A Practical IP Spoofing Defense Through Route-Based Fltering

Jelena Mirkovic, Nikola Jevtic and Peter Reiher

Abstract

We present the design and evaluation of the Clouseau system, which together with route-based filtering (RBF) acts as an effective and practical defense against IP spoofing. RBF's performance critically depends on the completeness and the accuracy of the information used for spoofed packet detection. Clouseau autonomously harvests this information and updates it promptly upon a route change. RBF information is inferred by filters applying randomized drops to TCP data traffic, which arrives from suspicious or previously unknown sources, and observing subsequent retransmissions. No communication is required with packet sources or other RBF routers, which makes Clouseau (and RBF) suitable for partial deployment. We show through experiments with a Clouseau prototype that the operation cost is reasonable and the legitimate TCP connections do not experience large delays because of randomized drops. The inference process is resilient to subversion by an attacker who is familiar with Clouseau. We motivate our work by showing that RBF brings instant benefit to the deploying network, and that it can drastically reduce the amount of spoofed traffic in the Internet if deployed at as few as 50 chosen autonomous systems.

1 Introduction

IP spoofing accompanies many security attacks, such as flooding denial-of-service (DDoS) and vulnerability scanning, and hinders the design of simple, cost-effective defenses. These threats use spoofing to blend the attack with the legitimate traffic and thus avoid identification of attack machines. A simple defense approach collects statistics of source IP sending behavior and uses them to spot either persistent large senders (to be blacklisted as suspicious) or recurring moderate users (to be given high-priority as long-term clients). IP spoofing can defeat this approach in four ways: (1) random spoofing in the entire IPv4 space creates numerous records than cannot be stored and processed, (2) an attacker can spoof a limited set of addresses consistently to fake moderate-sending patterns that do not stand out from legitimate traffic, (3) an attacker can briefly spoof each of many small address sets in turn — when one set is blacklisted the attacker moves to another set, and (4) an attacker can assume a good user's identity (by spoofing his IP address) and ruin that user's reputation. IP spoofing is also essential to *reflector attacks*, where an attacker spoofs the victim's IP in legitimate requests for service (e.g., DNS requests) that are sent to numerous public servers. The servers flood the victim with replies, creating a denial-of-service effect.

It is impossible to completely eliminate spoofing without deploying sender address verification such as ingress filtering [6] or SPM [11] at every network in the Internet, which is unrealistic. Practical spoofing defenses focus instead on limiting attacker's choices of spoofed addresses and spoofed traffic destinations. This reduces the severity of DDoS and reflector attacks and simplifies defenses.

In [9], Park et. al show how route-based filtering (RBF) can be an effective spoofing defense if deployed at a vertex cover of the autonomous system (AS) map. A router deploying route-based filtering keeps an *incoming table*, which links each source IP address with the expected incoming interface. Packets that arrive on unexpected interface are dropped as spoofed. So far, no practical approach was proposed for constructing incoming tables and maintaining them in face of routing changes. Also, while results from [9] indicate that vertex-cover deployment prevents spoofing from 84% of ASes, the required number of filters is prohibitively large for a realistic deployment.

The main contribution of our paper is the design of the Clouseau system, which builds incoming tables and updates incoming information when routing changes occur. Table contents are inferred by routers applying randomized drops to TCP data packets that arrive from suspicious or previously unknown sources, and observing subsequent retransmissions. No communication is required with packet sources or other RBF routers, making Clouseau (and RBF) suitable for partial deployment. We show through experiments with a Clouseau prototype that the operation cost is reasonable and legitimate TCP connections do not experience large delays because of randomized drops. The inference process is resilient to subversion by an attacker who is familiar with Clouseau.

Further, our paper revisits the analysis of RBF effectiveness and makes the following contributions:

1. We show that RBF can reduce spoofing to 2.5% when deployed at as few as 50 carefully chosen ASes. Ideal deployment points are ASes that are highly-connected and appear on many source-destination paths. Further investigation of top-50 filtering ASes in the period 2001—2005 shows that about 34—40 of them belong to the same 18 tier-1 Internet Service Providers. We identify these ISPs as the ideal target for RBF deployment.

2. We show that the benefits of RBF are directly experienced by filtering ASes and their customers, which creates excellent deployment incentive.

2 Route-based filtering

A route-based filter detects spoofed packets by comparing each packet's incoming interface with the *expected* interface associated with the packet's source IP in the incoming table. Figure 1 illustrates a case where filtering router R has information about 3 sources in its incoming table: sources A and B arrive over interface 2 and source C arrives over interface 1. For a moment, we will ignore parts of this network that are drawn in dashed lines. Source A is compromised and sends two spoofed packets to destination D. In the first packet, A spoofs C's IP address; this packet is detected by R as spoofed since it arrives on interface 2, while the expected interface is 1. In the second packet, A spoofs B's IP address. R cannot detect that this packet is spoofed since its incoming interface is same as the expected interface for B. This example illustrates an important property of RBF: If the route from source S1 to destination D1 overlaps the route from source S2 to destination D2 (possibly D2 $\neq$ D1), S1 and S2 can spoof each other and avoid detection by RBF routers placed downstream from the overlap. Clearly, the placement of filtering routers will greatly influence filtering effectiveness. For instance, if router Q (from Figure 1) were chosen for filter deployment instead of R, all three sources could spoof each other. If, on the other hand, P and R were both filters no spoofing would be possible between A, B and C.

An alternative RBF design maps source IP-destination IP pairs in the incoming table to an expected interface. Park et al. [9] call this a *maximal filter* while RBF with source-only information is called a *semi-maximal* filter. Maximal filter storage has a cost $\mathcal{O}(N^2)$, where N is number of possible sources and destinations, while a semi-maximal filter costs $\mathcal{O}(N)$. The effectiveness of maximal filters is only slightly higher than that of semi-maximal filters [9], which is insufficient to warrant enormous storage cost. Still, there may be situations where a semi-maximal filter would mistakenly filter out legitimate traffic, when a maximal filter would not. Figure 1 illustrates this case, with dashed items included. Let source A reach destination D via P-R-Q and destination E via T-R-Q. This creates two expected interfaces for A at R. If R is a semi-maximal filter and only stores one expected interface. then some legitimate traffic will be filtered out as spoofed. If R is a maximal filter it would properly record that A's packets come via interface 2 for destination D and via interface 3 for destination E, so this argument speaks in favor of maximal filters. On the other hand, semi-maximal filters can be extended to properly handle the above case at a much lower storage cost — by allowing multiple expected interfaces. The price we pay is lower filtering accuracy as multiple expected interfaces create more holes for the spoofed traffic to pass through. It is difficult to estimate how frequent is the above routing scenario in the Internet. We limit our discussion to semi-maximal filters with a single expected interface but this design can be extended to support multiple interfaces if necessary.

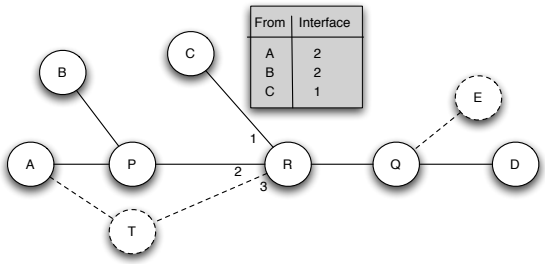

Figure 1: Illustration of route-based filtering

2.1 Analyses setup and objectives

This section investigates the effectiveness of RBF under very sparse deployment. We assume that each incoming table entry relates to one source IP and postpone discussion of entry aggregation for Section 5. We also assume that each RBF node performs ingress filtering [6], which forces outgoing packets from the deploying network to carry a source address from this network's address range. As ingress filtering is much simpler and cheaper than RBF this is a reasonable assumption. Our goal is to: (1) evaluate who experiences direct benefit of RBF — filters, their customers or some remote third party (2) define the optimal strategy for choosing ASes that deploy RBF to minimize the number of deployment points for a given target effectiveness, and (3) determine a small, critical set of ASes whose participation in RBF is crucial for overall spoofing reduction in the Internet.

Like [9], we perform effectiveness analysis on the AS map. RBF is deployed at all border routers of the chosen ASes, but we consider the size of each AS when we calculate filtering effectiveness; our measures are at the IP level while analyses in [9] are at the AS level. IP-level analysis is more accurate because (1) the spoofing capacity of an AS is proportional to its size, size of the reachable targets and number of IP addresses this AS can spoof, and (2) the analysis makes a realistic assumption that the whole IPv4 space can be spoofed, not only IP addresses allocated to ASes.

The connectivity information and the size of IP space allocated to each AS is obtained using BGP information collected by the RouteViews project [1]. Only about 0.15% of all the ASes participate in the RouteViews project, so the inferred connectivity data is a subset of AS-to-AS links observed in the real world. [1]

We assume that all the routers in a given AS have the same forwarding table which means that there is a single route for all the sources from this AS to a given destination. In reality, this may not be true for ASes that span large geographic regions, but each such AS can be transformed into several smaller, directly connected ASes, each with a single forwarding table, that satisfy the assumption. Such a transformation

[1] We also experimented with additional connectivity data from UCLA's Internet Topology project (http://irl.cs.ucla.edu/topology), which provides information about 40% more links than the RouteViews project. We did not notice any significant impact on RBF effectiveness when these additional links were added to the AS map.

would not change results of our analysis. We populate forwarding tables using the following steps:

1. Fully populate the forwarding tables of the ASes participating in the RouteViews project with nexthop information specified in as_path entries.
2. Partially populate forwarding tables of ASes that do not participate in the RouteViews project, but appear in as_path entries of other participants. For example, if AS 1 participates in RouteViews project and reaches destination X via ASes 2, 3 and 4, path 1-2-3-4 will exist in BGP dump logs of AS 1. In step 1, we specify 2 as nexthop in AS 1 for destination X. In step 2 we will insert 3 as nexthop for AS 2 to X, 4 as nexthop for AS 3 to X and we will mark 4 as the origin AS for X.
3. Populate the remaining forwarding entries by calculating shortest path route on the AS map for each vacant entry.

Let IP_{allc} be the set of IP addresses allocated to ASes and $IPv4$ be the set of all possible IP addresses. During the analysis, we observe packets sent from source address $s \in IP_{\text{allc}}$ to destination address $d \in IP_{\text{allc}}, d \neq s$, spoofing the address $p \in IPv4$, $p \neq s$. We also assume that RBF can detect and eliminate packets that spoof non-allocated IP addresses (i.e., $p \notin IP_{\text{allc}}$). We show how Clouseau can help RBF detect such addresses in Section 3.

We note that there are three dimensions of spoofing that filtering may reduce: (1) attacker dimension — how many machines s are usable for spoofing, (2) spoofing dimension — how many spoofed addresses p can be placed in packets that are not filtered by RBF, and (3) target dimension — how many destinations d receive spoofed packets. Reducing the attacker's options in any of these dimensions is beneficial and would simplify defenses. Limited attacker dimension would result in a few machines being usable for spoofing any address to any target — in this case defenses could concentrate on monitoring and policing traffic in portions of the Internet containing usable machines or on marking this traffic as suspicious and serving it with low priority. Limited spoofing dimension would mean that every machine on the Internet can only spoof a small amount of addresses to any destination. This forces the attack traffic to stand out in statistics of source IP behavior and facilitates easier attack detection. Finally, with limited target dimension only a few targets would be exposed to arbitrary spoofing from anyone. Those unfortunate sites can improve their protection by deploying RBF themselves.

2.2 Where to filter?

Two main benefits of reduced spoofing would be: (1) a node would be less likely to be a target of diverse spoofed traffic (although packets that spoof addresses from a specific set may still be able to reach this node) and (2) a node's address would be less likely to be misused in reflector attacks. We capture these two benefits with *target measure* and *stolen address measure*. Target measure tells us how many (s,p) combinations are possible for a given d. Stolen address measure tells us how many (s,d) combinations are possible for a given p. We calculate these values relative to the number of combinations possible without filtering, so they range from 0 to 1, with lower values denoting better protection.

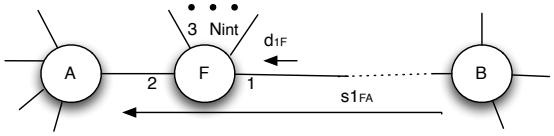

Figure 2: Illustration of RBF effectiveness

Intuitively, the performance of RBF for a given deployment pattern will depend on two main factors: the connectivity of chosen filters and the coverage of (s,d) paths by filters. Connectivity degree of a filter, i.e., the number of neighbors it has in the AS map, influences its ability to detect spoofed packets. A filter with N_{int} interfaces sees the Internet as N_{int} groups of potential sources, each group associated with a common interface. The filter cannot prevent spoofing within each group but it can prevent spoofing between groups (and also spoofing within the non-allocated IP space). We expect that better connected ASes can better discriminate among spoofed addresses and reduce p dimension of spoofing. Coverage of paths ensures that a filter can examine (and filter) traffic on these paths, which reduces s and d dimensions of spoofing. We express the coverage of a given filter (or set of filters) via the *popularity* measure, defined as the proportion of (s,d) pairs whose route contains this filter (or at least one filter out of the set).

Figure 2 illustrates how connectivity and popularity affect target and stolen address measure. A, B and F are ASes, and F is an RBF filter with N_{int} interfaces. Each interface i connects F to a group of sources IP_{i} that reach F over this interface. Node A is a first-hop neighbor connected to F via interface 2, while B is a remote node that reaches F over interface 1. Let d_{iF} be a portion of all destinations that a source from group IP_{i} reaches via F, and let s_{iFx} be a portion of all sources from group IP_{i} that reach some AS x via F. In reality, d_{iF} will be different for each source in the source group but to keep the following formulas simple we will assume a common value for all sources. If F is the only filter in the Internet, how does its filtering affect F's, A's and B's target and stolen address measures?

If F is a target of spoofed traffic it will be able to differentiate between packets that arrive from one source group but spoof the address of another group or of F itself. Attackers can only spoof addresses from within their source group. Additionally, attackers within F can spoof any address to another node in F. Let IP_x be a set of IP addresses within an AS x. For simplicity, in the following formulas we ignore the fact that a source does not spoof its own address. A tar-

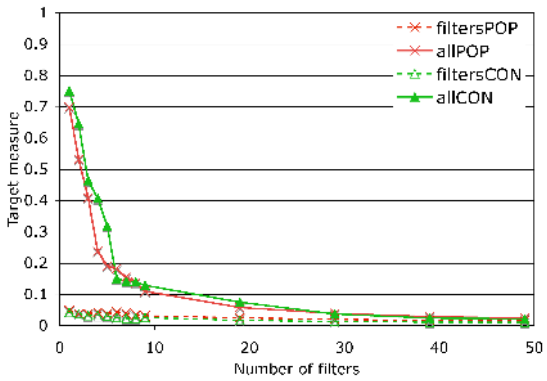

Figure 3: Target measure

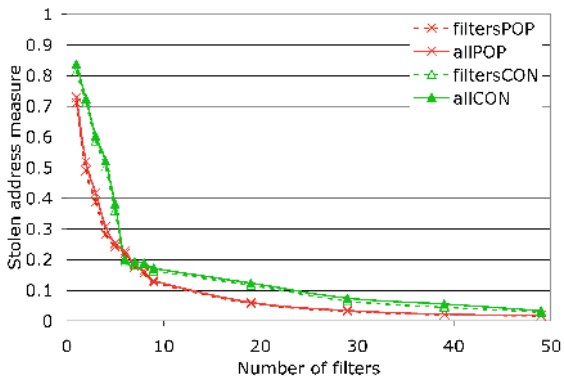

Figure 4: Stolen address measure

get measure for a host in F will be

$$\frac{|IP_{\mathrm{F}}| \cdot |IPv4| + \sum_{i=1}^{N_{\mathrm{int}}} |IP_{\mathrm{i}}|^2}{|IP_{\mathrm{allc}}| \cdot |IPv4|}. \quad (1)$$

Assuming that IP_{F} is a small portion of IP_{allc}, target measure is minimized when the sum in the second term is minimized. For a fixed N_{int}, this happens when the distribution of sources over interfaces is balanced, i.e. $|IP_{\mathrm{i}}| \approx |IP_{\mathrm{allc}}|/N_{\mathrm{int}}$. Source distribution depends on Internet routing dynamics and is bound to change frequently. Still, highly connected filters have an excellent chance of achieving a low target measure, since they have both a high N_{int} and can balance the source distribution by advertising their prefixes to many neighbors.

If A is the target of spoofed traffic, only the portion of that traffic going over F is subject to filtering. The fraction of (s,p) combinations that cannot be filtered is:

$$\frac{|IP_{\mathrm{F}}|^2 + \sum_{i=1}^{N_{\mathrm{int}}} |IP_{\mathrm{i}}| \cdot (s_{\mathrm{iFA}} \cdot |IP_{\mathrm{i}}| + (1 - s_{\mathrm{iFA}}) \cdot |IPv4|)}{|IP_{\mathrm{allc}}| \cdot |IPv4|}. \quad (2)$$

We see that A's target measure can be comparable to F's if A is single-homed or it is multi-homed but only advertises its prefix to F, since s_{iFA} would then be 1 for each source group except the one containing A. The expression for B's target measure is identical to (2), when A is replaced with B. Still, remote nodes are likely to be less protected than nodes closer to a filter since a smaller portion of their incoming paths will cross the given filter (i.e. $s_{iFB} \ll s_{\mathrm{iFA}}$). With regard to target measure, the deploying AS gets the most benefit from RBF, which is a good deployment incentive. Other nodes receive protection proportional to a fraction of their incoming routes which cross F. If RBF were deployed as an infrastructure service intended to reduce spoofing in the Internet, the optimal deployment strategy would favor filters with high connectivity and popularity.

If F's addresses are spoofed in the reflector attack, F will only be able to filter out those packets that it can see, so the stolen address measure for each address from F is:

$$\frac{|IP_{\mathrm{F}}| \cdot |IP_{\mathrm{allc}}| + \sum_{i=1}^{N_{\mathrm{int}}} |IP_{\mathrm{i}}| \cdot |IP_{\mathrm{allc}}| \cdot (1 - d_{\mathrm{iF}})}{|IP_{\mathrm{allc}}|^2}. \quad (3)$$

The A's stolen address measure is:

$$\frac{|IP_{\mathrm{F}}|^2 + (\sum_{i=1}^{N_{\mathrm{int}}} |IP_{\mathrm{i}}| \cdot (1 - d_{\mathrm{iF}}) + |IP_2| \cdot d_{2\mathrm{F}}) \cdot |IP_{\mathrm{allc}}|}{|IP_{\mathrm{allc}}|^2}. \quad (4)$$

where IP_2 relates to the set of addresses in A's source group in Figure 2. The stolen address measure for B is similar to that of A when 2 is replaced with 1. The dominant second term in the stolen address measure depends strongly on the coverage of paths *in the whole Internet*, so popular nodes are the best filter choices to reduce this term. We conclude that while any reasonably well-connected node can lower its target measure by deploying RBF, the only approach to significantly lower the stolen address measure (and prevent reflector attacks) requires deployment of RBF on popular ASes that comprise the Internet core. Fortunately, this deployment also lowers target measure for all Internet participants and thus handles all aspects of spoofing.

Figures 3 and 4 show how the target and stolen address measure change with the number of filters. We show the averages of target and stolen address measures for filters (dotted lines) and for all Internet participants (solid lines). We performed this measurement using RouteViews logs from May 31, 2005. Filters are chosen (1) by connectivity or (2) by popularity. Both target and stolen-address measures decrease dramatically with first 6 filters. 50 filters bring both measures to less than 3% for all participants. While adding more filters further improves the protection, we clearly reach a point of diminishing returns. Filtering performance is comparable for both approaches of choosing filters, but popularity filters have a slightly higher impact on effectiveness than connectivity filters. Lists containing top-50 ASes selected by connectivity and popularity show around 50% overlap, which indicates that well-connected ASes are also popular. There is a noticeable difference between target measures experiences by filters and by all participants. Filters reap high benefits by instantly lowering their target measure, while non-filters only receive benefit when there is enough path coverage. Stolen address measure, on the other hand, is comparable for filters and non-filters as it depends only on path coverage.

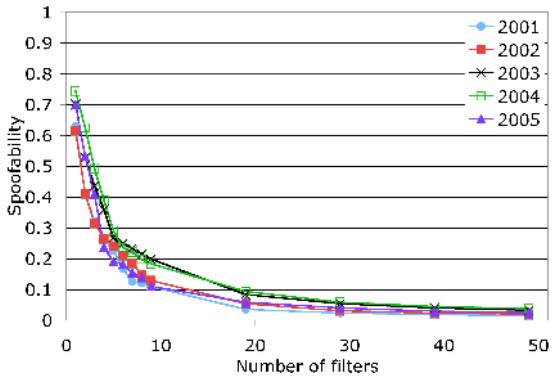

Figure 5: Spoofability in period 2001-2005

2.3 Filter membership

We now examine the ASes that are members of top-50 category and the organizations that own these ASes. We link organizations to ASes by using ARIN Whois service [4]. Our goal is to identify organizations that are both long-term members of top-50 list and that contribute significantly to spoofing reduction. These organizations would be prime candidates for RBF deployment.

To select long-term members for RBF deployment, we examine the dynamics of the top-50 list membership over 5 years, in the period of 2001—2005. For each year we select a day and an hour at random, and build the AS map from RouteViews logs. We then apply popularity selection to build the top-50 AS list. We introduce a new measure called *spoofability*, which represents a fraction of (s, d, p) combinations that are possible in spite of filtering, allowing us to express filtering effectiveness as a single number. Figure 5 shows spoofability measure for 2001—2005 as 1—50 filters are deployed. We see that spoofability curves have a similar shape in all 5 years — the first filter decreases spoofability to values around 70%, the impact of filters on spoofing reduction diminishes with their position on the top-50 list and 50 filters bring the spoofability down to below 3%. In the top-50 lists from 2001 to 2005, there are total of 82 unique ASes that belong to 41 organizations. 25 ASes and 18 organizations appear in all five lists, and they are consistently highly ranked in all lists. We place these ASes and organizations in the TopMembers list and show them in Figure 6.

We now examine a hypothesis that the TopMembers list contains the nodes crucial for RBF deployment. We measure spoofability for May 31, 2005 when (1) we have filter deployment only at ASes from the list, and (2) we choose filters from the vertex cover but we leave out ASes from the TopMembers list. When only the ASes from the TopMembers list deploy RBF, the spoofability is reduced to 8%. 19.5% ASes cannot send spoofed packets to any destination (except subnet-spoofed packets that RBF does not handle), and 34.7% source-destination paths cannot contain spoofed traffic. Without TopMembers deployment, more than 700 filters are needed to reach comparable filtering performance. Both these arguments speak in favor of deploying RBF on

Top Organizations
AT&T Global Network Services
AT&T WorldNet Services
Abovenet Communications, Inc
Asia Pacific Network Information Centre
Bell Canada
Cogent Communications
Global Crossing
Level 3 Communications, LLC
Merit Network Inc.
NTT America, Inc.
Qwest
RIPE Network Coordination Centre
SBC Internet Services
Savvis
Sprint
Time Warner Telecom
UUNET Technologies, Inc.
XO Communications

Top ASes	
174	3356
209	3549
237	3561
701	3786
702	4134
1221	4323
1239	4766
1267	6461
2647	7018
2686	
2828	
2907	
2914	
3215	
3269	
3320	

Figure 6: TopMembers list

ASes that belong to those large Internet participants, as an infrastructure service that brings benefit to everyone in the Internet.

3 Clouseau design

We have demonstrated that RBF can dramatically reduce spoofing even in sparse deployment. To make RBF practical we need an efficient and accurate approach for an RBF filter to autonomously build and maintain incoming tables. We separate the functionality of a filtering router into two parts: (1) populating incoming table entries and updating them when routing changes occur and (2) filtering spoofed packets using incoming table information and ingress filtering. The Clouseau system will handle the first functionality and RBF will handle the second. Packets are first checked by RBF and classified as "matching" if they arrive on expected interface, or "offending" if they arrive on unexpected interface or if their source address cannot be found in the incoming table.

Assume that an incoming table is initially complete and correct — we will shortly show how this can be achieved by Clouseau. Under perfectly stable routing and in the absence of spoofing, all packets will be classified as matching and will be forwarded to their destination. When a routing change occurs, some packets may start arriving on an unexpected interface because they now follow a different route from the one that existed when the incoming table was built. These packets will be classified as offending. Instead of dropping these packets, we would like to learn that the route has changed and update the corresponding incoming table entry to reflect this change. On the other hand, when spoofed packets arrive at an RBF router, they may also arrive on an unexpected interface and be classified as offending. We want RBF to quickly be told that these packets should be dropped.

Clouseau helps RBF learn whether the cause for offending packets was a route change or a spoofing attack by running an *inference process* on the corresponding incoming ta-

ble entry (if there is no entry, Clouseau will create one). If Clouseau concludes that the cause is a routing change, it updates the incoming interface information — this is called a *validating* decision. If Clouseau concludes that the cause is a spoofing attack, it marks the entry as fresh and places an inference ban on this entry for T_{ban} seconds — this is called a *filtering* decision. During the inference ban, RBF will drop offending packets on this entry and Clouseau will not be activated. When the ban expires, the fresh mark is removed and offending packets can again start the inference process. Since some offending traffic is forwarded to the destination as a part of the inference process, the inference ban is necessary to limit this amount and ensure that an RBF router works mostly in the filtering mode. Clouseau handles packet that spoof non-allocated IP addresses, by running the inference process to fill a non-existing table entry. The outcome of the process inserts the non-allocated address with "no interface" into incoming table.

During the inference process, Clouseau exploits the reliable delivery feature of TCP to test whether offending TCP data packets are generated by a TCP sender or by an attacker spoofing a given source IP. This test is performed by dropping some offending packets and forwarding the rest. All the inference structures and variables we mention in the following text are created and maintained per an incoming table entry that is undergoing the inference process. Clouseau records all the dropped packets in the *DroppedQueue* and it records some of the forwarded packets in the *ForwardedQueue*. We only record some unique identifier of a packet, e.g., a sequence number. If the sources of the offending traffic are TCP senders, they will learn about data packet drops either through duplicate acknowledgments from the receiver or when the retransmission timers expire. These senders will retransmit all the packets from the DroppedQueue. Some small number of packets in the ForwardedQueue may also be retransmitted if some senders or receivers do not use selective acknowledgments or if there is a congestion between the filter and the packet's destinations. If the offending traffic contains some spoofed packets, the attacker has no way of learning which packets were dropped and which have arrived at the target. A simple attacker that spoofs random addresses will fail to repeat all the packets from the DroppedQueue, which will be a signal to Clouseau to detect the spoofing. An attacker who is aware of Clouseau and wants to repeat some packets to invoke a validating decision will likely repeat too many from the ForwardedQueue, if we set our system correctly. We deal with this type of attacker in section 6.1.

To differentiate between the legitimate source and the attacker, Clouseau assigns one *valid* point to the incoming table entry each time a packet from the DroppedQueue is repeated, and one *spoofed* point when a packet from the ForwardedQueue is repeated. Repeated packets are marked as "verified" and are always forwarded to their destination.[2] Repeating a verified packet from the DroppedQueue does not increase the valid score, while repeating a verified packet from the ForwardedQueue increases the spoofed score. Inference process ends with a validating decision when Clouseau gathers V valid points. It ends with a filtering decision when Clouseau gathers S spoofed points or upon a timeout which is T_{inf} seconds. We note here that Clouseau was specifically designed with asymmetric traffic in mind and does not require both directions of a given source's communication to be visible.

We now provide more detail about Clouseau operation and explain some design decisions.

- How does Clouseau determine which packets to place into which queue? Let L_{DQ} be the length of the DroppedQueue, L_{FQ} be the length of the forwarded queue and $N = L_{\text{DQ}} + L_{\text{FQ}}$. Clouseau generates a random permutation of numbers 1, ..., N and marks the first L_{DQ} slots as dropped and the rest as forwarded. The first N unique data packets are placed into queues according to the permutation, and the other packets are forwarded without being recorded.
- How long are the queues? $L_{\text{DQ}} = V + M$, where V is the threshold for validation and M is some small margin of error, to handle the case when some TCP connections that have experienced drops die, maybe because they were aborted by the application. In our tests and in later discussion we set $M = 0$ but we note that in practice it may be wise to set it at a small positive value. $L_{\text{FQ}} = \alpha \cdot S$, where S is the threshold for filtering decision and $\alpha \geq 1$. Intuitively, the larger the value of α the easier it is to catch an attacker who repeats packets. Tests and analysis in later sections give further guidance on how to set queue sizes and threshold values.
- Why do we need the ForwardedQueue and should S be greater than 0? The ForwardedQueue is necessary to catch the attacker who is familiar with Clouseau and repeats some spoofed packets. Without it, even the simple attack strategy of sending each spoofed packet twice would lead to a validating decision. S should be greater than 0 for two reasons: (1) if a TCP sender does not use selective acknowledgments, some packets from the ForwardedQueue will be repeated; Clouseau must recognize this behavior as legitimate, and (2) some packets forwarded by Clouseau may be dropped further downstream; Clouseau cannot tolerate arbitrary downstream loss (this would result in inability to catch attackers) but should tolerate small, sporadic losses.
- Can Clouseau process a sample of packets and forward the rest to lower its operating cost? While the initial N packets could be sampled, once the first packet is placed into any queue, Clouseau must examine all following packets to detect repetitions. One argument in favor of sampling packets that are placed into queues could

[2]While it may seem justified to drop repeated packets that are found in the ForwardedQueue, we have no guarantee that the original packets (that were forwarded) have reached their destination. If they were dropped downstream of the filtering router, maybe due to congestion, dropping them again would severely delay TCP traffic.

be to ensure low packet loss and thus minimize TCP connection delay due to dropping. In our performance evaluation we do not implement sampling.

- Why do we end inference process on a timeout? It is important to resolve the inference process quickly so that an RBF router could spend most of the time filtering spoofed traffic. The timeout ensures that the length of the inference process is bounded. Timer value T_{inf} must be large enough to guarantee the validating decision for the slowest legitimate traffic. We discuss how to set this value in Section 4.1.3.
- What happens with non-TCP packets and TCP control packets? If they are classified as offending, these packets trigger the inference process and are filtered by RBF during the inference ban. We cannot use them for the inference decision because TCP's reliable delivery mechanism will not repeat all control packets, and we have no expectation of reliable delivery for non-TCP packets. During the inference these packets are forwarded without being recorded. If there are few or no TCP data packets to sample during the inference process, Clouseau will end this process on timeout with a filtering decision. Since the majority of traffic in the Internet is TCP, this situation is unlikely to occur, but in future work we will examine alternative methods of dealing with non-TCP and TCP control traffic.

4 Performance evaluation

This section presents both the theoretical evaluation of Clouseau performance and experimental results. The theoretical evaluation addresses how to set Clouseau parameters, and the experiments illustrate how performance depends on parameter setting. We have implemented Clouseau as a loadable kernel module in RedHat 7.3 Linux. The module uses netfilter hooks to capture packets as they travel through the IP stack. We tested Clouseau in real deployment in the Emulab testbed [2]. Our tests are small scale and are not meant to demonstrate scalability, but rather to illustrate how performance is affected by parameter settings. Scalability is addressed in Section 5. In our tests we set values of α to 2 and T_{inf} to 2 seconds, and vary queue sizes and threshold values.

4.1 Connection delay

Introducing Clouseau into the packet processing path has two effects on legitimate TCP traffic: (1) increased round-trip time (RTT) and (2) connection delay because of congestion response to packet drops. If queues are implemented as hash tables, Clouseau operation is a simple table lookup and counter update, and should not add significant delay to round-trip time. Connection delay due to packet drops is inevitable, as Clouseau has to drop some packets during the inference process, but should be minimized.

We measure RTT increase and congestion delay due to drops using a topology shown in Figure 7. RBF and Clouseau are deployed at router RC. In tests we generate

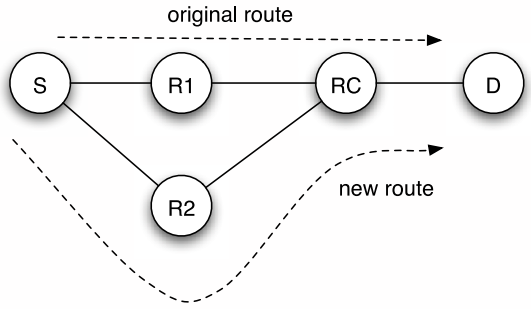

Figure 7: Topology used in the experiments

10 parallel TCP connections from S to D. Connections are telnet-like and send around 100 packets per second each. The route between S and D is originally S-R1-RC-D and changes to S-R2-RC-D after 20 seconds. This invokes the inference process. The route is changed by executing an ip route change command, so there is no route stabilization delay, such as might be seen in real operation when a link fails. The tests are performed with full incoming tables so the only source of connection delay is the inference process invoked by the routing change.

4.1.1 RTT increase

The additional processing delay introduced by an RBF/Clouseau router consists of (1) RBF delay for incoming table lookup, experienced by all packets and (2) Clouseau delay for searching the DroppedQueue and ForwardedQueue and for counter update on match, experienced by TCP data offending packets when route change occurs. We note here that both delays will depend heavily on the implementation, so the measurements in this section should be viewed more as ballpark estimates of expected processing delay than as accurate prognosis of RBF/Clouseau performance in real routers. In our implementation, RBF delay includes kernel module delay and can simply be measured by running a ping utility with and without a kernel module. Since Clouseau runs the inference process only on TCP data packets, ICMP packets generated by ping will not experience any Clouseau delay. These tests showed 4.39μs round-trip increase, which translates to 2.195μs RBF delay.

We measure Clouseau delay by first ensuring that no packets are dropped by Clouseau (through the use of a special flag) and then artificially delaying the inference decision indefinitely, which forces each TCP data packet to experience queue search delay. In the experiment, large data traffic is used to fill the queues completely. After that, a series of short packets are sent from S to D and the RTT is measured for these packets with and without Clouseau. We subtract RBF delay from the difference. We vary the sum of queue sizes, N, from 1 to 2000, which is the maximum our kernel memory can support. Our tests show the average Clouseau delay of around 20 μs per packet when $N \geq 1000$. Lower values of N did not produce any measurable delay.

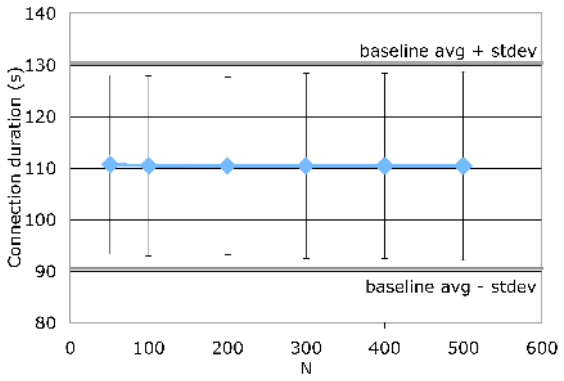

Figure 8: Connection delay vs queue size

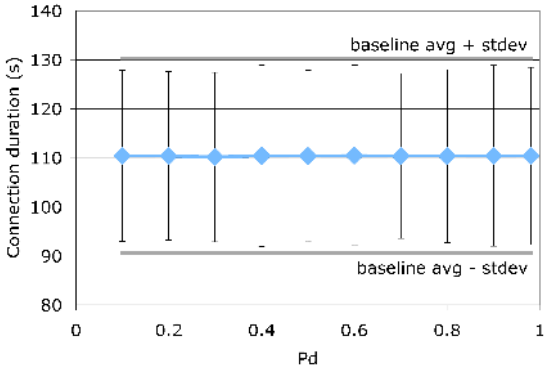

Figure 9: Connection delay vs p_d with short queues (N=100)

4.1.2 Connection delay due to drops

We measure connection delay due to drops by recording connection duration for each of 10 TCP connections, with and without Clouseau. In the first set of tests we measure how the connection delay changes as queue sizes change, but we keep the $p_d = \frac{L_{DQ}}{N}$ at 0.1. Each test is run 20 times. We show the results in Figure 8 — the y-axis shows the mean connection duration and error bars indicate one standard deviation difference from the mean. We also draw the baseline mean $\pm$ stdev of connection duration. The results show no noticeable delay due to 0.1 drop rate and are not sensitive to queue size increase.

In the next tests we measure how the connection delay changes as we increase the size of the DroppedQueue, but keep $N = 100$, thus effectively increasing the percentage of packets dropped. We show the mean $\pm$ stdev of connection duration in Figure 9. The results indicate that connection delay is not sensitive to packet drop rate and that connection duration stays within bounds of the baseline case. Note that this trend should hold only *as long as there is a sufficient number of connections in the offending traffic mix to share the drops, and if the queues are not too long to lead to sustained connection losses over time*. Figure 10 shows similar tests with N=1000.[3] We see that the connection duration starts increasing linearly as p_d goes over 0.4 and we expect that for longer queues this knee in the trend would move to the left, and with higher number of connections in the mix it would move to the right. Still, for reasons we will explain in the next section, it is good to keep p_d below 0.25.

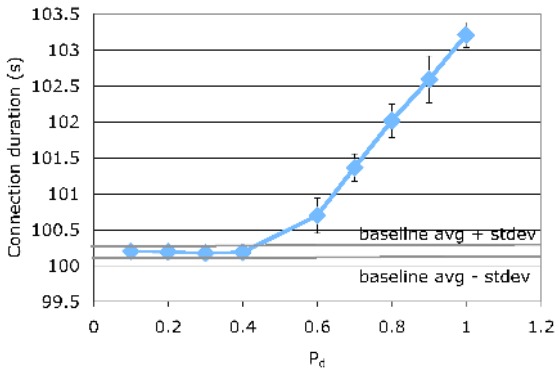

Figure 10: Connection delay vs p_d with longer queues (N=1000)

4.1.3 Inference delay

We call the time needed to reach a validating or filtering decision the *inference delay*. We first look at factors that influence this delay in case of route change or spoofing, and we then show experimental evaluation. T_{filter}, the time to reach a filtering decision is bounded from above by T_{inf}. In the case of packet-repeating attacker this time would be lower than T_{inf} and would depend on the attack dynamics and the time needed to repeat S packets from the ForwardedQueue. T_{valid}, the time to reach a validating decision is the time required to collect V valid points. Assume that the k-th packet from the DroppedQueue is the last one to be verified (this will be a packet from the slowest connection). T_{valid} is then the sum of time needed to fill the k-th slot in the DroppedQueue — $T_{\text{DQfill}}(k)$ and and the time to verify the k-th packet — $T_{\text{verify}}(k)$. If we assume the worst case $k = N$, $T_{\text{DQfill}}(k) \approx \frac{N}{R}$, where R is the rate of offending packets. $T_{\text{verify}}(k)$ is the sum of the time needed for the sender to receive the notification of the loss, $T_{\text{notify}}(k)$, measured from the moment the original packet is dropped, and the time it takes the retransmitted packet to arrive to the RBF router $T_{\text{sender(k)/Clouseau}}$. Combining these two, $T_{\text{verify}}(k)$ is the time needed for the sender to receive the loss notification, measured from the moment the original packet is *sent by the sender*. A TCP sender receives loss notification either through Retransmission Timer timeout or after the arrival of

[3]These tests were run on the same topology and with same legitimate traffic but with different hardware than experiments shown in Figures 8-9. This is because we run experiments on a shared testbed and cannot guarantee which hardware will be assigned to the experiment. Different hardware unfortunately produces different baseline values.

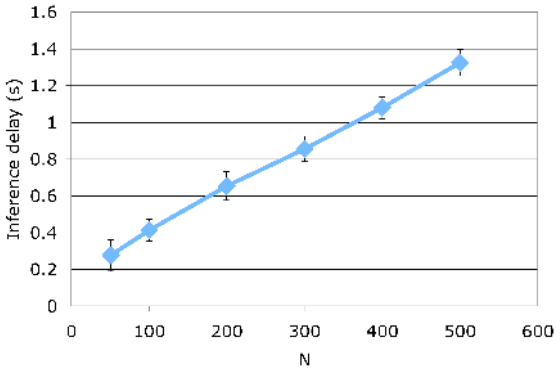

Figure 11: Inference delay vs queue size

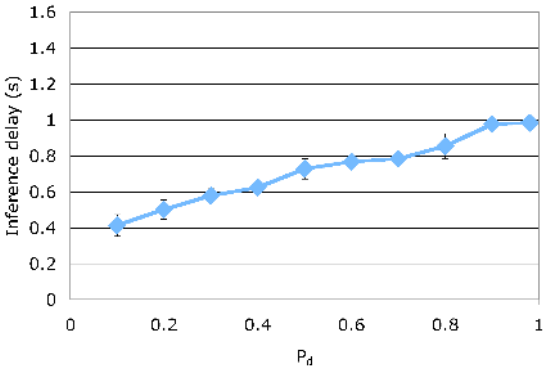

Figure 12: Inference delay vs p_d

three duplicate acknowledgments, so:

$$T_{\text{verify}}(k) \approx \begin{cases} RTO \approx 2 \cdot RTT & \text{on RTO timeout.} \\ \frac{3}{R_C(k)} + RTT & \text{on 3-dup ACKs.} \end{cases}, \quad (5)$$

where RTO is the Retransmission Timer value, RTT is the estimate of the roundtrip time on the TCP connection that generated the k-th packet, and $R_C(k)$ is the packet rate of k-th connection. Obviously, the case of three duplicate ACKs is preferable to the timeout case since it ensures faster decision. To avoid timeouts, Clouseau must ensure that no connection has more than 1 out of 4 packets dropped. We can control the level of drops by controlling the queue sizes: $L_{\text{DQ}}/N < \frac{C}{4}$, where C is the total number of connections at a given incoming table entry, undergoing the inference process. As Clouseau should be installed at major routers in the Internet backbone, we expect that values of C will be so large that this condition holds for any queue sizes. Otherwise, it would be wise to assume C=1 and ensure that $\frac{L_{\text{DQ}}}{N} < 0.25$. The expected value of T_{valid} is then $T_{\text{valid}} = \frac{N}{R} + \frac{3}{R_C(k)} + RTT$.

In [3], Shakkottai et al. measure the distribution of RTT in four sets of backbone traces and conclude that this distribution is heavy-tailed, with values ranging from 10ms to well beyond 10s, and with peaks around 100ms. In a mix with a large number of simultaneous connections and a high cumulative packet rate, such as is expected to be seen at a large backbone router, the value of T_{valid} should be dominated by the RTT value of the slowest connection. In our experiments RTT $\ll \frac{N}{R}$ so we expect T_{valid} to be dominated by the speed of filling the queues. The inference timeout T_{inf} must be larger than the maximum validation time $Max[T_{\text{valid}}]$. The worst case for route change validation occurs when the N-th packet is the last one to be verified and the sender is notified of the loss upon Retransmission Timer timeout: $Max[T_{\text{valid}}] \approx \frac{N}{R} + Max[2 \cdot RTT]$ Since RTT distribution is heavy-tailed, the worst-case RTT can be arbitrarily large, so T_{inf} would have to be set to a very large value. Instead, we will make an optimistic estimate of RTT value based on RTT distribution graphs in [3]. From these graphs, an overwhelming majority of flows have RTT less than 500ms. Taking this value as the worst-case RTT estimate, and assuming that the packet rate R is large, which agrees with our target deployment at large backbone routers: $Max[T_{\text{valid}}] \approx 2 \cdot Max[RTT] = 1s$. Thus T_{inf} should be at least 1s. Since in our tests $R = 1000$ and N varies from 50 to 1000, we set T_{inf} to a larger value of 2 s.

We now show how the inference delay measured in tests in Section 4.1.2 depends on queue size (Figure 11) and how it depends on p_d (Figure 12). Inference delay increases linearly both with queue size and with increase in percentage of packets dropped. This is expected, as both factors lead to linear increase in packets that need to be validated. Since all our connections had the same RTT, we did not see much variation in inference delay, but a real-world deployment would result in larger variance of this measure.

4.2 Congestion

Clouseau's inference process can handle low levels of downstream offending packet loss, about $\frac{1}{\alpha}$. If loss is greater than that, Clouseau will collect more than S spoof points during the inference and will wrongly reach a filtering decision. High packet loss may occur when routes change due to congestion or when there is a large delay in route convergence. The filtering decision will be followed by an inference ban of T_{ban} seconds and legitimate traffic will be dropped during this time. When the ban expires, Clouseau will start the inference process anew, which will again result in a filtering decision if the congestion has not cleared. When the congestion clears, Clouseau will successfully reach a validating decision. For a reasonably small α (e.g. 2 — 4), packet loss higher than $\frac{1}{\alpha}$ will itself create severe delays in legitimate traffic, so Clouseau will not make the situation much worse. The adverse effect occurs when the congestion clears and Clouseau is still filtering packets due to inference ban. In the worst case scenario, Clouseau will place an inference ban just before the congestion ends. The maximum amount of time that the congestion effect will be prolonged for the legitimate traffic is T_{ban}. On the other hand, in case of a persistent random spoofing attack, $\frac{T_{\text{ban}}}{T_{\text{inf}}+T_{\text{ban}}}$ is the percentage of time spent in successfully filtering the attack (the rest is spent

in the inference process). The tradeoff in setting T_{ban} is then in choosing a sufficiently low value to minimize unwanted drops when congestion is high, and choosing a high enough value to maximize the time spent doing filtering.

4.3 Cascaded filters

If there is a large filter deployment in the Internet, it is possible that some routing changes will trigger inference process in more than one filter simultaneously. In this case, some packets forwarded by upstream filters will be dropped by downstream ones. This may lead to erroneous filtering decisions in upstream filters and unwanted legitimate traffic drops. The core reason for this problem is filter synchronization. We implement a token-passing approach to break this synchronization in Clouseau by having the filtering router place a well-known value[4] in the TOS field of offending packets that it forwards. A downstream filter that sees marked packets delays its inference process for T_{delay} seconds. This way, several routers may initially capture the token but only the router closest to the source of the offending packets will be the one to keep the token. To prevent an attacker from delaying the inference process indefinitely by stamping his packets with the well-known value, we impose the limit on the total delay a filter is willing to experience before it starts the inference process regardless of marked packets. When a filter completes its inference process, it stops marking TOS field in forwarded packets (releases the token) and the downstream routers will attempt to capture it. We have implemented and tested this approach with up to 10 synchronized filters. It successfully prevents concurrent inferencing without introducing any noticeable slow-down into the inference process or legitimate TCP connections.

4.4 Packet processing

We measured the processing cost of RBF and Clouseau by measuring CPU load at deploying machines. Each machine is a Pentium III with 850 MHZ CPU and 256 MB of RAM. When there are no offending packets RBF adds about 5% to the CPU load. With offending packets Clouseau adds about 50% to the CPU load within the first second following the route change, when the inference process is run. The load goes down when the inference process ends. We also perform tests with attack traffic spoofing the legitimate address and we vary the attack rate and Clouseau queue sizes to measure how CPU load changes when these factors change. The additional CPU load remained at 50—55% in all the tests and did not show any trend when attack rate or queue sizes were changed.

5 Scalability

The biggest issue for RBF and Clouseau scalability is storage. In case of RBF, storage is needed for the incoming table. Each entry stores the corresponding source address (32 bits), the expected incoming interface (12 bits based on the current connectivity information) and a fresh mark (1 bit). A straightforward approach that stores one source IP per entry would require 2^{32} entries, which is infeasible. Clouseau could aggregate addresses that share $/24$ address range into one entry, as this is usually the smallest address range that can be assigned to a subnet and all machines from the subnet are expected to share routes to any given destination. Still, 2^{24} entries may represent too large a demand for router memory, so more aggregation is needed. Today's routers apply sophisticated aggregation techniques to minimize space needed by the routing and forwarding table entries, while still allowing for fast prefix lookup. Many of those aggregation approaches can be reused by RBF to aggregate the incoming table aggressively. The expected size of the incoming table should be comparable to the size of today's forwarding tables. Another way to minimize storage requirements for the incoming table is to integrate incoming and forwarding information in the forwarding table for those entries that have the same outgoing and incoming interfaces. An auxiliary incoming table would then keep only the information about addresses that have asymmetric routes to the filtering node.

While entry aggregation lowers storage cost for RBF, it complicates Clouseau. When a route change occurs, it may affect only a part of the addresses in the aggregated entry. Clouseau then needs to infer which part is affected and how to split the entry. A simple approach is to assume that all addresses for a $/24$ address range must arrive on the same interface. When offending packets start arriving, Clouseau builds a new entry for each $/24$ address range and removes the fresh mark from the original incoming entry. New entries can be aggregated if their prefixes allow this and if they share the same new candidate interface. When the inference ends, the address space for the old entry is adjusted to account for the split and the new entries are marked as fresh. We plan to carefully investigate splitting and reaggregation of incoming entries in the future work.

Clouseau also needs storage for DroppedQueue, ForwardedQueue and various counters. The smallest size of a queue entry is 32 bits, if we take TCP sequence numbers as unique packet identifiers and we assume that the probability of two sources sharing the same sequence number range is low. Still, if the total size of queues is $N = 1000$, this would map into 4KB of memory for each entry that undergoes the inference process. In the worst case, all entries in the incoming table could simultaneously go into inference, which would blow up memory and CPU requirements. This could be amended by randomly selecting a limited number of entries to undergo the inference process, and forwarding offending packets for other entries until their turn comes to start inference. Another way to address this, which we plan to investigate in future work, is to use Bloom filters to implement queues, to minimize storage requirements.

6 Attacks on Clouseau

In this section, we assume that the attacker is familiar with Clouseau and the values of all the Clouseau parameters. The attacker's goal may be (1) to trick Clouseau to falsely recog-

[4] An alternative would be to choose a secret value but this would have to be communicated to other filters which requires shared trust.

nize a spoofing attack as a route change or (2) to perform a successful spoofing attack on a given victim in spite of Clouseau. The attacker who aims to trigger a false validating decision will attempt to guess and resend some spoofed packets. If he succeeds, Clouseau will change the incoming interface for the spoofed source. This may both lower filtering effectiveness and lead to some legitimate packet drops if the legitimate traffic starts arriving on the old, now unexpected, interface. We analyze this case in three attack scenarios below: *packet-repeating*, *permutation* and *collusion* attack.

The attacker who aims to avoid Clouseau filtering will attempt to outrun Clouseau by frequently changing the addresses spoofed in packets. The attacker may choose to reuse an address after some cycling period. If the change period is lower than T_{inf} and the cycling period is higher than T_{ban}, the offending packets will always hit incoming table entries that are not marked fresh. This will guarantee that more than $\frac{\alpha \cdot S}{N}$ fraction of packets is forwarded to the destination, possibly creating a successful attack. We analyze this case in the *cycling attack* scenario, below. Note that random-spoofing is a subset of a cycling attack.

6.1 Permutation and packet-repeating attacks

A simple attempt to bypass the Clouseau inference process and invoke incoming interface change with spoofed packets is to repeat each spoofed packet.

Lemma 1 Repeating packets more than once is suboptimal for the attacker.

Proof: Observe the sequence of K identical packets — one original and $K-1$ repetitions. If the original packet is placed in the DroppedQueue, only the first repetition brings one valid point. If the packet is placed in the ForwardedQueue, each repetition brings one spoof point. Finally, if the packet was not placed in any queue, the attacker does not gain anything by repeating this packet. Thus, the attacker's chances to deceive the system can only decrease for $K > 2$. □

We now introduce the definition of a permutation attack and prove that a packet-repeating attack is a special case of the permutation attack. Observe a sequence of N unique spoofed packets that completely fills Clouseau queues, according to the pre-computed random permutation. The attacker selects R out of these packets and repeats them in some chosen order before the inference timeout.

Lemma 2 The outcome of the inference process is determined only by the order of R repeated packets and not the repetition time.

Proof: The algorithm reaches a decision when the first of the valid or spoof points reaches the corresponding threshold. As long as the repeated packets arrive in the same order, the points are gained in the same manner, and the algorithm reaches identical decisions. □

This allows us to focus the analysis only on the order of duplicate packets, ignoring their original occurrences. Further, only the repeated packets whose originals were recorded in the queues influence the decision process. From lemma 2 follows that a packet-repeating attack is a special case of a permutation attack. We assume that the attacker knows the queue sizes, and hence he may choose to send less then N unique packets. Let the number of unique packets be $n < N$. This does not increase or decrease the attacker's chances to fool Clouseau compared to the case where he sends at least N unique packets, but chooses not to repeat $N - n$ of them.

We now focus on analyzing the strategy where the attacker repeats a subset of R out of N original packets, and introduces a permutation into the repeated packets. The attacker's goal is to guess a favorable permutation that helps him gain V valid points before collecting S spoofed points. Since the selection of packets inserted in the DroppedQueue and ForwardedQueue was chosen randomly over a set of all permutations, no permutation chosen by the attacker is more likely to succeed than any other.

Lemma 3 Probability that the attacker wins after exactly k repeated packets, $V + S - 1 \geq k \geq V$, is

$$p_k(N, V, S) = \frac{V}{k} \frac{\binom{N-V}{k-V}}{\binom{N}{k}}.$$

Proof: The number of subsequences of length k chosen from N unique elements is $N \cdot (N-1) \ldots (N-k+1) = \binom{N}{k} \cdot k!$. The attacker's winning sequences need to end with one of the V packets, and the number of different such sequences is $V \cdot (k-1)! \binom{N-V}{k-V}$. The lemma follows. □

Note that for $k > V + S - 1$, the attacker cannot win after element k, because by that time he will have gained at least S spoof points and the filtering decision will have been reached.

Theorem 1 The optimal attacker's strategy for defeating Clouseau is to repeat at least $V + S - 1$ unique packets, and the probability of success is

$$p_{\text{success}} = \frac{\binom{N-V}{S-1}}{\binom{N}{V+S-1}}$$

Proof: From lemma 3, the attacker can win at any $V+S-1 \geq k \geq V$. Since each of those probabilities is positive, the optimal attacker strategy calls for repeating at least $V+S-1$ packets, after which Clouseau is guaranteed to reach some decision. The probability of success for an attacker is thus,

$$p_{\text{success}} = \sum_{k=V}^{V+S-1} \frac{V}{k} \frac{\binom{N-V}{k-V}}{\binom{N}{k}} = \frac{\binom{N-V}{S-1}}{\binom{N}{V+S-1}} = \frac{\binom{V+S-1}{V}}{\binom{N}{V}}. \quad (6)$$

The value of the sum represents a total fraction of sequences of size $V + S - 1$ containing exactly V packets in the DroppedQueue, and value $\frac{\binom{N-V}{S-1}}{\binom{N}{V+S-1}}$ is a fraction of permutations in which V is chosen from the first $V + S - 1$ elements relative to the total number of permutations in which V can be chosen from all N elements. □

Clouseau parameters α, S and V determine the system's robustness against the permutation attack. In the following

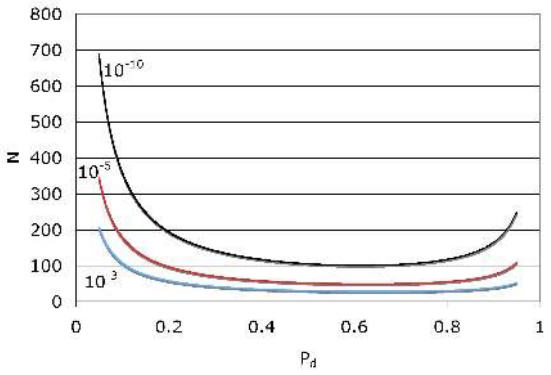

Figure 13: Required queue size N for $p_{\text{success}} < 10^{-3}$, $p_{\text{success}} < 10^{-5}$ and $p_{\text{success}} < 10^{-10}$ for varying p_d

analysis, we assume that $\alpha = 2$. The probability of the attacker's success, p_{success} from equation (6), can be made arbitrarily small by selecting proper parameter values. Assume a constant ratio of dropped packets relative to the total queue sizes: $p_d = \frac{L_{\text{DQ}}}{N} = \frac{V}{N}$. Then, $S = \frac{1}{2}(1-p_d)N$.

Lemma 4 The probability for the attacker's success p_{success} can be made arbitrarily small by increasing the queue sizes, and in particular,

$$p_{\text{success}} < b{\cdot}c^N = \frac{2\cdot(1-p_d)}{\sqrt{1+p_d}} \left(\frac{(1-p_d)^{1-p_d}\left(\frac{1+p_d}{2}\right)^{\frac{1+p_d}{2}}}{\left(\frac{1-p_d}{2}\right)^{\frac{1-p_d}{2}}} \right)^N.$$

Proof:

$$\begin{aligned} p_{\text{success}} &= \frac{\binom{V+S-1}{V}}{\binom{N}{V}} = \frac{(N-V)!(V+S-1)!}{N!(S-1)!} \\ &= \frac{(N-V)}{N}\frac{\Gamma(N-V)\Gamma(V+S)}{\Gamma(N)\Gamma(S)}. \end{aligned}$$

Above we used that $n! = \Gamma(n+1)$. To upper bound p_{success}, we use Feller's bounds [13] on Stirling's approximation,

$$\Gamma(x) = \sqrt{2\pi}x^{x-\frac{1}{2}}e^{-x}e^{s_x}, \text{ where } \quad 0 \le s_x \le \frac{1}{12x}. \qquad (7)$$

The claim follows. □

We note that $c < 1$, so p_{success} decreases exponentially with increase in N. In Figure 13, we show how the minimum queue size N necessary for $p_{\text{success}} < 10^{-3}$, $p_{\text{success}} < 10^{-5}$ and $p_{\text{success}} < 10^{-10}$, depends on p_d. The optimal p_d which minimizes memory cost for target p_{success} is 0.6. With sampling we can easily ensure that the actual packet drop rate is low while keeping $\frac{L_{\text{DQ}}}{N} = 0.6$.

Since the probability of the attacker winning is always greater than zero, it is still possible that in some rare cases (or after sufficient number of attempts) Clouseau can be tricked. One way to detect this would be to change the relationship of RBF and Clouseau to periodically trigger inference on entries that do not receive any offending packets, to revalidate them. This inference could be triggered with a higher V to make the test strong, and a very small p_d so the damage to legitimate traffic due to drops would be minimized.

6.2 Collusion attack

Clouseau differentiates between a spoofing attack and a routing change by relying on the attacker's inability to learn which packets were forwarded and which were dropped. An attacker who has a helper at the other side of a Clouseau router could trick Clouseau by sending a sequence of spoofed packets to his helper and asking him to report which packets have arrived. The attacker would then resend the missing packets, gain V valid points and lead Clouseau to reach a validating decision. In this attack, the attacker and helper actually reimplement the TCP mechanism. They behave, from Clouseau's observation point, as a legitimate TCP sender and receiver, which makes this attack particularly hard to detect.

One approach to detect the collusion attack is to create an impression of reality at a sender's side that is different from the one at the receiver's side, so that the attacker cannot benefit from his helper's communication. Clouseau creates separate realities by introducing a fake event observable by the source (whose address is in the offending packets), but not by the receiver. Some number of offending packets are subject to a fake event and some information about these packets is remembered in a *FakeEvent* table. Note that this table has similar functionality as the DroppedQueue and ForwardedQueue and that an offending packet can now be either forwarded, dropped or subjected to a fake event. This event must meet two requirements (1) a legitimate TCP sender can generate a response that cannot be faked by the source of spoofed packets with the helper's messages, and (2) a legitimate TCP sender can recover from the fake event and resume its communication without further interference from Clouseau. One type of TCP event satisfies these requirements — generation of a fake packet loss. Clouseau forwards an offending TCP data packet but informs a source that the packet has been dropped by generating a fake triple-duplicate acknowledgment for a packet immediately preceding this one. Clouseau can generate fake acknowledgments without a TCP connection record, inferring proper header values from the captured packet, and remember only a sequence number in FakeEvent table. A legitimate TCP sender will immediately resend the "missing" packet after acknowledgments have been received, provided that the fake acknowledgments from Clouseau arrive before the receiver's acknowledgment for this packet; this is likely since Clouseau is on path from sender to the receiver. An attacker will not receive fake acknowledgments and will fail to repeat some packets from FakeEvent table. Clouseau differentiates between the attacker and the legitimate TCP sender by counting a number of packets from FakeEvent table that were not repeated some time after the fake event. It reaches a filtering decision when this number goes over some set threshold. When a packet from a FakeEvent table is repeated its identifier can be deleted from the table. TCP receiver will discard this packet as duplicate and the connection will proceed as normal. This approach is resilient to packet drops. If a forwarded packet is dropped en route to the receiver, it will still be properly repeated after the

sender receives fake acknowledgments. Clouseau can minimize the danger of fake acknowledgment drops by sending more than 4 of them for each packet.

6.3 Cycling attack

An attacker who frequently changes spoofed addresses avoids Clouseau filtering, since offending packets hit incoming table entries that are not marked fresh. This triggers the inference process during which a large portion of offending packets is forwarded to the destination. This attack cannot be handled by adjusting Clouseau parameters. Instead we first detect the attack and identify affected entries, then apply custom-handling to packets that are recognized as a part of the attack. We observe two kinds of a cycling attack: (1) many-to-few, where spoofed packets flow to a few targets, and (2) many-to-many, where spoofed packets flow to diverse targets. Many-to-few spoofing occurs in many flooding DoS attacks. It easy to create, frequent and very damaging to the target, so it is important that Clouseau handles this kind of cycling attack. Many-to-many spoofing creates a lot of spoofed traffic flowing everywhere, but poses no additional threat to Internet security. If the traffic is excessive, many-to-many spoofing could create congestion in the core. It would be impossible to handle this attack by trying to separate offending-spoofed traffic from offending-legitimate traffic, since the attack traffic has no specific signature. Instead, a Clouseau router that experiences a sustained high offending packet rate, and does not detect a many-to-few attack, could mark all its incoming entries fresh and perform aggressive filtering. This would create some collateral damage to traffic whose route changes during the attack, but would successfully reduce congestion in the core.

Clouseau detects a many-to-few attack by monitoring offending traffic destinations. Attack detection occurs when a large portion of the offending traffic has common destinations. The destinations are monitored at some chosen prefix granularity, and those that receive the highest offending packet rates are stored in the *target set*. Clouseau also keeps with each incoming entry a total count of offending packets received at this entry and how many of these were going to destinations from a target set. For detection, we will use counts only for those entries that have recently experienced a filtering decision, where "recently" denotes a period since the last counter reset.

Let TS denote the target set, FIT the set of incoming table entries that recently experienced a filtering decision, and $IT(s)$ denote an entry associated with the source s. Let p_s^d denote an offending packet with source IP s and destination IP d and $n_c[x]$ is exponentially weighted average of the count of the elements x that satisfy the criteria c. The detection criteria for a many-to-few attack is:

$$\frac{\sum n_{d\in \text{TS}, IT(s)\in \text{FIT}}[p_s^d]}{\sum n_{IT(s)\in \text{FIT}}[p_s^d]} > f_{\text{TS}}, \tag{8}$$

where sum is calculated over all incoming entries and f_{TS} is a threshold for attack detection. This criteria tests if the majority of the offending packets that invoked a recent filtering decision had a common set of destinations. When the attack is detected, Clouseau places special marks at all entries that signify that entries are fresh only when an offending packet has a destination from TS. This means that all offending packets going to the target set will be filtered by RBF, bypassing Clouseau. Other offending packets are treated as usual. The end of many-to-few attack can be detected when the condition from the equation (8) is no longer met or when the offending packet rate to the target set subsides. We use an exponentially weighted average of counters to avoid rash attack end detection in the case of pulsing attacks. The reset period — T_{reset} should be sufficiently large so that the memory of filtering decisions is kept for at least several minutes. When the attack end is detected, the special fresh mark is removed and Clouseau reverts to its normal operation. Route changes occurring during the attack can be validated using offending packets that travel to destinations other than the target set. Given the proposed deployment at popular routers, we expect that many sources will use these routers to reach a large number of destinations, and will have no problems validating route changes during cycling attacks.

The attacker could attempt to misuse the above proposed modification to Clouseau to create a denial-of-service on legitimate traffic by triggering attack detection and leading Clouseau to mark all entries as fresh for a given target set. This attack only affects sources that experience a route change during the attack and that cross the Clouseau router only to send traffic to destinations in the target set. We expect that portion of such sources would be too small to create an interesting target for the attacker.

7 Related work

There is extensive work on IP traceback which aims to detect a packet's origin by using packet header marks placed by routers on the packet's path. As these approaches do not filter spoofed traffic, we do not cover them in this section.

There are several spoofing defense approaches that use concepts similar to route-based filtering. They keep the incoming table where each source address is associated with some chosen parameter's value, learned in a reliable manner. This parameter is inspected in incoming packets and observed values should match the expected value from the incoming table. Mismatching packets are considered spoofed.

Hop-count filtering (HCF) [10] infers the expected hop count value between each source and an HCF-server using TTL values in TCP packets, and filters out packets whose hop count is not close enough to the expected value. HCF can be deployed at the end-host that is a victim of a DDoS attack and is thus easier to deploy than RBF. It offers good protection against random spoofing, but a smart attacker can learn the expected hop count value by trial and error and completely defeat HCF.

Stack-Pi [8] associates a source address of the packet with a mark in the packet's IP header. This mark is jointly created by Stack-Pi routers that forward the packet, by shifting

the value in the IP identification field and appending to it a small secret mark. During a flooding DDoS attack, marks can be used by the target host to identify paths that carry a large volume of traffic, allowing filtering based on mark values. This filtering inflicts collateral damage to legitimate users that share a path with an attacker. Although marked packets can be filtered close to the attack's victim, marking routers must be deployed widely to ensure mark diversity. The spoofed traffic consumes Internet resources before being filtered at the destination.

Spoofing prevention method (SPM) associates a source-AS or a destination-AS of a packet with a cryptographic secret exchanged between the source and the destination AS, and carried in the packet. Packets are checked for the proper mark as they exit the source AS and on entry to the destination AS. The main advantage of SPM over RBF is that it is an end-to-end protocol and as such has lower deployment cost than RBF, which has to be deployed at well-connected routers. The main disadvantage is that SPM can only help filter traffic that spoofs an SPM address to an SPM destination, so the protection that SPM offers to a deploying network depends on the degree of SPM deployment in the Internet. Packets that spoof non-SPM addresses or spoof SPM addresses to non-SPM destinations cannot be filtered. Also, filtering occurs at the destination, which means that Internet resources between the attackers and the destination are still consumed by spoofed traffic.

In [7], Li et al. proposed the SAVE protocol, where participating sources generate periodic advertisements about their prefixes, and send them to all destinations. Filters use the advertisements to populate incoming tables by recording the arrival interface as the valid interface for the prefixes in the advertisement. SAVE advertisements are also generated when a source experiences a routing table change, which correctly updates incoming tables for some, but not all, routing changes. Incomplete tables significantly lower filtering effectiveness, limiting the practicality of SAVE for real deployments.

Recently proposed Inter-Domain Packet Filters (IDPF) [12] are built using local BGP advertisement information. IDPFs filter less spoofed traffic than route-based filters because they mark multiple incoming interfaces as expected, and cannot detect which interface is really used by a source. IDPF approach further faces problems in some specific routing scenarios, e.g., in presence of selective announcements. Conversely, Clouseau will properly detect expected single or multiple interfaces in any routing scenario.

8 Conclusions

IP spoofing is a commonly used attack tool that complicates and often defeats defenses against many security threats. Two major problems have prevented design of practical spoofing defenses. First, the most optimistic results suggested that effective combating of spoofing required deployment at several thousand ASes, at least. Second, no workable method of producing the data structures required to perform effective filtering had been proposed.

This paper has addressed both of these core problems. We have shown that deployment of RBF at as few as 50 ASes can lead to effective filtering, and we have demonstrated a feasible method of creating the information needed to perform that filtering. The Clouseau system has a number of advantages for this purpose, including being self-contained on sites that deploy it, requiring no changes in protocols or cooperation with any other sites, offering strong deployment advantages to the groups that need to deploy it, having little or no impact on normal Internet traffic, and being resilient to attacks intended to subvert its operation. These claims are validated by analysis and extensive experimentation with a working Clouseau prototype. Thus, this paper demonstrates a reasonable system whose deployment at 50 ASes could largely reduce spoofing in the Internet.

References

[1] University of Oregon Route Views Project, http://www.RouteViews.org

[2] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in the Proceedings of the OSDI02, Dec. 2002, USENIX Association, pp. 255—270.

[3] S. Shakkottai and R. Srikant and N. Brownlee and A. Broido and kc claffy, "The RTT Distribution of TCP Flows in the Internet and its Impact on TCPbased Flow Control," CAIDA Technical Report, TR-2004-02.

[4] ARIN WHOIS Database, http://www.arin.net/whois/

[5] NLANR – National Laboratory for Applied Network Research, http://www.nlanr.net

[6] P. Ferguson, and D. Senie. "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2267

[7] J. Li and J. Mirkovic and M. Wang and P. Reiher and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol," Proceedings of INFOCOM, June 2002.

[8] A. Perrig and D. Song and A. Yaar. "StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks," Carnegie Mellon University Technical Report, CMU-CS-02-208, February 2003.

[9] K. Park and H.Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," In Proceedings of SIGCOMM 2001.

[10] C. Jin and H. Wang and K.G. Shin, "Hop-count filtering: an effective defense against spoofed DDoS traffic," Proceedings of the 10th ACM conference on Computer and communications security, 2003.

[11] A. Bremler-Barr and H. Levy, "Spoofing Prevention Method," In Proceedings of INFOCOM'05, March 2005.

[12] Z. Duan and X. Yuan and J. Chandrashekar, "Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates," Proceedings of INFOCOM'06, April 2006.

[13] W. Feller, "An Introduction to Probability Theory," Wiley 1968.