MORP4: Monitoring active network space dynamically

Iliana Xygkou*, Jithin K. Sojan*, Dhruv Rauthan*, Thomas Holterbach*,†, Alberto Dainotti*
*Georgia Tech †University of Strasbourg

ABSTRACT

A *network telescope* passively monitors traffic reaching Internet address space that is not assigned to any hosts but is advertised to the global routing system. This traffic is by definition *unsolicited*. For more than two decades, network telescopes have enabled research breakthroughs by allowing global visibility into a wide range of Internet phenomena. However, network telescopes are afflicted by two main issues: progressive erosion, due to the increasing scarcity and commercial value of address space, and blacklisting.

To overcome these issues, we propose MORP4, a programmable data plane framework implementing a "dynamic" network telescope. MORP4 adaptively tracks unused space of an organization's network with configurable time and space granularity and captures only traffic directed towards unused addresses. MORP4 enables an organization not only to "recover" unused space for capturing unsolicited traffic but it also provides greater utility than a static telescope by countering blacklisting and monitoring addresses adjacent to used ones. We provide an implementation of MORP4 in P4 and Python, and deploy it on an actual Tofino switch. We show that it can detect unused (IPv4) address space at the finest granularity (/32) while operating at line rate and leaving enough resources for other applications running on the switch.

1 INTRODUCTION AND MOTIVATION

A network telescope, or simply *telescope*, is a research infrastructure, also used for cybersecurity operations, that passively monitors traffic reaching Internet address space that is not assigned to any hosts but is advertised to the global routing system (*i.e., dark address space*). This traffic is by definition *unsolicited* (also known as Internet background radiation—IBR) and is constituted of an evolving mix of diverse traffic components originating from across the whole Internet [3].

For more than two decades, network telescope instrumentation has enabled significant research breakthroughs, by allowing global visibility into a wide range of Internet phenomena: the automated spread of malicious software such as Internet worms or viruses [23, 16, 19, 17]; random spoofed source denial-of-service attacks [18]; large-scale botnet activities [6, 20]; macroscopic Internet blackouts due to natural disasters [9], network failures [2] and state censorship [7]; trends in IPv4 address space utilization [8, 10]; bugs and misconfigurations in popular applications [3], etc. Measurement and analysis of such macroscopic phenomena are of key relevance for the security and reliability of the Internet infrastructure.

However, two major issues affect this type of infrastructure broadly. First, due to the increasing scarcity and commercial value of IPv4 address space, the size of (even the largest) telescopes has been progressively eroding over the years, to the point that some organizations stopped operating them or reduced them to sizes that severely impact their research and educational utility [24, 4]. Another major issue experienced by telescopes is blacklisting by malicious actors: lists of network telescope address blocks have been circulating [1] and are used by these actors and hardcoded in malware to avoid probing them—again, causing a loss of the infrastructure's utility.

To overcome these issues, we propose MORP4, a programmable data plane framework implementing a "dynamic" network telescope. Our solution is based on deploying programmable switches at the ingress/egress points of the organization's network to adaptively track its unused space with configurable time and space granularity and capture only traffic directed towards unused addresses. MORP4 enables an organization not only to "recover" unused space for capturing unsolicited traffic, but also counters blacklisting, since it dynamically monitors addresses intermittently used or mixed with used ones.

The intuition at the basis of MORP4, is that many organizations tend to have subnet blocks that are internally assigned but are in fact *sparsely* used. Moreover, organizations interested in operating network telescopes—*e.g.,* academic and research institutions—often own significant portions of IPv4 address space. This phenomenon is particularly evident in the US, where many research institutions received generous IPv4 address block allocations in the early life of the Internet. Figure 1 shows that more than 40% autonomous systems (ASes) of US academic organizations originate the equivalent of a /16 IPv4 block ($\approx 65k$ addresses) in BGP[1]. We analyzed one week of NetFlow records from a large educational and research network's border routers (in Sep. 2023) and found that more than 70% of the /24 blocks in the address space it originated did not appear in any flow during that week. These blocks might be either completely unused but allocated to organization's units[2] or potentially silent only for certain periods of time, and thus in either case not suitable to statically assign to a telescope. While Netflow records are obtained from packet sampling, and thus are only indicative of upper bounds, they highlight a phenomenon that is well known to network engineers and especially in academia, which contrasts with the increasing unavailability of unused blocks/addresses for dedicated allocation to network telescopes. MORP4 addresses this contrast by tapping into existing resources *seamlessly*, without requiring their dedicated allocation, and preserving the confidentiality of traffic towards any monitored address as soon as it becomes in use.

We provide an implementation of MORP4 in P4 and deploy it on an actual Tofino switch. We show that it can detect unused (IPv4) address space at the finest granularity (/32) while operating at line

Georgia Tech - Technical Report - Internet Intelligence Lab 2024-01, ,
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] 812 ASNs associated with the tag "Academic" and located in the US according to bgp.tools.
[2] This includes the rare case where public addresses are used internally but not routed towards the rest of the Internet.

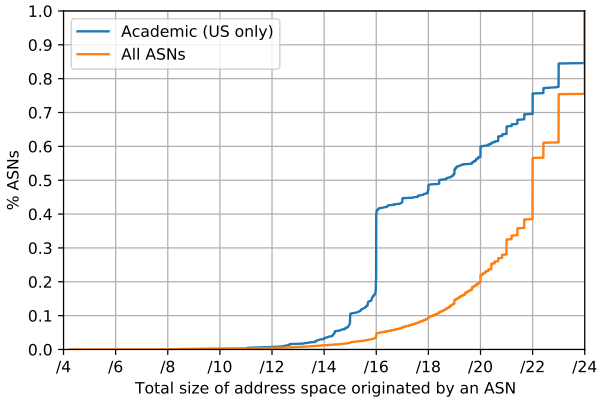

Figure 1: Distribution of the reachable address space of (a) academic organizations in the U.S. (#ASNs=812) and (b) all ASNs (#ASNs=75258). For (a), around 96.7% of them advertise on BGP less space than a /14, while only AS7377 (USCD) advertises more space than a /10. Similarly, for (b), around 98.9% of the ASNs announce on BGP less space than a /14, whereas only 124 ASNs (0.16%) announce more space than a /10.

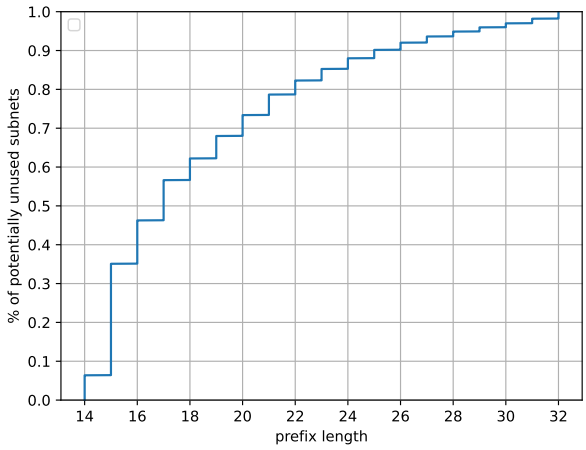

Figure 2: Analysis of the marginal gain in potentially unused IP addresses in the network of Table 1. We discover higher proportion of the total unused space when we increase the processing address granularity, but with a lower rate as we approach the finest granularity of /32.

Address granularity	24	26	28	30	32
% of originated space potentially unused	74.39%	77.79%	80.2%	82%	84.54%

Table 1: Analysis of the potentially unused IP addresses in a large US research network which advertises on BGP prefixes accounting in total for 4860160 addresses (~ 1.15 of a /10). The first row denotes the address granularity at which we process the subnets to discover inactive ones, and the second row provides the portion of the total subnets which are found inactive.

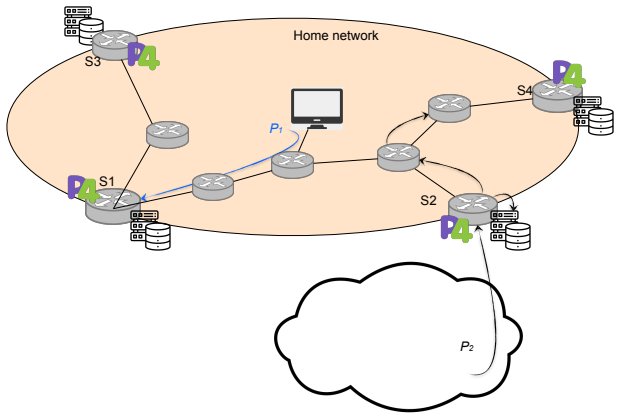

Figure 3: Overview of MORP4. P1 is an outbound packet. P2 is an incoming packet destined towards an unused address.

rate and leaving enough resources for other applications running on the switches.

2 MORP4 OVERVIEW

2.1 MORP4 overview

In Figure 3 we present an overview of the deployment of our system MORP4. MORP4 is made of a control-plane component running on an SDN controller and a data-plane component running on programmable switches. The data-plane component must run on the switch(es) in the ingress and egress points of a network under the organization's administration, so that it can observe the traffic exchanged between that network and the rest of the Internet. In addition, since each switch will dump the captured IBR traffic on a dedicated monitoring port, a *capture* host is attached to it to store the traffic (and, if desired, perform on-the-fly processing).

MORP4 can be deployed in two different ways. In *inline* mode, it is deployed on programmable switches that are also responsible for forwarding all the network's traffic at the respective ingress/egress points. In many practical scenarios, this would require replacing non-programmable routers. In *monitoring* mode instead, the routers forwarding production traffic share a copy of the packets with the MORP4 programmable switches (*e.g.,* through an optical split or a SPAN port). This deployment allows an organization to not replace production routers and offers the advantage to not risk any interference with production traffic.

MORP4 is highly configurable by the network administrators, depending on their internal policies (network allocations, size of

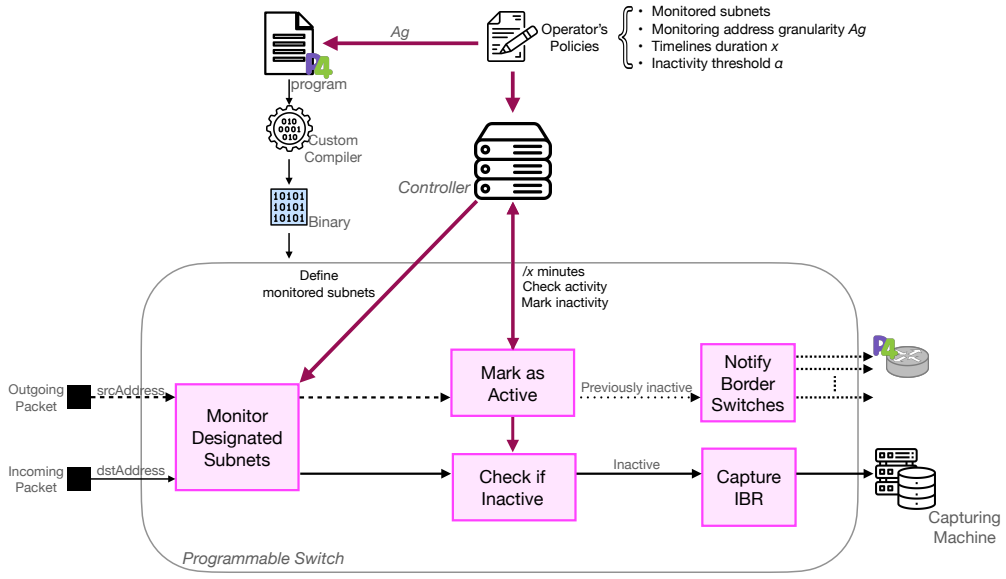

Figure 4: Workflow of a packet traversing our system. The operator's policies define which prefixes should be monitored. (a) Outgoing packet: We will check whether its source IP address belongs to our monitored space and if so, we will set the state of the corresponding /A_g subnet to active and will notify the rest of the border routers upon change of that state. (b) Incoming packet: We will check whether its destination IP address belongs to our monitored space and if so, we will examine whether it's destined to an active or inactive subnet so that we capture it in the second case.

DHCP pools, etc.) and other considerations. Network administrators configure MORP4 with a list of prefixes within their network which they wish to monitor for potentially unused address space. We will refer to these prefixes as *monitored*. The administrators also configure the *address granularity* (A_g) at which they want MORP4 to discover unused address blocks, *e.g.,* /30 subnets. MORP4 monitors both inbound and outbound traffic to/from the network at each ingress/egress point, and maintains the *state* of each individual *subnet* of size A_g in each monitored prefix. This state can either be *active* or *inactive*: we mark a subnet as active when we observe at least one of its hosts generating traffic and inactive otherwise. The basic intuition is that an outgoing packet is an indication of a subnet with at least one active host[3]. Incoming packets instead do not provide evidence of a subnet being active: they can either be part of normal traffic towards an active host or constitute unsolicited traffic (*i.e.,* IBR).

MORP4 will mark a subnet as inactive only after a configurable timeout period T_o (*e.g.,* 1 week or 1 hour) during which it does not observe any outgoing packets from the subnet addresses. When a subnet is inactive, MORP4 logs the inbound traffic—*i.e.,* mirrors packets on its dedicated monitoring port without interfering regular forwarding—since it is unsolicited and thus IBR. In Section 3.6, we show that on a Tofino-1 switch MORP4 is capable to work all the way down to A_g = /32 subnet granularity for IPv4, *i.e.,* track individual addresses.

Our approach offers several advantages that address important issues to advance telescope's state of the art and makes them more robust and capable:

MORP4 expands a telescope size by allowing an organization to "recover" large portions of their unused space, thus contrasting the steady reduction of monitored unused space experienced by research and educational networks. Using the same Netflow record dataset described in Section 1, in Table 1 we show that (for the week we analyzed) the hypothetical gain of unused address space when identifying "silent" subnets increases with the address block granularity of our search. E.g., while taking into account only /24 blocks leads to potentially recover more than 3.6M addresses (74.4% of the organization's network address space), tracking individual addresses yields more than 4.1M (84.5%). MORP4 can be configured

[3]A packet might carry a spoofed source address. However, this case does not represent a significant issue. We expand on this possibility in Section 3.

to detect unused space down to IPv4 /32 granularity (*i.e.,* individual addresses).

MORP4 makes telescopes more robust to blacklisting. A dynamic telescope monitors space that is changing over time but more importantly contiguous and potentially intermingled with actively used addresses (*i.e.,* the actual targets of malicious actors), thus making blacklisting practically infeasible. Depending on the configuration and the specific scenario, a dynamic telescope will also monitor addresses previously used (*e.g.,* a month or a week before), again rendering blacklisting practically impossible.

MORP4 generates more scientifically and operationally valuable data. By monitoring a mix of largely unused address blocks—and thus less interesting to attackers based on scans or potentially even blacklisted—and addresses part of populated address blocks, it not only captures relevant IBR that other telescopes might miss but it also offers the opportunity to continuously compare IBR directed towards these two different categories of space. This data labeling offers a precious pre-classification of IBR, potentially separating noisy traffic from more interesting traffic.

MORP4 reduces the attack surface with respect to traffic confidentiality and data/infrastructure integrity and availability. Real-world deployments of telescope infrastructure can be ad-hoc and/or use unconventional software, hardware, and configurations that can present vulnerabilities not found in more traditional environments. *(i)* Moving most of the functionalities to hardware (*i.e.,* a programmable switch) and *(ii)* entirely automating the list generation *(iii)* using direct observation of active traffic instead of indirect/manual sources, will reduce the attack/incident surface. Finally, our code is published as open-source (with a BSD-style license) [25]. We implement MORP4 in only $\sim$ 300 lines of $P4_{16}$ code and $\sim$ 200 lines of *Python* code: Networks deploying our system can easily inspect the code and independently compile it to deploy it on their switches.

2.2 Challenges

Dynamic transformation of IPv4 space. Inside a network of an organization devices such as personal computers, mobile phones, etc, enter and leave abruptly, and are not statically assigned IP addresses. Therefore, an approach which relies on the complement of a list of the currently assigned IP addresses to obtain the inactive address space, would require immediate updates on that list when a new device entered or exited the network. To tackle this issue, MORP4 passively monitors outbound traffic at the network's egress points, and automatically and instantly learns that a host has become active based on their network activity.

Data Privacy. A further implication of the previous issue is the data privacy of the network users (*i.e.,* we need to ensure that MORP4 does not log sensitive data). That means, when a host becomes active, we should instantly stop logging any packets destined to it. MORP4 achieves that by declaring a subnet as active when it observes an outgoing packet originating from that subnet. After that, it will not store any incoming packets destined to that active subnet. However, if MORP4 has observed no outgoing packets from a subnet in the past configurable timeout period T_o, the subnet has become inactive, and therefore it is safe to log packets towards it.

Accurate Declaration of an Inactive Prefix. Ideally, a subnet should be declared as inactive after a timeout period T_o with no outgoing packets originating from it. This essentially translates to a timer per subnet which keeps track of how much time has passed since the last outgoing packet from that subnet. Within the context of a data- and control-plane framework, the control plane should check the data plane every second for indication of activity from every subnet. Such a procedure is time- and bandwidth-consuming, and would lead to inaccuracies because of the unpredictable transmission delay between the controller and the switch. Therefore, we propose a time bin-based approach in which the controller retrieves this indication of activity of each subnet every configurable time bin of x minutes, which is long enough for the controller to conclude this process for all monitored subnets. Moreover, the controller maintains a counter for each subnet which tracks the number of consecutive time bins that the switch has observed no outgoing packets from that subnet. When the counter reaches a predefined value α (*i.e.,* the timeout $T_o = \alpha \cdot x$ minutes has expired), the controller will declare the subnet as inactive, and update its state to such on the switch.

Race Conditions between Control Plane and Data Plane. The process of the controller's retrieval of indication of activity of a subnet, and the potential subsequent update of its state in the switch to inactive is time-consuming and during that the switch may process a great amount of packets. In the scenario, where the switch has not observed an outgoing packet from a subnet for α time bins of x minutes, the controller (1) will retrieve the indication of activity at the α^{th} time bin and will determine the subnet as inactive, and then (2) will update its state on the switch to such. However, between (1) and (2) the switch may have processed an outgoing packet from that subnet indicating it is now active. MORP4 addresses this issue with the use of two different data plane structures whose values the controller retrieves and updates per subnet in an order that alleviates the race condition.

Consistent state between the switches. Most networks comprise of multiple ingress/egress traffic points. As a result, we deploy MORP4 on all border switches to monitor the inbound/outbound traffic and to obtain an accurate and global view of the state of the network subnets. However, the introduction of additional switches incurs the following scenario: (1) switch A observes an outgoing packet from a subnet for the fist time in the current time bin, (2) switch B observes traffic towards that subnet. We need to ensure that between the time points (1) and (2) switch B learns about the activation of the subnet, and consequently does not log incoming traffic destined to it in the case the subnet was previously inactive. The controller will periodically perform the aforementioned procedure of collecting the indication of activity per subnet and updating its state in all switches. However, in order to accelerate the process and prevent storing sensitive information, MORP4 employs the mechanism of communication between the border switches. When a switch observes an outgoing packet from a subnet for the first time in the current time bin, it will notify the rest of the switches about that activity through specially tailored packets which introduce negligible overhead in the network.

Minimal memory overhead on the switch. A data-plane-only alternative to MORP4 could use timestamps. This would remove

the need for the controller's periodic actions, and introduce greater accuracy in declaring a prefix as inactive immediately after T_o has expired. However, storing a timestamp value requires 32 bits of SRAM to be able to operate accurately at a second-level. Given that we would need to store a timestamp value for every subnet, the memory overhead of such an approach would be bery high. For that purpose, MORP4 uses just 2 bits (*i.e.,* 16 times less SRAM) to keep track of the state of each subnet. Furthermore, inconsistencies between the different border switches would occur due to transmission delays and inconsistent internal clocks. Therefore, we choose the time bin-based approach of MORP4 to introduce as small memory overhead as possible on the switch to leave space for more applications to be deployed on it.

3 DESIGN AND IMPLEMENTATION

In this section, we describe how we designed and implemented the MORP4 system to address the challenges we described while enabling configurability.

3.1 Single-switch scenario

Our design must take into account various constraints and trade-offs. E.g., a programmable switch has both limited resources to maintain state and limited flexibility in logical operations. However, as we show in this section, it is highly beneficial for certain decisions to happen directly in the data plane. For this reason, MORP4 splits responsibilities between the controller and the switch: in a nutshell, the controller makes the decision to turn an active address to inactive, whereas the switch is responsible for detecting when an inactive address should be turned to active.

For each address, each component keeps the state in their respective memory and needs to stay consistent. Figure 5 shows how the two components interact with each other in a time-binned fashion to coordinate and achieve adjustable monitoring granularity. At bootstrap, the state of each address of each monitored prefix is set to "active" to prevent logging traffic towards addresses potentially in use. The controller then periodically checks for any activity from each address by querying the switch in discrete time bins of x minutes each, where x is fixed and configurable (*e.g.,* 60 minutes). For each time bin and address, the switch keeps an active/inactive *flag* indicating if the switch observed at least one outgoing packet from that address in that time bin.

The controller retrieves the flags of all addresses every time bin, and sets a (previously active) address to inactive if it did not have any active flags for α consecutive time bins, where $\alpha = T_o/x$. When the controller changes an address state, it forces that change also on the switch, which will trigger the switch to start logging inbound packets destined to the address. Traffic directed to an inactive address is unsolicited and thus IBR. However, since an address might become active at any point in time, the switch will immediately turn an address' state to active—and stop logging any inbound traffic towards it—as soon as it observes a packet originating from it.

On the one hand, a state change from active to inactive happens with a maximum latency that is coarse: *i.e.,* between T_o and $T_o + x$ compared to when the address actually stopped sending packets. This latency is configurable and acceptable, since for telescope

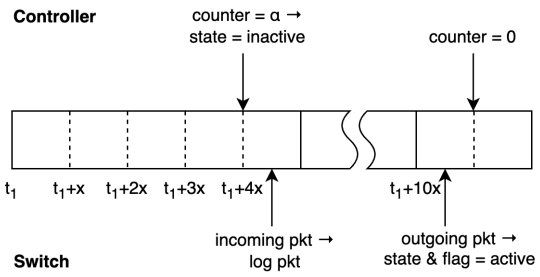

Figure 5: Evolution of the state of an address where $\alpha = 4$. After α time periods of no record of outgoing packets, the controller declares the address as inactive. As a result, the switch starts logging incoming packets towards that address. However, when the switch observes at least one outgoing packet, it will immediately set the state of the address to active. Finally, the controller retrieves the flag value for that time bin, and since it is active, it resets its counter.

applications it is not critical that new monitored addresses are added quickly. An operator might instead want to be conservative in picking a T_o large enough to avoid logging packets somehow related to the activity of the host that was using an address before becoming inactive. On the other hand, our design ensures that a state change from inactive to active happens at line rate, *i.e.,* before the triggering packet exits the switch. This condition guarantees that incoming response packets to a host suddenly using a previously inactive IP address are *not* logged by the switch due to its state still appearing as inactive at the time a response packet is received.

3.2 Multi-switch scenario

Most networks running network telescopes and where we envision deploying MORP4 (*e.g.,* universities and other research and education networks) will have a single location that all the ingress and egress traffic traverses. However, other networks might present ingress/egress points in multiple locations, which will require our system to be deployed on multiple switches and to operate in a distributed manner.

A challenge arising in this scenario is how to synchronize the active/inactive state of the addresses across all the switches with sufficient time granularity to prevent a switch from capturing non-IBR traffic. This issue is possible due to asymmetric routing: when the triggering outbound packet of a host starting to use a previously inactive IP address traverses a MORP4 switch, there is no guarantee that a related inbound packet will enter the network at the same location, as it might rather enter through a different switch. It is thus necessary to update every switch's state for that address to active *before* they have a chance to observe an inbound packet due to the address sudden activity.

We solve this problem by having MORP4 switches notify through special control packets all the other MORP4 switches of the state change for an address. We illustrate this mechanism through a sample scenario in Figure 6: the control packet p_c is generated directly by the switch $S1$ and is sent at the same time the packet

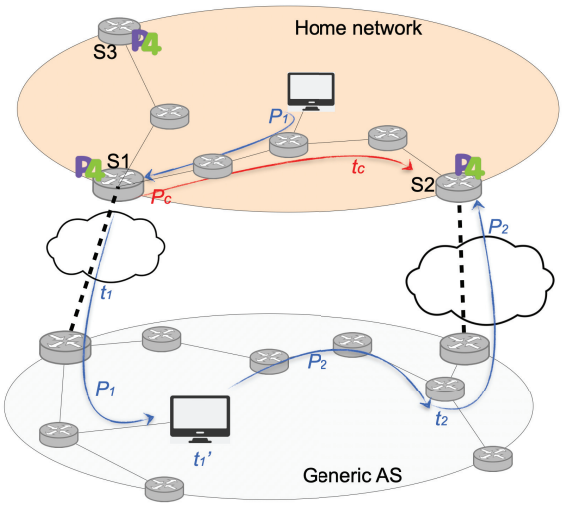

Figure 6: Under the multi-switch scenario, asymmetric routing, in theory, poses a challenge on how to avoid logging regular (*i.e.,* non-IBR) traffic. When the switch $S1$ observes an outgoing packet p_1 from a previously inactive address, it will update its state to active, and notify all the other MORP4 switches of this modification with control packets p_c. Consequently, the switch $S2$ will not log the response packet p_2. This holds under the assumption that the time required for p_c to reach $S2$ (t_c) is less than the time needed for the original packet p_1 to reach the destination (t_1), to then be processed (t_1'), and for the resulting response packet p_2 to travel back to $S2$ (t_2) (*i.e.,* , $t_1 + t_1' + t_2 < t_c$).

p_1 (which triggered the detection) is forwarded. The assumption is that the sum of the time t_1 for p_1 to reach a host outside of the monitored network (which will traverse at least one inter-domain link), the time t_1' for the destination host to process p_1 and send a response packet p_2, and the time t_2 for the corresponding response packet p_2 to reach back any of the other MORP4 switches (again, at least another inter-domain link) *e.g., S2*, is shorter than the time t_c for p_c to travel from $S1$ to $S2$. (In our implementation we send p_c three times to take into account accidental packet loss.)

While the likelihood that $t_1 + t_1' + t_2 < t_c$ is remote, there is no mathematical guarantee it will not happen—*e.g.,* because of extreme delays or packet loss within the home network. This rare event would cause only a single packet to be erroneously captured and logged, which alone is highly unlikely to expose any significant sensitive information. As future work, we plan to experiment with introducing locking and delay mechanisms—either for $p1$ through the controller or in the packet logging pipeline on the traffic capture host.

Finally, in a multi-switch scenario, at each time bin, the controller will read the flags associated with each address from each MORP4 switch in order to keep track of the addresses use over time: If at least one switch observed outgoing packets from an address, the controller will reset its timeout counter and switch its corresponding state to active if previously inactive. Conversely, if there have not been any outgoing packets for α time bins in any switches, the controller will force the address state to inactive in all switches.

3.3 Data-plane Implementation

3.3.1 Data structures. The network administrators wish to monitor a subset of their network for unused IPv4 subnets. We design our system to support a maximum of 2^{22} subnets. When considering the finest possible granularity of /32 for an IPv4 subnet, this limit translates into the ability of tracking each individual address in a set of prefixes cumulatively as large as a /10 block. This choice is consistent with the size of the space that ASes originate in practice, which we showed in Section 1.

Our design uses two tables, implemented as arrays of registers, to store the state and flag (for the current time bin) of each subnet. We call these tables the *State table* and *Flag table*. Subnets from the same monitored prefix are stored sequentially: their position in both arrays is computed as an offset from a base index. We use a third table, the *Monitored prefixes table*, to *(i)* check if the source (destination) address of an outbound (inbound) packet entering the switch belongs to the monitored prefixes and, if so, *(ii)* to obtain the base index and offset needed to locate the corresponding subnet's registers in the State and Flag tables.

Longest prefix matching in P4 is implemented through TCAM Match-Action tables. To implement the Monitored prefixes table, we use a TCAM Match-Action table with the network prefix (network address and netmask) as a key and the corresponding base index as the value. However, due to P4 limitations, we must store the prefix's netmask also as a value. While this in principle would be redundant, in the actual implementation it is necessary, since when calculating the position of the register in the tables (by summing base index and offset) the netmask component of the key that matched in the TCAM cannot be retrieved.

The base indexes are pre-computed by the controller as follows,

```
base_index = 0
for prefix in monitored_prefixes:
    if prefix == examined_prefix:
        break
    # number of /A_g subnets in the prefix
    num_subnets = pow(2, A_g - prefix.pfx_len)
    base_index = base_index + num_subnets
```

where *monitored_prefixes* is the list of the monitored prefixes and *examined_prefix* is the prefix whose *base_index* we wish to compute. Both arrays for the State and Flag tables have the same length: *e.g.,* for a set of monitored prefixes cumulatively summing up to the equivalent of a /10 address block, this would be 2^{A_g-10} (*i.e.,* $\sim 4.2M$ cells for $A_g = 32$).

For each packet, the switch computes the index for the register arrays as

$$index = base\_index + (ip\_address >> (32 - A_g)) \;\&\; network\_mask,$$

and attaches it as (P4) metadata to the packet.

3.3.2 Control flow. When a packet arrives, the switch determines if it is *inbound* or *outbound* based on the switch port the packet is received from (*i.e.,* , from a port connected to the internal network or to the rest of the Internet). However, a switch might also observe a custom *control* packet from another switch signaling it has observed an outbound packet for a previously inactive subnet. Depending on

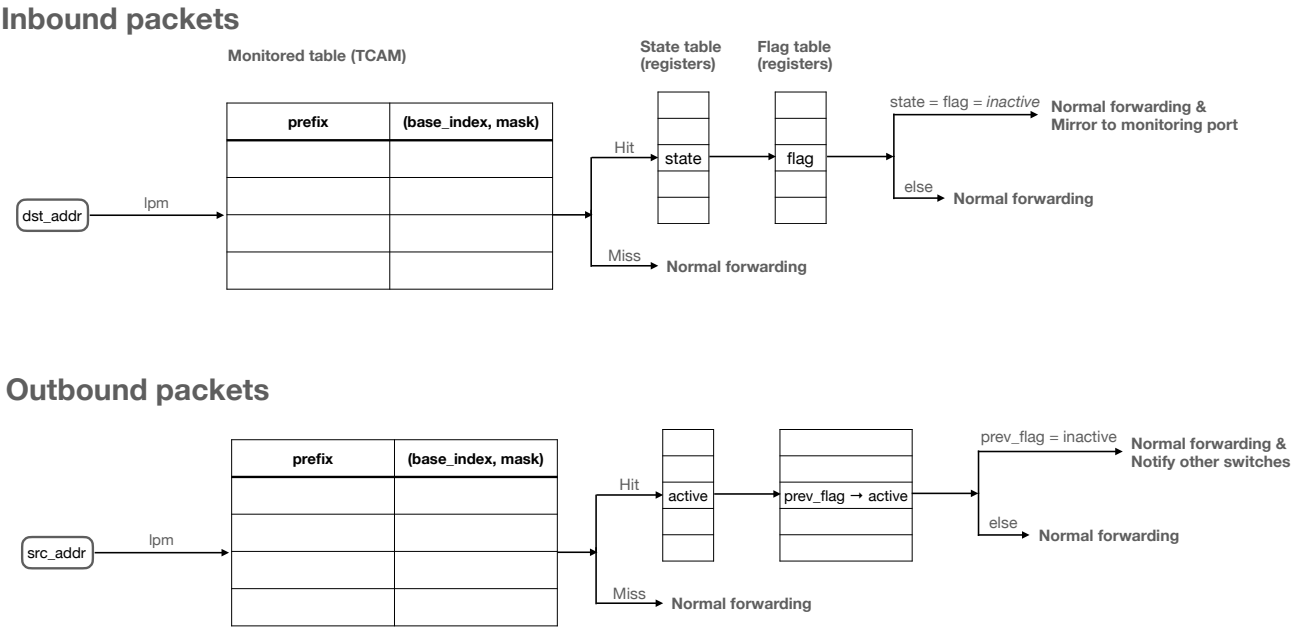

Figure 7: *(i)* Pipeline workflow for inbound packets. We examine if the destination address of the incoming packet belongs to a monitored prefix, and subsequently if both the state and the flag of the corresponding address are inactive. If so, we consider the packet IBR and mirror it to the monitoring port of the switch. In both cases, we normally process the original packet and forward it towards its destination. *(ii)* Pipeline workflow for outbound packets. We examine if the source address of the outgoing packet belongs to a monitored prefix, and subsequently set both the state and the flag of the corresponding address to active. If the previous value of the flag was inactive, we send control packets to the rest of the border switches to notify about the activation of the address. We normally process the original packet and forward it towards its destination.

their characteristics, packets will follow one of the three following control flows:

- *Control packet.* The first stage of a MORP4 switch determines if the packet under examination is a control packet. We mark control packets setting a custom value in the Protocol ID IP header not used by any known protocols and store the source address of the original outbound packet in their custom header. The switch extracts this address and sets the corresponding State and Flag tables' registers to 1. It then drops the control packet, since there is no purpose in forwarding it.
- *Incoming packet.* An incoming packet is destined to either an active or an inactive address. To determine the address' state, the packet checks the values of the registers of the State and Flag tables, and if both of them are equal to 0, the address is inactive (active otherwise). In both cases, the packet will be processed further and forwarded normally. However, in the first case, no active host is expecting the packet and therefore MORP4 will also replicate the packet on the monitoring port for logging.
- *Outgoing packet.* An outgoing packet is an indication of an active host. Thus, the packet will set the corresponding State and Flag tables' registers to 1, and then will be forwarded normally. However, if the previous Flag table's register was equal to 0, we also need to notify the other MORP4 switches with a control packet. To implement this functionality directly in the data-plane (*i.e.,* since using the controller would introduce latency (see Section 3.2)) we trigger the production of a clone of the packet which the switch transforms into a control packet and multicasts to the other MORP4 switches. The transformation step includes *(i)* the truncation of the packet so that it does not introduce unnecessary overhead into the network and *(ii)* the addition of a custom transport layer control header which contains only the field *target_address*, *i.e.,* the IP address that has been observed as active.

3.4 Optionally rate-limiting IBR

A packet destined to an inactive address is considered unsolicited traffic (IBR) and contains valuable information. However, telescope infrastructure can occasionally experience denial of service (DoS) due to an overwhelming amount of traffic towards one or more of its addresses. These events might be caused by synthetic traffic generators used for testing purposes whose packets escape the network under test, or might be due to other types of misconfiguration or even intentional attacks. In any case, storing the full set of packets (instead of *e.g.,* a few samples) from these events provides little value for both research and operations while it can potentially overload the data capturing infrastructure (in the case of MORP4, the monitoring host) and the associated storage.

Identifying IBR traffic directly in the switch's data plane, before it reaches the dedicated IBR capturing components, creates the opportunity to easily address this issue. A Tofino switch is capable of handling Tbps rates, whereas the capturing host can be provisioned to handle typical IBR loads, *i.e.,* with peaks around 10Gbps. We introduce in MORP4 an optional IBR rate-limiting feature, which rate limits first at the whole network level and then at a /24-subnet level the packets that are replicated on the monitoring port. The first filter ensures that the total replicated IBR traffic does not overwhelm the capturing host. Additionally, we use the second filter to isolate the traffic destined to different /24 subnets so that a heavily probed subnet does not saturate the available capturing rate and thus does not prevent logging packets towards other subnets.

We implement this feature using meters provided by the P4 language for rate-liming. For the first step, we define the *Global Inactive* meter, and for the second step we define an *Inactive* array of meters of length equal to $2^{14} (\sim 16k)$, *i.e.,* the number of /24 subnets for a set of monitored prefixes cumulatively summing up to the equivalent of a /10 address block. Similarly to computing the index for the State and Flag tables, *(i)* the controller pre-computes the $inactive\_base\_index$ by setting $A_g = 24$ and inserts it as an additional value in the *Monitored prefixes table*, and *(ii)* the switch derives the index for the *Inactive* array of meters by setting $A_g = 24$ and replacing $base\_index$ with $dark\_base\_index$.

3.5 Control-plane Implementation

The control-plane is responsible for *(i)* declaring a subnet as inactive, *(ii)* ensuring consistent subnet states across all switches, and *(iii)* adjusting the logging policies for each inactive subnet.

As mentioned in Section 3.1, our system operates in time bins of x minutes. At each time bin, the controller queries each MORP4 switch for the flags of all (monitored) subnets. For a given subnet, if it results active in at least one switch and its previous state in the controller was inactive, the controller updates the state of that subnet in *all* switches. Otherwise, the controller will increment the counter of consecutive inactive time bins, and if it reaches the threshold value α, it will set the state of the subnet to inactive in *all* switches.

3.6 Resource utilization

We implement the data-plane pipeline of MORP4's in $\sim$ 300 lines of $P4_{16}$ code and the control plane in $\sim$ 200 lines of *Python* code.

Resource	Tofino-1				Tofino-2			
	$A_g = 26$		$A_g = 32$		$A_g = 26$		$A_g = 32$	
	w/o	*w/*	*w/o*	*w/*	*w/o*	*w/*	*w/o*	*w/*
#Stages	6	8	6	8	7	8	7	8
SRAM	1.4%	3.3%	7.8%	9.8%	1.1%	2.3%	4.8%	6%
Hash bits	1.8%	2.4%	2.1%	2.6%	1.1%	1.4%	1.2%	1.6%
TCAM	0.7%	0.7%	0.7%	0.7%	0.4%	0.4%	0.4%	0.4%
VLIW instruction	3.4%	3.9%	3.4%	3.9%	2.5%	2.8%	2.5%	2.8%
Meter ALU	4.2%	8.3%	4.2%	8.3%	5.0%	7.5%	5.0%	7.5%
Ternary Xbar	0.5%	0.5%	0.5%	0.5%	0.3%	0.3%	0.3%	0.3%
Exact Xbar	1.2%	1.6%	1.2%	1.6%	0.9%	1.0%	0.9%	1.0%
Gateway	5.7%	6.8%	5.7%	6.8%	5.6%	6.3%	5.6%	6.3%

Table 2: Resource utilization of the implementation of MORP4 in Tofino-1 and in Tofino-2 for two different values of the address granularity A_g: 26 and 32.

Furthermore, we provide two implementations of MORP4 which run on Intel Tofino-1 and Tofino-2 switches respectively.

Table 2 shows the resource usage of our implementation in Tofino-1 and in Tofino-2 for two different values of the address granularity A_g, /26 and /32. Tofino-1 and -2 have 12 and 20 pipeline stages correspondingly. MORP4 uses at most 8 stages when deployed with the rate limiting option, leaving room for other applications to run in the switch. Moreover, the main resource that MORP4 requires is the SRAM for the State and Flag tables register arrays whose size depends on A_g (2 bits per monitored $/A_g$ subnet), and thus can be adjusted accordingly to satisfy specific memory constraints. Note that this is an average across stage. In reality, with $A_g = 32$, the State and Flag table almost saturate the maximum size of stateful objects dedicated to two respective stages (91.43% in Tofino-1 and 69.6% in Tofino-2), indicating that we cannot achieve finer granularity unless in the implementation we split these table in half, at the cost of using two more stages.

However, even at the finest granularity, MORP4 uses only $\sim 8.4M$ SRAM bits to monitor 2^{22} addresses in total and—as shown by the relatively low values shown in Table 2—allows for extensions and other applications to run on the switch.

We note that in the Tofino-2 the maximum size of a stateful object decreased. Thus, to still be able to operate at address granularity $A_g = 32$ we split each of the *State* and *Flag* register arrays into two halves, which increase the number of occupied stages in MORP4's Tofino-2 implementation.

4 RELATED WORK

In this section we will discuss the related work on discovering unused IPv4 space in a production network and its utilization for honeynets. Moreover, we will refer to the literature around *gray space*'s analysis.

Discovering unused space. Cooke et al. [5] propose *Dark Oracle*, a system to detect unused and unreachable addresses within a network using external (BGP) and local routing data (OSPF), and host configuration data (DHCP). However, access to the latter can be challenging, and their system's classification accuracy is highly dependent on retrieving the updated data sources (which can be inaccurate or unstable) promptly. Mizoguchi et al. [14, 15] recover unused IP addresses by requesting the unassigned addresses from the DHCP server, and assign them to network sensors. This approach is constrained within a DHCP segment and its expansion towards more diverse collected intelligence requires the placement of additional sensors in different segments of the network. MORP4 tackles these issues by monitoring the inbound/outbound traffic at line rate and updating an address' state immediately, while it monitors the whole network's space at its ingress/egress points.

In a data-driven direction, Shimoda et al. [22] propose *DarkPots*, a system of virtualized honeypots which utilize the unused addresses of a network. In their implementation, they place two mirroring switches, one before and one after the firewall and border gateway router. The switch between the gateway and the local network mirrors the traffic to the *Vacancy Checker* component that monitors which addresses produce traffic, and removes them from a list of unused addresses provided by the network administrators. The other switch mirrors the traffic to the *Forwarder* component which forwards the traffic to honeypots if it is destined to an address within the updated list of unused addresses. However, if an address becomes active, but the *Forwarder* is not yet aware of that due to the time cost of the synchronization with the *Vacancy Checker*, the former will misdirect legitimate traffic to honeypots and potentially interfere with the connection of the internal host. Moreover, it is not clear whether an address is added back to the list of unused addresses if it is no longer assigned to a host. The authors extend their work in [21] to detecting unused pairs of IP addresses and ports, to include into their analysis scope even active hosts. They focus on TCP flows initiated by an external host and classify them as malicious when they observe no TCP SYN/ACK responses for a specified time interval. In the scenario, in which an active host becomes temporarily unavailable (*e.g.,* , due to reboot) and is unable to respond to the TCP SYN packet before the timeout expires, the honeypot will take over an otherwise legitimate session. MORP4 addresses these challenges since it updates an address' state at line rate, and considers an address inactive after a configurable timeout period T_o with no outgoing packets from that address has expired. Thereby, addresses are **dynamically** removed from and **added to** the network telescope. The network administrator is also able to adjust the value of T_o to be as conservative as they wish to avoid interfering with normal traffic.

Gray space analysis. Jin et al. [12] introduces the term *gray space* to refer to IP addresses within a network which have produced no traffic over a defined time period (but they could have been active before that or could become active after that). These addresses are dynamic and randomly distributed, and therefore external actors cannot easily filter them out from their scanning lists. The authors apply the above heuristic on month-long NetFlow traces of a large campus network to focus on flows towards gray addresses, and study the patterns in scanning activity. Jin et al. [13] extend the previous work by comparing the NetFlow records of the *gray space* to the ones of the active space to develop an algorithm to detect and monitor malicious activity. Finally, Eduard et al. [11] use the notion of *grey space* when they create rules to apply on NetFlow data and classify one-way traffic into seven different classes. MORP4 is able to significantly contribute to such research initiatives by providing insight into the payload of the packets destined to inactive space.

5 DISCUSSION AND FUTURE WORK

add alternative solutions: eg broadcom (hard to program not very much flexibility –IX Logging only IBR traffic. As mentioned in Sec. 3.2, deploying MORP4 in multiple switches requires a synchronization mechanism which ensures that all switches update a previously inactive address' state to active when at least one switch observes an outbound packet from that address. We implement this with control packets from a MORP4 switch to all other MORP4 switches upon state change of an address. However, the highly improbable scenario in which inbound traffic towards a newly activated host arrives at a switch which has not yet received the corresponding control packet could cause the switch to log normal traffic.

Eliminating this race condition requires an approach in which we buffer the mistakenly captured inbound packet for an adequate amount of time (*e.g.,* 1 second) and examine again if it is indeed destined to an unused address before we finally store it. In future work, we aim to assess the idea of leveraging DPDK to process packets at very high speed, and queues to buffer these packets. *where to move this, since it's not no longer an idea? –IX*.

Furthermore, according to our current implementation a MORP4 switch sends out 3 control packets to each other MORP4 switch to reduce the effects of packet loss conditions. Nevertheless, this approach does not guarantee the delivery of the packet in a switch since all 3 of the control packets could be dropped in case of an extremely congested network, faulty links, *etc.*. Therefore, a more reliable strategy should include the acknowledgement of reception of a control packet. We leave this exploration to future work.

Addressing spoofed IP addresses. Spoofing can have significant effects on our detection of unused IPv4 space as addresses that are not actually in use might appear as active because of a spoofed source IP address in an outgoing packet. MORP4 reduces the possibility of occurrence of this case by examining outgoing packets based on the port on which the switch receives them. By doing so, only hosts within the network can tamper with an address' state with spoofed packets. To handle this issue, we envision an approach in which upon observation of an outbound packet from a previously inactive address, a MORP4 switch will immediately *ping* this address and only the reception of the expected response will result in changing the address' state to active. We plan to investigate this idea in future work.

REFERENCES

[1] Manos Antonakakis et al. "Understanding the Mirai Botnet". In: *USENIX Security Symposium*. 2017.

[2] Karyn Benson et al. "Gaining Insight into AS-level Outages through Analysis of Internet Background Radiation". In: *ACM CoNEXT Student Workshop*. 2012.

Alberto Dainotti* [4pt] *Georgia Tech †University of Strasbourg‡

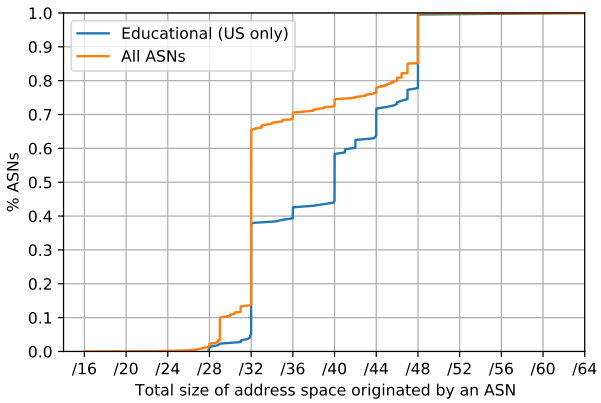

Figure 8: Distribution of the reachable IPv6 address space of (a) academic organizations in the U.S. (#ASNs=221) and (b) all ASNs (#ASNs=32565). For (a), around 96.8% of them advertise on BGP IPv6 space between a /28 and /48, while for (b), around 98.9% of the ASNs announce on BGP IPv6 space between a /28 and /48. We exclude from the graph 11 (0.03%) ASNs that announce less IPv6 space than a /64.

[3] Karyn. Benson et al. "Leveraging Internet Background Radiation for Opportunistic Network Analysis". In: *ACM IMC*. 2015.

[4] CAIDA. *STARDUST Workshop series: 2012, 2019, 2021.* "https://www.caida.org/workshops/?workshopserieslisting=DUST&show_all=1".

[5] Evan Cooke et al. "The dark oracle: perspective-aware unused and unreachable address discovery." In: *NSDI*. Vol. 6. 2006, pp. 8–8.

[6] Alberto Dainotti et al. "Analysis of a "/0" stealth scan from a botnet". In: *ACM IMC*. 2012.

[7] Alberto Dainotti et al. "Analysis of country-wide Internet outages caused by censorship". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement*. IMC '11. Berlin, Germany: ACM, 2011, pp. 1–18. ISBN: 978-1-4503-1013-0. DOI: http://doi.acm.org/10.1145/2068816.2068818.

[8] Alberto Dainotti et al. "Estimating Internet Address Space Usage Through Passive Measurements". In: *ACM SIGCOMM Computer Communication Review* 44.1 (2014).

[9] Alberto Dainotti et al. "Extracting Benefit from Harm: Using Malware Pollution to Analyze the Impact of Political and Geophysical Events on the Internet". In: *ACM SIGCOMM Computer Communication Review* 42.1 (2012), pp. 31–39.

[10] Alberto Dainotti et al. "Lost in Space: Improving Inference of IPv4 Address Space Utilization". In: *IEEE Journal on Selected Areas in Communications (JSAC)* 34.6 (2016), pp. 1862–1876.

[11] Eduard Glatz and Xenofontas Dimitropoulos. "Classifying internet one-way traffic". In: *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '12. London, England, UK: Association for Computing Machinery, 2012, pp. 417–418. ISBN: 9781450310970. DOI: 10.1145/2254756.2254821.

[12] Yu Jin et al. "Gray's anatomy: Dissecting scanning activities using IP gray space analysis". In: *Usenix SysML07* (2007).

[13] Yu Jin et al. "Identifying and tracking suspicious activities through IP gray space analysis". In: *Proceedings of the 3rd Annual ACM Workshop on Mining Network Data*. MineNet '07. San Diego, California, USA: Association for Computing Machinery, 2007, pp. 7–12. ISBN: 9781595937926. DOI: 10.1145/1269880.1269883.

[14] Seiichiro Mizoguchi, Yoshiaki Hori, and Kouichi Sakurai. "Monitoring Unused IP Addresses on Segments Managed by DHCP". In: *2008 Fourth International Conference on Networked Computing and Advanced Information Management*. Vol. 1. 2008, pp. 510–515. DOI: 10.1109/NCM.2008.245.

[15] Seiichiro Mizoguchi et al. "Darknet Monitoring on Real-Operated Networks". In: *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. 2010, pp. 278–285. DOI: 10.1109/BWCCA.2010.82.

[16] David Moore and Colleen Shannon. "The Spread of the Witty Worm". In: *IEEE Security and Privacy* 2.4 (2005), pp. 46–50.

[17] David Moore, Colleen Shannon, and kc Claffy. "Code-Red: A Case Study on the Spread and Victims of an Internet Worm". In: *ACM Internet Measurement Workshop*. 2002.

[18] David Moore, Geoffrey Voelker, and Stefan Savage. "Inferring Internet Denial-of-Service Activity". In: *USENIX Security Symposium*. 2001.

[19] David Moore et al. "Inside the Slammer Worm". In: *IEEE Security and Privacy* 1.4 (2003), pp. 33–39.

[20] Elias Raftopoulos et al. "How Dangerous Is Internet Scanning? A Measurement Study of the Aftermath of an Internet-Wide Scan". In: *Traffic Monitoring and Analysis Workshop*. 2015.

[21] Akihiro SHIMODA, Tatsuya MORI, and Shigeki GOTO. "Extended Darknet: Multi-Dimensional Internet Threat Monitoring System". In: *IEICE Transactions on Communications* E95.B.6 (2012), pp. 1915–1923. DOI: 10.1587/transcom.E95.B.1915.

[22] Akihiro Shimoda, Tatsuya Mori, and Shigeki Goto. "Sensor in the Dark: Building Untraceable Large-Scale Honeypots Using Virtualization Technologies". In: *2010 10th IEEE/IPSJ International Symposium on Applications and the Internet*. 2010, pp. 22–30. DOI: 10.1109/SAINT.2010.42.

[23] Stuart Staniford et al. "The Top Speed of Flash Worms". In: *ACM Workshop on Rapid Malcode (WORM)*. 2004.

[24] *UCSD Network Telescope*. http://www.caida.org/projects/network_telescope/.

[25] Iliana Xygkou et al. *Dynamic Network Telescope*. "https://github.com/InetIntel/dynamic-telescope".