Multicast protocols traditionally require that routers store information about the delivery trees. Recently, source-routed in-packet Bloom-filter (iBF) based multicast has been proposed as a remedy to this: instead of storing state in the network, the delivery tree is encoded in the packet itself using a Bloom filter. The packets are then forwarded based on the in-packet information instead of requiring that per-delivery tree state is stored in the network. It is believed that moving the state from the network to the packets makes Internet-wide multicast feasible.

In-packet Bloom filter (iBF) forwarding has also been advocated as a secure forwarding solution. It is, for example, believed that the architecture makes it impossible to send traffic to a receiver who has not explicitly allowed traffic from the particular source. This has led to claims that iBF forwarding would be inherently secure to distributed denial of service attacks. The security of iBF forwarding, however, has not yet been studied and these claims have not yet been proven.

This thesis takes the first steps in the security analysis of iBF forwarding. The goal is twofold: (1) to find out what information a malicious entity controlling a botnet can get about the network, and (2) to determine how this knowledge could be used to launch attacks against availability of some network node. The analysis is done using analytical methods and high level simulations.

The main result of this thesis is to show that the security of iBF forwarding has been exaggerated in the literature: iBF forwarding does not have inherent protection against DDoS attacks. The thesis formulates attacks that allow an attacker controlling a botnet to send unsolicited traffic to the intended target. It is also shown that the security mechanisms proposed for iBF forwarding do not give full protection against these attacks. While this does not mean that iBF forwarding is fundamentally insecure, the found attacks reveal that there is a true need for further security research of iBF forwarding.
I am grateful to Tuomas Aura for providing the funding for this thesis. I sincerely hope that the investment turned out to be worth it.

I want to thank both my supervisor Tuomas Aura and my instructor Mikko Särelä for all the time, advice, and feedback they gave me. Their guidance truly helped me in this process.

Finally, I would like to thank Peter Sjödin whose comments helped improving the thesis.

Espoo, June 30, 2011

Markku Antikainen
Abbreviations and Acronyms

AIP       Accountable Internet Protocol [3]
AS        Autonomous System
ASN       Autonomous System Number
BF        Bloom filter
BGP       Border Gateway Protocol [27]
CAIDA     The Cooperative Association for Internet Data Analysis
CDF       Cumulative Distribution Function
DoS       Denial of service
DDoS      Distributed denial of service
FHID      Forwarding-hop identifier
FRM       Free Riding Multicast [26]
iBF       in-packet Bloom filter
LIPSIN    Line Speed Publish/Subscribe Inter-networking [16]
LANES     Logical Address space Network Extensions [37]
MPLS      Multi-Protocol Label Switching [28]
MPSS      Multi-Protocol Stateless Switching [38]
PDF       Probability Distribution Function
Variables and Functions

Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_x$</td>
<td>Attacker controlled node $x$ (i.e. a bot)</td>
</tr>
<tr>
<td>$B_x$</td>
<td>Bloom filter $x$</td>
</tr>
<tr>
<td>$b_i$</td>
<td>The value of the $i$th bit in a bit vector</td>
</tr>
<tr>
<td>$fpr$</td>
<td>False positive rate (i.e the probability of a false positive)</td>
</tr>
<tr>
<td>$m$</td>
<td>Bloom filter length in bits</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of hash functions applied to an element that is added to a Bloom filter</td>
</tr>
<tr>
<td>$T$</td>
<td>Target of the attack</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Mean</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Bloom filter fill ratio</td>
</tr>
<tr>
<td>$\sigma^2$</td>
<td>Variance</td>
</tr>
</tbody>
</table>

Functions

- $Hyp(\mu, \sigma^2)$: Hypergeometric distribution with mean $\mu$ and variance $\sigma^2$.
- $N(\mu, \sigma^2)$: Normal distribution with mean $\mu$ and variance $\sigma^2$.
- $\pi(B)$: Permutation function that permutes the bit vector $B$ (i.e., a P-box).
- $Z(\ldots)$: $Z$-function that is used to generate FHIDs. See Section 2.5.2.
# Contents

Abbreviations and Acronyms iv  
Variables and Functions v  
Unrelated Quotation viii  

## 1 Introduction
1.1 Problem statement and methodology 2  
1.2 Scope of the thesis 2  
1.3 Structure of the thesis 3  

## 2 Background
2.1 Bloom filters 4  
2.1.1 Basic operations 4  
2.1.2 False positive rate 5  
2.1.3 Fill factor $\rho$ 7  
2.1.4 About the BF notation used in this thesis 7  
2.2 Bloom-filter multicast forwarding 8  
2.3 Forwarding-hop identifiers 12  
2.4 Previously known forwarding anomalies and DoS problems 13  
2.4.1 Chain reaction anomalies 13  
2.4.2 Target path attacks 14  
2.5 Proposed mechanisms to increase DoS resistance 16  
2.5.1 Limits on fill factor 16  
2.5.2 Cryptographic and flow specific FHIDs 16  
2.5.3 Global and local $k$ 17  
2.5.4 Permutations 18  
2.5.5 Summary of the proposed iBF variants 21  

## 3 Security evaluation 22  
3.1 Connectivity graph 23  
3.2 Correlation attack 26
3.2.1 Attack model ........................................... 26
3.2.2 The Attack ............................................. 27
3.2.3 Evaluation .............................................. 29
3.3 Injection attack .......................................... 35
3.3.1 Attack model ........................................... 35
3.3.2 Acquiring information about a permutation ............ 36
3.3.3 Performing the injection ................................ 38
3.3.4 Chaining the injection attacks ......................... 40
3.3.5 Evaluation .............................................. 40
3.4 Injection attack with unknown topology .................... 44
3.4.1 Attack model ........................................... 44
3.4.2 The Attack ............................................. 44
3.4.3 Evaluation .............................................. 49
3.5 Injection attack when permutations change as often as FHIDs 51
3.5.1 Attack model ........................................... 51
3.5.2 The attack ............................................. 51
3.6 Injection attack and flow specific FHIDs .................... 53

4 Discussion .................................................. 55
4.1 Proposed forwarding architectures and their resistance against injection attack ........................................... 56
4.2 Implications of the described attacks ...................... 56
4.3 Evaluation of the methodology ............................ 58

5 Conclusion .................................................. 59

Bibliography ................................................... 61

A CAIDA policy annotated AS-graph .......................... 65

B Derivation of Equation 3.2 ................................ 67
B.1 Problem .................................................. 67
B.2 Solution .................................................. 67
“It is always possible to aglutenate multiple separate problems into a single complex interdependent solution. In most cases this is a bad idea.”

The 5th Fundamental Truth of Networking [7]
Chapter 1

Introduction

The dominant communication pattern in the current Internet is one-to-one messaging. Even though the one-to-one messaging, called unicast, fulfils the needs of most network applications, it is not well suitable for applications such as live video streaming where the same information needs to be delivered to multiple receivers. The problem is that when the same packet is sent to several receivers, multiple copies of the same packet are transmitted over the same links, which leads to inefficient network usage and increased congestion. This problem can be solved by using a one-to-many communication pattern, i.e. single-origin multicast, which achieves the same as several unicast transmissions. In multicast messaging, it is sufficient to send just one copy of the packet to the network. The network then replicates the packet so that each receiver receives a copy. This messaging pattern reduces the network overhead especially near the sender and in the core networks because multiple copies of the packet are not sent over any link. [9, 24, 25]

Multicast has been a hardy perennial among the network research topics for over two decades [9, 24, 25]. One of the problems with many of the multicast proposals is that they require a state in the network: routers need to store information about the delivery trees or about the recipients of the multicast transmission. Recently, source-routing with in-packet Bloom filters (iBF) has been proposed as a remedy to this problem (e.g. [16, 33, 38]). In these proposals, the multicast delivery tree is encoded into a probabilistic space-efficient data structure called Bloom filter, which is then included in the packets sent to the multicast group. It is believed that moving the state from the network to the packets will make Internet-wide multicast scalable to large number of groups [16].

The properties of Bloom filters make iBF forwarding different from many previously proposed multicast solutions [16, 25]. Firstly, a Bloom filter is a probabilistic data structure that allows false positives. In the context of iBF forwarding this means that a packet may also be forwarded to nodes that are
CHAPTER 1. INTRODUCTION

not part of the delivery tree. Secondly, even though the whole delivery tree is encoded in the packet, the receiver cannot use this information to determine the location of the sender or the other recipients. This discovery has led to claims that it is impossible to fabricate a valid Bloom-filter header, which describes the forwarding path, and that the only way of getting an address is to explicitly ask it from the recipient or from a trusted entity [29, 34]. Based on these claims, iBF forwarding has been advocated as a denial-of-service (DoS) resistant forwarding technology [29].

However, despite these rather bold claims, the security of iBF forwarding has not yet been studied thoroughly. Recent studies have only focused on how to make iBF forwarding resistant to accidental anomalies that stem from false positives in the forwarding [29, 32–34]. Potential problems caused by a creative malicious entity have not yet been studied in detail, which raises a question about the accuracy of the security claims.

1.1 Problem statement and methodology

This thesis studies attacks that a malicious entity can perform in an environment that utilizes in-packet Bloom-filter (iBF) forwarding. The goals of this thesis are threefold:

1. to analyse what information a malicious entity can get about the network by investigating the Bloom filters in the packets,
2. to study how this knowledge can be used to cause attacks against availability of some network element (i.e. a denial of service attack against a router, an end host, or a single link), and
3. to identify existing and possibly new mechanisms to mitigate the attacks.

The thesis will use analytical methods and simulations to evaluate the attacks and proposed protection methods.

1.2 Scope of the thesis

This thesis focuses on examining attacks against in-packet Bloom-filter forwarding and proposed security mechanisms. There is another family of forwarding protocols that utilize Bloom filters but where the delivery path is not encoded into a Bloom filter (e.g. [13]). These are not considered in this thesis.

Due to the fact that iBF forwarding research is still ongoing and the technology has not yet been standardized or deployed widely, the security analysis
is done using analytical tools and high level simulations. Additionally, even though the main focus of this thesis is to identify security weaknesses, the thesis will also discuss the implications of the attacks and how to protect against the attacks.

1.3 Structure of the thesis

The rest of this thesis is divided into three chapters. Chapter 2 starts by describing the background for the thesis including fundamentals of Bloom filters, in-packet Bloom-filter forwarding, and the security of Bloom filter forwarding. It also contains the literature survey and enumerates the differences in proposed iBF forwarding schemes. Chapter 3 goes into the actual security analysis and attacks. The chapter first defines a concept called connectivity graph, which will be used for generalizing the study of different types of iBF forwarding schemes. After this, the chapter introduces novel attacks which are feasible under increasingly tight security assumptions. Finally, Chapter 4 summarizes the findings, discusses the implications of the found attacks, and concludes the thesis.
Chapter 2

Background

2.1 Bloom filters

Bloom filter (BF) [4] is a probabilistic space-efficient data structure that can be used to verify whether an element is a member of a set. The structure offers a way to perform membership queries which may result in false positives (claiming falsely that an element is a member), but never false negatives (claiming falsely that an actual member is not a member). The basic operations of Bloom filters include membership testing and adding elements to the set. By default, Bloom filter does not support removing of elements. There are several different Bloom filter variants that enable also other operations, such as element removal [6, 36]. In this work, the main focus is on basic Bloom filters and these variants are not discussed in more detail.

2.1.1 Basic operations

Bloom filter \( B = (b_0, \ldots, b_{m-1}) \) is an array of \( m \) bits that represents a \( n \)-element set \( S = \{x_0, x_2, \ldots, x_{n-1}\} \). Initially, the empty filter is filled with 0s. An element can be mapped to the Bloom filter by utilizing \( k \geq 1 \) hash functions, each of which maps the element to an integer in range of \([0, m-1]\). An element is added to the Bloom filter by applying the hashes to the element one at the time, and by setting the bits at the resulting positions in the Bloom filter to 1. Similarly, the membership testing can be done by calculating the hashes of an element and checking whether all the resulting bit positions in the Bloom filter are set to 1. Algorithms 1 and 2 show the pseudocode for these operations.

Figure 2.1 gives a graphical illustration of the basic Bloom filter operations: initially, elements \( x \) and \( y \) are added to the Bloom filter by setting bits in positions 0,3,8,9 and 10. After this, the membership of elements \( z \) and \( w \)
CHAPTER 2. BACKGROUND

Figure 2.1: Basic Bloom filter operations. Initially, elements $x$ and $y$ are added to the Bloom filter. Afterwards, the membership of elements $z$ and $x$ is tested. However, the membership test of $w$ returns false positive.

Algorithm 1 Adding an element to a Bloom filter

Input: Bloom filter $B = (b_0, \ldots, b_{m-1})$; new element $e$

\[
\text{for } i \leftarrow 0 \ldots k-1 \text{ do }
\quad j \leftarrow h_i(e)
\quad b_j \leftarrow 1
\text{end for}
\]

Algorithm 2 Membership testing from a Bloom filter

Input: Bloom filter $B = (b_0, \ldots, b_{m-1})$; element to be tested $e$

Output: Returns TRUE or FALSE depending on if the element is in $B$

\[
\text{for } i \leftarrow 0 \ldots k-1 \text{ do }
\quad j \leftarrow h_i(e)
\quad \text{if } b_j = 0 \text{ then}
\quad \quad \text{return } \text{FALSE}
\quad \text{end if}
\text{end for}
\quad \text{return } \text{TRUE}
\]

2.1.2 False positive rate

Whenever Bloom filters are used, it is important to know the probability that a membership test of a non-member element returns a false positive.
This probability, called false positive rate or fpr, is affected by following parameters:

- filter size in bits ($m$),
- number of hash functions ($k$), and
- number of elements added to the filter ($n$).

The classic formula for the false positive rate, which initially appeared at [22], can be derived as follows. The probability that a bit is not set by a specific hash function is $1 - (1/m)$. After $k$ hash functions, the probability that any one of them is not set is $1 - (1/m)^k$. Similarly, after adding $n$ elements to the Bloom filter, the probability that a given bit is still zero is $1 - (1/m)^{kn}$. Thus, the false positive rate, $fpr$, describing the probability that a Bloom filter falsely claims that an element is a member of the set, is the following:

$$fpr = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \quad (2.1)$$

The efficiency of the Bloom filter can be optimized by choosing the parameters so that they minimize the $fpr$ value. The optimal value for $k$ is:

$$k_{opt} = \frac{m}{n} \ln 2 \quad (2.2)$$

Furthermore, it is possible to derive the required length $m$ for the Bloom filter for a given false positive rate:

$$m = -\frac{n \ln fpr}{(\ln 2)^2} \quad (2.3)$$

Even though these equations appear quite widely (e.g. [6, 36]), the accuracy of these formulas has been recently questioned. In 2008, Bose et al. [5] kindly remarked that “[The proof] is not quite correct”. The problem is that Equation 2.1 is based on the assumption that events “bit $b_i$ is set to one” and “bit $b_j$ is set to one” are independent. This, however, is not the case: it is fairly easy to see that the Equation 2.1 gives false answers by testing it with small values of $k$, $m$ and $n$.

Despite the fact that the classic equation was known to be incorrect, the problem of calculating the exact probability of false positives turned out to be difficult. An exact formula was initially given by Bose et al., but it was impractical due to its complexity [5]. Later, Christensen et al. were able to provide a numerical solution for the problem [8].

However, in this thesis, Equation 2.1 is used. The reason for this is twofold:
1. Equation 2.1 makes it possible to use analytical methods, which would not be possible if results from [5, 8] were used.

2. The error of the Equation 2.1 was shown in [5] to be

\[ O \left( \frac{k}{\sqrt{\ln m - k \ln p}} \right) \].

Thus the error is negligible when \( k \ll m \). This is the case when speaking about Internet-wide multicast: for example in [33], Bloom filters of 800 bits were considered.

### 2.1.3 Fill factor \( \rho \)

An important concept recurring in this thesis is the fill factor \( \rho \) of a Bloom filter. It tells the proportion of 1-bits in a Bloom filter, namely \( \rho = |B| / m \) where \( |B| \) is the \( L_1 \) norm of the Bloom filter \( B \) essentially telling the number of 1-bits in it.\(^1\) Fill factor is relevant for two reasons:

- As it will be described, when Bloom filters are used in forwarding, their maximum fill factor needs to be limited to some \( \rho_{\text{max}} \). Also, as will be explained later, a high fill factor is often an advantage for the attacker and thus the security analysis can be done with the assumption that \( \rho \approx \rho_{\text{max}} \).

- It is often easier to perform calculations assuming Bloom filters that have a certain fill factor than to calculate the fill factor distribution using the parameters \( k, m \) and \( n \).

The mean value for the fill factor \( \bar{\rho} \) of a Bloom filter can be derived from the equations showed earlier:

\[ \bar{\rho} \approx \left( 1 - \frac{1}{m} \right)^{nk} \].

This result is approximate due to the reasons explained in Section 2.1.2.

### 2.1.4 About the BF notation used in this thesis

In this thesis, Bloom filters are denoted with capital \( B_x \) with the identifier \( x \) of the filter marked as a subscript. All Bloom filters are treated as bit vectors. In the context of Bloom filters, symbols \( \& \), \( \lor \), and \( \oplus \) are to denote bitwise AND, bitwise OR, and bitwise XOR operations, respectively. Additionally,\(^1\) The \( L_1 \) norm is also known as taxicab distance, rectangular distance, and Manhattan distance. In relation to bit vectors, it is often called the population count.
CHAPTER 2. BACKGROUND

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg B_a$</td>
<td>Bitwise NOT (i.e. complement of $B_a$).</td>
</tr>
<tr>
<td>$B_a \land B_b$</td>
<td>Bitwise AND of two Bloom filters</td>
</tr>
<tr>
<td>$B_a \lor B_b$</td>
<td>Bitwise OR of two Bloom filters</td>
</tr>
<tr>
<td>$B_a \oplus B_b$</td>
<td>Bitwise exclusive-or (XOR) of two Bloom filters</td>
</tr>
<tr>
<td>$B_a \cdot B_b$</td>
<td>Dot product of two bit vectors. Namely, the number of 1-bits in the bit vector $B_a \land B_b$</td>
</tr>
<tr>
<td>$B_a \supseteq B_b$</td>
<td>Shorthand notation for $B_a \land B_b = B_b$. It must be noted that due to false positives this does not necessarily mean that any same elements have been added to the Bloom filters. Thus, it must not be confused with the subset-operator of the added elements.</td>
</tr>
<tr>
<td>$e \in B_a$</td>
<td>The membership test for element $e$ returns true. However, because of the false positives this does not necessarily mean that the element $e$ has been added to $B_a$.</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of the Bloom filter notation used in this thesis.

operations $\supseteq$ and $\subseteq$ are used as shorthand notations to denote that two Bloom filters contain 1-bits in the same positions, namely

$$B_a \supseteq B_b \iff B_a \land B_b = B_b. \quad (2.6)$$

This, however, must not be confused with the subset-operation because of the false positives — the fact that two Bloom filters have corresponding bits set to one does not mean that same elements have been added to them. Table 2.1 summarizes the Bloom filter notation used in this thesis.

### 2.2 Bloom-filter multicast forwarding

One of the first works suggesting the use of Bloom filters in multicast forwarding is by Grövall [13]. In this work, each outgoing interface of a multicast router has a Bloom filter which encodes the multicast-group addresses that are associated with the interface. When a packet is received, the destination address in the packet is compared to each Bloom filter on the router and forwarded to the matching interfaces. This mechanism does not store any individual addresses in the router. However, addition and removal of multicast addresses is required and thus a Bloom filter variant capable of these operations must be used [36].

Ratnasamy et al. [26] proposed a multicast protocol that uses Bloom filters slightly differently: instead of storing Bloom filters that describe mul-
Figure 2.2: Multicast tree encoding in Free Riding Multicast. Links in the delivery tree are added into the iBF. The table shows the elements added to the iBF when the receiver is either AS\textsubscript{5}, AS\textsubscript{7}, or both.

Because the routing is source routing, the border router of the source must know in which ASes there are multicast-group members. The solution for the multicast group management was to include group membership information in the BGP advertisements: ASes would encode the group identifiers of all groups they belong to in a Bloom filter, which is included in the BGP advertisements. Since the BGP advertisements are essentially flooded in the

\footnote{The term “in-packet Bloom filter” was coined later in [29].}

\footnote{To be exact, the amount of state that a router must store is proportional to the amount of interfaces the router has. This is, however, radically different situation than in many previous proposals where per-multicast tree state was required [16].}
Algorithm 3 iBF forwarding as proposed in [26].

<table>
<thead>
<tr>
<th>Input:</th>
<th>iBF $B_x$ describing the delivery tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Packet which is received from AS $ASN_f$ is being forwarded.</td>
</tr>
</tbody>
</table>

for each $ASN_n \in$ neighbours \ $ASN_f$ do

if $B_x$ contains link ($ASN_{self} \rightarrow ASN_n$) then

forward packet to $ASN_n$

end if

end for

network, all border routers could test whether an AS belongs to a certain multicast group. The source AS can then utilize this information when it is constructing the multicast tree.

Despite the novel ideas presented in [26], the work leaves many unanswered questions. Among these are the implications of the potential false-positive forwarding decisions and the security of the architecture. Additionally, the requirements for the border routers, which have to inspect and construct the multicast trees on per-packet basis, raise questions on the scalability of the architecture.

LIPSIN (Line Speed Publish/Subscribe Inter-networking [16]) proposed an iBF based forwarding plane into a publish-subscribe (pub-sub) networking architecture. In pub-sub networking paradigm the networking primitives are changed from $send(data, destination)$ and $receive(source)$ into $publish(data, identifier)$ and $subscribe(identifier)$. In other words, the networking becomes data-centric, which is in contrast to the current end-point centric networking. In-packet Bloom filter forwarding was proposed for the pub-sub paradigm because the iBFs can be used to hide the location of the data source. This is a desired feature for pub-sub networking where the sender and the receiver should be decoupled from each other [11].

The actual forwarding in LIPSIN is relatively similar to what Ratnasamy et al. [26] proposed. Each link in the network is assigned a link identifier that is $m$-bit bit string and has up to $k$ bits set using $k$ different hash functions. The Bloom filter describing the delivery tree is then constructed by bitwise ORing the relevant link identifiers together. Using Bloom filter terminology, the link identifiers are elements that are added to a set. From the forwarding point of view, the main difference of LIPSIN compared to [26] was that LIPSIN proposed that all forwarding should be based on iBF while Ratnasamy et al. proposed an incremental fix for inter-domain multicast. Additionally, unlike in [26] where link identifiers are public knowledge, the link identifiers in LIPSIN are secret and periodically changing.

Essentially, LIPSIN presented a forwarding fabric for data-centric publish-subscribe networking where the communication primitives are changed from...
CHAPTER 2. BACKGROUND

Figure 2.3: Multicast forwarding and branching in LANES. The tree consists of three unicast paths that are sent with iBFs $B_a..B_c$. The packet is branched at $N_b$ and $N_c$ according to the nodes’ branching table. The branching table is the only state that the routers need to store.

send-receive to publish-subscribe. One of the central goals of the proposed paradigm shift was to fix the imbalance of power between the sender and the receiver: unlike the current Internet architecture, the proposed publish-subscribe architecture would allow content delivery only if the receiver has explicitly allowed it [2]. In-packet Bloom-filter forwarding appeared to be the solution for this kind of networking because a forwarding identifier given to a publisher could be treated as a permission to send traffic. Additionally, iBF forwarding was deemed to create DoS resistance to the network because an attacker would not be able to fabricate forwarding identifiers. However, despite the indisputable impact of LIPSIN, some details were left unexplained such as an analysis of the hinted DoS resistance. Additionally, the description on how the forwarding identifiers are created was quite sketchy: the function was outsourced to a black-box called topology manager, which knows the full network topology.

LANES [37] is another iBF forwarding proposal for a future pub-sub Internet architecture. Unlike in [16, 26], in LANES iBFs are intended to be used only for unicast routing. Multicast is implemented on top of the unicast forwarding so that each router that performs branching has to store multicast state. Figure 2.3 illustrates how forwarding and branching works in the LANES architecture: even though the multicast tree has three recipients, only unicast path $N_a \rightarrow D_a$ is initially encoded into the iBF. When the packet arrives at router $N_b$, the packet is branched based on the router’s branching table: the packet is replicated and sent towards $D_b$ using the iBF $B_b$ while the original packet is forwarded unmodified towards $D_a$. The packet is branched similarly at $N_c$ by sending a copy of the packet towards $D_c$ using the iBF $B_c$. One of the benefits of LANES is that the sender does not have all control over the route selection.

While LIPSIN and LANES proposed forwarding planes into future Inter-
net, Bloomcast [34] brought iBF forwarding back into IP networks. It proposed an inter-domain multicast protocol that would use IP for signalling. In Bloomcast, a subscriber node wanting to join a multicast tree can send a \texttt{BC\_JOIN} packet to the publisher (i.e. source) in the multicast group over the IP. When the \texttt{BC\_JOIN} packet is forwarded, the Bloom filter required for sending packets back to the subscriber is being collected hop by hop. Finally, when the publisher receives the \texttt{BC\_JOIN} packet, it can add the subscriber to the multicast group simply by bitwise ORing the collected Bloom filter with the other Bloom filters received from the other subscribers. Another difference of Bloomcast compared to LIPSIN was that, instead of attaching identifiers to links, inbound-outbound interface pairs are given identifiers. This does not change the overall behaviour of iBF forwarding but it was argued in [29] that this would make the forwarding more secure.

In addition to the already mentioned proposals, iBFs have appeared also for example in [30, 38]. [30] promotes iBF based switching for data centers while [38] proposes a protocol called Multi-Protocol Stateless Switching (MPSS) that combines iBF forwarding and Multi-Protocol Label Switching (MPLS [28]). Even though these applications show the convenience of in-packet Bloom-filter forwarding, they are not particularly interesting for our discussion. This is because the protocols are intended for use only in a single administrative domain where all nodes can be trusted.

\subsection*{2.3 Forwarding-hop identifiers}

How the forwarding path is constructed varies between the different iBF forwarding proposals: for example in LIPSIN [16], the paths are composed of unidirectional links (link identifiers), while in Bloomcast the path are constructed from inbound-outbound interface pairs. In this thesis, these identifiers are called \textit{forwarding-hop identifiers} (FHID). Regardless of what the FHIDs identify, forwarding happens similarly: the identifier simply tells the direction to which the packet should be forwarded.

FHID can be used to to identify for example

- the next-hop node,
- the next-hop unidirectional link (i.e. ordered pairs of nodes),
- the next-hop bidirectional link (i.e. unordered pairs of nodes),
- ordered pairs of inbound and outbound interfaces, or
- unordered pairs of inbound and outbound interfaces (i.e. the identifier is the same for both directions).

Figure 2.4 illustrates some of these: in Figure 2.4a FHIDs identify next-hop routers; in 2.4b, the next-hop links; and in 2.4c, the inbound-outbound
2.4 Previously known forwarding anomalies and DoS problems

Forwarding anomalies that may occur in iBF forwarding were initially studied in [29] and later in [33, 34]. These works identified six anomalies that can be categorized into two groups: chain reaction anomalies and target path attacks\(^4\) [34]. This section explains these anomalies and what may cause them.

2.4.1 Chain reaction anomalies

As already discussed in Section 2.1.2, there is always a possibility of false positive results with Bloom filters. In iBF forwarding these false positive results cause packets to be forwarded to neighbours that are not part of the intended delivery tree. In the worst case, false positives they may cause

---

\(^4\)Even though these are called attacks, the analysis in [29, 33, 34] mainly discusses accidental occurrences of the anomalies and not attacks done by a malicious entity.
anomalies that greatly increase the traffic amount or reduce the forwarding efficiency of the network. These anomalies include 100% fill factor, packet storms, loops, and flow duplication. [34]

100% fill factor is an anomaly where attacker sends a packet with an iBF that has all bits set. The iBF would match to every FHID in the network thus possibly creating a never ending packet storm. A simple way of protecting the network against this attack would be simply to limit the number of 1-bits that there can be in the iBF. This is discussed in more detail in Section 2.5.1.

Packet storm is caused by an event where routers forward a packet on average to more than one false interface. If this happens, the packet is forwarded to more than one extra router at each hop thus generating a chain reaction. It was calculated in [34] that the average number of false positives in a given router is \( \rho^k \times (d - b - 1) \), where \( \rho \) is the fill factor, \( k \) is the number of hash functions, \( d \) is the number of neighbours, and \( b \) is the number of outgoing branches in the forwarding tree. Thus, with values \( k = 5 \) and \( \rho = 0.5 \), nodes with degree \( d > 32 \) would each cause more than one false positive. Packet storm is a risk especially in topologies that have a clique of high degree nodes. Internet AS-topology is an example of such topology.

Loops are anomalies where a false positive causes a packet to be forwarded back to a router the packet already has visited. False positives may be accidental, but an attacker may also try to cause loops on purpose. Loops are harmful because they use a lot of network resources and can be used to amplify traffic. This can be seen in Figure 2.5a where the destination receives the looping packet once per each time the packet traverses the loop.

Flow duplication is an anomaly where traffic amplification happens without a loop. This can happen when a false positive cause a packet to be forwarded to another branch of the same multicast tree. This is illustrated in Figure 2.5b and 2.5c.

2.4.2 Target path attacks

Three target path attacks have been identified in literature: injection attack, correlation attack, and replay attack [29, 34]. Unlike in chain reaction anomalies, in target path attacks the exploitation of false positives does not have large emphasis. Additionally, target path attacks are always initiated by a malicious entity and often require that the attacker controls a botnet.
CHAPTER 2. BACKGROUND

Even though target path attacks have been mentioned in the literature [29, 34], the attacks have not been studied in detail. In fact, this raises a question whether these attacks are even real threats. This thesis, however, will formulate target path attacks and thus show that they indeed are a real threat for iBF forwarding.

**Injection attack** is an attack where an off-path attacker tries to send data to an existing delivery tree. The attacker may have some information of the delivery tree to which it is injecting traffic (e.g. iBF for the delivery tree). This attack is trivial if the attacker knows the FHIDs as in FRM [26] where FHIDs are not secret.

In **Correlation attack** the attacker infers FHIDs from several iBFs. Using the information gained this way, the attacker can for example create loops to the network. This attack is also called *computational attack* [29].

**Replay attack** is an attack where a valid iBF is used to send traffic that is not supposed to be sent with that particular iBF. An attacker can, for example, flood unsolicited traffic using an iBF that is only meant for signalling.
2.5 Proposed mechanisms to increase DoS resistance

2.5.1 Limits on fill factor

As already discussed, the most obvious way to attack against any iBF forwarding system is to send a packet with a Bloom filter that has all bits set. A proposed solution for this is to limit the fill factor $\rho$ of the Bloom filter. This limit, called the maximum fill factor ($\rho_{\text{max}}$), defines the maximum proportion of set bits in the filter. If the proportion of set bits exceeds the limit, the forwarding nodes drop the packet. Typically $\rho_{\text{max}} \approx 0.5$ since it yields the best efficiency in terms of number of elements per the false positive rate [34].

Additionally, the number of set bits in the iBF given to the publisher node could be bounded from below so that no publisher ever gets an iBF with less than $\rho_{\text{min}}$ of the bits set. The motivation behind this restriction could be to limit the amount of information an attacker can get by investigating a given iBF. $\rho_{\text{min}}$ also limits how the sender or some other node can combine several delivery trees by bitwise ORing them together: for example if $\rho_{\text{min}} \approx \rho_{\text{max}}$ the nodes cannot combine delivery paths at all. This essentially means that the multicast group management is then moved from the nodes to a separate entity constructing the multicast iBFs.

2.5.2 Cryptographic and flow specific FHIDs

Cryptographic FHIDs were initially investigated in [29] as a solution for target path attacks (see Section 2.4). The idea is that instead of maintaining a fixed forwarding table for the FHIDs, a $Z$-function would be used to dynamically generate the FHIDs on a per-packet basis. This would make it possible to utilize information from the packets when generating the FHIDs thus allowing iBFs that are valid only for certain packet flows.

The creation of forwarding-hop identifiers, both static and cryptographic\textsuperscript{5}, can be formalized as a function $Z : A \rightarrow I$, where $A$ is a tuple of arguments and $I$ the resulting forwarding-hop identifier. Depending on the iBF forwarding scheme, the arity of the tuples in $A$ may vary. For example, in the base case where the FHID identifies the next-hop interface, $i_{\text{out}} \in A$. Similarly, if the FHID identifies inbound-outbound interface pairs, the argument tuple would be $(i_{\text{in}}, i_{\text{out}}) \in A$. It must be noted that in this formalization $Z$ is considered to be a global function. This means that if all parameters are

\textsuperscript{5}According to this specification, static FHIDs are just a special case of cryptographic FHIDs.
Table 2.2: Samples of the arguments that may be given to the Z-function

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{next}$</td>
<td>next-hop router</td>
</tr>
<tr>
<td>$l_{out}$</td>
<td>outbound link (bidirectional)</td>
</tr>
<tr>
<td>$i_{out}$</td>
<td>outbound interface</td>
</tr>
<tr>
<td>$i_{in}$</td>
<td>inbound interface</td>
</tr>
<tr>
<td>${i_{in}, l_{out}}$</td>
<td>a set containing inbound-outbound interface pair (bidirectional)</td>
</tr>
<tr>
<td>$t$</td>
<td>time; FHID is changed once in $\Delta t$</td>
</tr>
<tr>
<td>$K$</td>
<td>secret key of a router</td>
</tr>
<tr>
<td>$f$</td>
<td>flow identifier unique to each connection flow</td>
</tr>
<tr>
<td>$s$</td>
<td>identifier of the source of the packet</td>
</tr>
</tbody>
</table>

globally known, also the FHIDs are globally known. To make the FHIDs secret, routers can for example use a secret key $K$ as one of the arguments.

Naturally, the FHID creation function may also take other arguments than just the input and output interfaces. To make the identifiers valid only for a certain amount of time, a time argument can be applied: $(i_{in}, l_{out}, K, t) \in A$. It is also possible to use information from the packet header in the argument tuple: [29] proposed using a flow identifier $f$, which is unique to each communication flow, in FHID creation.

This thesis uses the sequence $A$ to describe different iBF forwarding schemes. Table 2.2 summarizes variables that may be used in FHID creation.

### 2.5.3 Global and local $k$

Typically in Bloom filters the bit vector representing an element is constructed by applying a fixed number of hash functions to the element. In other words, all of the bit-vectors that represent elements are constructed using the same number of hash functions.

However, varying the number of used hash functions has been proposed in the context of iBF forwarding [33, 34]. These proposals are motivated by the varying false positive rate in network nodes of different degree: if the same values of $k$ is used at a low degree node and a high degree node, the false positive forwarding rate of the high degree node is greater. In the worst case, this could result in a packet storm in highly connected part of the network (see Section 2.4.1). If the nodes independently chose the $k$, each node could set the value so high that the average number of false positives is less than one and packet storms are avoided.

If routers can independently decide how many hash functions they use to
2.5.4 Permutations

Permutations have been proposed as a solution to accidental and maliciously created forwarding loops and flow duplications [33].\(^6\) In the context of iBF forwarding, permutations are similar to P-boxes that are present in many cryptographic functions [35]: permutation essentially just shuffles the bits in the iBF. This gives protection against loops and flow duplications because iBF becomes a function of the the original iBF and the path that the packet has traversed. This is illustrated in Figure 2.6: even though a packet is forwarded from $C$ back to $A$ due to a false positive, the packet is dropped with high probability at $A$ because the iBF is changed to $(\pi_C \circ \pi_B \circ \pi_A)(B_x)$.

Despite the simplicity of this operation, permutations can be applied to iBF forwarding in various ways. First of all, a permutation can be either completely static or static for a certain time period. If static permutations are used, the permutations may be applied either at the inbound interfaces, outbound interfaces, or both. Figure 2.7 provides illustration on how static permutations can be implemented in iBF forwarding. It must be emphasized, that the figure does not enumerate all possible ways to implement the permutations.

Analogously to the FHID creation, the permutation functions could also be created dynamically using information for example from the packet header as an input. This would make it possible to use unique permutation for each traffic flow. However, keyed bit permutation is relatively time consuming.

---

\(^6\)Even though [33] claims that permutations also solve maliciously created forwarding loops and flow duplications, the work focuses only on accidental occurrences of these. Whether permutations protect against maliciously created anomalies has not been studied thoroughly.
to do with software [19], which makes it infeasible for large scale packet forwarding. Some efficient hardware instructions have been proposed [14] — for example, Shi et al. proposed the BFLY instruction in [31] which could be used to do arbitrary bit permutations with a single instruction by utilizing a butterfly network. The problem with the hardware solutions is, however, that in iBF forwarding the permuted bit-vectors are typically long and thus would require custom designed hardware, which would increase the overall cost of the system. Due to these reasons, we do not considered dynamic permutation as a realistic option in Internet wide forwarding. This means, that in practise there are only two options how the permutations can be implemented: they can either be completely static or static for a certain time period.
Figure 2.7: Samples how static permutations can be implemented. Packets arrive to input interfaces \(I_0 \ldots I_{i-1}\) and are forwarded through to the output interfaces \(O_0 \ldots O_{j-1}\). The \(M_n\) functions check whether to forward the packet to the corresponding interface (i.e. they check whether the FHID is included in the iBF).
2.5.5 Summary of the proposed iBF variants

Table 2.3 summarizes the proposed variants of iBF forwarding. It must be noted that the table is not complete: for example as it can be seen for in Figure 2.7 (which is not complete either), there are many different ways how to implement permutations.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FHID</td>
<td>static</td>
<td>FHIDs is the same for all flows. In other words, the Z-function, which is used to generate the FHIDs, does not have any flow-specific arguments.</td>
</tr>
<tr>
<td></td>
<td>cryptographic</td>
<td>FHIDs are calculated dynamically for each flow. This essentially means that the Z-function has a flow-specific argument (e.g. a flow identifier).</td>
</tr>
<tr>
<td>Hashing</td>
<td>global $k$</td>
<td>All routers use $k$ different hash functions to calculate the FHIDs.</td>
</tr>
<tr>
<td></td>
<td>local $k$</td>
<td>The number of hash functions used to generate FHIDs may vary between routers.</td>
</tr>
<tr>
<td>Permutation</td>
<td>none</td>
<td>No permutations are implemented in the forwarding nodes.</td>
</tr>
<tr>
<td></td>
<td>static</td>
<td>Permutations are static to each node, interface, or interface pairs (see Figure 2.7).</td>
</tr>
<tr>
<td></td>
<td>dynamic</td>
<td>Permutations are dynamically calculated analogously to cryptographic FHIDs. Dynamic permutations are assumed infeasible for large scale (i.e. Internet wide) forwarding.</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td></td>
<td>The maximum allowed fraction of 1 bits in an iBF. $\rho_{\text{max}}$ is a value between $[0, 1]$, where 1 essentially means that the fill factor has no upper limit.</td>
</tr>
<tr>
<td>$\rho_{\text{min}}$</td>
<td></td>
<td>The minimum fraction of 1 bits in an iBF given to a publisher. $\rho_{\text{min}}$ is a value between $[0, \rho_{\text{max}}]$, where 0 means that the fill factor of an iBFs given to a publisher has no lower limit.</td>
</tr>
</tbody>
</table>

Table 2.3: Summary of the proposed iBF enhancements
Chapter 3

Security evaluation

As described in Chapter 2, a wide range of different iBF forwarding schemes have been proposed. The large number of possible schemes makes the security analysis challenging: it is simply not feasible to enumerate all possible iBF forwarding schemes. To tackle this, we start this chapter by introducing a concept called connectivity graph, which can be used to normalize iBF forwarding schemes. After the introduction of connectivity graphs (Section 3.1) comes the actual security evaluation of iBF forwarding.

In some of the iBF forwarding proposals, the security has been based on the secrecy of the static FHIDs [16]. If the Bloom filters are not permuted (see Section 2.5.4), the secrecy of the FHIDs is the only security mechanism protecting for example against intentionally created forwarding loops. The security evaluation starts in Section 3.2 by describing how an attacker can elicit information about the FHIDs when iBFs are not permuted. The attack shows that iBF forwarding cannot be secured simply by assuming that the FHIDs are secret. Even though this result is expected, detailed analysis on the attack had not been done before.

Sections 3.3–3.6 describe how an attacker who is controlling a botnet can perform an injection attack, where it can send unsolicited traffic to the target from several bots. Section 3.3 presents the injection attack in a scenario where not many security mechanisms are being used. After presenting the basic idea of the attack, Sections 3.4–3.6 re-evaluate the situation with more strict assumptions and more sophisticated security mechanisms in place. Throughout these sections, it is assumed that the attacker has full control over its bots and that the bots are located at random network locations (i.e. sparsely). The presented attacks show that the assumption that has appeared in the literature (e.g. in [16, 29]) concerning the DoS resistance of iBF forwarding is exaggerated.
CHAPTER 3. SECURITY EVALUATION

3.1 Connectivity graph

We would like to analyse the DoS resistance of iBF forwarding in general. This is complicated because there are minor variations of the iBF forwarding protocol. In particular, FHIDs are used to identify a large variety of different entities from nodes to links and link-pairs (see Section 2.3). To tackle this problem, we define the connectivity graph, which can be used to normalize the network graphs regardless of what the FHIDs identify. The idea is to create a graph where the vertices present forwarding hops instead of network nodes. The edges between the nodes represent ways in which packets can traverse the forwarding hops.

**Definition 1.** Connectivity graph is a directed graph presented as a pair of sets \((V; E)\) where \(E \subseteq V \times V\). \(V\) represents the forwarding hops, which are annotated with forwarding-hop identifiers, and \(E\) edges between the forwarding hops. For every unidirectional forwarding-hop identifier in the network, there exists one vertex in the connectivity graph. \((v_i, v_j) \in E\) iff it is possible for a packet to traverse hop \(v_j\) right after hop \(v_i\).

Figure 3.1 illustrates how connectivity graphs can be constructed from a network topology with different types of forwarding-hop identifiers. The FHIDs in the figure are illustrated with shading. It can be seen that the connectivity graph essentially describes what kind of FHID sequences are possible in routing.

Depending on what the FHIDs identify, the resulting connectivity graph can be very different. Table 3.1 illustrates this by showing the connectivity graph sizes of two topologies: Internet AS-graph and full 5-ary tree\(^1\) of depth 7. The table shows the sizes of the connectivity graphs when FHIDs are used to identify either the next-hop node, next-hop link, or inbound-outbound link pairs. What is notable is that even though the number of nodes is in the same order of magnitude in the two topologies, the connectivity graphs differ hugely in size. The cause for the differences in the connectivity graph sizes is that the Internet AS-graphs has a very dense core where nodes may have thousands of neighbours. On the other hand, in a simple tree topology the node degrees are uniform (excluding the leaf nodes and the root of the tree) and relatively low.

It will be shown later in this thesis that the size of the connectivity graph may also affect the difficulty of certain DoS attacks. The explanation is rather intuitive: it is more difficult for the attacker to have access to iBFs for intersecting paths in a topology that has a large connectivity graph. However,

\(^1\)Full \(k\)-ary tree is a rooted tree where each node excluding the leaves has exactly \(k\) children.
it will be also seen that a large connectivity graph does not necessarily mean that the topology is resistant to the presented attacks.
Figure 3.1: Connectivity graph of a network topology with different types of FHIDs.

Table 3.1: Sizes of connectivity graphs in Internet AS-topology (33 508 nodes and 75 001 bidirectional links) and in a full 5-ary tree of depth 7 (19 531 nodes and 39 060 links). For more detail about the AS-graph and routing policies, see Appendix A.
3.2 Correlation attack

The security of some iBF forwarding schemes, such as LIPSIN [16], is only based on the secrecy of the FHIDs. If the iBFs are not permuted (see Section 2.5.4), the secrecy of the FHIDs is the only thing preventing an attacker from creating loops and sending traffic to arbitrary recipients. This section discusses the possibility that an attacker can learn the identifiers that routers use to generate the Bloom filter. Finding a way of computing the FHIDs would mean that the secrecy of the FHIDs alone is not enough to secure the forwarding and that the security model proposed for example in LIPSIN [16] is broken.

We consider scenarios where the identifiers (i.e. FHIDs) are either:

- node identifiers identifying the next-hop router,
- link identifiers identifying the next hop link, or
- in-out identifiers identifying input-output interface pairs.

In this section, we assume that the identifiers are either completely static or static for a certain time (i.e. identifiers are not unique to each flow, see Section 2.5.2) and that the Bloom filter permutations (see Section 2.5.4) are not used. Our hypothesis is that an attacker can gain information about the used link identifiers assuming that it knows the network topology and has control over a small number of bots. The motivation here is to show that iBF forwarding scheme that uses secret FHIDs but not permutations is insecure: if it is possible to learn the FHIDs, the attacker could perform several attacks described in Section 2.4, such as creating forwarding loops.

3.2.1 Attack model

Goal of the attacker: To learn FHIDs used in the network.

Resources of the attacker: $n$ bots that are placed at random network locations. The attacker has full control over the bots and can send iBF forwarded messages between any of them.

Prior knowledge of the attacker: The attacker has full knowledge of the network topology.

FHID: FHIDs are static and identify either the next-hop router, next-hop link, or inbound-outbound link pairs. Using the notation defined in Section 2.5.2, the FHIDs could for example be determined by a function $Z(i_{in}, i_{out}, K)$.

Used security mechanisms: FHIDs are secret.

\footnote{The parameters for the $Z$-function may vary depending on whether FHIDs are used to identify nodes, links, or interface pairs.}
Figure 3.2: Prerequisite for the correlation attack: the attacker must be able to construct paths between its bots so that the paths intersect. In this figure it is assumed that the FHIDs identify links.

3.2.2 The Attack

The prerequisite of this attack is that the attacker can send iBF forwarded messages between any two of its bots. If the attacker can choose two pairs of bots \((A_a_1, A_a_2)\) and \((A_b_1, A_b_2)\) so that the paths \(A_a_1 \rightarrow A_a_2\) and \(A_b_1 \rightarrow A_b_2\) intersect, it can get a Bloom filter that matches the FHID of the intersection point simply by calculating the bitwise AND of the two Bloom filters describing the paths. This is illustrated in Figure 3.2: assuming that FHIDs identify links, the attacker can obtain a Bloom filter matching the shared link by calculating \(B_1 \wedge B_2\).

However, the bit-vector resulting from the bitwise AND is not necessarily the exact representation of the FHID. Broder et al. [6] provided a formula for calculating the mean number of 1-bits that two Bloom filters have at matching positions (i.e. the dot product of the two bit-vectors \(B_1\) and \(B_2\)). The mean value of the dot product of two Bloom filters presented as bit-vectors \(B_1\) and \(B_2\) which describe the sets \(S_1\) and \(S_2\), respectively, is

\[
B_1 \cdot B_2 = m \left( \bar{\rho}_1 + \bar{\rho}_2 - 1 + \left( 1 - \frac{1}{m} \right)^k \left( |S_1| + |S_2| - |S_1 \cap S_2| \right) \right), \tag{3.1}
\]

where \(\bar{\rho}_1\) and \(\bar{\rho}\) denote the expected fill factor of the Bloom filters \(B_1\) and \(B_2\), respectively.

Even though it may be possible to get the exact representation of the element at the intersection of two Bloom filters simply by bitwise ANDing them together, this is relatively unlikely. More accurate representation of the element at the intersection can be derived if there are more than two paths that intersect at the same point. Furthermore, the accuracy is increased if the paths have only one common element; namely given \(n \geq 3\) Bloom filters describing sets \(S_1..S_n\) such that \(\forall a, b : |S_a \cap S_b| = 1\). The question
then becomes: how many Bloom filters describing sets like this are required in order to learn the exact representation of the element in the intersection $S_1 \cap S_2$. Assuming that all of the Bloom filters have fill factor $\rho$, the median number of required Bloom filters $c$ is approximately

$$c \approx 2 + \frac{\ln (1 - \sqrt{0.5})}{\ln \rho},$$

(3.2)

where $d$ is the number of 1-bits in $B_1 \land B_2$ that do not describe the element at the intersection $S_1 \cap S_2$ (i.e. the number of false 1-bits). The derivation of Equation 3.2 can be found from Appendix B. It must be noted that this equation applies only if the Bloom filters do not share any other elements but the one at the intersection $S_1 \cap S_2 \cap \ldots \cap S_n$. In other words, Equation 3.2 applies only if $\forall a, b : |S_a \cap S_b| = 1$ even though in practise the attacker can use also Bloom filters that share more than one element. Table 3.2 shows sample values calculated using Equation 3.2. It can be seen from the table that two Bloom filters is enough to get the intersection of the sets accurately when the fill factor is low.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Resulting values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>$k$</td>
</tr>
<tr>
<td>256</td>
<td>11</td>
</tr>
<tr>
<td>256</td>
<td>11</td>
</tr>
<tr>
<td>256</td>
<td>11</td>
</tr>
<tr>
<td>1024</td>
<td>11</td>
</tr>
<tr>
<td>1024</td>
<td>11</td>
</tr>
<tr>
<td>1024</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3.2: Illustration on how different variables affect the expected value of $B_1 \cdot B_2$ when $|B_1 \cap B_2| = 1$. Value $c$ shows the approximate number of Bloom filters required to obtain the identifier of the element at the intersection (see Equation 3.2).

Since the attacker wants to learn individual FHIDs, it will want to construct pairs of paths between the bots so that the paths intersect at one network location. Depending on what kind of FHIDs are used, the paths between the bots must intersect either at one, two or three consecutive routers. Figure 3.3 illustrates this: if FHIDs identify next-hop routers, the two paths must intersect at one router. Similarly, if FHIDs identify next-hop links, the paths must intersect at two consecutive routers, and if FHIDs are specific to inbound-outbound interface pairs, the paths must intersect at three consecutive routers in order to have one FHID in common. This can also be explained using the connectivity graph described in Section 3.1: since each
FHID in the connectivity graph is a vertex, it is sufficient to find two paths that have an intersection in the connectivity graph.

What makes this attack interesting is the fact that, given \( n \) bots, the attacker can create \( \frac{1}{2}n(n - 1) \) different bidirectional paths between them. With these paths, it is possible to use

\[
\frac{1}{4}(n - 1)n \left(\frac{1}{2}(n - 1)n - 1\right) \in O(n^4)
\]  

(3.3)

different path pairs when calculating the intersection points. This hints that the number of FHIDs that an attacker can compute grows rapidly as a function of the botnet size.

\subsection{Evaluation}

The CAIDA AS-level Internet topology [1] was used to evaluate this attack (see Appendix A for details about the topology dataset). The topology was used to determine two things:

1. How often two paths that are between two pairs of bots intersect in such a way that it is possible to compute detailed information about the FHIDs in the intersection?

2. With what accuracy is it possible to determine individual FHIDs? How often does the attacker learn an FHID accurately?
The number of intersecting paths

The AS-level topology, which contains 33,508 nodes, was used to determine how often two paths that are generated between two pairs of bots intersect in such a way that it is possible to compute information about the FHID at the intersection of the paths.

The evaluation was done as follows:

1. Initially, \( n \leq 16000 \) bots were placed at random network locations without replacement.
2. The shortest path between all bots was calculated. If there were several shortest paths between the two nodes, one path was picked at random.
3. The number of FHIDs that were contained in more than one path was calculated. As discussed earlier, depending on the type of the FHIDs, the paths must intersect either at 1, 2, or 3 consecutive nodes in order to give information about a single link identifier.

The experiment was repeated 15 times for each botnet size. Figures 3.4a–3.4c show the results. It must be noted that the evaluation does not show the number of FHIDs that can be computed accurately. On the contrary, the result show on how many FHIDs the attacker can gain some information. How accurately the attacker can determine the FHIDs is evaluated later in this section.

The evaluation shows that the effect of the attack is very different depending on what FHIDs identify. When FHIDs identify nodes, attacker gains information on 10\% of the FHIDs with only 2000 bots, but when FHIDs identify inbound-outbound interface pairs, the attacker needs 14,000 bots to gain information on 10\% of the FHIDs. This also hints that the size of the connectivity graph has an effect on the attack. Additionally, it hints that the attack efficiency is highly dependent on the topology.

However, it must be emphasized that the objective of an attacker is not necessarily to learn as many FHIDs as it can. Instead, if the goal of the attacker was to perform a DoS attack utilizing a loop, it would be enough to known only very few particular FHIDs. Because of this, the implications of the attack are serious: regardless of what FHIDs identify, the attacker does not necessarily need many bots to compute FHIDs that could be used for example to create loops.

\(^3\)Since the dataset describes a policy-annotated AS-level topology, all used paths were valley free (i.e. policy compliant). See Appendix A for more details.
The accuracy of the obtained FHIDs

The CAIDA AS-topology was also used to determine the accuracy at which it is possible to learn FHIDs. For this evaluation, it was assumed that the FHIDs are link identifiers. The other cases were not investigated.

The evaluation was done as follows:

1. Initially, each link in the network was assigned a 256-bit long FHID that was generated using \( k = 24 \) hash functions (i.e. a global \( k \) was being used). The number of hash functions was chosen so that fill factors larger than 0.5 would be rare. The fill factor distribution can be seen in Figure 3.6.

2. \( n \in \{100, 500, 1000\} \) bots were placed at random network locations.

3. The shortest path\(^4\) between all bots was calculated. If there were several shortest paths between the two nodes, one path was picked at random. For each shortest path, an iBF \( B_{s,d} \) was created by bitwise ORing the FHIDs on the path together.

4. For each link, a BF matching the FHID of the link was calculated by bitwise ANDing all iBFs that describe a path going through the particular link. Finally, the differences of the obtained BFs and the actual FHIDs were calculated.

Figure 3.5 shows the CDF of the accuracy at which the attacker can learn the FHIDs. The figure shows the number of extra 1-bits in the obtained FHID compared to the real FHID. It can be seen that the number of the bots affects how accurately the attacker can obtain the FHIDs: with 100 bots, 20% of the obtained FHIDs are known accurately while, with 2000 bots, nearly 60% of the obtained FHIDs are known accurately.

The wavy shape of Figure 3.5 is a result of the fill factor distribution. Because a global \( k \) is being used, the iBF fill factor distribution has clear peaks (see Figure 3.6). These peaks are then reflected to the number of extra bits present in the learned FHIDs, which can be seen in Figure 3.5.

The evaluation shows that the presented method for acquiring information about the FHIDs is feasible. With only 100 bots, a comparably large fraction of the FHIDs were learned accurately. The results show that it is wrong to assume that secret FHIDs would alone make iBF forwarding secure. Even when FHIDs are used to identify inbound-outbound interface pairs, which is the most difficult case for the attacker, it is possible to learn a significant number of FHIDs with a reasonable-size botnet. Additionally, the attack presented in this section is the most obvious way to compute FHIDs

---

\(^4\)Since the dataset describes a policy annotated AS-level topology, all used paths were valley-free (i.e. policy compliant). See Appendix A for more details.
— it may be possible that even better results could be achieved with more sophisticated attacks.
Figure 3.4: Number of FHIDs from which it is possible to gain information with $n$ bots.
Figure 3.5: Cumulative Distribution Function (CDF) showing the number of extra 1-bits in the computed FHID when FHIDs identify unidirectional links.

Figure 3.6: Distribution of the iBF fill factors when a global $k = 24$ was being used in the Internet AS-topology (see Appendix A).
CHAPTER 3. SECURITY EVALUATION

3.3 Injection attack

It has been claimed that permutations of the Bloom filter give protection against an injection attack where the attacker injects unsolicited traffic to an existing delivery tree [34]. This claim is based on the idea that it is difficult to obtain information about individual permutations that are used in the network. This section, however, shows how an injection attack is possible if the attacker controls a bot network and knows at least one iBF that leads to the victim.\(^5\) After this section, Sections 3.4–3.6 further elaborate this attack and describe how the attack can be carried out even under stricter security assumptions.

Apart from the definition of the attack model and the evaluation (Sections 3.3.1 and 3.3.5), the attack is described in three parts. At the beginning, Section 3.3.2 describes how an attacker can learn permutations between two network nodes. When the attacker knows certain permutations between its bots, it can perform the actual injection. This is described in Section 3.3.3. Finally, Section 3.3.4 discusses how the attack could be made more effective by chaining the injections.

3.3.1 Attack model

**Goal of the attacker:** To launch unsolicited traffic from bots \(A_{0..n-1}\) against a victim \(T\).

**Resources of the attacker:** \(n\) bots \((A_{0..n-1})\) that are placed at random network locations. The attacker has full control over the bots and can send iBF forwarded messages between any of them.

**Prior knowledge of the attacker:** The attacker has full knowledge of the network topology. Additionally, the attacker has a valid forwarding identifier that can be used to send traffic from bot \(A_A\) to the target \(T\).

**FHID:** FHIDs identify either the next-hop router, next-hop link, or inbound-outbound link pairs and they are static for time \(\Delta t\). Using the notation defined in Section 2.5.2, FHIDs could for example be determined by a function \(Z(i_{in}, i_{out}, t, K)\).\(^6\)

**Used security mechanisms:** FHIDs are secret. The iBF is permuted at each router. The permutation for each router is static but secret.

\(^5\)Due to the fact that most Internet applications require some signalling channel also towards the source of the data, we believe that it is justifiable to require that the attacker has at least one valid path towards the victim.

\(^6\)The parameters for the \(Z\)-function may vary depending on whether FHIDs are used to identify nodes, links, or interface pairs.
3.3.2 Acquiring information about a permutation

When permutations are used, the in-packet Bloom filter that the receiver observes is different from the iBF that the sender uses to send the packet. This is due to the fact, that each router on the path permutes the iBF with its own secret permutation function. Thus, when a packet traverses several hops, the iBF is permuted with the composition function of the permutations on the path. For example, an iBF $B$ will change to $(\pi_b \circ \pi_a)(B)$ when the packet traverses through two routers.

An attacker can learn some composition permutations partially as follows. Whenever a host $H_1$ is sending messages to host $H_2$, it is possible to gain information about the permutations of the 0-bits in the Bloom filter just by setting each 0-bit to 1 one bit at the time. At the same time, the receiver node $H_2$ can notice the positions of the changed 1-bits. However, the permutations of the original 1-bits cannot be determined this way. Figure 3.7 illustrates this: while it is possible to learn the permutations for the 0-bits, the attacker cannot know how the 1-bits are permuted. Thus, in order to attain the permutation of the 1-bits, the attacker must repeat the process with different iBFs. This can be done once in $\Delta t$, which is the validity period of the FHIDs.

It is rather difficult to analytically determine the number of iBFs needed before the attacker gets all of the individual bit permutations (i.e. before the permutation function is learned completely). This is because the random variable describing an event “the permutation of bit $b_i$ is solved after investigating $j$ iBFs” is dependent on the number of other solved permutations. However, Figure 3.8 shows empirically derived probability density function of the number of iBFs required before all of the bit-mappings are solved. It can be seen from the figure that the length of the Bloom filter does not have much effect on the required number of iBFs: quadrupling the iBF size.
CHAPTER 3. SECURITY EVALUATION

Figure 3.8: Probability density functions showing the number of iBF updates required before a permutation can be accurately determined. The plots show results of 10,000 simulations done with both 256 and 1024 bit Bloom filters and $\rho = 0.5$.

Figure 3.9: The fraction of known bit-permutations after $n$ different iBFs with 25% and 50% full Bloom filters. The middle plots show the median fraction, while the error bars show 5% and 95% percentiles with 256 bit long Bloom filters. The continuous lines visualize function $f(n) = 1 - \rho^n$ with $\rho = 0.25$ and $\rho = 0.5$ respectively.
Figure 3.10: A sample topology and its connectivity graph illustrating the prerequisites the for attack: path \( B_a \rightarrow T \) must intersect the path \( B_b \rightarrow B_c \) at two consecutive routers. Additionally, if permutations are used, path \( B_a \rightarrow B_c \) must follow the two other paths as illustrated.

From 256 bits to 1024 bits while keeping the fill factor the same increased the median value only by 2 from 10 to 12.\(^7\) Additionally, Figure 3.9 shows the median fraction of known bit permutation after \( n \) different iBFs (i.e. the fraction of known bit mappings in the P-box). It can be seen from the figure that the median fraction follows the equation \( 1 - \rho^n \). It was observed that the used Bloom filter length does not affect this value. However, the variance of the fraction of known bit-permutations decreased as a function of Bloom filter length.

### 3.3.3 Performing the injection

From here on, it is assumed that the attacker can obtain the composition permutation between any two bots using the method described earlier.

Figure 3.10 shows a sample topology that is used to illustrate the attack. The objective of the attacker is to send unsolicited traffic from bot \( A_b \) to the target \( T \) when the attacker has one valid iBF (Bloom filter \( B_a \) in the figure), which describes path \( A_a \rightarrow T \). The requirements for this attack are that the paths \( A_b \rightarrow A_c \) and \( A_a \rightarrow T \), represented as Bloom filters \( B_a \) and \( B_b \), traverse through adjacent nodes in the connectivity graph as illustrated in Figure 3.10b. If FHIDs identify inbound-outbound interface pairs, this means that the paths must overlap at two consecutive routers. Additionally,

\(^7\)However, the attacker must send \( O(m) \) packets, where \( m \) is the iBF length, before it is granted to learn a permutation completely.
if permutations are used, the path $A_a \rightarrow A_c$ must follow the two other paths as shown in the figure.

If the permutations were not used and if the fill factors of the Bloom filters were low, the attacker could construct a Bloom filter which could be used to send traffic from $A_b$ to $T$ simply by bitwise ORing $B_a$ and $B_b$. Permutations make the injection attack more difficult, but not impossible: the key insight of the attack presented here is that the attacker does not need to know the permutation table of any single router in order to perform an injection. Instead, the attacker can exploit the knowledge that the same permutations are applied to paths sharing a common part.

If the permutations in the topology are denoted with $\pi_a$, $\pi_b$, $\pi_c$, and $\pi_t$ as shown in Figure 3.10c, the attacker can obtain following two permutation compositions and their inverses using the method described earlier:

- $\pi_c \circ \pi_a$ (i.e. from $A_a$ to $A_c$)
- $\pi_c \circ \pi_b$ (i.e. from $A_b$ to $A_c$)

The goal of the attacker is to create an iBF that can be used to send traffic from $A_b$ to $T$. This is possible with an iBF $B_x$ that satisfies following equations:

\[
B_x \land B_b = B_b \quad (3.4a)
\]
\[
\pi_b(B_x) \land \pi_a(B_a) = \pi_a(B_a) \quad (3.4b)
\]

Satisfying the first equation makes a packet sent from $A_b$ traverse to the intersection. The second equation enables the packet to traverse the end of the path from the intersection to the target. It must be noted that, due to the first equation, the packet using $B_x$ is also forwarded from $A_b$ to $A_c$. This, however, does not matter since the goal is just to be able to send traffic to $T$.

Using only the known information, namely the known permutations and iBFs, the attacker can construct an iBF satisfying Equations 3.4 as follows:

\[
B_x = (\pi_c \circ \pi_b)^{-1} ((\pi_c \circ \pi_a)(B_a)) \lor B_b \quad (3.5)
\]

It is clear that $B_x$ satisfies the first requirement (i.e. Equation 3.4a) because $B_b$ is simply bitwise ORed to $B_x$. It can also be seen that $B_x$ satisfies the second requirement, namely $\pi_b(B_x) \supseteq \pi_a(B_a)$:

\[
B_x \supseteq (\pi_c \circ \pi_b)^{-1} \circ (\pi_c \circ \pi_a)(B_a) \quad \Leftrightarrow \quad (3.6a)
\]
\[
\pi_b(B_x) \supseteq \pi_b ((\pi_c \circ \pi_b)^{-1} \circ (\pi_c \circ \pi_a)(B_a)) \quad (3.6b)
\]
\[
= (\pi_b \circ \pi_b^{-1} \circ \pi_c^{-1} \circ \pi_c \circ \pi_a)(B_a) \quad (3.6c)
\]
\[
= \pi_a(B_a) \quad (3.6d)
\]
The intuition behind this attack can be explained as follows. If $A_a$ and $A_b$ can send such packets to $A_c$ that the iBFs received by $A_c$ look the same, the iBFs in the packets must be the same also after the initial permutations $\pi_a$ and $\pi_b$.

### 3.3.4 Chaining the injection attacks

The attacker can also perform several injections one after another in order to gain a path to the target. Figure 3.11 illustrates a chained injection where the attacker injects traffic to one of its bots from which it is possible to inject traffic to the intended target. The attacker can inject traffic to the target from bot $A_d$ by first performing injection to $A_b$. These injections can be chained as long as the fill factor of the resulting iBF does not exceed $\rho_{\text{max}}$.

### 3.3.5 Evaluation

One of the central requirements for this attack is that the attacker is able to construct forwarding paths between its bots in such a way that the paths intersect with the valid path to the intended target. The effect of this topological requirement was evaluated using the measured Internet AS-level topology (see Appendix A for details) and with a 5-ary tree topology. The 5-ary tree topology was a full k-ary tree where $k = 5$, meaning that each node excluding the leaves has exactly five children. The depth of the tree was set to 7, which yields a network with 19,531 nodes out of which 15,625 are leaves. The bots and the target were randomized from the leaf nodes (i.e. the leaves were considered to be end-hosts and the other nodes routers). The evaluation with
both topologies was done assuming that FHIDs represent inbound-outbound interface pairs. In other words, the paths illustrated in above in Figure 3.10 had to intersect at two consecutive nodes to allow the injection.

In the case of the AS-topology, all of the calculated paths are shortest policy compliant paths. If there were several such paths, a random one of them was used. In other words, it was assumed that the attacker has no control over the path selection even though it knows what paths are being used.

The evaluation with both topologies was performed as follows.

1. Initially, \( n + 1 \) bots \((A_0, n)\) and one target \((T)\) were selected randomly. One of the bots, \( A_0 \), was assumed to have a valid path to the target (this corresponds to bot \( A_a \) in the figures).
2. The intersections of the valid path and all paths from \( A_0 \) to other bots were calculated. These intersections are represented partial paths from \( A_0 \) to other bots.
3. For every bot \( A_x \in A_{1..n} \), the shortest path to other bots (excluding bot \( A_0 \)) was calculated. If one of the shortest paths calculated from \( A_x \) intersects with one of the partial paths calculated previously with at least two nodes, it is possible to perform injection from this node.
4. The number of nodes that could perform the injection was counted.

This test was done with 100, 500, 1000, 5000, and 10000 bots in the AS-topology. In the 5-ary tree topology, smaller botnet sizes 10, 50, 100, and 200 were used. The test was repeated 1600 times with all botnet sizes in both topologies. The results can be seen in Figures 3.12–3.13, which show what fraction of the bots can inject traffic to the target — Figure 3.12 shows results for the AS-topology while 3.13 illustrates the attack efficiency in the tree topology.

It can be seen that the attack becomes more efficient when the attacker gains more bots: the cumulative distribution functions (CDF), which show the fractions of nodes that can flood the target, become less flat as a function of the botnet size. This means that when the number of bots is increased, a larger fraction of them can perform the injection. In the AS-topology, the attacker requires much more bots than in the tree topology in order to increase the attack efficiency: the median case in the AS-topology is that the attacker can inject traffic to the target only from less than 10% of the bots. This drastic difference between the two topologies can be explained by the fact that the AS-topology has dense core and many different paths through it while in a k-ary tree there is always only one path to the target and two paths are more likely to use it.

The impact of the the attack varies depending on the location of the bot \( A_0 \) which has a valid path to the target \( T \). Here, the target \( T \) and the bot
$A_0$, which has a valid path to the target, were selected randomly. It is likely that the attack would become more dangerous if the attacker could select the bot having the valid forwarding path to the target or if there were several of such bots. Additionally, the efficiency of the attack can be improved if $\rho_{max}$ is low and the injections can be chained as described in Section 3.3.4.
CHAPTER 3. SECURITY EVALUATION

Figure 3.12: Injection attack efficiency in AS-level topology [1] containing 33,508 nodes in total.

Figure 3.13: Injection attack efficiency in a full 5-ary tree of depth 7. The bots were selected from the $5^6 = 15,625$ leaf nodes.
3.4 Injection attack with unknown topology

This section takes the attack presented in the previous section one step forward and shows how the attack can be done without knowing the network topology. This makes the attack more complicated since the attacker does not initially know from which bots it can inject traffic to the legitimate delivery tree. In other words, the attacker does not initially know which paths intersect with the one valid path leading to the target. This section shows that it is possible to gain information about these intersecting paths without any prior knowledge of the network topology.

3.4.1 Attack model

Goal of the attacker: To launch unsolicited traffic from bots $A_{0..n-1}$ against a victim $T$.

Resources of the attacker: $n$ bots ($A_{0..n-1}$) that are placed at random network locations. The attacker has full control over the bots and can send iBF forwarded messages between any of them.

Prior knowledge of the attacker: The attacker has a valid forwarding identifier that can be used to send traffic from bot $A_a$ to the target $T$.

FHID: FHIDs identify either the next-hop router, next-hop link, or inbound-outbound link pairs and they are updated periodically after time $\Delta t$. Using the notation defined in Section 2.5.2, FHIDs could for example be determined by a function $Z(i_{in}, i_{out}, t, K)$.

Used security mechanisms: The FHIDs are secret. The iBF is permuted at each router. The permutation for each router is static but secret. The network topology is kept confidential.

3.4.2 The Attack

The attack presented here is essentially a way to deduce whether two or more paths intersect (i.e. a correlation attack). After knowing that paths between four nodes, three bots and the target, intersect as shown in Figure 3.10, it is possible to perform an injection attack similarly as explained in the previous section. The only difference compared to the attack described in the previous section is on how many nodes the different paths must intersect.

---

8The parameters for the $Z$-function may vary depending on whether FHIDs are used to identify nodes, links, or interface pairs.
Deducing whether two paths intersect

The key for determining whether two paths intersect is to find a way for an attacker to determine if two or paths contain the same FHID — if two paths contain the same FHID, they also intersect. This method, however, changes the topological requirements of the attack. Because the paths must share at least one FHID, they must intersect at three consecutive routers, assuming that FHIDs identify inbound-outbound interface pairs. This is shown in Figure 3.14. The requirement essentially means that that the paths $A_a \rightarrow T$, $A_a \rightarrow A_c$, and $A_b \rightarrow A_c$ must contain at least one common FHID. Naturally, this FHID cannot be trivially attained due to permutations.

Assuming the topology illustrated in Figure 3.14a, the attacker can gain information on whether two paths, for example, $A_a \rightarrow T$ and $A_a \rightarrow A_c$, have a common FHIDs by comparing the iBFs $(\pi_c \circ \pi_a)(B_a)$ and $(\pi_c \circ \pi_b)(B_b)$: if these iBFs contain many 1-bits at the corresponding positions, it is likely that the paths share FHIDs. Repeating the same process once in time period $\Delta t$ for the updated FHIDs, the attacker can gain more and more accurate information on the length of the shared part of the paths. It must be noted that here the attacker does not need to know the topology nor the individual permutations — it is sufficient to know the permutations between two bots just as in the previous section.

Once it is known that it is possible to gain information about the paths by comparing the dot products of different iBF pairs, what becomes interesting is that after inspecting several iBF pairs, with what certainty the attacker can say that the paths share or do not share a a common FHID. This can be calculated using information about the dot product of two random $m$-bit
long Bloom filters. The dot product of two random bit vectors follows the 
hypergeometric distribution\(^9\):

\[
P(X = x) = \binom{\rho_1 m}{x} \binom{m - \rho_1 m}{\rho_2 m - x} \binom{m}{\rho_2 m}^{-1},
\]

(3.7)

where \(m\) is the length of the iBFs, \(\rho_1\) and \(\rho_2\) are the fill factors of the two 
iBFs, and \(x\) is the dot product of the two Bloom filters. Hypergeometric 
distribution is a distribution telling the number of successes in a sequence 
taken from a finite population \textit{without replacement}. It is similar to the bi-
nomial distribution with the difference that binomial distribution calculates 
the number of successes \textit{with replacement}. The mean and variance of the 
hypergeometric distribution shown in Equation 3.7 are

\[
\mu = \rho_1 \rho_2 m
\]

(3.8a)

\[
\sigma^2 = \frac{m^2 \rho_1 \rho_2 (1 - \rho_2)(1 - \rho_1)}{m - 1}
\]

(3.8b)

Normal distribution offers a good approximation for the hypergeometric 
distribution when the fill factors are close to 0.5 [18, 23]:

\[
Hyp(\rho_2 m; \rho_1 m, m) \approx \mathcal{N} (\mu, \sigma^2)
\]

\[
= \mathcal{N} \left( \frac{\rho_1 \rho_2 m}{m - 1}, \frac{m^2 \rho_1 \rho_2 (1 - \rho_1)(1 - \rho_2)}{m - 1} \right)
\]

Knowing that normal distribution can be used to approximate the hyperge-
ometric distribution, it is easy to test whether the observations follow the 
proposed distribution. The attacker can for example use Student’s \(t\)-test to 
determine whether the observed dot products follow the distribution of the 
dot product of two random bit vectors. In other words, the null hypothesis 
could be \textit{“the dot product of the iBFs representing two paths is not larger 
than the dot product of two random bit-vectors of the same size”}.

Assuming that two \(m\)-bit Bloom filters both having a fill factor \(\rho\) contain 
one common element and thus always share at least \(c\) common bits. In this 
case, the value of the dot product follows the hypergeometric distribution

\[
P(X = x) = \binom{\rho m - c}{x} \binom{\rho m + c}{\rho m - c - x} \binom{m - c}{\rho m - c}^{-1}.
\]

(3.9)

\(^9\)Dot product of two random bit-vectors can be illustrated as follows: a \(m\)-bit long 
Bloom filter can be illustrated as an urn containing \(m\) balls that are either white (1-bits) 
or black (0-bits). The dot product of this and another \(m\)-bit long bit-vector containing 
i 1-bits, is then the number of white balls taken out of the urn with \(i\) takes \textit{without replacement}. The probability distribution of this is hypergeometric.
Figure 3.15 shows how many iBF pairs need to be investigated in order to gain 90% or 95% confidence on the null hypothesis using Student’s one-sample $t$-test. In the figure, it is assumed that the iBFs are 1024 bits long, all observed iBFs had fill factor $\rho = 0.5$, and all observations had mean and constant variance as described in Equation 3.8. The figure shows that it is relatively easy to see if the iBFs share many elements (i.e. when the number of common bits, $c$, is large). However, when $c$ is small, e.g. $< 10$, the required amount of iBF pairs increases rapidly.

In practise, the attacker does not need to know whether two paths intersect with high confidence. On the contrary, it is only enough that the attacker can target its resources to the most promising attack paths. This means that, using only a few observations, the attacker is able to sort the path pairs so that it can use its resources as efficiently as possible to the attack. If the attacker has more than one observation of each path pair, this sorting can be done based on the confidence intervals obtained with for example Student’s $t$-test using the null hypothesis described earlier: path-pair observations that suggest dropping the null hypothesis with high confidence are likely to be made on paths that share FHIDs.

The Student’s $t$-test, however, cannot be used if there is only one observation because positive sample standard deviation is required in order to use it. If there is only one observation of the dot products of each path pair, the sorting can be done based the probability that the dot product of two random bit strings results in a value lower than the observed value. In other words, the metric for sorting the observations is the value of the CDF of the probability distribution of Equation 3.7:

$$f(x, \rho_1, \rho_2) = P(X < x) = \sum_{k=1}^{x-1} \left( \frac{\rho_1 m}{k} \right) \left( \frac{m - \rho_1 m}{\rho_2 m - k} \right) \left( \frac{m}{\rho_2 m} \right)^{-1} \quad (3.10)$$

As can be seen above, the CDF is a function of the observed dot product $x$ and the fill factors $\rho_1$ and $\rho_2$ of the two bit strings.

**On the complexity of the attack**

As discussed in Section 3.2, there are $O(n^2)$ ways in which a bot pair can be chosen from $n$ bots. Additionally, in the Internet AS-topology, relatively many path pairs must be investigated before two paths sharing a FHID can be found when FHIDs are used to identify inbound-outbound interface pairs. A naive way of performing this attack would be following:

1. Find out the permutations for every bot pair $(A_i, A_j)$. This requires sending $O(n^2)$ messages.
2. Using the known permutations, calculate the dot product of the iBF of the valid path ($A_a \rightarrow T$) and the iBF of the path between each bot pair ($A_i, A_j$). This requires $O(n^2)$ messages.

3. Repeat the second step several times in order to get statistically significant results about the intersection sizes of the path pairs. It can be seen from Figure 3.15 that 5–40 repetitions usually give good confidence with 1024 bit iBFs.

4. Once statistically significant results are attained, the attacker can use the best paths for injection. Alternatively, the attacker start the attack before getting statistically significant results on what paths intersect. In this case the attacker can sort the attack paths and use the most promising ones as described earlier.

Even though the attack may seem to be very heavy to perform, it must be noted that each bot has to send only $O(n)$ messages. The computational complexity of the attack can also be slightly reduced if the attacker does not try to determine whether the valid path (i.e. $A_a \rightarrow A_c$) and the paths between each bot pair (i.e. $A_b \rightarrow A_c$ in the figures) intersect. Instead, the attacker could first calculate the dot product of iBFs $B_b$ and $B_c$. The reasoning behind this is that if the iBFs $B_c$ and $B_b$ have a large dot product, then it is more likely that these paths have a long intersection, which intuitively implies that there is a larger probability that also paths $A_a \rightarrow T$ and $A_b \rightarrow A_c$ (i.e. iBFs $B_a$ and $B_b$) intersect.
3.4.3 Evaluation

An injection attack that is done without knowing the network topology does not differ much from the case where the topology is known. The main difference is how the paths must intersect in each case. If the topology is known, it is sufficient that paths “touch” each other, i.e. their concatenation forms a continuous path, in the connectivity graph. If the topology is unknown, the paths must actually intersect in the connectivity graph. This means that, if FHIDs identify inbound-outbound interface pairs, it is enough that the paths intersect at two consecutive nodes when the topology is known. On the other hand, when the topology is unknown, the paths must intersect at three consecutive nodes. Thus, the evaluation of this attack can be done similarly to the evaluation of the injection attack when the topology is known. The only difference is that here the injection is considered to be possible only if the relevant attack paths intersect at three consecutive nodes.

Depending on the connectivity graph of the topology, the requirements that the paths $A_a \rightarrow T$ and $A_b \rightarrow A_c$ must have a common FHID may make the attack vastly more difficult. This can be seen from Figure 3.16, which shows the fraction of bots that can inject traffic to the target given that one (randomly selected) bot has a valid path to the target. In both figures, the FHIDs are used to identify inbound-outbound link pairs. Thus, two paths sharing a single FHID must intersect at three consecutive nodes. Figure 3.16a shows the situation in the AS-topology while Figure 3.16b shows the attack efficiency in full 5-ary tree that has depth 7. These figures can be compared to Figures 3.12 and 3.13, which illustrate the same situation but where the paths must intersect only at two consecutive nodes. It can be clearly seen that, in the AS-topology, which has a short diameter, dense core, and very large connectivity graph (see Table 3.1 and Appendix A), the attack is ineffective. However, the efficiency of the attack is rather good in the tree topology. This hints that the size of the connectivity graph may correlate with how difficult the injection attack is to perform. It must be noted that here the source of the valid path (i.e. $A_a$ in the path $A_a \rightarrow T$) was chosen randomly. If the attacker was able to choose the node from which the valid path leaves, the average attack efficiency would be higher.
Figure 3.16: Efficiency of the injection in different topologies when the attacker does not know the network topology. Injection becomes more difficult to perform because the paths must intersect in three consecutive routers when FHIDs are used to identify inbound-outbound link pairs. Similarly as in the case of known topology (see Section 3.3), the attack efficiency is affected much by the network topology.
CHAPTER 3. SECURITY EVALUATION

3.5 Injection attack when permutations change as often as FHIDs

The injection attacks presented in the previous two sections have been based on the assumption that the attacker can learn the permutation functions fully between any two bots. One might think that the attack would become much more difficult to perform if the permutation functions were changed as often as the FHIDs, namely once in $\Delta t$. In this scenario the attacker could solve only parts of the permutation functions. However, as it will be shown in this section, even this does not give full protection against the injection attack: if the average fill factor for unicast paths is not close to $\rho_{\text{max}}$, it still is be possible to perform the injection attack.

3.5.1 Attack model

Goal of the attacker: To launch unsolicited traffic from bots $A_{0..n-1}$ against a victim $T$.

Resources of the attacker: $n$ bots ($A_{0..n-1}$) that are placed at random network locations. The attacker has full control over the bots and can send iBF forwarded messages between any of them.

Prior knowledge of the attacker: The attacker has a valid forwarding identifier that can be used to send traffic from bot $A_n$ to the target $T$.

FHID: FHIDs identify either the next-hop router, next-hop link, or inbound-outbound link pairs and they are updated periodically after time $\Delta t$. Using the notation defined in Section 2.5.2, FHIDs could for example be determined by a function $Z(i_{\text{in}}, i_{\text{out}}, t, K)$.\(^{10}\)

Used security mechanisms: FHIDs are secret. The iBF is permuted at each router. Permutations are secret for each router and they are updated periodically at the same time with FHIDs. The network topology is kept confidential.

3.5.2 The attack

The attack presented in the previous section is based on the idea that an attacker can compute the permutation function that is used between any two bots. This is not possible, however, in this scenario due to the fact that the permutations change as often as the FHIDs. Because of the changing permutations, the attacker can only learn how the 0-bits of the iBF are permuted

\(^{10}\)The parameters for the $Z$-function may vary depending on whether FHIDs are used to identify nodes, links, or interface pairs.
between two bots — as explained in Section 3.3.2, it cannot compute how the individual 1-bits are permuted.

Despite the fact that the attacker cannot learn the permutation functions completely, the attack is still possible if the fill factors of the iBFs between the bots are small enough. Using the same reasoning as in Section 3.3.3, the attacker can construct an iBF that can be used to inject traffic towards the target as follows:

\[
B_x = (\pi_c \circ \pi_b)^{-1} ((\pi_c \circ \pi_a)(B_a \lor B_c)) \lor B_b \tag{3.11}
\]

Unlike Equation 3.5, which works if the attacker knows the permutation functions accurately, this equation works also if the attacker does not know how the permutation functions permute the individual 1-bits of the iBF. This is because even though the attacker does not know for example the permutation function \(\pi_c \circ \pi_a\) completely, it knows the result of \((\pi_c \circ \pi_a)(B_c)\). In other words, the attacker can calculate the result of permutation \((\pi_c \circ \pi_a)(B_a \lor B_c)\) because

\[
\begin{align*}
(p_c \circ p_a)(B_a \lor B_c) &= (p_c \circ p_a)((B_a \land \neg B_c) \lor B_c) \\
&= (p_c \circ p_a)(B_a \land \neg B_c) \lor (p_c \circ p_a)(B_c) \tag{3.12c}
\end{align*}
\]

The attacker can compute formula 3.12c because the left part of the formula involves only using the known part of the permutation function and the right part of the formula is known. The same logic applies also to the permutation \((\pi_c \circ \pi_b)^{-1}\) in Equation 3.11.

The only drawback of Equation 3.11 is that, because more iBFs are combined, there is greater risk of exceeding the maximum fill factor \(\rho_{\text{max}}\). If the fill factors are low, namely \(\rho_a + \rho_b + \rho_c \leq \rho_{\text{max}}\), it is always possible to construct an injection path towards the target. In practice, injection is often possible even when the fill factors are larger than this because the paths have common FHIDs and thus also the iBFs have many 1-bits at the same positions.
3.6 Injection attack and flow specific FHIDs

Heretofore in this thesis, it has been assumed that the FHIDs are static for some time $\Delta t$, which has made it possible to for example determine if two paths share the same FHID. However, many of the iBF proposals utilize cryptographic FHIDs [33, 34]. With the cryptographic FHIDs, the identifiers are generated with a $Z$-function that takes parameters such as an identifier of the source and a unique flow identifier. The idea is that the FHIDs are different for each traffic flow and thus the attacker is unable to concatenate paths, which would make the injection attack impossible. However, the use of flow specific FHIDs also presents rather exacting requirements for the architecture.

Cryptographic FHIDs protect against injection attacks because the iBFs are different depending on the sender and the flow identifier. As described earlier in this thesis, when FHIDs are static, they can be considered to be outputs of a function $Z(i_{in}, i_{out}, K, t)$, where $i_{in}$ and $i_{out}$ are the identifiers of the inbound and outbound interfaces, $K$ is a router’s secret key, and $t$ is timestamp. Cryptographic FHIDs also depend on flow-specific parameters such as $f$ and $s$ that identify the flow and the source of the packet. This gives protection against the injection attack since the FHIDs that are on the valid path ($A_a \rightarrow T$) work only if the packet is sent from $A_a$. Naturally, using cryptographic FHIDs means that the router must calculate the FHIDs for each packet.

Even though many iBF forwarding proposals employ flow identifiers in the $Z$-function, how the flow identifier is obtained is often brushed aside [29, 33]. A trivial way of obtaining a flow identifier for unicast flows would be that the receiver of the information could simply choose it. This may appear to be a secure solution since the sender of the information does not have any power over the selected flow identifier and thus also the sender cannot affect the resulting FHIDs and iBFs. However, in the injection attack, the attacker communicates between its bots. Even if the receiver was responsible for choosing the flow identifier, the attacker could choose the flow identifiers used in bot-bot communication. The attacker can also choose the same flow
identifier for all flows and it can copy the one from the valid path to the target. Thus the injection attack would not be more difficult to perform than the attacks presented in previous sections. On the contrary, if the receiver of the data can choose the used flow identifier, the injection attack presented in this thesis actually becomes easier. This is due to the fact that the injection attack involves finding out permutations between two bots (see Section 3.3.2) and, if the topology is unknown (see Section 3.4.2), potentially finding out shared FHIDs between several paths. These operations become easier if the attacker can freely create several valid iBFs between the bots since the requirement of waiting $\Delta t$ time between the new iBFs would be removed.

Flow specific FHIDs, however, give one major advantage for the victim of the attack: once the victim notices that it is under a DoS$^{11}$ attack, it can blacklist the bot that was originally given the valid iBF and flow identifier. The next time when the FHIDs are updated, the victim does not give an updated iBF to that bot. Because of this, the attack can be continued approximately only for time $\Delta t$ per each flow identifier the attacker has. Thus, for the attack to be more efficient, several bots must have a valid iBFs to the target. Out of these valid iBFs, only one is used at a time — each time the FHIDs are updated, the attacker moves to use the next iBF.

The forwarding architecture would be resistant to injection attacks if the flow identifiers were chosen by a trusted network element and if communication with fabricated forwarding identifiers were blocked near the sender. This is, however, a very strong requirement for the architecture: it would not be sufficient that flow identifier is chosen by a trusted entity, but also that falsified flow identifiers are blocked, which essentially means egress filtering. In other words, domains must validate that every packet originating from their network uses a flow identifier that is legitimate. The problem is the same if the identifier of the source of the packet is used as one of the parameters for the $Z$-function$^{12}$: the source node must perform egress filtering for all of the packets and validate that the source identifiers are not spoofed. Additionally, the node performing the egress filtering must naturally be trusted.

These requirements are analogous to IP spoofing in current Internet architecture. In order to prevent IP spoofing, all domains should be honest and perform egress filtering for data sent by untrusted end nodes. It is well known, that this is not the case: IP spoofing is still a problem [10]. This shows that basing the security of an architecture on this kind of assumptions may be unrealistic.

$^{11}$For the victim, the attack does not appear to be distributed (i.e. DDoS) even though in reality it is.

$^{12}$In data-centric networking this could be the identifier of the data-object.
Chapter 4

Discussion

The attacks presented in this thesis show that, unlike previously believed (see e.g. [29]), the iBF forwarding does not provide inherent protection against DoS attacks. This chapter discusses the implications of the presented attacks and summarizes the weaknesses in the proposed iBF forwarding schemes. The results suggest that the presented vulnerabilities cannot be fixed easily: even though it is not impossible to construct a secure iBF forwarding scheme, the requirements for such an architecture appear to be unrealistic. This also increases the significance of the found vulnerabilities.

This chapter is divided into three sections. At the beginning, Section 4.1 goes through some of the proposed iBF forwarding protocols and explains how they are affected by the injection attack. Section 4.2 goes into more detail in the implications of the injection attack and what it means for secure iBF forwarding system design. Finally, Section 4.3 presents a brief evaluation of the methodology used in this thesis.
4.1 Proposed forwarding architectures and their resistance against injection attack

Free Riding Multicast (FRM [26]) does not have any security or anomaly prevention mechanisms in it. The FHIDs used in the protocol are publicly known, which makes it vulnerable to a wide range of attacks including but not limited to injection attacks, forwarding loops, and packet storms.

The security of LIPSIN is based on two mechanisms: the FHIDs are secret, and every link has several FHIDs, out of which one is used based on an in-packet hop-counter. Additionally, loop avoidance is implemented statefully by caching recent packets. These mechanisms, however, do not provide protection against any of the attacks presented in this paper — as explained in Section 3.2, the security of the forwarding system cannot be based only on the secrecy of the FHIDs due to the fact that an attacker can compute FHIDs using a botnet.

Bloomcast [32–34] was one of the first iBF protocol proposals where security was one of the main goals. The security features of the protocol are mostly from [29]. Bloomcast uses dynamic FHIDs, static permutations, and maximum fill factor. The problem, however, is that Bloomcast uses the \((source, group identifier)\) pair as a flow identifier, which can be forged unless all ASes are honest and perform egress filtering. If the attacker can forge the flow identifiers, the system is vulnerable to the injection attack as described in Sections 3.3–3.6 in this thesis.

In LANES [37], iBF forwarding is only intended for unicast forwarding and multicast is solved by storing state in the network. Regardless, LANES is vulnerable to the injection attack. This is due to the fact that even though iBF forwarding is intended only for unicast, multicast cannot be inhibited and the attacker can still encode a multicast tree into the iBF. What may provide some protection to the LANES architecture is the fact that, because iBF forwarding is intended only for unicast, the parameters of the Bloom filters may be set in such way that only a small number of FHIDs can be added to the iBF. This would make the attack more difficult because the attacker could not add many FHIDs to the filters without exceeding the maximum fill factor. However, it does not provide full protection and the effectiveness of the protection is highly depended on the network topology.

4.2 Implications of the described attacks

This thesis shows that the claims initially presented in [16] and later elaborated in [29] about the DoS resistance of the iBF forwarding are exaggerated.
Even though sending unsolicited traffic is more difficult in iBF forwarding system compared to the current Internet architecture, existing iBF forwarding schemes do not solve the problem with any serious level of security.

As described in Section 3.6, the attacks do not work if dynamic FHIDs are used and if the attacker cannot influence how the FHIDs are formed for each flow. This can be stated so that the Z-function must have an argument that is both unique to each flow and that the attacker cannot influence. This requirement is strong especially in host-centric networks where a unique flow identifier is required:

- The requirement implies that there is a trusted entity or entities which generate the flow identifiers.
- Flow identifier forgery should be made impossible, which implies that trusted entities should be able to validate the flow identifiers. Assuming that all ASes can be trusted, this could be implemented as egress filtering. An alternative solution would be to sign the flow identifiers provided by a trusted entity (e.g. packet-level authentication [17]). However, public-key cryptography contradicts with the main ideas of iBF forwarding, which is inexpensiveness and efficiency in the forwarding nodes.

Alternatively, dynamic permutations (see Section 2.5.4) could be used to secure iBF forwarding. However, as described earlier in this thesis, dynamic permutations are relatively difficult to implement efficiently and cheaply thus making them less feasible. Furthermore, dynamic permutations have not yet been studied at all in the context of iBF forwarding.

iBF forwarding could also be secured with a solution similar to Accountable Internet Protocol (AIP [3]), which is an architecture proposal for IP networks that would solve problems such as source-address spoofing and DoS. As its name suggests, the proposal introduces accountability to the network by using self-certifying addresses [21], which are essentially plain public keys or hashes of them. One challenge of AIP, however, is that it needs to be deployed relatively widely before it can provide a serious level of security. Additionally, whether the Internet should have accountability as an in-built feature is a subject for debate.

This thesis has discussed iBF forwarding mainly in context of host-centric networking despite the fact that iBF forwarding has also been proposed as a forwarding fabric in data-centric networks (e.g. in [16]), where data junks are addressed instead of endpoints. One property of data-centric networks is that, because addresses point to data objects, hosts cannot necessarily influence where the data comes from [2, 15]. This makes caching more easy because the source of the data becomes irrelevant — only the content of the data matters. It has also been claimed that this kind of architecture
would make DoS attacks more difficult to carry out because the often re-
quested data-objects would be cached at several network locations thus frus-
trating any DoS attempts on one data-object [2]. However, the research on
data-centric networking is still ongoing and conclusive evidence about the
feasibility of such architectures has not yet been presented.\footnote{For ongoing research projects, see www.fp7-pursuit.eu (Publish Subscribe Internet Technology) and www.ccnx.org (Content-Centric Networking).}

It must be emphasized that this thesis has only focused on situations
where the endpoints cannot be trusted. The results do not suggest that iBF
forwarding would be insecure in trusted environments, such as inside a single
domain as in [38]. Additionally, it must be emphasized that the results
of this work do not indicate that iBF forwarding is completely insecure.
On the contrary, the results indicate that iBF forwarding is not inherently
considerable more secure than traditional IP forwarding.

4.3 Evaluation of the methodology

The attacks presented in this thesis were evaluated analytically and with
high level simulations. The abstraction level of the thesis was motivated by
the fact that no standard iBF protocol has been specified yet — low level
evaluations would have required several assumptions that would have made
the results less general and in the worst case even partially questionable.
The chosen methods also allowed simultaneous investigation of several iBF
protocols regardless of their minor differences.

However, the chosen methodology may have introduced inaccuracies for
the evaluation results. It is very difficult to determine accurately the real
complexity of the attacks without detailed knowledge of the protocol and
the environment where the protocol is being used. It was, for example,
oberved that the network topology has a large effect on the severity of
the attacks. Additionally, even small implementation details, such as the
number of used hash function $k$, may affect the difficulty of carrying out the
presented attacks, which reduces the generality of the evaluation presented
in this thesis.

It can be said that, while the high level of the analysis made it possible to find new vulnerabilities in the design of iBF forwarding, the lack of de-
tailed specifications made it hard to evaluate the difficulty of the attacks. If more detailed protocol descriptions were available, low-level analysis, such as packet-level simulations, could provide more credible evaluation results.
Chapter 5

Conclusion

This thesis is, according to the author’s knowledge, the first work analysing the security of iBF forwarding against intentional attacks. The results show that iBF forwarding is not as secure as has been claimed and that iBF forwarding is particularly vulnerable to attacks that employ botnets. This can be seen as a reminder that security claims should be presented with extreme care due to the fact that vague claims are often shown false after a more careful study.

The main contributions of this thesis are

1. to present a concept called connectivity graph that enables a normalized way of investigating different iBF forwarding schemes,

2. to show that it indeed is possible to gain information about the used FHIDs (i.e. correlation attack), and

3. to show that the security mechanisms presented in the literature (i.e. in [16, 29, 34]) do not provide sufficient security against an injection attack and an attacker controlling a bot network can inject traffic to a victim assuming that the attacker has a single valid path to it.

The connectivity graph, which is the first contribution of this thesis, turned out to be useful in the study of iBF forwarding. It does not only help conceptualizing iBF forwarding, but it also shows that regardless of what FHIDs identify, iBF forwarding can be described using the same notation. The possibility of normalizing different iBF forwarding schemes also implies that the fundamental properties of different iBF forwarding schemes are the same regardless of what the FHIDs identify.

The second contribution of this thesis was to show that secret FHIDs do not provide security due to the fact that it is possible to gain information about individual FHIDs by investigating the iBFs. Even though this result was expected, the attack has not appeared earlier in the literature.
Clearly the most notable finding of this thesis is the injection attack due to the widespread conception that this kind of attack is not possible in iBF forwarding. The presented attack shows that the currently used security mechanisms do not give protection against unsolicited traffic: the attack is possible even with an unknown topology and periodically changing FHIDs. Because of this, all of the proposed iBF forwarding schemes are vulnerable to this attack. Flow-specific FHIDs, which are used in the most recent iBF forwarding proposals, could provide protection against the attack but only if the end-hosts cannot influence the chosen flow identifiers. This, however, is a strong requirement due to the fact that it suggests that a trusted entity would generate the flow identifiers and it would be impossible to forge one.

These results, however, must not be seen as a conclusion of the security analysis of iBF forwarding. On the contrary, this thesis takes only the first steps in the security analysis. Possible future research topics include:

- To analyse whether a malicious attacker can construct forwarding paths where packets are routed into a loop, thus enabling traffic amplification. Permutations appear to protect against infinite forwarding loops, but also finite loops can be used in this kind of attacks.

- To study further what trusted entities an iBF forwarding system requires, and to what extent it is possible to secure an iBF forwarding architecture assuming that there are malicious entities in the network.

Regardless of the many open security questions and the findings of this thesis, nothing indicates that iBF forwarding would be particularly insecure. The findings of this thesis merely show that unlike was previously believed, iBF forwarding is not inherently more secure than the current networking solutions. Additionally, even if more vulnerabilities similar to the ones presented in this thesis were found from iBF forwarding, it could still be used in trusted environments, e.g. as an intra-domain underlay protocol [38]. Because of this, it is easy to believe that iBF forwarding has a future also outside academic research communities.
Bibliography


Appendix A

CAIDA policy annotated AS-graph

This thesis uses policy annotated AS-graph produced by CAIDA (The Co-operative Association for Internet Data Analysis) [1]. The dataset contains the Internet topology on autonomous system (AS) level. The links are policy annotated meaning that each link is either of type client-provider, provider-client, peer-peer, or sibling-sibling. The policies represent the business relationships between the ASes: client-provider relationship means that the client pays to the provider for the bandwidth while peer-peer relationship implies mutual traffic exchange. Sibling-sibling relationship is more rare and is observed mainly between ASes belonging to the same organization.

In policy graphs, a valid policy-compliant path between two nodes is valley-free. A valley-free path has three parts [12]:

1. Uphill part of the path, which is a sequence of domains that have client-provider or sibling-to-sibling relationship.
2. In the middle there is one or zero peer-to-peer relationship connections.
3. After the peak of the route comes the downhill path, which contains only provider-client and sibling-to-sibling relationships.

The dataset used in this document is generated at 20th January, 2010 [1]. The dataset contains 33 508 ASes in total. Table A.1 shows the key figures of the dataset. Figure A.1 shows the degree distribution of the graph. It can be noted, that the degree distribution is power-law distributed, which is typical for scale-free topologies [20].
Dataset date: 20th January, 2010
Amount of ASes: 33,508
Links in total: 75,001
Average degree: 2.24
Average hop-count between two nodes: 3.9 (average of 1000 randomly selected node pairs)

Table A.1: Characteristics of the used AS-relationships topology

Figure A.1: Node degree distribution of the Internet AS-topology.
Appendix B

Derivation of Equation 3.2

B.1 Problem

Given following:

- $n$ distinct sets $S_1..S_n$ such that $\forall a, b : |S_a \cap S_b| = 1$ and
- $n$ Bloom filters $B_1..B_n$ describing the sets. All of the Bloom filters have the same fill factor $\rho$.

What is the expected number of sets required (i.e. the value of $n$) in order to get the exact Bloom filter that describes the element at the intersection $S_1 \cap S_2$.

B.2 Solution

The aim is to get an approximate value for the median number of Bloom filters that are required in order to learn the exact Bloom filter representation of the element in $S_1 \cap S_2$. Knowing the expected dot product of the two Bloom filters (i.e. $B_1 \cdot B_2$), it also possible to calculate how many "extra" 1-bits there are in $B_1 \land B_2$ that do not belong to the $S_1 \cap S_2$. Let us denote the expected amount of extra 1 bits with $d$:

$$d = B_1 \cdot B_2 - m \left(1 - \left(1 - \frac{1}{m}\right)^k\right)$$

The probability that a given bit is one in a Bloom filter equals to the fill factor of the filter, $\rho$. Thus, the probability that the $n$th Bloom filter $B_n$ is the first filter having the given bit set to 0 follows geometric distribution $P[X = n] = (1 - \rho) \times \rho^{n-3}$, where $n \geq 3$, and $X$ is the random variable.
The CDF of this distribution is $P[X \leq n] = 1 - \rho^{n-2}$. The median of the distribution is easy to obtain by setting the CDF to 0.5 and solving $n$.

An approximate solution for the problem can be derived by treating all of the $d$ bits can be treated independently. It other words, it is assumed that an event “bit $b_a$ is 1 on BF $B_x$” and “bit $b_b$ is 1 on BF $B_x$” are assumed to be independent. From this, it follows that the approximate median value for $n$ is:

$$(1 - \rho^{n-2})^d \approx 0.5 \quad \Leftrightarrow \quad (B.1)$$

$$\rho^{n-2} \approx 1 - \sqrt[0.5]{0.5} \quad \Leftrightarrow \quad (B.2)$$

$$n \approx 2 + \frac{\ln (1 - \sqrt[0.5]{0.5})}{\ln \rho} \quad (B.3)$$

The error that follows from the assumption is reasonable. This can be seen in Figure B.1 that shows the approximate value derived using Equation B.3 and the actual value derived empirically with 256 and 1024-bit long Bloom filters. Despite the decision to treat each bit independently the error is not large.
Figure B.1: The number of Bloom filters, which share a single element, required to elicit the exact bit representation of the element at the intersection of the Bloom filters. Figures show the actual mean and median values (derived empirically) and the approximate median got with Equation B.3.