to maintain per group state, which leads to additional complexity, and scalability and security problems.

Bloom filter based multicast [111, 62] has been proposed as a way to create scalable multicast by pushing the group management to the source and thus removing the per group state at the transit routers. In Bloom filter based multicast, the links belonging to a multicast tree are encoded as a Bloom filter, which is then placed in the packet header. Each router checks for the presence of outgoing links and forwards the packet accordingly. Our work builds on this work.

1.2.1 Multicast objectives

We work with the following objectives for the architecture:

Large number of groups: The architecture should support large number of groups, at least hundreds or thousands of groups per host in the Internet.

The number of potential uses for a general multicast architecture, source specific or any-source, is large. Hence, the number of potential multicast groups is also large. As an example, every chat client, RSS-feed, phone service, mobile node, etc. would potentially use the multicast architecture. Furthermore, it is likely that such an application would not be built to switch from unicast mode of communications to multicast when the number of receivers changes. Instead, it would use the multicast protocols for all communications, including those with only one (or even zero)

recipients.[2]

A multicast architecture should be designed with the potential for hundreds or thousands of simultaneous multicast sessions per host in the Internet. As there are billions of hosts in the Internet, the architecture needs to scale to trillions of simultaneous groups. This also means that many of the groups will be small most of the time, containing only one, or two (or even zero) recipients.

Source and receiver control: A source should have control over the set of receivers. A recipient should have control over groups it receives.

Security: The multicast infrastructure should be resistant to denial-of-service attacks. It should also prevent traffic injections to multicast groups by outsiders, or amplification attacks that use infrastructure to enhance the strength of a denial-of-service attack.

1.3 Rendezvous

Rendezvous is another well known problem in publish/subscribe and any-source multicast. The task is to match publishers and subscribers (or sources and receivers in case of multicast). Consider a situation in which there is a new source or receiver to a particular group. The network has to locate the corresponding receivers or sources that may reside anywhere in the network.

A number of solutions to this problem have been proposed. Simple ideas, such as using a rendezvous point [43] are not scalable to large number of groups (and large networks). Flooding and network of rendezvous points [43, 44] are not scalable to large numbers of dynamic groups either. While distributed hash tables have many useful properties, as proposed e.g. in i^3 [129] and ROFL [22], they suffer from policy incompliant routing, when used in inter-domain context. Policy routing based on names [50, 70] is a promising approach, because it piggybacks on the existing policy routing infrastructure. This means that the name queries and the actual data transfer will take the same route in the network providing fate sharing between the name routing and the actual data transfer. Additionally, the existing infrastructure provides an existence proof that policy routing can

[2]The thesis is agnostic on the issue whether a separate unicast communication mode is needed. We merely state that for communications with the potential for many end points, multicast is the natural choice even when there are only 0 or 1 recipients.

work with large inter-network with thousands of operators. However, the state requirement scales linearly to the number of groups (for some nodes in the network), which may be problematic in the case of incremental deployment.

1.4 PSIRP architecture

This thesis was born from our work on the PSIRP project [104]. PSIRP is a publish subscribe internet routing paradigm project aiming for a clean slate internetworking architecture based on publish/subscribe principles. Its purpose is to design a new information centric internetworking architecture. The architecture is designed to operate with publish/subscribe primitives at all networking layers.

The PSIRP architecture is based on the logical separation of rendezvous, topology management, and forwarding functions. The role of the rendezvous is to match publishers and subscribers, while the topology management is responsible for the creation of multicast trees from the publisher to a set of subscribers. Bloom filter based multicast [62] is used to create a fast multicast based forwarding plane. In this thesis, we use the IP routing and forwarding as the topology layer for our publish/subscribe architecture.

1.5 Contributions

In this thesis, we create an instantiation of the PSIRP architecture by combining BloomCasting, a technique for Bloom filter based inter-domain multicasting, and the rendezvous system with current IP architecture. Our work on a incrementally deployable rendezvous architecture is a combination of policy routing on the edges and hierarchical overlay between islands of deployment. It forms the basis for the PSIRP rendezvous architecture. BloomCasting is used as a secure and scalable technique for multicasting and IP unicast is used for signaling purposes. In our architecture, anycast is used in the rendezvous for locating (information) objects. Unicast is used for signaling and path collection and Bloom filter-based multicast for all data traffic.

One of our goals is to take the clean slate design from PSIRP and find a

way to fit it together with the current Internet architecture. During the course of this work, we show that such a combination can be used to solve or mitigate a number of architectural issues in the current Internet, including mobility, multicast, and resistance to distributed denial-of-service attacks.

We show that Bloom filter based forwarding suffers from anomalies that have severe effects on infrastructure reliability and security, namely packet storms, forwarding loops, and flow duplications. We show that the problems can be solved by varying the parameters of the Bloom filters locally in each router and by permuting the Bloom filters at each router. We also propose a technique for creating Bloom filters that are cryptographically secure and can be used as capabilities.

1.6 Structure of this thesis

In the next chapter, we present the necessary background for our contributions. We continue by presenting BloomCast, our receiver driven network architecture and conclusions in the subsequent chapters, followed by the published articles.

2. Background

In this section, we cover the relevant related work on multicast, Bloom filters, distributed denial-of-service security, mobility, and rendezvous. In the end of the Section, we also provide a brief summary of two security techniques used in this thesis: hash chains and bit permutations.

2.1 Multicast

Multicast [32, 31] was originally designed as a general network layer service enabling anyone to send to and receive from any group. A simple example network using any-source multicast is shown in Figure 2.1. Two sources S_1 and S_2 both send data to group G. A set of receivers R receive traffic from group G. Each router maintains forwarding state for every group it needs to forward, and as shown in the illustration in some cases a router needs to maintain group specific state per source. The table on the left shows that all multicast packets to group G $(*, G)$ are forwarded to router 2 and multicast packets (S_1, G) are forwarded to routers 3 and 4.

One of the key questions in the any-source multicast (ASM) model is: How does the network locate all sources and receivers of a given group. For example, if R_3 started sending to group G, how will the routers know where to forward the packets. Many different multicast protocols and architectures have been proposed (see e.g. [35] for a survey) There are two main models: dense mode and sparse mode multicast.

Dense mode multicast, e.g. Protocol Independent Multicast - Dense Mode PIM-DM [1] floods the network when a new source sends to a group and relies on the routers to prune unneeded links afterwards. It, however, creates group specific state to every router in the network and cannot scale to a large number of groups.

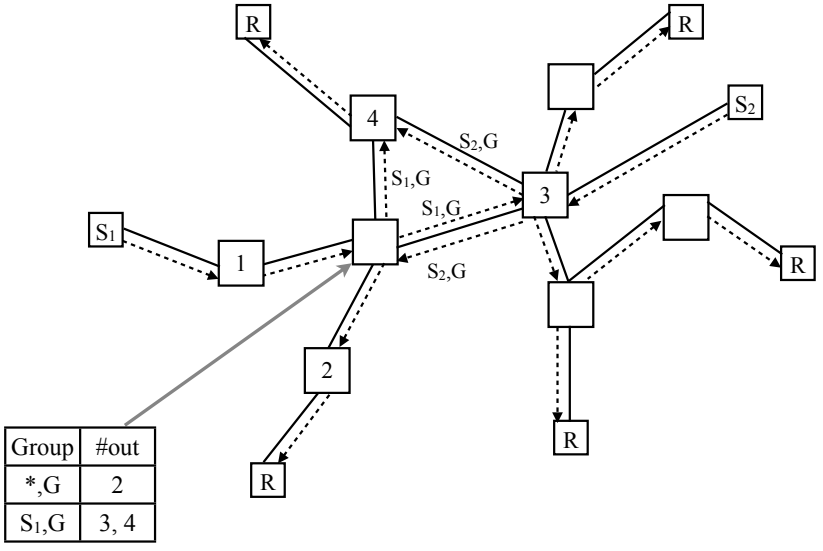


Figure 2.1. (a) Example of any-source multicast with two sources S_1 and S_2 both sending to group G.

A sparse mode multicast protocol uses a rendezvous point (or an equivalent) to act as an intermediary between sources and destinations. In Protocol Independent Multicast - Sparse Mode (PIM-SM) [43, 17, 66, 44, 41] the designated router of a receiver sends a join to the rendezvous point of that group. Originally, a source's packets are tunneled to the rendezvous point. After this, in two phases the trees can be pruned so that there is a shortest path tree from the source to the set of receivers. First, the rendezvous point sends a join toward the source, removing the need for tunneling and then a router on the receiver network can send a source specific join to the source, creating a shortest path route between source and receiver.

Core-Based Trees [14, 94] form a single tree with a core-router as the root of the tree. Joins are forwarded towards the core until a router with group specific state is found and each router creates group specific state on when it passes the join forward. When a packet is sent to a group by a group member, the packet is forwarded using the tree rooted at the core-router so that each router forwards the packet to all interfaces that are marked as receivers for the group excluding the interface were the packet came from. If the sender is not part of the group, the packet is tunneled to the core router. This setup saves state compared to PIM-SM, but still requires state in a router for every group that it belongs to.

The main benefit of ASM is that it provides a general many-to-many

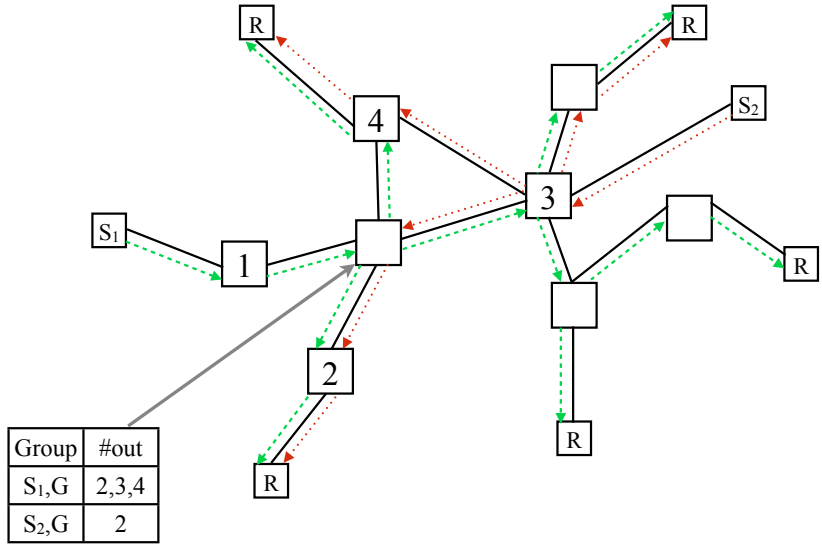


Figure 2.2. (a) Example of source specific multicast with two sources S_1 and S_2 sending to group (S_1, G) and (S_2, G) respectively. Each has its own set of receivers.

communications architecture. However, the address space is too small for large scale use and, hence, requires complicated distributed multicast address space management. It also suffers from many other deployment problems [34], especially in the case of inter-domain use. These include the combined requirement for per group state in routers and no control over the routing state, which causes severe security problems, and insufficient tools for group management in general.[1]

As an example of such problems, anyone can send to or join into an arbitrary group address. Sending to or joining into an empty group has been noted as a difficult security problem [119] that creates a denial-of-service vulnerability since the multicast (rendezvous) architecture must search the entire network for potential receivers or senders.

2.1.1 Source specific multicast

Source specific multicast (e.g. Protocol Independent Multicast - Source Specific Multicast PIM-SSM [18]) provides a solution to the multicast address allocation and rendezvous problem by using the combination of the

[1]We understand that many of the multicast deployment problems are fundamentally economic in their nature, such as the lack of control for costs in transit (state) or source (transmission). We believe that our architecture does offer a better design in such terms as well. However, such considerations are beyond the scope of this thesis.

IP source address with the multicast address as the group address. Only the host residing at the source address can send to the group and joins are routed towards the source using IP forwarding (MBGP [16] forwarding tables can also be used). When the join packet is forwarded, each router creates forwarding state for the group and forwards the packet towards the source until a router with group specific state for that group is found. Figure 2.2 shows an example network using source-specific multicast.

Source specific multicast is a step toward a deployable multicast architecture. It resolves the group address management problem, allowing each host with an IP address to be a source for millions of groups. However, the security problems related to per group state in routers still remain [119].

XCast [20] is a source-specific multicast protocol that uses unicast routing directly. The set of destination IP addresses is placed in the packet header and routers route the packet using unicast routing to all destination IP addresses. This, however, leads to additional forwarding complexity because routers need to parse variable length sets of destination addresses and, in branching points, re-construct the headers with a subset of these addresses.

Scalable and adaptable multicast [145] improves PIM-SSM scalability by combining it with XCast. Only a subset of multicast routers are chosen to maintain state for a given multicast group and packets are routed from one stateful multicast router to the set of next multicast routers using XCast. This reduces the state requirement per router, but has the same problems as described in XCast.

2.1.2 Multicast Security

While the end-to-end security considerations of multicast have received considerable attention (see e.g. [93, 24, 53, 98, 123, 64, 106, 54]), relatively little has been written about the security of the network multicast architecture: RFC 4609 [119] lists issues such as join flooding, sending to empty groups, and disturbing existing groups by sending to them, that are security problems for multicast architecture.

A few factors complicate multicast security compared to unicast. First, the number of entities in communications is larger, which creates additional complexity that manifests itself as security problems in group key management, source authentication [13, 63], and receiver authentication,

etc. This also means that there is natural multiplication of traffic, which can be useful for denial-of-service attacks.

Second, the multicast architectures described above are based on the stateful multicast model. This model places group management in the network, making it difficult for the source to control who can or who cannot receive. This combination forces the network infrastructure to handle problems that would be better left to the application, such as source and receiver authentication. Application requirements for authentication may differ.

Third, while hierarchical routing can help unicast routing to scale with sub-linear growth [67], multicast group identifiers are topology independent, and the number of potential multicast groups grows exponentially with the number of receiver nodes in the network. There have been attempts at helping multicast forwarding plane scale better by aggregating multiple multicast groups into smaller number of trees (see e.g. [105, 130]).

There have also been attempts to solve problems related to multicast using overlay architectures, such as secure-i^3 [2]. While distributed hash tables spread load efficiently across the system, they lack policy-compliant paths and control over who is responsible for a particular group.

2.2 Bloom filter-based forwarding

Bloom filter-based source routing has been suggested as a solution for multicast forwarding [111, 62, 131, 149]. Using Bloom filters to encode the multicast forwarding tree into the packet removes the need for the routers to maintain state per multicast tree. This solves an important scalability problem in multicast and can be used to enhance the control receivers and senders have over multicast communications. By decoupling group management from multicast routing, it also makes it possible for sources to deploy different end-to-end group management solutions, depending on need.

We next introduce Bloom filters and then go through the relevant related work in using in-packet Bloom filters for packet forwarding.

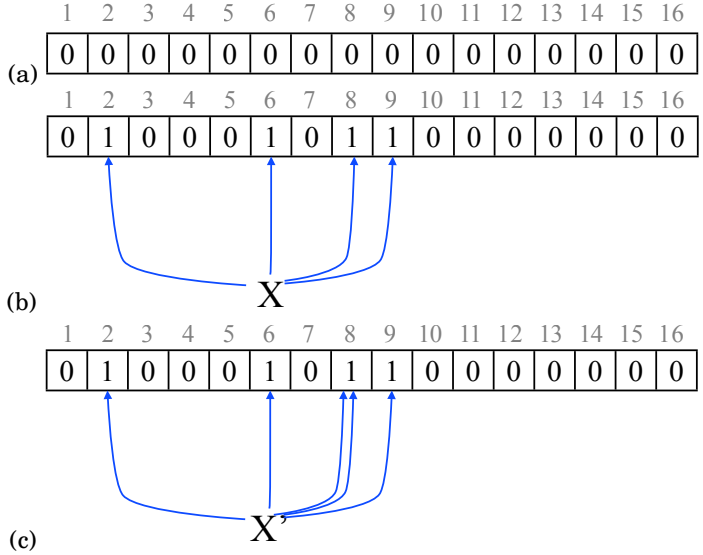


Figure 2.3. (a) Empty Bloom filter (b) X is added to Bloom filter by setting the hash value array positions to 1, $k = 4$. (c) shows a hash collision. Two hashes for element X' both yield location 8, $k = 5$.

2.2.1 Bloom filters

A Bloom filter [19] is a probabilistic data structure. It is an m-bit long array and initially all bits are set to 0 as shown in Figure 2.3 (a). Elements can be added into a Bloom filter by computing a set of array positions in the Bloom filter that are set to 1, as shown in Figure 2.3 (b). The presence of an element is tested by checking if those array positions are set to one.

This means that new elements can always be inserted into a Bloom filter, but no elements can be deleted. False positives are possible, i.e. a membership test can return positive even if the element has not been added to the Bloom filter. However, false negatives are not possible.

Each element e is represented with k positions in the array. For example k separate hash functions can be used to compute k array positions - each hash function giving output $[0, m - 1]$. The element can be encoded as an m-bit long vector in which the array positions denoted by the k hash values are set to 1 as shown in Figure 2.3 (b). Two hash values can collide, as shown in Figure 2.3 (c) bit 8. As the figure shows, $k = 5$ hash values are inserted into the Bloom filter, but only 4 array positions are marked to one.

An element can be added to a Bloom filter by bitwise ORing the element's m bit vector together with the Bloom filter. The presence of an

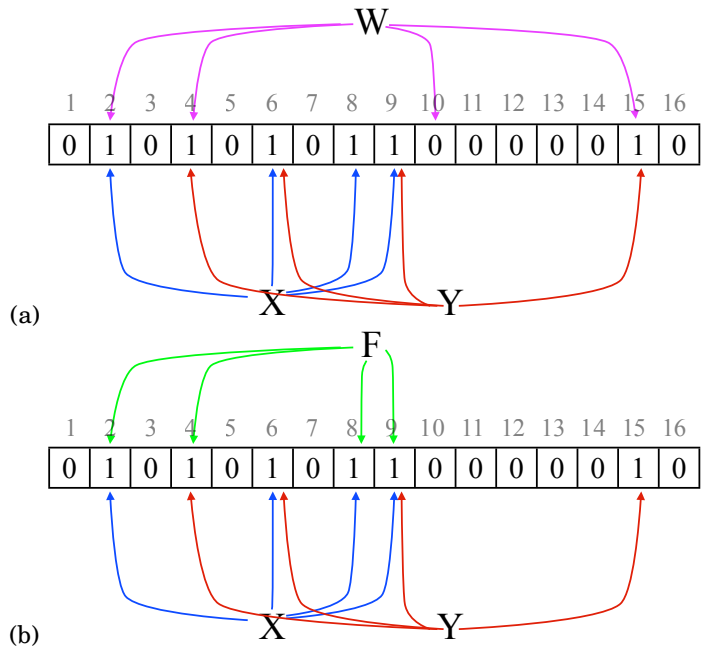


Figure 2.4. (a) Shows a Bloom filter to which elements X and Y have been added. The corresponding array positions denoted by the blue and red arrows have been set to 1. The element W is not in the Bloom filter, since the bit in array position 10 is 0. (b) Shows the same Bloom filter and element F that has not been added to the Bloom filter. However, the test for membership indicates that F has been added, since all the corresponding array positions are set to 1. F is a false positive

element is tested by checking if the k array positions are set to one. This can be efficiently done with $F \in B : m \wedge e = e?$, where m is the Bloom filter and e the tested element in m-bit long form.

Figure 2.4 (a) shows a Bloom filter after elements X and Y have been added to it. The membership of an element, such as W, is tested by checking if each array position set to 1 in W is also set to 1 in the Bloom filter. The membership testing for W shows that W is not a member in the filter, since the bit in array position 10 is set to 0.

When an element has not been added to the Bloom filter, but the array positions of the element are set to 1, a false positive happens. As an example, Figure 2.4 (b) shows a false positive. Only two elements, X and Y, have been added to the Bloom filter. F is denoted by the array positions $\{2, 4, 8, 9\}$, which have all been set to 1 due to X and Y. Hence, membership testing will indicate that F is in the Bloom filter, while it has, in fact, not been added.

False positive probability fpr is the probability that a membership test for an element will return true for an element not added to the Bloom

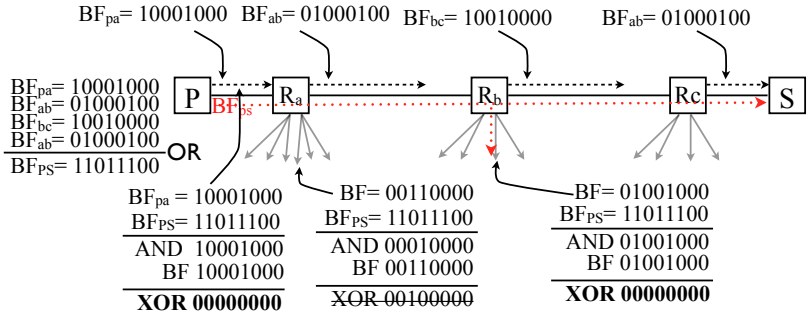


Figure 2.5. Bloom filter based forwarding. Each link has a directional identifier, e.g. BF_{ab}. BF_{PS} on the left shows the construction of a Bloom filter for path from P to S. The other columns show how each router tests the presence of a link in the Bloom filter. Red dotted line shows the path that the packet is forwarded.

filter. The probability of a false positive is $fp = \rho^k$, where k is the number of hashes used. Fill factor ρ is the proportion of array positions set to 1 in the Bloom filter, e.g. the fill factor in the Figures 2.4 (a) and (b) is $\rho = 6/16 = 0.375$, since 6 array positions are set to 1 out of 16.

Three variables that affect the fill factor are the number of elements n, the number of hashes used k, and the length of the Bloom filter m. Choosing these three variables well, results in a Bloom filter that has a fill factor of ≈ 0.5. Intuitively, having a filter with approximately 50% of array positions set to 1 maximizes entropy and minimizes redundancy in the Bloom filter[2].

2.2.2 Multicast forwarding with Bloom filters

To encode a multicast tree as a Bloom filter, each link on the forwarding tree is locally named as a Bloom filter element. These links are then inserted into the Bloom filter, as shown in Figure 2.5. The resulting Bloom filter is placed into the packet header making the Bloom filter a compact representation of the source tree from source to the set of receives. The routers forward the packet by checking which outgoing links are included in the Bloom filter, i.e. for each outgoing link a router makes a membership test.

Each router names all its links locally with an m-bit long string with k bits set to one[3], with $k \ll m$.

[2]Not every Bloom filter with fill factor 0.5 represents a good choice of values.

[3]We assume that each of the k bits is in a random position. Collisions are allowed. Hence, it is possible that the resulting string has less than k bits set to 1.

The length (m) of the Bloom filter is constrained by the available size. The longer the filter, the larger the packet overhead is from using Bloom filters, but also more links can be encoded to a single Bloom filter. As an example, with $m = 256$,i.e. 32 bytes and $k = 5$, $\approx \binom{m}{k} \approx 5 \cdot 10^{12}$ different link identifiers.

A Bloom filter for a multicast tree is formed by bitwise ORing the links together, as shown in Figure 3.3. For practical and security reasons, explained in detail below, it is useful to enforce a maximum fill factor, e.g. slightly above 50%. If the number of links is greater than can be inserted into the Bloom filter without exceeding the fill factor, at least three possible solutions exist. In this thesisi we use the second, i.e. splitting.

(1) Using a longer Bloom filter. The capacity of Bloom filter, given a target false positive probability and k grows linearly with the size of the Bloom filter. Hence, by doubling the size of a Bloom filter, it is possible to approximately double the number of links the Bloom filter can contain. (2) Splitting the multicast tree into subtrees that are small enough to fit [111]. Arbitrarily large multicast trees can be supported by splitting them into several separately encoded multicast trees, each with its own Bloom filter. For large groups, this approach may be needed. The source will have to send a copy of every packet separately for each group. (3) Using virtual links [62]. It is possible to encode a path or a tree in the network so that it will be treated as a single link for the Bloom filter. As an example, an MPLS [115] path or tree could be used as a single virtual link in a Bloom filter.

The source needs to acquire the information required for encoding the multicast tree into a Bloom filter. There are at least the following three approaches and in this thesis we use the third, i.e.collecting the Bloom filter into the signaling message.

(1) A centralized topology manager (such as PCE [42]). A topology manager collects the link identifiers in addition to the routing and service related information. When a path or a tree is needed, a router sends a request to the topology manager which first computes the path or the tree and then the Bloom filter for it. Off path computation of the route means less work for the routers, but more for the centralized element. It also means that path computation doesn't share fate with the actual path taken by the packets.

(2) Adding the required functionality to link state (e.g. OSPF [92]) or path vector routing protocols (such as BGPv4 [113]) enables routers to do

the Bloom filter computation by themselves without any additional signaling. The link identifiers can either be computed from well known identifiers, e.g. IP addresses or AS number, or the functionality of signaling the link identifiers can be added to the routing protocol.

Using link state or path vector routing protocol to compute the Bloom filter at the source, as is done in FRM [111], requires that the local routing protocol state truly reflects the current state of the network. With Bloom filter based forwarding, stale state means that the packets will be lost, compared to IP hop-by-hop forwarding, in which no router needs to have a complete and up-to-date view of the whole path. Additionally, the use of path aggregation (common in path vector protocols) breaks the path computation, since no exact path information is available. Otherwise, the source will not have the required information to determine the necessary path information.

(3) The required information can also be collected hop-by-hop with a signaling message [149]. Collecting the required information forwarding information on path is relatively fast as it uses the same path as the actual data traffic and the signaling message can be piggy-bagged in e.g. connection initialization message. However, a separate forwarding protocol for signaling messages is required. Symmetric paths can be setup with a single signaling message that collects a bi-directional Bloom filter for the path. While, in the inter-domain case, this may violate some existing local routing policies, it also ensures fate sharing between both directions of traffic. Creating asymmetric paths based on existing local routing policies is also possible, but requires additional signaling.

In this thesis, we use a signaling message that collects the required path information into a Bloom filter. IP forwarding is used for forwarding the signaling packet. The process of collecting a Bloom filter is explained in more detail in Section 3.

When Bloom filters are used for packet forwarding, measuring false positive probability can be misleading. This is because every router that receives a packet, tests for the presence of all its links (except the incoming link) and the degree of routers varies. Instead, we propose that false positive rate fpr should be measured. False positive rate is the average number of false positives per packet in a given router $fpr = \rho^k \cdot (d - b - 1)$, where d is the degree of the node, i.e. the number of neighbors the router has, b is the number of branches in the multicast tree at the router, and -1 removes the incoming link. If a router received the packet due to false

positive, the false positive rate is $fpr = \rho^k \cdot (d-1)$.

As an example, consider three routers with degree 4, 64, and 1024, a Bloom filter with $m = 256$, $k = 5$, and $\rho = 0.5$. The false positive probability is $2^{-5} \approx 0.03$. The false positive rates are ≈ 0.13, 2, and 32 false positives per packet, respectively.

2.2.3 Related work on Bloom filters

FRM [111] is an inter-domain any-source multicast architecture that separates group membership management and forwarding. Each AS encodes the multicast groups it wants to receive into a Bloom filter that is communicated to other ASes in BGP advertisements. With this, each AS knows the AS receivers for all multicast groups. A source AS computes the forwarding tree from the BGP routing table and encodes it as a Bloom filter, where the link identifiers are computed directly from the AS numbers. The approach is problematic, partly because it requires all ASes to have complete and up-to-date view of the Internet topology, and partly because the link identifiers are public, which causes security problems discussed in the next Section.

LIPSIN [62] proposes a Bloom filter based forwarding method for PSIRP, a publish subscribe based internet, architecture [29, 28, 30]. The architecture is divided into three parts: rendezvous, topology, and forwarding, and explained in more detail in Section 3. LIPSIN develops the Bloom filter based multicast forwarding fabric for the PSIRP architecture.

MPSS [149] is a technique for combining MPLS with Bloom filter based forwarding. The resulting design enables zero signaling path configuration and removes required per path or tree state in provider routers.

AOM-based Bloom filter forwarding [131] uses the IP forwarding information base to re-encode the Bloom filter after each forwarding hop. While it ensures loop freeness, there is a heavy per-packet processing overhead for routers due to Bloom filter re-encoding, which requires longest prefix matching for each IP address encoded into the Bloom filter.

A number of extensions to Bloom filters have been proposed over the years. Variable-length Signatures [81] allows partial signatures, where only $q \leq k$ bits need to be set. Popularity Conscious Bloom Filters [150] varies the number of hash functions as a function of the data item popularity to reduce the average number of false positives. This is similar to our idea of varying the k locally in each router according to its degree (see

Section 3.5. Recently, deletable Bloom filters [116] have been proposed as a memory efficient add-on to allow probabilistic element deletions, enabling for instance to remove already processed outgoing links as the in-packet Bloom filter traverses the network. They may provide a useful addition to the techniques described in our work. However, further work is required to study how they can be combined with the bit permutation technique we use for loop prevention, see Section 3.5.

2.2.4 Security and reliability issues in Bloom filter based forwarding

In existing solutions [111, 62, 149], link identifiers are static and a Bloom filter is, hence, defined per multicast tree. Each router that receives a packet will test the Bloom filter for all its outgoing links. A false positive causes the packet to be forwarded to a neighbor that was not an intended recipient, which will then test the Bloom filter for all its outgoing links. False positives cause forwarding anomalies that affect both network reliability and network availability. We identified three forwarding anomalies, namely packet storms, forwarding loops, and flow duplication Publication VII.

Packet storms: In a network with unicast communication and constant node degree d, i.e. each node has d neighbors, the bandwidth overhead caused by false positives is $\frac{(d-2)\cdot fpr}{1-(d-1)\cdot fpr}$, i.e. the percentage of false positive links traversed per a single actual link in the encoded path. Table 2.1 shows some of these values for a 256-bit Bloom filter. The bandwidth overheads caused by false positive packets have been calculated for constant node degree $d = 10$ and $d = 20$. As can be seen from the column $d = 20$, the bandwidth overhead can grow to infinity.

If the false-positive rate of a Bloom filter exceeds one, $fpr = (d-1) \cdot \rho^k \geq 1$, false positives will cause an unlimited traffic explosion in the network. This is because each false positive will, on the average, cause more than one false positive in the next forwarding node. A malicious node could utilize this property to attack the network infrastructure, causing a potentially devastating denial-of-service attack. Theoretically, when we limit the filter fill factor to $\rho \leq 50\%$, the network will be stable only if $(d-1) \cdot 2^{-k} < 1$. (Note that, excluding the ingress link, there are $d-1$ potential outbound links at each forwarding node.)

Let $k = 5$ and consider high-degree core nodes in the network. For example, if a node has 1000 neighbors, a packet routed to it will cause,

Table 2.1. False positive rate, maximum multicast tree size and bandwidth overhead as functions of k (Bloom filter length $m = 256$, fill factor $\rho = 50\%$ and network node degree $d = 10$ and $d = 20$).

		Max tree	Bandwidth overhead	
k	fpr	size	$d = 10$	$d = 20$
4	6.3 %	44	114.3 %	∞
5	3.1 %	35	34.8 %	138.5 %
6	1.6 %	29	14.5 %	40.0 %
7	0.8 %	25	6.7 %	16.5 %
8	0.4 %	22	3.2 %	7.6 %
9	0.2 %	19	1.6 %	3.7 %
10	0.1 %	17	0.8 %	1.8 %

on the average, over 30 false positives to be sent out. Moreover, every false positive received by such a high-degree node will cause another over 30 copies of it to be sent. This kind of amplification would cause a packet storm in the highly connected core part of the network.

While practical network topologies have few very-high-degree nodes, inter-domain routing in the Internet has enough of them to be vulnerable to such an instability. There are hundreds of ASes with 32 or more peering relations[4].

Forwarding loops: In addition to the bandwidth overhead, the literature identifies forwarding loops as another potential consequence of the false positives. A loop can arise from a single false positive that causes a packet to be sent back to a node which it has already traversed, as shown in Figure 2.6. Even rare occurrences of such loops can severely disrupt a network since every packet in the afflicted flow gets stuck in the loop until its TTL is zero. In addition to the packets congesting the loop, a copy of each packet will be sent to the intended downstream tree every time it goes around the loop. A malicious sender may intentionally construct packets that contain such infinite loops. Assume that the sender has a Bloom filter to a target host or network. It only needs to add few more links to the existing Bloom filter to create a loop somewhere on the path toward the target.

FRM [38] found that false positive rate above 0.2% causes a sharp decline in bandwidth efficiency due to forwarding loops. LIPSIN [62] proposes several solutions: cached state in the routers, a TTL field in the

[4]See e.g. Caida Data set: as-rel.20091215.a0.01000.txt, http://as-rank.caida.org/

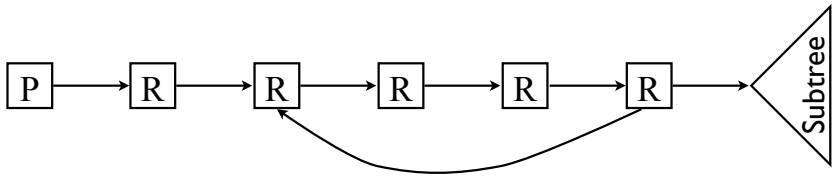


Figure 2.6. Forwarding loop with Bloom filters causes a copy of each packet to be sent to the subtree, each time a packet goes through the loop.

packets, and valley-free network topology. These ideas have major limitations. The cached state would make the protocol less scalable with the number of flows and, thus, also create another denial-of-service vulnerability. The TTL field would end the loop after a finite number of rounds. That might suffice for loops that occur accidentally with low probability. In DoS attacks, however, even a small number of rounds in the loop can create significant traffic amplification to the downstream tree. Valley-free [49] networks prevent loops in theory[5] but are vulnerable to another routing anomaly that will be presented next.

Flow duplication: Loops are not the only anomaly that can arise because of the false positives in probabilistic packet forwarding. Figure 2.7(a) shows how a false positive can cause a packet to be forwarded for a second time to a subtree even though the forwarding topology is loop-free.

Figures 2.7(b)-(c) presents rather artificial constructions in which the number of packets grows according to the Fibonacci sequence and as powers of two. The numbers indicate the number of copies of each packet that traverse the link. These examples are important for two reasons. First, they shows that exponential growth is possible even when packets have low TTL values and when the valley-free forwarding rule is observed.[6] Second, more importantly, such extremely anomalous cases could be constructed by malicious senders. While accidental flow duplication is clearly not as serious a problem as accidental forwarding loops, the duplication becomes an important consideration when we consider secure ways of preventing intentional attacks.

Targeted attacks

We evaluated the security of the basic Bloom filter forwarding approach in Publication V. Bloom filter forwarding suffers from three vulnerabil-

[5]In practice even valley free networks experience loops e.g. due to misconfiguration and routing dynamics.

[6]The exponent is limited by the number of tiers in the routing architecture.

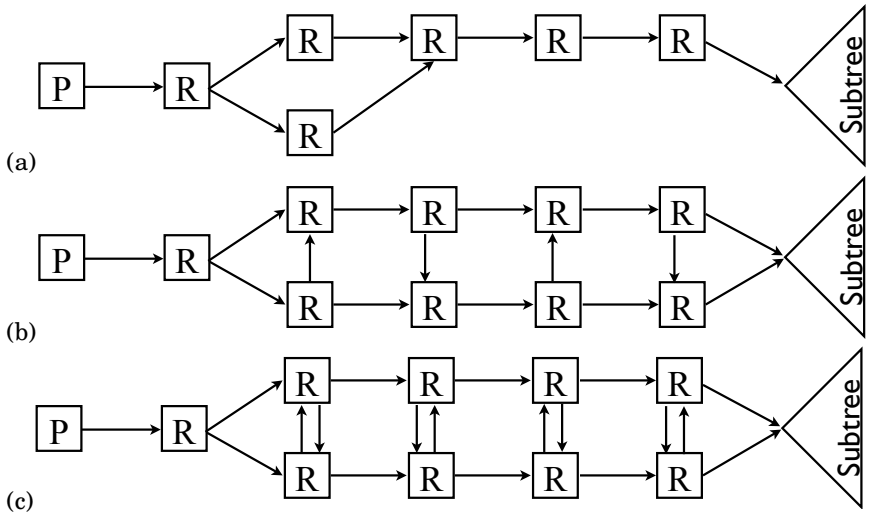


Figure 2.7. Flow duplication. (a) Shows a single duplicate link, which causes the duplication of every packet in the packet flow to the whole subtree. (b) and (c) show how duplication links can cause the number of duplicate packets to grow (b) in Fibonacci numbers and (c) exponentially to the number of links on the shortest path.

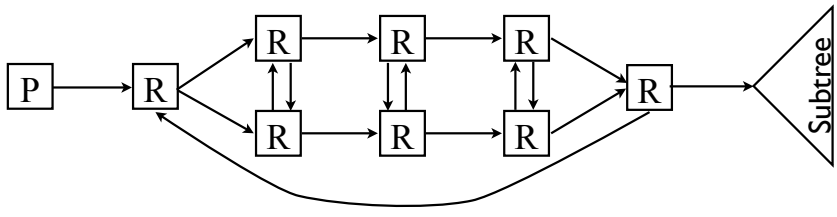


Figure 2.8. Flow duplication and loop can be combined for devastating effect on network. With each loop the number of packets flowing in the network and towards the subtree grows exponentially.

ities: Bloom filter replay attack, computational attack, and injection attack.

First, while a given Bloom filter works only from its source to its sink(s), the same Bloom filter can be used also for traffic other than it was meant for. We call this a *Bloom filter replay attack*. Second, while the used encoding helps to hide the link identifiers, correlation between Bloom filters is still possible, creating a *computational attack*; see below. Third, while each Bloom filter is directly usable only by the source and any *en-route* nodes, if an attacker can figure out another Bloom filter that passes through any of the en-route nodes, it can *inject* traffic to the delivery tree.

In the computational attack, an attacker collects valid, related Bloom

filters and analyses them. Wherever the bit patterns are similar among a group of Bloom filters, it is likely that any reoccurring bits represent a partial graph common to those Bloom filters. Hence, knowledge over a large number of ⟨source, sink(s), Bloom filter⟩ triples may allow an attacker to create valid Bloom filters towards a target. By merging correlation pairs from multiple sites (e.g., using bots), DDoS attacks might well be possible. While the introduction of the LIT construction makes this attack computationally more expensive, especially when d is large, the attack appears to remain practical.

2.3 Distributed denial of service attacks

The aim of a distributed denial-of-service (DDoS) attack is to deny target service to and from the Internet by creating extreme levels of congestion. One could compare this to a large group of truckers or tractor drivers congregating in all the highways towards a major city[7], thus blocking access to it. The difference is that a single attacker could hijack the automated driving computers of all those cars and instruct them to do it from the other side of the world. The basic methods mitigating DDoS attacks are (i) traffic classification and filtering, (ii) replication, and (iii) hiding.

2.3.1 Classification and filtering

Packet classification methods strive for a way to order packets in such a way that good traffic gets service first and bad traffic gets filtered out. Classification (and hence filtering) can either be proactive or reactive and can be placed either in (some elements in) the network or in the packet header.

Capabilities

The capability approach divides the network traffic into packets requesting permission to send and packets with a permission (capability). Routers reserve only a small amount of bandwidth (e.g. 5%) for capability requests to the server. The capability can be constructed in several ways. The design choice affects the amount of processing and state required in routers, and the additional state required in packet headers.

The capability can be originated by the server [5], e.g. a value in a hash

[7] Such things happen, e.g. french farmers blocked access to Paris by filling the highways http://abcnews.go.com/Business/wireStory?id=10486291

chain. This approach requires verification points in the network to store flow specific state. Another possibility is to collect the capability hop-by-hop into the capability request packet [144, 147, 146], which removes the need for flow specific state, but increases the size of the capability. Fastpass [139, 140] proposes a system, in which the server can delegate the right to provide capabilities to third parties enabling massive replication of request-for-capability service. End user involvement [132] and other methods of access control can also be used.

Phalanx [36] combines capabilities with an overlay, hiding the recipients from senders. It utilizes a set of mailboxes through which traffic must traverse to be delivered to the recipient. The ISP builds a filtering ring around its perimeter that blocks traffic that does not comply with this requirement. The mailboxes receive traffic from the client, and the recipient requests these packets explicitly from the mailboxes. In addition, Phalanx utilizes multiple paths to reduce effects of an attack on a single mailbox.

Argyraki et al. argue that capability based systems have a flaw called *denial of capability (DoC)* [6]. An attacker can flood the request channel and thus deny legitimate users the chance of getting a capability. Puzzles have been shown to mitigate denial of capability attacks [137, 138, 96]. The case for public work [27] proposes a public work function that can be verified by anyone.

Other proactive classification and filtering methods include creating a congestion market [21] by using explicit congestion notification [109] re-feedback. Using symmetry of packet flows, typical in legitimate traffic [72], allows automatic detection of harmful flows and can potentially be enforced in NICs. An even more radical proposal is to create a network architecture where the default is off [12] and hosts need an explicit permission to send. A Bloom filter can be used to efficiently encode and spread lists of who is allowed to send to whom. For scalability, return traffic is routed using source routing and reverse path. However, the connection setup times tend to be long, as it takes tens of seconds for filters to propagate in an Internet sized network.

DDoS defense by offense [133, 135] uses the existing fair queueing in routers to implement filtering by encouraging good clients to send more traffic (increasing congestion). Puzzle auctions [136] can be used for filtering request and leveling playing field between attackers and legitimate senders. In a similar approach, decongestion control [107] is a method in

which hosts always send at full rate and vary the coding of the data in packets as a response to congestion, instead of varying sending rate.

There are also proactive overlay approaches that attempt to protect the target host by requiring traffic to travel through an overlay, which filters out bad traffic [65, 3, 125, 75, 121]. A filtering ring is built around the server that blocks all traffic except for those coming from authorized nodes in the overlay. The verification is done using simple non cryptographic methods, such as source address or destination address filtering, destination port filtering, etc.

Reactive Filtering

Reactive filtering is applied to flows that the receiver indicates are bad traffic. The receiver can, for example, indicate a source IP address or a set of source IP addresses together with the destination address(es) of the recever that it wishes to be blocked. A router then installs a rule that drops all packets that match that rule.

It would be preferable to do the filtering close to the source. This way the bad packets would not consume network resources and congestion on path. Also, filtering may be easier to do close to the source, where the flows are still relatively small and easily distinguished. Unfortunately, this is difficult, since it requires deployment both at the source and at the destination. Source address spoofing also makes it more difficult to push the filtering to the other edge. The idea of using regions of mutual trust [124, 147, 146] has been proposed as a partial solution to this dilemma.

Some benefit can also be gained by filtering within the receiver's operator [76, 82]. Here the focus is on organizing an efficient method for filtering out the bad traffic within the operator network while enabling other flows to continue as smoothly as possible. It is, however, limited by the available network bandwidth the operator has.

Early proposals pushed the filtering towards source hop-by-hop (see e.g. [58]). While this sidesteps the problem of identifying the source, it produces an unbearable burden for the routers in the core of the Internet, since both the traffic rates and the number of flows that need filtering are highest in these routers.

Alternatively, the filtering can be pushed directly from the receiving edge to the sending edge [8, 7, 120, 57, 80, 4]. Such an approach requires that the receiver can correctly identify the source and send a filtering request. Ingress filtering [45], source address validation [141] have been

proposed to prevent source address spoofing. However, obtaining full deployment is necessary and difficult.

AITF [8, 7], edge-to-edge filtering architecture [57], and StopIt [80] utilize intermediate trusted routers to add information about the packet's origin. Holding the Internet Accountable [4] proposes a new address structure that is composed of an AS part and a host part, both cryptographically verifiable. This makes it possible to reliably determine the origin of a packet and use shut-off messages directly to the source NIC.

Leveraging good intentions [120] utilizes the fact that most hosts participating in an attack are actually owned by well meaning owners and used for bad purposes by an outsider, who has gained control of the machine from afar. Thus, the actual owners are, at least, not opposed to the idea that an end-to-end protocol stops an outsider from using their machine for an attack. Hence, it proposes a separate end-to-end control protocol, which lets a host signal the other end to stop sending for a short period of time.

2.3.2 Hiding and replication

Hiding and replication are two techniques that have the potential to help in DDoS resistance and it's easy to see why. If the attacker cannot find the target, or if the target has more resources than the attacker, the DDoS attack will likely fail. As an example, it has been estimated that the replication and hiding of DNS root servers using anycast increases the robustness of the system. Each root server is replicated to multiple locations and each server uses the same IP address. This way, the IP routing fabric itself, hides all but the closest server for each host in the Internet.

Steps toward a DoS resistant Internet [51] proposes hiding clients from other clients and servers from other servers. This is accomplished with the separation of client and server addresses and, preventing connections between clients and clients or servers and servers.

Data can be replicated with relative ease. DONA [69] shows that this can be done reactively when demand for a certain piece of data increases. Anycast [97] has been used in some individual cases to replicate and hide services, e.g. DNS root servers [52]. Recent interest in cloud computing [9] is a step towards running services on platforms that automatically scale the system to demand.

(Secure) Internet Indirection Infrastructure (s)i3 [129, 2] proposes an

overlay which enables communicating hosts to keep their IP addresses hidden. The overlay utilizes a distributed hash table called Chord. Hi3 [95] separates control plane from data plane using the Host Identity Protocol [90, 91] and utilizing an overlay for connection setup. Firebreak [46] uses anycast to hide servers behind a set of proxies.

2.4 Mobility

The Internet architecture uses IP addresses for two somewhat conflicting purposes: as topological address used by the Internet routing to deliver packets to their destinations and as connection identifiers at transport and application layers. This dual nature causes problems when nodes are mobile. There is a large number of proposals that provide mobility support on the Internet (see e.g. [78] for an overview).

The minimum necessary security requirements for mobility management stay the same regardless of where in the stack the mobility management system is placed. When a mobile node moves, it needs to signal the correspondent node about the new location. The signaling must be authenticated to ensure that it is, indeed, the mobile node that originated it. Additionally, the correspondent node needs to verify that the mobile node is where it claims to be. The latter requirement is called return routability testing.

The existing IETF macro-mobility protocols, like Mobile IP (MIP) [99], Mobile IPv6 (MIPv6) [61] and HIP [91], are designed in a way that only the peers and rendezvous nodes participate in the mobility signaling. The protocols aim at disentangling the two purposes of IP addresses. MIP and MIPv6 uses home addresses as connection identifiers and current-care-of-address as routing identifier, whereas HIP uses a separate host identifier, cryptographically generated from a public key, for (transport layer) connection identification.

The return routability test is used to ensure that the mobile node is where it claims to be. The corresponding node sends a test message to the new location, requiring the mobile node to answer to it. This exchange requires at least 1.5 round trip time, causing a noticeable delay for delay sensitive protocols. Additionally, micro-mobility protocols like HMIP [26], FMIP [68], Cellular IP [23] suffer from scalability problems related to location update security mechanisms and the intermediate mobility anchor

points.

To overcome these limitations some overlay approaches, such as i^3 [127], Secure-i^3 [2], and Hi^3 [95] have coupled the mobility and the packet forwarding plane with each other, above the IP layer. Due to the use of Distributed Hash Tables, these systems cannot ensure policy compliant paths. However, when the mobility management is realized with support of the overlay routing infrastructure, it results in a more DoS resistant packet delivery.

We show in Publication VI that with an internetworking architecture based on source routing and Bloom filters [19], the return routability test can be bypassed by collecting the return source route into the signaling packet en route. Coupled with a hash chain based authentication, communication can resume after a single message, in the most common case when only one end point moves simultaneously.

2.5 Rendezvous

Locating groups, publishers, or subscribers is a key problem for large scale publish subscribe and multicast systems.

Hierarchical location based names (e.g. IP) improve scalability by enabling aggregation within the routing architecture. Routing on DNS based names has been proposed in Triad [50] and is also used in CCN [59]. While the former uses policy routing, the latter utilizes DNS and resolves the names into IP addresses if the data is not found locally. However, hierarchical location based naming limits the applicability of the names, since objects need to conform to location and organizational boundaries.

Self-certifying and flat names are another potential solution for naming [114, 84, 69]. They make secure name resolution easier and have the potential for logical separation of data (or objects more generally) from the location it is found in. This makes it possible for new information structures to emerge after the data is created. However, flat naming structures do not allow for simple aggregation of the name based routing. Instead, in the default free zone, policy routing scales linearly to the number of registered objects. This is especially problematic, if many nodes (e.g. ASes) need to participate in the default free zone.

Distributed hash tables (DHT) (see e.g. [112, 128, 117, 83]) have been proposed as a method for scalable distributed lookup. In a DHT, each

participating node has one or more (hash-based) identifiers. The nodes maintain a set of pointers to other nodes in the DHT in such a way to ensure relatively efficient routing to any DHT node.

However, DHTs suffer from a few key problems, especially in inter-domain scenarios. First, the owner of an object has little control over the node that stores it. Second, the requesting node (or his provider) has little control over the route the request takes. These are problems, since the lack control over location and routing means lack of control over dependability, privacy, the legal system, and many other concerns.

Each object stored in a DHT also has a hash-based identifier and is stored in a node that is directly determined by the identifier, e.g. the node with the smallest node identifier greater than the object identifier (or with the greatest node identifier smaller than the object identifier). The different DHT based service proposals vary on what the stored objects are and what they are used for.

Oceanstore [73] uses a DHT for storing pointers to the location where data is stored and combines this with probabilistic Bloom filter [19] based system for routing to local copies for faster lookups. Internet Indirection Infrastructure [127, 2] uses a DHT for storing *(id,addr)* and *(id,id)* pointers. These are used for building unicast, multicast, anycast, mobility, and service composition services. CoDoNS [110] uses a DHT with caching for fast DNS lookups. DHTs have also been proposed for resolving tags to object-records [134] and services (sid) to end points (eid) to IP addresses [11]. ROFL [22] uses a hierarchical DHT based on Canon [48] for routing on flat labels.

We now present Canon [48] and DONA [70] in more detail, since the techniques are used in our rendezvous design.

2.5.1 Canon

Canon is a hierarchical distributed hash table (DHT) and designed to maintain the uniformity of load while allowing fault isolation and security, effective caching and bandwidth utilization, adaptation to the underlying physical network, hierarchical storage and hierarchical access control.

In Canon, each domain forms a local DHT. A number of DHTs can be combined together to form a DHT ring by adding routing pointers between the domains. The process is shown in Figure 2.9. A number of such

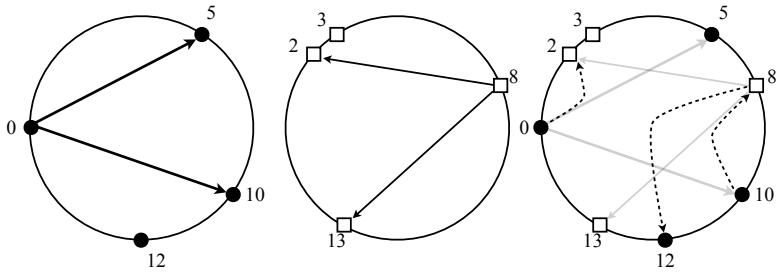


Figure 2.9. Combining two Canon rings.

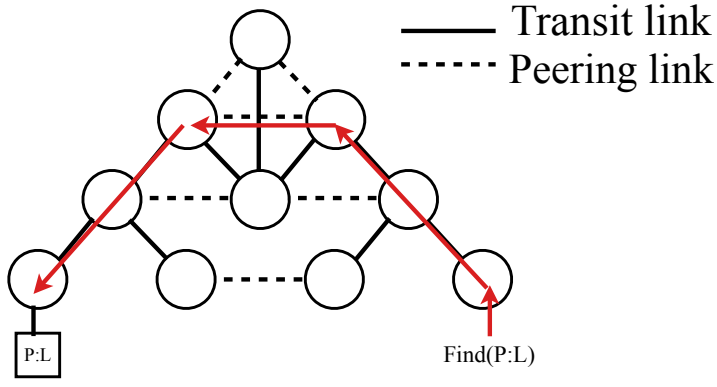


Figure 2.10. DONA routing is based on local AS policies.

rings can, again, be combined to form even larger ring. The outcome of the process is a hierarchical DHT structure, in which, at each level, the local ring has full routing state for object pointers stored locally and some pointers to the ring above in hierarchy for global routing.

It is important to notice that at each hierarchy level, the routing may pass through any node in any of the sub-domain forming the ring.

2.5.2 DONA: Data-Oriented Naming Architecture

Data oriented (and beyond) network architecture (DONA)[70] proposes a name resolution service that aims for persistence, availability, and authenticity of data. It is based on the idea of self-certifying names. The names are of the form P:L, where P is a hash of the publisher's (principals) public key and the second part L is a string of bits chosen by the publisher to represent a particular piece of data. It could be, for example, a cryptographic hash of the contents.

The name resolution infrastructure is organized using the AS transit/peering hierarchy, as in TRIAD [50]. Each AS operates a resolution han-

dler (RH) which hosts use for REGISTER(P:L) and FIND(P:L) operations.

When a client sends a FIND(P:L), the RHs route the query to a nearby copy. The REGISTER(P:L) message sets the necessary routing state using policy based routing, i.e. it emulates BGP routing policies. An RH that receives a register message from a child, forwards it to its peers and transit providers, if there is no copy with shorter path in its routing table. An RH that receives a register message from peer can forward the message or not, depending on the local routing policies. Figure 2.10 shows an illustration of the routing process.

The state requirements for a fully deployed DONA are reasonable, approximately 500 PCs in each tier-1 AS. This is a relatively minor cost compared to the total costs of running a large AS.

However, the situation is different for partial deployments, especially deployments in which many of the tier-1 ASes do not participate. In such scenario, which is likely when a system is incrementally deployed, the network is essentially partitioned into DONA sub-trees that will need to interconnect through the (non-deploying) tier-1 ASes.

Such a partial deployment of DONA with tier-1 ASes missing would require every AS on top of its local tree to hold routing information for all the data registered in the DONA system. It would also magnify the amount of register traffic, since each pair of ASes on top of their respective DONA trees would need to exchange their full routing state. In Publication III we showed that, in fact, tier-1 ASes may have incentives not to participate in such a system.

2.6 Techniques

In this section, two techniques used in this thesis are described: bit permutations and hash chains. The bit permutations are used on the Bloom filters to prevent loops and hash chains are used for authenticating multicast and mobility signaling between source and receiver.

Hash chains

Hash chains [77, 126] are based on cryptographic message authentication codes, such as SHA-1 [37]. A hash chain, as shown in Figure 2.11, is a collection of values $V_i, i = 1 \ldots N$ such that each value V_i is a cryptographic hash of the previous value in the chain V_{i-1}, except for the firsti value V_1. The owner first reveals the last value V_N (called a hash anchor)

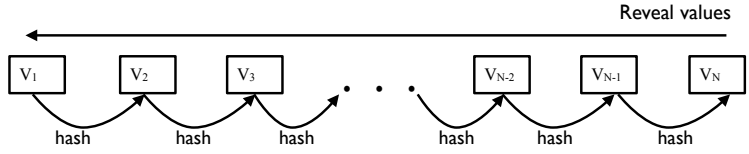


Figure 2.11. The hash chain is constructed by repeated hashing of values starting from V_1 and continuing to V_N. The values are revealed starting with $V_N, V_{N-1}, \ldots$. Upon receiving value V_i, the receiver can verify that hash(V_i)=V_{i+1}.

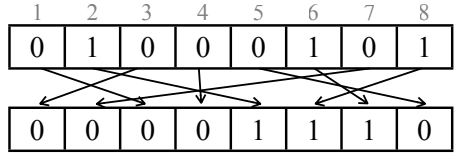


Figure 2.12. Example of a bit permutation. The position of each bit is permuted, e.g. the bit in position 1 is permuted to position 3.

to the recipient(s) and by subsequently revealing the values $V_{N-1}, \ldots$ he can prove that he is the owner of the hash-chain, since only the owner knows the next values in the chain. Hash chains have been used for authentication in many network protocols, see e.g. [100, 56].

The simplicity of hash chains comes at a price. Once a hash chain value V_i is revealed, anyone can produce the values $V_{i+1}, V_{i+2}, \ldots$. Hence, a simple protocol in which the owner merely reveals the next hash chain value as a proof of hash chain ownership is vulnerable to man-in-the-middle attack unless time synchronization is used, as in TESLA [100].

Bit permutations

Bit permutations are a common building block in block ciphers. A bit permutation takes an array of bits and rearranges them so that the position of each bit changes. Figure 2.12 shows an example of a bit permutation. Static bit permutations perform the same permutation to all input arrays. It is relatively simple to make a static permutation in hardware.

Keyed permutations are bit permutations that perform a random permutation based on a key. While there are techniques for performing keyed permutations in hardware, they require relatively large number of gates [122, 55].

3. BloomCasting with PSIRP

Our work has focused on building a variant of the PSIRP architecture that is designed to utilize the current Internet architecture as the topology layer of the architecture. We combine the PSIRP rendezvous design with BloomCasting, a scalable multicast architecture for the Internet. One part of our contribution is an alternative design of the PSIRP architecture that works together with the current Internet architecture. In the rest of this section, we first introduce the PSIRP architecture, followed by the BloomCasting multicast architecture.

3.1 PSIRP architecture

The PSIRP (Publish-Subscribe Internet Routing Paradigm [104]) architecture is a clean slate internetworking architecture designed to support publish/subscribe based communications natively in the network. Information is the primary named entity across all layers, and the primary function of the network is to locate and deliver (named pieces of) information.

The basic network primitives in the PSIRP architecture are publish and subscribe. With publish, an application can offer a piece of information and with subscribe an application can request a piece of information. To create a network layer based on the publish/subscribe model, the PSIRP architecture is divided into three main architectural elements: Rendezvous, Topology, and Forwarding. Figure 3.1 shows the integrating the PSIRP architecture into the Internet architecture with BloomCasting.

The topology function is used to create and maintain the delivery trees used to forward the publications. The trees can be created reactively, when there is a need. Delivery trees, or parts of them (e.g. a virtual link)

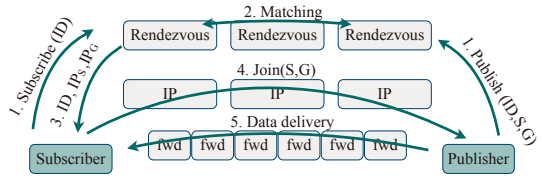


Figure 3.1. Shows the components and their relationship in our BloomCasting based PSIRP architecture. (1) The publish and subscribe operations happen asynchronously in any order. After there is both a publish and subscribe, the rendezvous matches these (2) and sends a source-specific multicast group specifier to the subscriber (3). The subscriber sends a join message to the publisher (4) which then adds the receiver into the multicast delivery tree (5).

can also be created proactively where it is likely to be useful. The purpose of a separate forwarding plane from topology management is to have an efficient and scalable forwarding plane for delivering packets along the delivery trees. In the next sections we detail how the existing Internet can be used as a topology plane for the PSIRP architecture.

The rendezvous is used for matching the publishers and the subscribers, and is divided into a global rendezvous system and rendezvous points. In the rendezvous service model, a publisher creates a scope in a rendezvous point and publishes information in that scope, i.e. announces to the rendezvous point that certain pieces of information are available.

Our design is based on the following key design concepts:

A1: Everything is information: The network is based on a common naming of information through all layers, as opposed to naming hosts in the current architecture. While we recognize that there are resources (e.g. computation, memory, bandwidth) that per se are not information, communicating about the availability of such resources, or requesting the use of such resources is information.

We create a common and global information namespace usable across layers and applications and define an information item as the simplest and smallest unit transmitted by the network. Each information item has a rendezvous identifier (RId) that is statistically and globally unique. The RId space is a flat cryptographically generated name space.

A2: Information is scoped: Information exists in one or more scopes. A scope is a collection of information administered by a single entity called rendezvous point. A scope is information structure by itself and hence identified with an RId called scope identifier (SId).

A scope provides a method for structuring information into collections. This increases the scalability of the global rendezvous system by dividing

the problem of locating information into smaller more manageable parts. It gives an application a tool to group information together and allows publishers to tailor access control policies for various groups of information.

A3: Equal control: Publishing information is sender-controlled while retrieval of information is receiver-controlled, provided that access has been granted. Thus, communication will only take place with agreement from both parties. With it, the PSIRP architecture provides a balance of power between publisher and subscriber. It offers a new set of network services that shifts the network from send/receive between endpoints to a publish/subscribe model of information [40, 25].

3.1.1 Global Rendezvous

Our goal is a general architecture for locating objects in a network that can scale to large numbers of objects in an inter-domain environment. These objects can be hosts, networks, multicast groups, information, services or other identifiable objects residing in the Internet and using the same rendezvous architecture. The solution needs to be efficient and scalable so that it can serve the multitude of different objects residing in the Internet. Additionally, we require incremental and partial deployability.

The rendezvous point publishes the availability of an Rid to the global rendezvous system. However, we assume that there is a some cost involved in adding and updating Rids to the global rendezvous space. This provides an incentive for publishers to use the global rendezvous space for scopes and maintain their own publications within the scope. This reduces the number of items published and maintained in the global rendezvous. Hence, we believe that the global rendezvous system would be mainly used for publishing scopes, i.e. Sids, instead of individual Rids. The publication of an SId to global rendezvous system is only required if a global reachability of the scope is wanted.

The provided system is flexible, in that it allows a publisher to add an individual information item to the global rendezvous space, if it deems it important enough. Neither does it require the publication of a scope in the global rendezvous space. Virtual private information networks can be built by maintaining private rendezvous system that hosts the private scopes and only providing a globally reachable scope to the rendezvous point that provides access control to the private network.

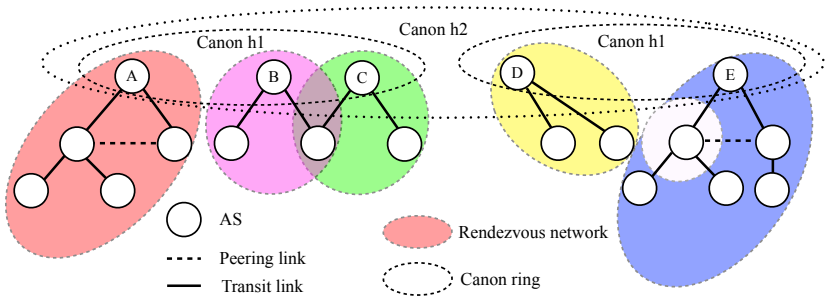


Figure 3.2. The figure shows our rendezvous architecture. The rendezvous networks use local policy based routing and the overlay is organized using Canon, a hierarchical DHT.

The global rendezvous system, described in Publication IV, is used for registering and requesting SIds. It implements anycast service by routing SId requests to one of the rendezvous points servicing that particular SId. The rendezvous point then identifies a particular source for the requested RId. Our global rendezvous design is based on the assumption of incremental deployment. We divide the problem into two architectural components: rendezvous networks and an interconnection overlay. Figure 3.2 shows the rendezvous system.

First, we assume that each AS runs a rendezvous node that is responsible for rendezvous within its network. Furthermore, ASes can make direct rendezvous peering or transit agreements (similar to DONA [70]). We call a group of ASes interconnected via such agreements rendezvous networks. Within rendezvous networks, each AS advertises the availability of scopes to its rendezvous peers and rendezvous providers. Hence, each AS will maintain state for all the scopes announced by its peers and by its customer hierarchy.

As a single rendezvous network may not cover the whole Internet, an overlay is used for interconnecting the rendezvous networks together for global reachability. The overlay is built with Canonical Chord [48] [1].

A rendezvous request then takes the following path: (1) it is first sent to a local rendezvous node. It either locates the information or forwards it to its local rendezvous provider. (2) In the local rendezvous network, each rendezvous node checks if if has a pointer for the object. If it does, it sends a reply via the same route the request traversed. Otherwise, it sends the request to its rendezvous provider. (3) Once the packet reaches the root

[1]Canonical Chord is also used in ROFL [22]. We apply the design over willing rendezvous service providers, which ensures that enterprise ASes only appear as endpoints of any communication path through the rendezvous architecture.

of the rendezvous network, it is forwarded through the interconnection overlay and (4) finally to a rendezvous network in which an RP hosting the SId is registered in.

In the standard operation, the first node that finds a cached entry of the SId $\longrightarrow$ RP location mapping sends a response back to the source the same route that the request traversed. This makes it possible for intermediate nodes to cache the information. This return route can, for example, be encoded using the Bloom filter based forwarding on the rendezvous overlay and network.

Some SIds may be multihomed, i.e. either the rendezvous point is multihomed or the SId is hosted by multiple RPs. In the first case, the RP registers the SId via multiple providers and in the second case each RP registers the SId. The overlay locates an RP in anycast fashion.

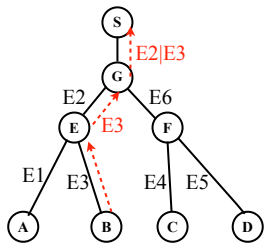
SIds can also be mobile. As an example, a mobile node can run an RP locally. It registers a SId that can be used by other nodes to reach it to the rendezvous system. When it moves, it re-registers the SId.

There are a few choices for ensuring that the rendezvous system does not provide stale information. First, the rendezvous node responsible for the SId can set “No caching" bit in the replies, ensuring that other rendezvous nodes do not cache the pointer. Second, if caching of the SIds is allowed, the responsible rendezvous node collects the Bloom filter based source routes to those rendezvous nodes that cache the SId.

When the RP makes a location update for the SId, the update is routed through the rendezvous network and the overlay until it reaches the responsible rendezvous point. The responsible rendezvous point then sends an update that deletes the state to its rendezvous peers and providers. If caching is allowed, the responsible rendezvous point sends a location update to all nodes caching the state.

3.2 BloomCasting

BloomCasting is designed as a secure source-specific multicast architecture, which transfers the membership control and per group forwarding state from the multicast routers to the source. The BloomCasting architecture is divided into *multicast group management* and *multicast forwarding*. Joining a group and maintaining group state is done with end-to-end signaling, so that intermediate routers collect path information for



Source S

Receiver	BF
A	E1|E2
B	E2|E3
C	E4|E6
D	E5|E6
ABCD	E1|E2|E3| E4|E5|E6

Figure 3.3. The left side shows a multicast Join message using iBFs. The right side shows a simplified Membership Table MT(S) that contains the Bloom filters for A, B, C, and D. The separated bottom row shows how to combine the Bloom filters into an iBF.

multicast source routing. Similar to [111, 62, 149], it uses source routing and encodes the forwarding tree into an in-packet Bloom filter (iBF).

To join, a host sends a join request (`BC_JOIN`) towards the source S. Intermediate routers record forwarding information in the packet, so that when the packet reaches S, it will contain a *collecting* iBF for the source-receiver path. By combining the iBFs for all the receivers, the source will have an iBF encoding for the whole multicast tree. When a host does not wish to receive packets for the group anymore, it sends an authenticated leave message to S. Upon processing this packet, the source will reconstruct the Bloom filter for the group leaving out the recently pruned path. The operation is illustrated in Figure 3.3. The process and the techniques used are explained in more detail below.

Data packets are routed using the *forwarding* iBF placed in the `BC_FW` header. Each intermediate router takes its forwarding decision by querying it with the question: which of my outgoing links are present in the iBF? It then forwards the packet to the corresponding peers. Eventually, the packet reaches all the receivers, following the sequence of routers the `BC_JOIN` packets traversed, in reverse order.

3.3 Group Membership Management

Group membership management includes the joining, leaving, and maintenance of multicast groups, and this is the main task of the control plane. Along this discussion, we show how BloomCasting encodes a multicast tree into the iBF.

Joining a group: When a host joins a multicast group, it sends a BloomCast Join (`BC_JOIN`) message towards the source. The packet con-

Algorithm 1: Adding edge-pair labels (E) and permuting collect and forward iBFs at transit routers.

```
Collect_iBF (C):
    E ← Z_K(S,G,R_p, R_c, R_n);
    C ← C ∨ E;
    C ← Permute_K(C);

Forward_iBF (F):
    foreach outgoing link i do
        F ← Permute_K(F);
        E ← Z_K(S,G,R_n, R_c, R_p);
        if E ∧ F = E then
            Send F → i;
        end
    end
```

tains the following information: (S,G) specifying the multicast group and a *collecting* iBF. The latter is used for collecting the forwarding information between the source and the receiver. Finally, it also contains a hash chain anchor for future signaling security, see Section 3.6 for details.

In each router, the next hop for the `BC_JOIN` message towards S is found from the routing information base.[2] As the message travels upstream towards the source, each router records forwarding information into the packet by inserting the edge pair label E into the *collector* iBF. After this, for loop prevention and increased security, it performs a bit permutation on the *collector* iBF (see Section 3.5.3). Finally, it selects the next hop usptream towards S. The operation is shown in Algorithm 1. Unlike traditional IP multicast approaches where the forwarding information is installed in routers on the delivery tree, in BloomCast, transit routers do not keep any group-specific state.

Once the `BC_JOIN` message reaches the source, it contains sufficient information so that the source can send source-routing style packets to the recently joined host. The source stores this information in the Membership Table (MT), as shown in Figure 3.3. The source can now send packets to the multicast tree by combining iBFs for the group, by bitwise ORing them together.

[2]Just like in standardized IP multicast protocols, this forwarding decision can be taken according to the RIB created by BGP or according to the Multicast RIB created by MBGP [15].

Leaving a group: When a receiver wishes to leave the group, he sends a BC_LEAVE towards S, including the next element from the hash chain it used when joining the group. On-path routers forward the packet to S. As no further processing is needed in intermediate routers, unlike pruning packets in IP multicast, BC_LEAVE packets are always routed to the source.

S verifies the hash and removes (or de-activates) the entry in the Membership Table. Single message hash authentication, vulnerable to man-in-the-middle attacks, is sufficient, since the hash is only used to verify that the host wishes to leave the group. As a final step, it recomputes the forwarding iBF of the delivery tree. An example of a forwarding iBF is shown in Figure 3.3 at the separated bottom row of the table.

Refreshing membership state: The iBFs in the MT may become stale, because of route failures, because the receiver has moved to a new location, or because a router changes the keys it uses to compute the edge-pair labels (see 3.5.1). Keys are expected to change periodically (e.g., every few hours) to increase security by excluding brute force attacks, see Publication V.

This means that the iBF needs to be recomputed with a new BC_JOIN packet. When making the forwarding decision, during a transition period routers need to compute edge-pair labels for both the old and the new key. If they find that an edge-pair label computed with the old key is present in the iBF, they set a flag in the BC_FW header indicating that the receiver should send a BC_JOIN again, as the iBF will soon become invalid. When a packet is to be forwarded on a failed link, the router sends an error message back to the source.

3.4 Multicast Forwarding

So far, we have discussed how hosts join and leave multicast groups. We now show how data packets are forwarded between the source and the receiver.

As we saw previously, iBFs for each receiver border router are stored separately in the Membership Table. We also saw the basic concept of deriving the forwarding iBF from the MT information; now we extend that with new details.

For each group, the source stores one or more iBFs in its BloomCast Forwarding Table (BFT). Several iBFs can be stored because the number

of receivers is too large for a single iBF or because the traffic is split towards several routers alread at the source[3]. In practice, the capacity of a packet-size iBF is limited in order to guarantee a certain false positive performance. In case of large multicast groups, several iBFs are created, one for each partial multicast tree, and duplicate packets are sent to each next hop.

The source creates one copy of the packet for each next hop for (S,G) in the BFT. It creates a `BC_FW` header, fills it with the corresponding iBF, and sends it to the next hop router.

Each router makes a forwarding decision based on the iBF. First, it applies the reverse permutation function to the iBF, replacing the iBF with the result. Then, it checks for the presence of peer routers by computing one edge-pair label for each peer R_n, based on the R_p and R_c[4] and on group identity (S,G) found in the IP header as shown in Algorithm 1. In the final step, the router checks whether the iBF contains the edge-pair label, by simple bitwise AND and comparison operations.

3.5 Techniques

We now describe the techniques we use to solve security and reliability issues in Bloom filter-based forwarding, identified in Section 2.2.4. Cryptographic edge-pair labels are used to make it difficult for an attacker to guess a Bloom filter for a path or tree. Varying k is used to prevent packet storms, which could happen either because of malicious users or by chance. Per-hop bit permutations on the in-packet Bloom filters are used to prevent loops and flow duplication.

3.5.1 Cryptographic edge-pair labels in Bloom filters

We proposed a technique called Z-formation in Publication V. It combines two ideas to provide bi-directional in-packet Bloom filters that can simultaneously act as capabilities [5] and forwarding identifiers.

Instead of maintaining a fixed forwarding table containing the Link IDs (or LITs) for each outgoing interface, the edge-pair labels are dynami-

[3]This improves forwarding performance, as the false positive probability increases with the number of iBF inserted elements.

[4]The router uses the same inputs as in the `BC_JOIN`, hence the R_p and R_n switch places due to direction change

cally computed for each forwarded packet. A router uses the function Z to compute the edge-pair label using (i) Flow identifier[5] F_1F_2, e.g. (IP_S,IP_G) from the packet, (ii) a periodically changing local secret K, (iii) the incoming and outgoing interface indices (In, Out). Each edge-pair label $O = Z(F_1, F_2, K(t_i), In, Out)$ is an m-bit long string with k bits set to one. The Z function is any function capable of producing k cryptographically secure hashes from the given input, e.g. a fast, spreading, hash function that yields the bit locations for the edge-pair labels [71, 148]. The method can be applied independently at each router, having no impact on the protocol.

Secure iBF forwarding: When a data packet arrives at a forwarding node, the node extracts F_1, F_2 from the packet. With F_1F_2, the incoming interface index, and the current $K_i(t)$ value, it computes the LIT for each outgoing link. If the iBF matches the on-the-fly generated edge-pair label, the packet is forwarded along the interface. Dynamic edge-pair label computation can be easily done in parallel for each interface. Note that forwarding nodes are freed from storing any per-flow state or traditional FIB table lookups. Only the *seed* of the secret K and the currently accepted values $K_i(t)$ need to be maintained and these are independent of the number of sources, groups, receivers, or even peers[6]. Figure 3.4 shows Bloom filter collection and in Figure 3.5 packet forwarding with Bloom filters.

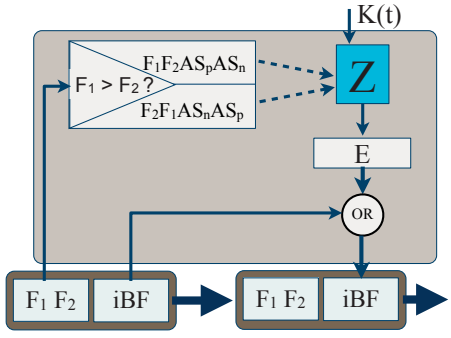


Figure 3.4. Collecting function in a single router: The packet contains the iBF and a two part flow identifier F1F2, where the order of F1F2 depends on the direction of the traffic (e.g. source and destination IP addresses). Comparing the value $F1 > F2$ lets the router order the inputs to the function Z so that the inputs are in the same order for both directions. Function Z then computes the edge-pair label, which is bitwise ORed to the iBF before the packet is forwarded to the next hop router.

[5]Not to be confused with Flow label field in IPv6 [108]
[6]However, the router still needs to have a local enumeration of its peers.

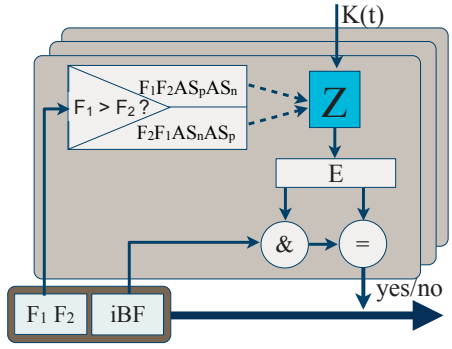


Figure 3.5. Forwarding function in a single router: The router compares F1 and F2 to determine the order of inputs to the Z-function. It then computes the edge-pair label and compares it to the iBF by checking if bitwise AND with the iBF is equal to the edge-pair label.

Bi-directional edge-pair labels: Figure 3.5 shows a two part flow identifier F_1, F_2. The flow identifier can be e.g. IP_{src} and IP_{dst} and it can be used to create the bidirectional path with a single Bloom filter. The Flow Id is ordered in the packet header F_1F_2 in one direction and F_2F_1 in the other[7]. The router decides the order of inputs based on the relative numerical values of F_1 and F_2. In other words, if $F_1 > F_2$, we define a packet to be flowing downstream. Therefore $E = Z(F_1, F_2, AS_p, AS_n, K(t_i))$; else the packet flows upstream and therefore $E = Z(F_2, F_1, AS_n, AS_p, K(t_i))$, where E is the edge-pair label.

This enables the iBF routers to determine the relative direction of the packet as shown in the left upper corner of Figures 3.4 and 3.5. The method works, because switching the Flow ID labels F_1 and F_2 enables the routers to distinguish between the upstream and downstream relevant to edge-pair label computation.

When several Bloom filters are combined to form a multicast tree, the resulting Bloom filter will match the edges in the multicast tree, when the packet travels from the source to the set of receivers. A receiver (or intermediate router) can use the same Bloom filter to send control messages to the source.

[7]Another possibility is to use a separate bit in the packet header to signal the direction of the flow

3.5.2 Varying Bloom filter parameters

False positives occur randomly, hence, packet storms cannot be reliably detected and stopped reactively. To control them, a stability condition must be preserved, i.e. $d \cdot \rho^k < \alpha$, where $\alpha \leq 1$ is a preferred maximum average number of false positives per node, d is the node degree of the forwarding node and k is the number of bits set in the link masks.

Given a fill factor ρ, the average number of false positives is independent of the length of the Bloom filter. This suggests a two-fold solution: firstly, each node sets k locally based on the node degree and, secondly, the source sets the length m of the Bloom filter so that the fill factor $\rho \approx \rho_{max}$. (Either technique can also be applied separately.) The maximum fill factor ρ_{max} should be 50% or slightly more to maximize the information content of the Bloom filter.

Varying k: Each node sets k_{var} locally based on its degree d:

$$k_{var} = \lceil \log_2(d-1) \rceil + r \text{ for some small global integer } r.$$

This local condition guarantees that, at any node, the false positive rate is $fpr < \rho_{max}^r/(d-1)$. Thus, a false positive arriving at any node will cause, on the average, fewer than 2^{-r} further false positives to be sent. If we set $\rho_{max} = 50\%$ and r globally to some small value, this limits the bandwidth overhead caused by the false positives to $1/(2^r - 1)$. For example, for $r = 3$, the overhead is limited to 14%.

The variable k_{var} does not only prevent traffic storms in the high-degree core parts of the network; it also optimizes the usage of Bloom filter capacity so that there is no unnecessary safety margin and the filter capacity can be used to encode larger multicast trees. This is particularly important if the multicast group size exceeds the maximum capacity of a single filter and the group needs to be split into several trees. The simulations with Internet topology in Publication VII confirm this by showing about 50% reduction of fpr with variable k_{var} compared to static k (holding the Bloom filter size and multicast tree constant).

Varying Bloom filter length m: The number of elements added into the Bloom filter (together with the Bloom filter length m and the number of bits used to identify a link k_{var}) determines the fill factor ρ, which then determines the false positive probability. The higher the fill factor, the higher the probability of false positives. In other words, when encoding bigger multicast trees, we can expect a larger average number of false positives for a given Bloom filter length m_1. However, if we use a longer

filter length m_2, we could reduce the false positive probability for the same multicast tree, thus improving efficiency, with the penalty of larger packet headers. Vice versa, multicast trees containing only a couple of receivers would require fewer bits for efficient packet delivery and would save on per-packet overhead, which is important for applications where average payload sizes are small.

We propose a scheme to implement the variable-length Bloom filters. First, a long filter $longBF$ (e.g. $M = 8000$ bits) is created for the multicast tree. Then, the fill factor ρ, i.e. the number of 1-bits divided by the filter length, is computed. This allows us to compute what would have been theÄ optimal length of the filter: $m = \lceil -M \log_2(1-\rho) \rceil$. A filter of length m would result in a fill factor 50% for the tree. We then fold the long filter into one of length m as follows:

$$iBF[i] = \bigvee_{j=0 \ldots \lfloor M/m \rfloor} longBF[j \cdot m + i] \text{ for } i = 0 \ldots m-1.$$

(Note that the elements beyond the array boundaries are considered to have value 0. Also, the fill factor of the resulting vector may, by chance, be above 50%, in which case m should be decremented by one until the fill factor is below the limit.) The forwarding algorithm is modified in such a way that the k hash functions $f_{1 \ldots i}$ used to compute the locations of the 1-bits in the link masks are modified to be $f'_i(x) = f_i(x) \bmod m$.

3.5.3 Bit permutations on in-packet Bloom filters

Routing loop or flow duplication happens, when the same nodes forward the same packet (or its copy) multiple times. To prevent such anomalies, we utilize the fact that the packets differ in their history i.e. the path they have already traversed. Only one occurrence has traversed exactly the path that was intended by the routing algorithm while the others have taken some anomalous path and then rejoined the intended route.

The history-dependent forwarding can be achieved by accumulating information about the traversed path in the packet. For example, loops could be prevented by adding a hop counter or TTL value in the packet and including it as an input to the computation of the Bloom filter and link masks. This does not, however, prevent flow duplication where the path length is the same for both copies of the flow (e.g. Figure 2.7(a)). It would also complicate the process of collecting edge-pair labels into a signaling packet.

```
Algorithm 1: packet forwarding

Input: Edge pair label inputs;
  Permutations of the node;
  iBF in the packet header;
let π = Permutations[ingress link]
set iBF in packet to π(iBF);
foreach outgoing link l do
  let mask = LinkMasks[ingress link, l]
  if iBF & mask == mask then
    Forward packet on the outgoing link l
  end
end
```

Figure 3.6. Pseudocode for packet forwarding

Our solution is to *perform a bit permutation of the Bloom filter on each router*. The cumulative permutation of the filter along the forwarding path, in effect, makes the forwarding decisions dependent on the path already traversed by the packet. Every router selects a random permutation for its inbound links, and applies this permutation to the filters of all packets arriving on that link. The forwarding algorithm is shown in Figure 3.6.

There are a few reasons for choosing bit permutations as the technique for making forwarding dependent on the history of the packet. First, (static) bit permutations can be efficiently implemented in hardware. Second, performing a transformation on the filter does not consume any additional space in the packet header.

Upper bound on infinite loop probability: When a bit permutation is applied to the filter at every hop, the first false positive is just as likely to occur but, after that, the packet will usually not match the link masks of its old route. Every further hop along the looping path requires another false positive and, intuitively, we would expect a duplicate packet to be dropped soon. This intuition is slightly misleading as infinite loops are still possible: the set of bits tested by the routers around the loop fall into some cycles of the permutations, and the packet will go into an infinite loop if and only if all bits in these cycles happen to be 1.

Probabilities for some values m and K, where $K = \sum_{i=1}^{n} k_{var}$ are shown in Table 3.1. Recall that K is the total number of bits set in the link masks around the loop. We refer to Publication VII for a detailed analysis of a

Table 3.1. The probability of infinite loops with varying m and K given $\rho = 0.5$

K	m = 64	m = 128	m = 256	m = 800
3	$4.1 \cdot 10^{-4}$	$5.1 \cdot 10^{-5}$	$6.4 \cdot 10^{-6}$	$2.1 \cdot 10^{-7}$
5	$7.8 \cdot 10^{-6}$	$2.4 \cdot 10^{-7}$	$7.6 \cdot 10^{-9}$	$2.6 \cdot 10^{-11}$
7	$3.1 \cdot 10^{-7}$	$2.5 \cdot 10^{-9}$	$1.9 \cdot 10^{-11}$	$6.6 \cdot 10^{-15}$
13	$3.5 \cdot 10^{-10}$	$4.3 \cdot 10^{-14}$	$5.2 \cdot 10^{-18}$	$1.9 \cdot 10^{-24}$
17	$1.9 \cdot 10^{-11}$	$1.4 \cdot 10^{-16}$	$1.1 \cdot 10^{-21}$	$4.3 \cdot 10^{-30}$
21	$2.4 \cdot 10^{-12}$	$1.2 \cdot 10^{-18}$	$5.9 \cdot 10^{-25}$	$2.4 \cdot 10^{-35}$

theoretical upper bound on loop probabilities. Assuming that a minimum of 3 hops is needed for a loop[8], we can say that the probability of an infinite loop is vanishingly small as long as K is reasonably large (> 10). Hence, even small (multihomed) networks should use $k_{var} \geq 3$.

As an example, for $K = 13$ and $m = 256$, the probabilty of an infinite loop is $\approx 5 \cdot 10^{-18}$. To give some sense to the number, a billion Internet nodes, would each have to initiate five billion flows for a single such instance to happen. If each node initiated a single flow every second, on average, it would take approximately 160 years for a single infinite loop to happen.

Probability of duplicate flows with permutations: A duplicate flow can happen when a false positive causes a packet to be forwarded to a router that is part of the forwarding tree. When the random permutations described above are used, the false positives packet will not be automatically be forwarded down the tree from the node that received it. Because of the randomizing effect, a false positive must occur at every further hop. This is just as unlikely as the propagation of any false positive.

3.6 Mobility

The Bloom filter based multicast makes receiver mobility relatively simple. The receiver only has to send a new subscription message to the source (and delete the subscription to the old location). In the rest of this section, we describe an authentication mechanism that a receiver can use for authenticating itself with a single message. This method can also be

[8]We are aware that there are also 2 hop loops in the Internet routing [142, 143]. However, we assume that such loops can be detected and corrected relatively simply, since such loops arise from local erroneously configured routing policies, instead of from the dynamics of the distributed route computation.

used as a fast mobility signaling method for unicast traffic. Bicasting can easily be used for smooth handovers, since the forwarding fabric is based on multicast. The details and analysis of the BloomCasting mobility are included in Publication VI[9].

The iBF can be tied to a pair of IP addresses, whether they are the multicast source specific group address (S,G) or IP address pair belonging to mobile node (MN) and correspondent node (CN). In essence, all communications can be treated as multicast communications.

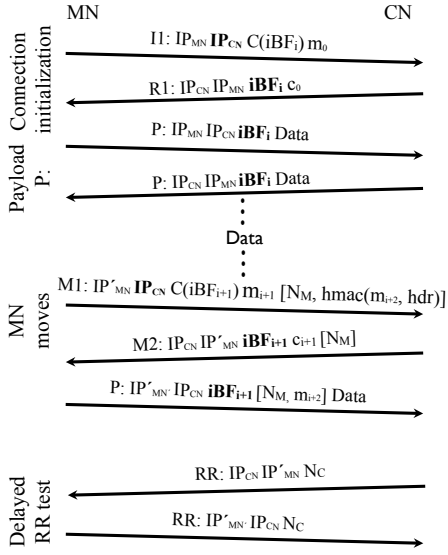


Figure 3.7. Protocol messages

The protocol is shown in Figure 3.7. MN sends an Initiation packet I to CN. The packet contains a hash anchor $m_0 = h^n(m)$ for a later mobility event authentication and a collecting iBF. As the packet is forwarded through the internetwork, each iBF router adds an edge-pair label to the iBF collector in the packet.

Finally, CN receives the packet that now contains the hash anchor and the collected iBF. It replies to MN with an init-reply (R) packet that contains the iBF, and its own hash chain anchor $c_0 = h^n(c)$. The packet is forwarded through the network with the iBF. Receiving the packet, MN stores the iBF and uses it to send packets to CN.

For scalability reasons, it may preferable to terminate the Bloom filter based path in a proxy close to the host, but not at the host itself. In

[9]However, the protocol diagram in Publication VI had some errors that have been corrected in the Figure 3.7 below

this case, the Bloom filter collection packet is still sent end-to-end, but the Bloom filter is collected starting from a proxy close to the MN and ending close the CN (or vice versa). The proxy will maintain per flow state in order to map between the IP address pair in the packet header and the local care-of-address or addresses that the recipient(s) have. When a packet forwarded with the Bloom filter reaches the proxy, it also verifies that the destination host is in the part of the network it administers, and forwards the packet.

3.6.1 Basic mobility

As MN moves to a new location, it sends a location update to CN, as shown in Figure 3.7. The location update contains an iBF collector $C(iBF)$, the next value in the hash chain $m_{i+1} = h^{n-(i+1)}(m)$, and an optional hash chain verification against man-in-the-middle attacks $N_m, hmac(m_{i+2}, hdr)$. The anchor values of the hash chains are carried in the first packets between the nodes. The intermediate iBF routers add the local edge-pair labels and the packet is delivered to CN.

CN verifies the authenticity of the packet by verifying the revealed hash value, and uses the collected iBF to send a reply packet to the MN. The reply packet contains the next value from the CN's hash chain $c_{i+1} = h^{n-(i+1)}(c)$ and the packet is forwarded using the newly collected iBF. Once MN receives the reply, it stores the new iBF and uses it to send to CN.

Our security solution prioritizes single message easy to compute authentication at the cost of not preventing man-in-the-middle attacks. However, such attacks require that the attacker is on the path between MN and CN and that it is able to capture the signaling packet. The optional hash chain check in the m_{i+2} (shown in Figure 3.7) prevents this, leaving only a short window of opportunity for eavesdropping for an attacker, after which the attacker will get caught. Once the values in the hash chain start to run out, it can be renewed by binding the new hash chain to the current one.

Bicasting can be used in the case of make-before-break. To do so, the mobile node signals the new location with willingness to receive *bicast* for a time. The sender bitwise ORs the two iBFs together and sends the subsequent packet with the resulting iBF. The iBF router at the bicast branching point automatically duplicates the packet to both destinations

due to the way the iBF has been constructed[10]. This ensures that the connection can be transferred smoothly from the old location to the new one without packet loss, or state requirements in the transit networks.

This leaves two alternative solutions. Firstly, the destination address in the packet header may contain the home-address of the MN, and the proxy stores the care-of-address, home-address pair. The proxy then swaps the care-of-address (CoA) to the header before forwarding the packet to MN. This approach requires special care in how the proxy handles the switch, to prevent an attacker from creating false state in the proxy.

Secondly, it is possible to use the CoA, instead of the home address. In this case, during mobility signaling two iBFs need to be collected, one for the old CoA and another for the new CoA. CN can then bicast data to MN by using the iBF collected using the old CoA. The new CoA has to be included in the packet header so that the final forwarding between the proxy and the MN can be done at both destinations.

3.6.2 Dual mobility

If both nodes are capable of moving, the beginning of the signaling after the MN moves is just as described above. The CN now knows that the MN can be reached via the collected iBF. It still needs to verify that the MN can be reached also directly using the MN's current IP address. To do this, the CN performs a delayed return routability test as shown in Figure 3.7. Before CN moves, it needs to verify MN's current IP address in order to send the location update. It is important to note that both MN and CN can start sending payload before the delayed return routability test.

In the case of two mobile nodes, it is assumed that the connection initiation happens to an IP address that is hosted by the MNs rendezvous agent (e.g. home-agent in MIPv6). Assuming two mobile nodes MN1 and MN2 move simultaneously, a rendezvous agent is needed. The rendezvous system described in Section 3.1.1 can be used for that.

Whenever a mobile node moves, it registers an SId to the global rendezvous system with it's location. If both mobile nodes move simultaneously, they can request the current location of the other mobile node from the global rendezvous system and then send an iBF collector to its current location.

[10]For this to work, the IP address used to compute the edge-pair labels has to be the same for both paths.

Table 3.2. Mapping of solutions to attacks

Attack - Technique	ρ_{max}	k_{var}	*z-F*	$P(iBF)$
Packet storms	+	+		
Loops	+		+	+
Flow duplication	+		+	+
Injection	+		+	
Correlation			+	+
Replay			+	

3.7 Security

In this Section, we analyze the security of Bloom filter based multicast in the context of infrastructure availability and node mobility.

3.7.1 Multicast

We now analyze how BloomCast mitigates the security threats against Bloom filter-based multicast forwarding as described in Publication V and Publication VIII. We focus on malicious host-initiated attacks. Table 3.2 presents an overview of the mapping between the available techniques and the attacks addressed. As can be seen, BloomCast uses four techniques to prevent the six security threats described in Section 2.2.4.

Packet storms are prevented with the combination of limiting the maximum fill factor ρ_{max} and the *varying* k_{var} *technique*. Globally enforced ρ_{max} values enable each router to compute k_{var} locally so that every Bloom filter with a valid fill factor produces, on average, less than 1 false positive. Since BloomCast collects the Bloom filters on path with the `BC_JOIN` packet, it is easy to set k_{var} locally. Additionally, this optimization of k reduces the actual false positive rate as shown in Publication VII.

Loops are a serious threat to any network infrastructure. In BloomCast, the combination of maximum fill factor ρ and z-Formation makes it difficult for an attacker to construct looping Bloom filters. The first removes the easy attack of just adding bits into the Bloom filter until every link matches and the z-Formation ensures that guessing the right Bloom filter is difficult (see Publication V) for details).

To prevent accidental loops, each router performs a bit permutation on the Bloom filter before performing the outport matching – when using

the Bloom filter for forwarding (and after matching – when collecting a Bloom filter). If a packet does go through a loop, either because of a false positive or a malicious source, the Bloom filter has been modified with a random permutation (a product of the permutations performed by the set of routers participating in the loop).

Using permutations ensures a high probability that the packet will not continue looping and that it will not be forwarded to the downstream tree for a second, or nth time. As an example, the chances of an infinite loop in a three node loop configuration with $\rho = 0.5$, $k = 6$, and $m = 256$ are in the order of $O(10^{-12})$. The chances that a packet will be forwarded through the subtree once are ρ^K, where $K = \sum k_i$ is the sum of all hash functions used in the subtree.

Flow duplication: Similarly to loops, the flow duplication can be effectively prevented with the combination of restricting fill factor ρ, edge-pair labels, and per hop bit permutations. The result gives an attacker ρ^K probability of creating a specific subtree by accident.

Packet injection attacks, correlation attacks, and replay attacks can be efficiently prevented using the z-Formation technique as described in Publication V. It uses cryptographically secure edge-pair labels. Since the Bloom filter is constructed using these dynamic edge-pair labels, the resulting Bloom filter becomes additionally bound to the IP source and destination pair, a specific time period, and the path (because the input port is used for edge-pair label computation and per hop bit permutations on Bloom filters). This makes it impossible to share iBFs from different points of network, at different time instants, or to different destinations.

Finally, as Z takes in both the *outgoing* and *incoming* interface indices as inputs, any given Bloom filter is tightly bound to the corresponding forwarding path or delivery tree. That is, this feature blocks the injection attack, preventing off-path attackers from sending data towards a delivery tree even if they know both the Flow ID and the Bloom filter.

Consequently, the best strategy for a successful *packet injection attack* is reduced to a brute force attack consisting of generating random labels and hoping that at least one of them reaches the target(s). An attacker needs malformed iBFs to cause h consecutive false positives to get packets forwarded through a valid iBF path of length h. The chances of success are shown in Figure 3.8. In a single attempt the chance of success is $p = {\rho_{max}}^{h \cdot k}$, which is very low for typical configurations (e.g., $p = 2^{-36}$ for $h = 4, k = 8, \rho = 0.5$, i.e., over 10^{10} attempts are required for a 0.5 probability

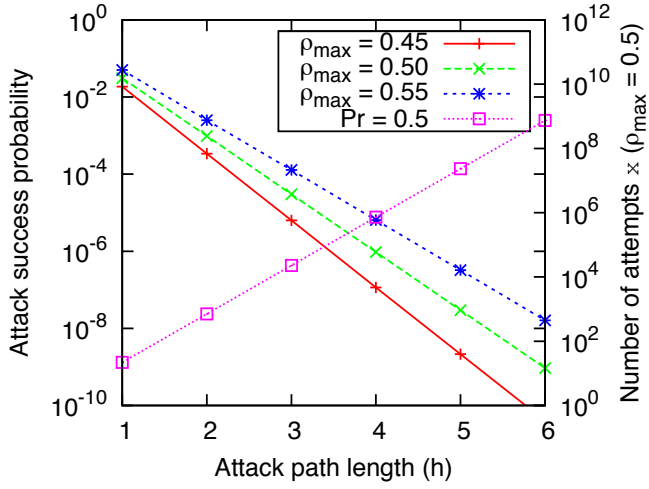


Figure 3.8. On the left axis, attack success probability for different ρ_{max}. On the right axis, the line with square points represents the attempts required to success with probability 1/2.

successful attack). Such brute force attacks can be easily detected, rate limited and pushed back, for instance after the false positive rate from a given source exceeds some threshold. Additionally, a forged iBF would work through the target path attack only as long as the period of validity for the secret keys the routers use. Also filtering based approaches become simpler, since the attacker cannot change the IP source and destination pair in the packet without making the Bloom filter invalid.

Source and receiver control: As the group management in BloomCast is end-to-end, the source has control over the receivers it accepts. If it wishes to, it can require receiver authentication before adding a receiver into the group. Similarly, multicasting to a receiver requires knowing the iBF that forms the path between source and destination. Since the iBF is cryptographically bound to (S,G), each router's secret key, and the path (via permutations and edge-pair labels), guessing an iBF for a path is difficult, as shown above.

Resource consumption attacks against the memory and processing capacity of routers do not become easier than they are in unicast forwarding. The routers do not need to maintain multicast state and the iBF collection and forwarding processing can be done at line speed in hardware and in parallel for each peer. The multicast source needs to maintain state for receivers. This is a needed feature, since this makes it possible for a source to exert control over who can and who cannot join the multicast

group. Simultaneously, it leaves the source vulnerable to an attacker who creates a storm of multicast join packets. A source can use a receiver authentication protocol, which pushes the authentication protocol state to the initiator (e.g., the base exchange of Host Identity Protocol [90] could be used for that purpose) to limit the state requirements to authenticated receivers.

False positive forwarded packets may compromise the *ephemeral secrecy* of the multicast data to non-group-members, i.e., some packets may reach unintended destinations. The time- and bit-varying iBFs contribute to spreading falsely forwarded multicast packets across different links over time, preventing thus a complete reception of a multicast packet flow.[11]

Anonymity of source is not an option in source specific multicast, since the group is identified with combination (S,G) where S is the sender address and G the group address. However, even though the protocol uses source routing, the actual paths, or nodes on path are not revealed to the source and the source can only use the related iBFs in combination with traffic destined to (S,G).

Receivers do not need to reveal their identifies or addresses to the network, or the source – the receiver (IP) address is not necessary in the protocol. The authentication, should the source require it, can be done end-to-end without revealing the identities to the intermediate routers. As the keys used to compute iBFs are changed periodically, correlation attacks between two or more Bloom filters used at different times become impossible. Similarly, since the edge-pair labels are tied to group identifier (S,G), an attacker cannot use a set of iBFs with different group addresses to determine whether the set contains one or more common receivers. These techniques effectively prevent traffic analysis and related vulnerabilities such as clogging attacks (cf. [10]).

3.7.2 Mobility

The core part of the BloomCasting technique is to bind the communication channel between peers, not only to IP-addresses, but also to the forwarding path between them. In this Section, we address the security aspects most important for mobility: hand-off security, hand-off latency and DoS vulnerability.

[11] *Data authenticity* is out of scope of the iBF forwarding service and can be provided by orthogonal security policies and group key management techniques (e.g., following the guidelines of [54]).

Mobile handoff has two main security requirements. The corresponding node has to know that the handoff message is authentic, i.e. sent by the mobile node (or someone authorized by the mobile node) and that the mobile node is indeed reachable from the address it claims to reside in. Without authentication an attacker can impersonate a mobile node and divert traffic to itself and without reachability test a mobile node can divert the traffic to a location it does not reside in, enabling a DoS attack.

In most cases, when CN receives a location update, the collected iBF and source address suffice and it does not need to make an additional reachability test. This minimizes latency during hand-offs since CN can continue sending packets to MN after receiving a single location update - the M2 message in the location update can be piggybacked in payload traffic. Existing protocols such as MIPv6 and HIP use 1.5 RTT to achieve the same.

The iBF determines the path to the destination AS. Earlier work [39] has shown that creating a valid iBF for a path without access to the secret keys is difficult. The protocol still allows MN to spoof its IP address, but only within the local domain it is located in. This provides incentives for ASes to deploy source address validation in their networks. If the AS level path changes, either node needs to renew the iBF using a location update. A return routability test is needed, if CN itself intends to move, because the security relies on the iBF that describes the path between CN and MN.

With the current mobility architectures, the mobile nodes must establish security associations with their peers. For example in MIPv6, MN needs to establish a security association with its home-agent, while in HIP, the peers establish a security association between each other. The iBF-based forwarding only requires a weak security association, based on hash chains, between the peers. The iBF-based forwarding could be coupled with existing IP-based mobility protocols. The mobility protocols can be optimized to utilize the security provided by iBF-based forwarding, or the forwarding fabric can be transparent to these protocols. Further work is needed to better understand the tradeoffs.

Figure 3.9 compares the *bindings* used in iBF mobility with MIP and HIP. The process is at the top, the forwarding identifiers on the bottom. In the case of MIP and HIP, the forwarding is bound to CoA, which is a single point in the network. Because of this, each solution needs a separate mechanism to verify that the node is actually where it claims to be. In

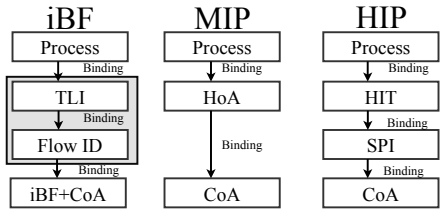


Figure 3.9. Comparing bindings between iBF, MIP and HIP

MIP, the verification requires return routability testing both directly and through the home agent, since there is no security association between MN and CN. In HIP, the verification is done directly using the security association between MN and CN. In the iBF based mobility, the packet forwarding is bound to the domain path (and CoA). The network does the binding to the domain level path and the weak security association between MN and CN ensures that other nodes cannot spoof the mobility signaling.

This binding of the flow to a network path has also benefits against denial-of-service attacks. As the path binding is done by the network, a faulty implementation in a host (e.g. web server) cannot be used for reflection attacks. It also makes source address validation more effective, as it is enough for the AS where the MN is located to validate source addresses to prevent IP address spoofing.

The binding between *Transport Layer Identifier (TLI)* and Flow ID can also enable the mobile node to seamlessly move between heterogenous networks. If the Flow ID is separated from the end point addresses, then the connection can be continued even as the node moves between IPv4, IPv6, and other types of networks.

4. Conclusions

In this thesis, we have proposed a publish/subscribe based network architecture for the Internet. While the PSIRP architecture is a pure clean slate architecture, our goal has been also to fit the architecture together with the existing Internet architecture. Our architecture is composed of rendezvous, Bloom filter based multicast forwarding, and the IP as a signaling mechanism.

The rendezvous system is composed of a hierarchical policy routing like component and an overlay using a hierarchical distributed hash table. This division enables incremental deployment, since locally each enterprise or ISP can peer with their neighbors and buy transit for rendezvous from their provider. It also enables those providers at the top of their local hierarchy to interconnect without requiring all the state to be shared between all participants. This is important. With full deployment, the number of operators that need to hold the full view of the routing table is relatively small, whereas in a scenario of incremental deployment, it may be much larger and contain many more small ISPs.

Combining hierarchical policy routing and an overlay using hierarchical distributed hash table, together with caching, allows the rendezvous system to scale and retain efficient route by name, while also ensuring that queries are routed only through network operators. Our evaluation shows that for large majority of cases (>95%) the latency from the rendezvous is less than 250ms and the utilization of rendezvous nodes is kept relatively low even in the most heavily loaded nodes.

Our work on Bloom filter based forwarding has shown that there are serious reliability and security problems in the earlier Bloom filter based multicast architectures and that these problems can be solved. The nature of false positives in Bloom filter based forwarding can cause serious anomalies in the network that can be used in an attack against the

network infrastructure or target end points. The anomalies are packet storms, in which consecutive false positives cause an exponential growth of traffic; forwarding loops that cause each packet in a flow to loop infinitely and an additional copy to the downstream receivers for each loop the packet takes; and flow duplication, which happens if false positives causes the path to intersect with some other part of the forwarding tree.

We have shown that these threats can be countered with four techniques that do not require per packet or per flow state in routers: (1) limiting the fill factor of a Bloom filter, (2) using cryptographically secure labels to denote path elements that are inserted into the Bloom filter, (3) varying the Bloom filter locally in the router parameters to ensure that the average number of false positives is low enough to prevent packet storms, and (4) use bit permutations on every router on the Bloom filter to carry information about the path the packet has traversed, and hence prevent the harmful side effects of loops and flow duplication.

The problem of building networks where information is a first class citizen is of importance for the future, as the amount of information created and consumed increases. In this work, we have worked on building such an architecture based on information centric anycast for locating information and stateless Bloom filter based multicast for data delivery. Still many open questions remain for future work. These include a comprehensive evaluation of the scalability of Bloom filter based forwarding and comparison to the stateful multicast techniques, as well as the potential for scaling up Bloom filter based multicast with the use of some state in the network. The deployment incentives and security of the rendezvous and Bloom filter based multicast also requires further study.

Bibliography

[1] A. Adams, J. Nicholas, and W. Siadak. 2005. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised). RFC 3973 (Experimental).

[2] Daniel Adkins, Karthik Lakshminarayanan, Adrian Perrig, and Ion Stoica. 2003. Towards a More Functional and Secure Network Infrastructure. Technical Report UCB/CSD-03-1242, Univ. California, Berkeley.

[3] David Andersen. 2003. Mayday: Distributed filtering for internet services. USITS: 4th USENIX Symposium on Internet Technologies and Systems .

[4] David Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. 2007. Holding the Internet Accountable. ACM Hotnets-VI .

[5] Tom Anderson, Timothy Roscoe, and David Wetherall. 2004. Preventing Internet denial-of-service with capabilities. Hotnets II pages 39–44.

[6] Katrina Argyraki and David Cheriton. 2005. Network capabilities: The good, the bad and the ugly. ACM HotNets-IV .

[7] Katrina Argyraki and David Cheriton. 2009. Scalable network-layer defense against internet bandwidth-flooding attacks. IEEE/ACM Transactions on Networking (TON) 17, no. 4, pages 1284–1297.

[8] Katrina Argyraki and DR Cheriton. 2005. Active internet traffic filtering: Real-time response to denial-of-service attacks. Usenix .

[9] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. 2009. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[10] Adam Back, Ulf Möller, and Anton Stiglic. 2001. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In: IHW '01: Proceedings of the 4th International Workshop on Information Hiding, pages 245–257. Springer-Verlag, London, UK. ISBN 3-540-42733-3.

[11] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. 2004. A layered naming architecture for the internet. In: SIGCOMM '04, pages 343–352. ACM, New York, NY, USA. ISBN 1-58113-862-8.

[12] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. 2005. Off by default. ACM HotNets IV .

[13] T. Ballardie and J. Crowcroft. 1995. Multicast-specific security threats and counter-measures. In: SNDSS '95: Proceedings of the 1995 Symposium on Network and Distributed System Security (SNDSS'95), page 2. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-7027-4.

[14] Tony Ballardie, Paul Francis, and Jon Crowcroft. 1993. Core based trees (CBT). SIGCOMM Comput. Commun. Rev. 23, no. 4, pages 85–95.

[15] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. 2007. Multiprotocol Extensions for BGP-4. RFC 4760 (Draft Standard).

[16] T. Bates, Y. Rekhter, R. Chandra, and D. Katz. 2000. Multiprotocol Extensions for BGP-4. RFC 2858 (Proposed Standard). Obsoleted by RFC 4760.

[17] N. Bhaskar, A. Gall, J. Lingard, and S. Venaas. 2008. Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM). RFC 5059 (Proposed Standard).

[18] S. Bhattacharyya. 2003. An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational).

[19] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13, no. 7, pages 422–426.

[20] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms. 2007. Explicit Multicast (Xcast) Concepts and Options. RFC 5058 (Experimental).

[21] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. 2005. Policing congestion response in an internetwork using re-feedback. In: Proceedings of the ACM Sigcomm'05, pages 277–288. ACM, New York, NY, USA.

[22] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. 2006. ROFL: routing on flat labels. In: Proceedings of the ACM Sigcomm'06, pages 363–374. ACM.

[23] A. Campbell, J. Gomez, S. Kimand A. Valko, C. Wan, and Z. Turanyi. 2000. Design, implementation, and evaluation of Cellular IP. IEEE Personal Commun. Mag. 7, no. 4.

[24] R. Canetti and B. Pinkas. 2000. A taxonomy of multicast security issues. IRTF Internet-Draft (draft-irtf-smug-taxonomy-01).

[25] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. 2001. Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19, no. 3, pages 332–383.

[26] C. Castelluccia. 2000. HMIPv6: A Hierarchical Mobile IPv6 Proposal. ACM Mobile Computing and Communication Review (MC2R) 4, pages 48 – 59.

[27] Wu chang Feng and E Kaiser. 2007. The Case for Public Work. IEEE Global Internet Symposium pages 43–48.

[28] D. Trossen (ed.). 2008. Architecture Definition, Component Descriptions, and Requirements (D2.3). Technical report. http://psirp.org/publications.

[29] D. Trossen (ed.). 2008. Conceptual architecture of PSIRP including subcomponent descriptions (D2.2). Technical report. http://psirp.org/ publications.

[30] D. Trossen (ed.). 2009. Update on the Architecture and Report on Security Analysis (D2.4). Technical report. http://psirp.org/publications.

[31] S.E. Deering. 1991. Multicast routing in a datagram internetwork. Technical Report STAN-CS-92-1415, Stanford University.

[32] S.E. Deering and D.R. Cheriton. 1990. Multicast routing in datagram internetworks and extended LANs. ACM Transactions on Computer Systems (TOCS) 8, no. 2, pages 85–110.

[33] L.P. Deutsch. 1973. Host names on-line. RFC 606.

[34] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. 2000. Deployment issues for the IP multicast service and architecture. IEEE Network 14, no. 1, pages 78–88.

[35] Christophe Diot, Walid Dabbous, and Jon Crowcroft. 1997. Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms. IEEE Journal on Selected Areas in Communications 15, no. 3, pages 277–290.

[36] C Dixon and T Anderson. 2008. Phalanx: Withstanding multimillion-node botnets. Usenix NSDI .

[37] D. Eastlake 3rd and P. Jones. 2001. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational). Updated by RFC 4634.

[38] A. Ermolinskiy. 2007. The Design and Implementation of Free Riding Multicast. Master's thesis, University of California, Berkeley.

[39] Christian Esteve, Petri Jokela, Pekka Nikander, Mikko Särelä, and Jukka Ylitalo. 2009. Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters. Proceedings of European Conference on Computer Network Defence (EC2ND) .

[40] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne Marie Kermarrec. 2003. The many faces of publish/subscribe. ACM Comput. Surv. 35, no. 2, pages 114–131.

[41] D. Farinacci and Y. Cai. 2006. Anycast-RP Using Protocol Independent Multicast (PIM). RFC 4610 (Proposed Standard).

[42] A. Farrel, J.-P. Vasseur, and J. Ash. 2006. A Path Computation Element (PCE)-Based Architecture. RFC 4655 (Informational).

[43] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. 2006. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601 (Proposed Standard). Updated by RFCs 5059, 5796.

[44] B. Fenner and D. Meyer. 2003. Multicast Source Discovery Protocol (MSDP). RFC 3618 (Experimental).

[45] P. Ferguson and D. Senie. 2000. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice). Updated by RFC 3704.

[46] Paul Francis. 2004. Firebreak: An IP Perimeter Defense Architecture.

[47] V. Fuller, T. Li, J. Yu, and K. Varadhan. 1993. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519 (Proposed Standard). Obsoleted by RFC 4632.

[48] Prasanna Ganesan, Krishna Gummadi, and Hector Garcia-Molina. 2004. Canon in G Major: Designing DHTs with Hierarchical Structure. IEEE Distributed Computing Systems (ICDCS'04). Proceedings pages 263–272.

[49] L. Gao. 2001. On inferring autonomous system relationships in the Internet. IEEE/ACM Transactions On Networking 9, no. 6, pages 733–745.

[50] M. Gritter and D.R. Cheriton. 2001. An architecture for content routing support in the internet. In: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems-Volume 3, page 4.

[51] Mark Handley and Adam Greenhalgh. 2004. Steps towards a DoS-resistant internet architecture. In: FDNA '04: ACM SIGCOMM workshop on Future directions in network architecture, pages 49–56. ACM. ISBN 1-58113-942-9.

[52] T. Hardie. 2002. Distributing Authoritative Name Servers via Shared Unicast Addresses. RFC 3258 (Informational).

[53] T. Hardjono, R. Canetti, M. Baugher, and P. Dinsmore. 2000. Secure IP Multicast: Problem areas, Framework, and Building Blocks. IRTF Internet-Draft (draft-irtf-smug-framework-01).

[54] T. Hardjono and B. Weis. 2004. The Multicast Group Security Architecture. RFC 3740 (Informational).

[55] Y. Hilewitz, Z.J. Shi, and R.B. Lee. 2004. Comparing fast implementations of bit permutation instructions. In: Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on, volume 2, pages 1856–1863. IEEE. ISBN 0780386221.

[56] Y.C. Hu, A. Perrig, and D.B. Johnson. 2005. Ariadne: A secure on-demand routing protocol for ad hoc networks. Wireless Networks 11, no. 1, pages 21–38.

[57] F Huici and M Handley. 2007. An edge-to-edge filtering architecture against DoS. SIGCOMM CCR .

[58] J Ioannidis and SM Bellovin. 2002. Implementing pushback: Router-based defense against DDoS attacks. In Proceedings of NDSS .

[59] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. 2009. Networking named content. In: Proceedings of the 5th international conference on Emerging networking experiments and technologies, pages 1–12. ACM.

[60] Van Jacobson. A new way to look at networking. http://video.google.com/videoplay?docid=-6972678839686672840.

[61] D. Johnson, C. Perkins, and J. Arkko. 2004. Mobility Support in IPv6. RFC 3775 (Proposed Standard).

[62] Petri Jokela, Andras Zahemszky, Christian Esteve, Somaya Arianfar, and Pekka Nikander. 2009. LIPSIN: Line speed Publish/Subscribe Inter-Networking. In: SIGCOMM.

[63] P. Judge and M. Ammar. 2002. Gothic: a group access control architecture for secure multicast and anycast. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1547 – 1556 vol.3.

[64] Paul Judge and Mostafa Ammar. 2003. Security issues and solutions in multicast content distribution: A survey. IEEE Network 17, pages 30–36.

[65] Angelos Keromytis, Vishal Misra, and Dan Rubenstein. 2002. SOS: secure overlay services. SIGCOMM 32, no. 4, pages 61–72.

[66] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci. 2003. Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP). RFC 3446 (Informational).

[67] L. Kleinrock and F. Kamoun. 1977. Hierarchical routing for large networks Performance evaluation and optimization. Computer Networks 1, no. 3, page 155.

[68] R. Koodli. 2008. Mobile IPv6 Fast Handovers. RFC 5268 (Proposed Standard). Obsoleted by RFC 5568.

[69] T Koponen, M Chawla, BG Chun, A Ermolinskiy, KH Kim, S Shenker, and I Stoica. 2007. A data-oriented (and beyond) network architecture. Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications pages 181–192.

[70] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. 2007. A data-oriented (and beyond) network architecture. In: SIGCOMM '07, pages 181–192. ISBN 978-1-59593-713-1.

[71] H. Krawczyk. 1994. LFSR-based hashing and authentication. In: Advances in Cryptology CRYPTO'94, pages 129–139. Springer.

[72] Christian Kreibich, Andrew Warfield, Jon Crowcroft, Steven Hand, and Ian Pratt. 2005. Using Packet Symmetry to Curtail Malicious Traffic. Hotnets-IV .

[73] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. 2000. Oceanstore: An architecture for global-scale persistent storage. ACM SIGARCH Computer Architecture News 28, no. 5, pages 190–201.

[74] M.D. Kudlick. 1974. Host names on-line. RFC 608. Obsoleted by RFC 810.

[75] J Kurian and K Sarac. 2006. FONet: A federated overlay network for DoS defense in the internet. Proceedings of Global Internet Symposium .

[76] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. 2004. Taming IP Packet Flooding Attacks. Sigcomm Computer Communication Review 34, no. 1, pages 45–50.

[77] L. Lamport. 1981. Password authentication with insecure communication. Communications of the ACM 24, no. 11, pages 770–772.

[78] D. Le, X. Fu, and D. Hogrefe. 2006. A review of mobility support paradigms for the internet. IEEE Communications Surveys & Tutorials 8, no. 1, pages 38–51.

[79] S.J. Leffler, R.S. Fabry, W.N. Joy, P. Lapsley, S. Miller, and C. Torek. 1986. An Advanced 4.4 BSD Interprocess Communication Tutorial. Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley .

[80] X Liu, X Yang, and Y Lu. 2008. To filter or to authorize: network-layer DoS defense against multimillion-node botnets. Proceedings of the ACM SIGCOMM 2008 conference on Data communication pages 195–206.

[81] Y Lu, B Prabhakar, and F Bonomi. 2005. Bloom Filters: Design Innovations and Novel Applications. In: In Proc. of the Forty-Third Annual Allerton Conference.

[82] A Mahimkar, J Dange, V Shmatikov, H Vin, and Y Zhang. 2007. dFence: Transparent network-based denial of service mitigation. NSDI'07 .

[83] P. Maymounkov and D. Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. Peer-to-Peer Systems pages 53–65.

[84] D. Mazieres, M. Kaminsky, M.F. Kaashoek, and E. Witchel. 1999. Separating key management from file system security. ACM SIGOPS Operating Systems Review 33, no. 5, pages 124–139.

[85] Jelena Mirkovic and Peter Reiher. 2004. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. Sigcomm Computer Communication Review 34, no. 2, pages 39–53.

[86] P.V. Mockapetris. 1983. Domain names: Concepts and facilities. RFC 882. URL `http://www.ietf.org/rfc/rfc882.txt`. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[87] P.V. Mockapetris. 1983. Domain names: Implementation specification. RFC 883. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[88] J.C. Mogul. 1984. Internet subnets. RFC 917.

[89] J.C. Mogul and J. Postel. 1985. Internet Standard Subnetting Procedure. RFC 950 (Standard).

[90] R. Moskowitz and P. Nikander. 2006. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational).

[91] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. 2008. Host Identity Protocol. RFC 5201 (Experimental).

[92] J. Moy. 1998. OSPF Version 2. RFC 2328 (Standard). Updated by RFC 5709.

[93] M.J. Moyer, J.R. Rao, and P. Rohatgi. 1999. A survey of security issues in multicast communications. IEEE network 13, no. 6, pages 12–23.

[94] K. Nagami, Y. Katsube, Y. Shobatake, A. Mogi, S. Matsuzawa, T. Jinmei, and H. Esaki. 1997. Toshiba's Flow Attribute Notification Protocol (FANP) Specification. RFC 2129 (Informational).

[95] P. Nikander, J. Arkko, and B. Ohlman. 2004. Host Indentity Indirection Infrastructure (Hi3). In: Proc. of SNCNW.

[96] B Parno, D Wendlandt, E Shi, A Perrig, B Maggs, and YC Hu. 2007. Portcullis: Protecting connection setup from denial-of-capability attacks. In: ACM Sigcomm.

[97] C. Partridge, T. Mendez, and W. Milliken. 1993. Host Anycasting Service. RFC 1546 (Informational).

[98] Pragyansmita Paul and S. V. Raghavan. 2002. Survey of multicast routing algorithms and protocols. In: ICCC '02: Proceedings of the 15th international conference on Computer communication, pages 902–926. International Council for Computer Communication, Washington, DC, USA. ISBN 1-891365-08-8.

[99] C. Perkins. 2002. IP Mobility Support for IPv4. RFC 3344 (Proposed Standard). Updated by RFC 4721.

[100] A. Perrig, R. Canetti, JD Tygar, and D. Song. 2002. The TESLA broadcast authentication protocol. RSA CryptoBytes 5, no. 2, pages 2–13.

[101] J. Postel. 1980. User Datagram Protocol. RFC 768 (Standard).

[102] J. Postel. 1981. Internet Protocol. RFC 791 (Standard). Updated by RFC 1349.

[103] J. Postel. 1981. Transmission Control Protocol. RFC 793 (Standard). Updated by RFCs 1122, 3168.

[104] PSIRP. PSIRP Homepage, available at: http://wiki.psirp.org.

[105] P.I. Radoslavov, D. Estrin, and R. Govindan. 1999. Exploiting the bandwidth memory tradeoff in multicast state aggregation. Technical Report USC/99-697, USC Computer Science Department.

[106] S. Rafaeli and D. Hutchison. 2003. A survey of key management for secure group communication. ACM Computing Surveys (CSUR) 35, no. 3, page 329.

[107] Barath Raghavan and Alex C Snoeren. 2006. Decongestion Control. Hotnets-V .

[108] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering. 2004. IPv6 Flow Label Specification. RFC 3697 (Proposed Standard).

[109] K. Ramakrishnan, S. Floyd, D. Black, et al. 2001. The addition of explicit congestion notification (ECN) to IP.

[110] V. Ramasubramanian and E.G. Sirer. 2004. The design and implementation of a next generation name service for the Internet. In: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pages 331–342. ACM.

[111] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. 2006. Revisiting IP multicast. In: ACM SIGCOMM. ACM.

[112] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. 2001. A scalable content-addressable network. In: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, page 172. ACM.

[113] Y. Rekhter, T. Li, and S. Hares. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard).

[114] R.L. Rivest and B. Lampson. 1996. SDSI-a simple distributed security infrastructure. Technical report, MIT.

[115] E. Rosen, A. Viswanathan, and R. Callon. 2001. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard).

[116] Christian Esteve Rothenberg, Carlos Macapuna, Fabio Verdi, and Mauricio Magalhães. 2010. The deletable Bloom filter: a new member of the Bloom family. IEEE Communications Letters 14, no. 6, pages 557 – 559. URL http://arxiv.org/abs/1005.0352.

[117] A. Rowstron and P. Druschel. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001. Proceedings, page 329. Springer.

[118] J. Saltzer et al. 1982. On the naming and binding of network destinations. In: Local Computer Networks, pages 311–317.

[119] P. Savola, R. Lehtonen, and D. Meyer. 2006. Protocol Independent Multicast - Sparse Mode (PIM-SM) Multicast Routing Security Issues and Enhancements. RFC 4609 (Informational).

[120] M Shaw. 2006. Leveraging good intentions to reduce unwanted network traffic. Proc. USENIX Steps to Reduce Unwanted Traffic on the Internet workshop .

[121] E Shi, I Stoica, D Andersen, and A Perrig. 2006. OverDoSe: A generic DDoS protection service using an overlay network. reportsarchive. adm. cs. cmu. edu/anon/2006/CMU-CS-06-114. ps .

[122] Z. Shi and R.B. Lee. 2000. Bit permutation instructions for accelerating software cryptography. In: Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on, pages 138–148. IEEE. ISBN 0769507166.

[123] Clay Shields and J. J. Garcia-Luna-Aceves. 1999. KHIP—a scalable protocol for secure multicast routing. In: SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, pages 53–64. ACM, New York, NY, USA. ISBN 1-58113-135-6.

[124] D.R. Simon, S. Agarwal, and D.A. Maltz. 2007. AS-based accountability as a cost-effective DDoS defense. In: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, page 9. USENIX Association.

[125] Angelos Stavrou and Angelos Keromytis. 2005. Countering DoS Attacks With Stateless Multipath Overlays. 12th ACM conference on Computer and communications security pages 249–259.

[126] D.R. Stinson. 2006. Cryptography: theory and practice. CRC press. ISBN 1584885084.

[127] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. 2002. Internet Indirection Infrastructure. In: Proc. of the ACM SIGCOMM'02, pages 73–88. Pittsburgh, PA, USA.

[128] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, page 160. ACM.

[129] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. 2002. Internet indirection infrastructure. Proceedings of the 2002 SIGCOMM conference 32, no. 4, pages 73–86.

[130] D. Thaler and M. Handley. 2002. On the aggregatability of multicast forwarding state. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1654–1663. IEEE.

[131] X. Tian, Y. Cheng, and B. Liu. 2009. Design of a scalable multicast scheme with an application-network cross-layer approach. IEEE Transactions on Multimedia 11, no. 6.

[132] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. 2003. CAPTCHA: Using hard AI problems for security. Advances in CryptologyÑEUROCRYPT 2003 pages 646–646.

[133] M Walfish, H Balakrishnan, D Karger, and S Shenker. 2005. DoS: Fighting fire with fire. 4th ACM Workshop on Hot Topics in Networks (HotNets) .

[134] M. Walfish, H. Balakrishnan, and S. Shenker. 2004. Untangling the Web from DNS. In: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1, page 17. USENIX Association.

[135] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. 2006. DDoS Defense by Offense. SIGCOMM pages 303–314.

[136] XF Wang and MK Reiter. 2003. Defending against denial-of-service attacks with puzzle auctions. Security and Privacy, 2003. Proceedings. 2003 Symposium on pages 78–92.

[137] XF Wang and MK Reiter. 2004. Mitigating bandwidth-exhaustion attacks using congestion puzzles. Proceedings of the 11th ACM conference on Computer and communications security pages 257–267.

[138] Brent Waters, John A Halderman, Ari Juels, and Edward W Felten. 2004. New Client Puzzle Outsourcing Techniques for DoS Resistance. 11th ACM conference on Computer and communications security pages 246–256.

[139] D Wendlandt, DG Andersen, and A Perrig. 2006. Fastpass: Providing first-packet delivery. Technical report CMU cylab .

[140] D Wendlant, DG Andersen, and A Perrig. 2009. Bypassing network flooding attacks using fastpass. Technical report, Carnegie Mellon University.

[141] J. Wu, J. Bi, X. Li, G. Ren, K. Xu, and M. Williams. 2008. A Source Address Validation Architecture (SAVA) Testbed and Deployment Experience. RFC 5210 (Experimental).

[142] J. Xia, L. Gao, and T. Fei. 2005. Flooding attacks by exploiting persistent forwarding loops. In: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, page 36. USENIX Association.

[143] J. Xia, L. Gao, and T. Fei. 2007. A measurement study of persistent forwarding loops on the Internet. Computer Networks 51, no. 17, pages 4780–4796.

[144] Abraham Yaar, Adrian Perrig, and Dawn Song. 2004. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. Security and Privacy pages 130–143.

[145] D.N. Yang and W. Liao. 2008. Protocol Design for Scalable and Adaptive Multicast for Group Communications. In: IEEE International Conference on Network Protocols, 2008. ICNP 2008, pages 33–42.

[146] X Yang, D Wetherall, and T Anderson. 2008. TVA: A DoS-limiting Network Architecture. IEEE/ACM Trans. on Networking 16, no. 6, pages 1267–1280.

[147] Xiaowei Yang, David Wetherall, and Thomas Anderson. 2005. A DoS-limiting Network Architecture. SIGCOMM .

[148] K. Yuksel, J.P. Kaps, and B. Sunar. 2004. Universal hash functions for emerging ultra-low-power networks. In: Proceedings of CNDS.

[149] Andras Zahemszky, Petri Jokela, Mikko Särelä, Sami Ruponen, James Kempf, and Pekka Nikander. 2010. MPSS: Multiprotocol Stateless Switching. In: Global Internet Symposium 2010.

[150] Ming Zhong, Pin Lu, Kai Shen, and Joel Seiferas. 2008. Optimizing data popularity conscious bloom filters. In: PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing, pages 355–364. ACM, New York, NY, USA. ISBN 978-1-59593-989-0.

Aalto-DD 49/2011

ISBN: 978-952-60-4149-0 (pdf)
ISBN: 978-952-60-4148-3
ISSN-L: 1799-4934
ISSN: 1799-4942 (pdf)
ISSN: 1799-4934

Aalto University
Name of the School
Department of Communications and Networking
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS