# Counting NATted Hosts by Observing TCP/IP Field Behaviors

Sophon Mongkolluksamee
National Electronic and Computer Technology Center (NECTEC)
Pathumthani, Thailand
Email: sophon.mongkolluksamee@nectec.or.th

Kensuke Fukuda
National Institute of Informatics
Tokyo, Japan
Email: kensuke@nii.ac.jp

Panita Pongpaibool
National Electronic and Computer Technology Center (NECTEC)
Pathumthani, Thailand
Email: panita@nectec.or.th

***Abstract*—With the prevalence of Network Address Translation (NAT), identifying a number of Internet users becomes a challenging task because many users share the same public IP address. This paper proposes a passive technique for estimating a number of Internet hosts sharing the same IP address, i.e., NATted hosts. Previous work by Bellovin [1] counted NATted hosts by observing a sequence of IPID fields in IP header. This technique only works on some operating systems with a global counter for the IPID sequence (e.g., Windows). Other operating systems that implement the IPID sequence on a per-flow or a random basis are not detected. The proposed technique overcomes this limitation by observing patterns of the TCP sequence number and the TCP source port, in addition to the IPID sequence. Our technique demonstrates more accurate estimate than the previous work in controlled experiments. Moreover, applying our technique on a collection of longitudinal traffic traces measured at a trans-Pacific link in 2001-2010, we find that the percentage of the NATted hosts is stably less than 2% over years.**

## I. Introduction

Network Address Translation (NAT) has become a popular tool for sharing the Internet connection to many devices, particularly in the IPv4 address exhaustion. The NAT gateway performs stateful mapping between several private IP addresses and a public IP address. Identifying a number of computers behind a NAT gateway is a challenging task because all computers will share a single IP address. This leads to difficulty in inventory checking, usage tracking, and billing processes.

There have been many attempts to detect NAT usage and to count a number of hosts behind NAT. In 2002, Bellovin [1] presented a technique for counting NATted hosts by observing sequences of ID field in IP header (IPID). Bellovin assumes that one IPID sequence represents a single host. However, this assumption is true only in Windows and FreeBSD implementations relying on a single IPID consecutive counter for all packets from the same host. Other operating systems (OS) may implement the IPID sequence as a per-flow counter (using a separate counter for each outgoing flow of packets) or a random number. Bellovin's algorithm will fail to detect such hosts.

In this paper, we propose a technique to tackle limitations of Bellovin's technique, namely counting NATted hosts which implement a per-flow IPID sequence and a random IPID sequence. In our technique, we identify an individual host as well as its OS based on sequences of IPID, TCP sequence number and TCP source port. After the validation of the proposed algorithm with synthetic and small dataset, we apply the proposed technique to longitudinal traffic traces called the MAWI traces [2] to study a long-term trend of NAT usage. As the results, we show that the percentage of the NATted hosts is stable and less than 2% in 2001-2010.

## II. Related work

There have been many attempts to detect NAT usage and to count a number of hosts behind NAT. Bellovin [1] observes patterns of IPID values and builds up a set of IPID sequences. When a new packet arrives, its IPID is appended to a selected sequence if the value matches within a predefined IPID gap and time interval. This technique works well with the global IPID implementation, but it over-estimates number of hosts if they use per-flow IPID. Moreover, it cannot identify hosts with random IPID.

Detecting NAT and counting NATted hosts using OS fingerprinting [3], [4] is a quick and easy method. The information in TCP/IP header such as TCP window size (WSS), initial TTL value, overall SYN packet size and Don't Fragment bit (DF) can be used to create a signature for identifying host OS. Nevertheless, this method cannot distinguish hosts that have the same OS.

Another piece of information that can distinguish hosts is the microscopic deviations in device hardware such as clock skews. Kohno et al. [5] measure the clock skews from ICMP timestamp and TCP timestamp option. This technique can count the number of hosts behind NAT, even if all of the hosts run the same OS, and even if the hosts use random or constant IPID. However, this method does not support fully passive measurement because some operating systems, such as Windows 2000 and Windows XP, disable TCP timestamp option by default.

Rui et al. [6] proposed an algorithm to passively detect hosts hidden behind NAT by analyzing traffic features, such as the number of packets sent, packets received, UDP packets, TCP packets, DNS request packets, FIN packets, RST packets, and SYN packets. The analysis uses directed acyclic graph support vector machine.

978-1-4577-2053-6/12/$31.00 ©2012 IEEE

Maier et al. [7] detected presence of NAT and estimated the number of hosts behind NAT using IP TTL and HTTP user-agent strings. The OS type, browser family, and browser version in HTTP user-agent string are used to distinguish hosts. However, this approach cannot give a good result if many hosts use the same OS and browser version.

## III. TCP/IP Fields Behaviors

There is a lot of useful TCP/IP information that can identify an individual host. In this study, we examine behaviors of IPID, TCP sequence number, and TCP source port for various types of operating systems.The 16-bit IPID field in the IP header identifies unique packets when fragmentation occurs. If there is no fragmentation, the IPID field is just a 16-bit counter generated by a sending host. The assignment of IPID values can be done in three ways: global IPID, per-flow IPID, and random IPID [8]. The 32-bit TCP sequence number is generated at the beginning of each TCP connection establishment. After that, it increases by the packet size (in bytes). The 16-bit TCP source port is assigned by a client computer when it tries to establish a TCP connection.

We setup five machines with different operating systems and have them download 20 files from a web server. Each download starts 20 seconds after a previous download. We capture download traffic and observe behaviors of these three fields. From this preliminary study, we find that different operating systems have different patterns of the three header fields which can be useful in distinguishing individual hosts. Summary of this study is shown in Table I.

- Windows XP, Vista and 7: Windows implements IPID as a global counter, meaning that all connections within the same host follows a single sequence. Windows uses a random number for starting sequence number of each TCP connection. For TCP source port, Windows will increase its source port linearly proportional to starting time of each connection.
- Linux 2.6: The IPID implementation of Linux is a per-flow counter. The graph of IPID sequence in Table I shows 20 lines of IPID for 20 TCP download sessions. For TCP sequence number and TCP source port, Linux implements both of them as counters.
- FreeBSD 8.1: FreeBSD uses a global counter for IPID and random numbers for TCP sequence number and TCP source port.
- MAC OS 10.6: Although MAC OS is based on FreeBSD, it has different TCP/IP stack implementation. It generates random IPIDs and random TCP sequence numbers but increases TCP source ports linearly for each connection.
- OpenBSD 4.6: OpenBSD is designed with security consideration. As a result, it generates IPID, TCP sequence number, and TCP source port in a random fashion.

## IV. Methodology

To estimate the number of NATted hosts from network traffic, we modify Bellovin's algorithm, which uses only IPID information, to include other TCP information, namely TCP sequence number and TCP source port. As observed in Section III, each OS has its own implementation for selecting a starting TCP sequence number and a TCP source port for each TCP connection. This characteristic can help us distinguish individual hosts when it implements the IPID field in a per-flow or a random manner. Our technique consists of two phases, sequence construction phase and host classification phase.

### A. Sequence Construction Phase

We process a packet trace file to collect IPIDs, TCP sequence numbers, and TCP source ports of all packets. We try to order IPID and TCP sequence number into monotonic increasing sequences. Let $\mathbb{D}$ be a set of IPID sequences and $\mathbb{C}$ be a set of sequences of TCP sequence numbers (called TCP sequences, for short). Conditions for sequence construction are given below.

$$\begin{aligned}\mathbb{D} = \{D_i = (d_{i,1}, d_{i,2}, ..., d_{i,n}, ...)| \\ d_{i,n}(t) - d_{i,n-1}(t) \leq timelim_{IPID} \text{ and} \\ d_{i,n} - d_{i,n-1} \leq gaplim_{IPID}\}\end{aligned} \quad (1)$$

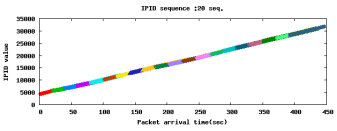
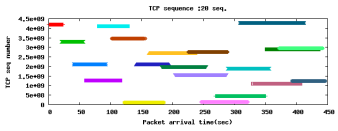
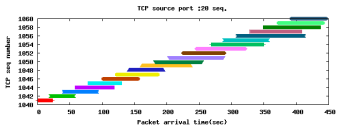
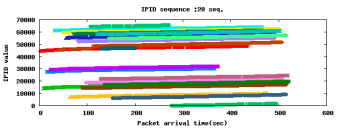
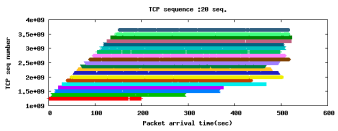
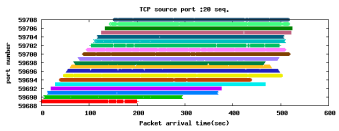
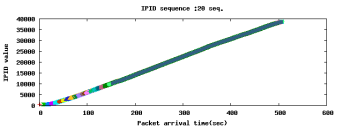
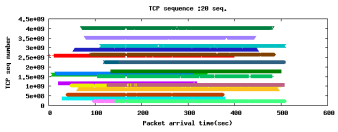
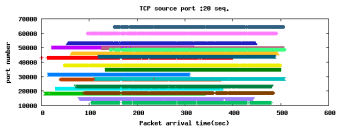
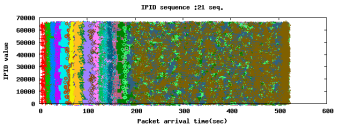
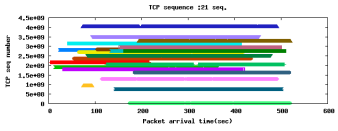
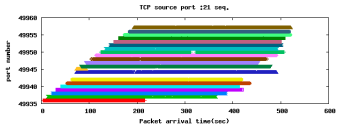
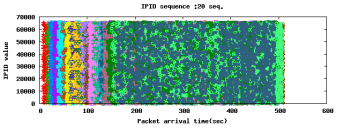
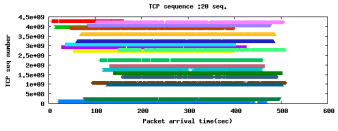
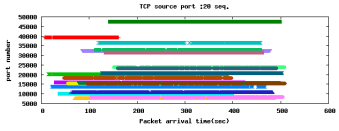
$$\begin{aligned}\mathbb{C} = \{C_i = (c_{i,1}, c_{i,2}, ..., c_{i,n}, ...)| \\ c_{i,n}(t) - c_{i,n-1}(t) \leq timelim_{TCPseq} \text{ and} \\ c_{i,n} - c_{i,n-1} \leq gaplim_{TCPseq}\}\end{aligned} \quad (2)$$

$d_{i,n}$ indicates an $n^{th}$ IPID value in a sequence $D_i$, and $d_{i,n}(t)$ is the arrival time of a packet containing this $d_{i,n}$ IPID value, likewise for $c_{i,n}$ and $c_{i,n}(t)$. In case of IPID, if $d_{i,n} - d_{i,n-1} = 1$, we call it a *perfect* order. If $d_{i,n} = d_{i,n-1}$, we label it *duplicate*. Otherwise, we label it *out-of-order*. In addition, we discard the packet with $d_{i,n} = 0$ because some device may set IPID to zero for all packets [1].

Constructing sequences of IPID is a little more complicated than those of TCP sequence number because IPID could have big-endian or little-endian byte order depending on the OS. We cannot tell the endianness by looking at a single IPID. Thus, we create two candidate sets, one for big-endian sequences and the other for little-endian sequences. For each new IPID, we search for the best-match sequence from the little-endian set first. If the best match is found, the IPID will be appended to this best-match sequence. If the best match is not found in the little-endian set, we search from the big-endian set. If the best match is not found in both candidate sets, we create a new sequence for this IPID in both sets.

At this step, we have three sets of candidate sequences, the big-endian IPID sequence set ($\mathbb{D}_{BE}$), the little-endian IPID sequence set ($\mathbb{D}_{LE}$), and the TCP sequence set ($\mathbb{C}$). We perform sequence cleanup on these three sets to remove effects of random numbers. First, sequences that have less than *MemberCri* members are discarded. These short sequences are usually formed by random numbers and are too short to be a sequence. Second, sequences whose ends are within *gapfac*gaplim* of one another, and whose arrival time are within *timefac*timelim* seconds of one another, as according

TABLE I
CHARACTERISTICS OF IPID,TCP SEQUENCE NUMBER, AND TCP SOURCE PORT IN POPULAR OPERATING SYSTEMS

| OS | IPID<br>x: time (sec), y: IPID number | TCP sequence number<br>x: time (sec), y: TCP sequence number | TCP source port<br>x: time (sec), y: TCP source port |
|---|---|---|---|
| Windows XP, Vista, 7 |  |  |  |
| Linux 2.6 |  |  |  |
| FreeBSD 8.1 |  |  |  |
| MacOS 10.6 |  |  |  |
| OpenBSD 4.8 |  |  |  |

to [1] are merged. Third, IPID sequences with more than *OutOfOrderCri%* of their members being *out-of-order* are removed. Having many out-of-order members usually means a sequence is formed by random chances. Lastly, we re-examine remaining sequences and remove any sequences that are still too short, i.e., with less than *MemberCri2* members. We use a parameter set shown in Table II based on [1], [4].

TABLE II
PARAMETERS USED IN SEQUENCE CONSTRUCTION PHASE

| Parameter | IPID | TCP seq number |
|---|---|---|
| *timelim* (sec) | 5 | 30 |
| *gaplim* | 64 | 1460 |
| *timefac* | 80 | 70 |
| *gapfac* | 80 | 50 |
| *MemberCri* | 5 | 5 |
| *MemberCri2* | 50 | 50 |
| *OutOfOrderCri* (%) | 50 | - |

### B. Host Classification Phase

To distinguish an individual host from one another, we have to classify relationship among IPID, TCP sequence number and TCP source port. This phase consists of three steps.

*1) Associating TCP sequence numbers with IPID sequences:* We identify a set of TCP sequences $\hat{\mathbb{C}}$ that are associated with each IPID sequence $D_i$. Let $IPID(c_{j,n})$ refers to an IPID value of a packet with a sequence number $c_{j,n}$. A TCP sequence number $c_{j,n}$ is defined to be associated with an IPID sequence $D_i$ if $IPID(c_{j,n}) \in D_i$. Moreover, a TCP sequence $C_j = (c_{j,1}, c_{j,2}, ..., c_{j,n}, ...)$ is defined to be associated with an IPID sequence $D_i$, if more than *assoc%* of its members are associated with $D_i$. $C_j \lhd D_i$ denotes such relationship. In our experiments, we set *assoc* to 50%. Using these definitions, we could obtain multiple TCP sequences that are associated with an IPID sequence $D_i$. So for each $D_i$, $\hat{\mathbb{C}}_i = \{C_j | C_j \lhd D_i\}$ We are interested in $N = |\hat{\mathbb{C}}_i|$, a number of TCP sequences in the set $\hat{\mathbb{C}}_i$. There are three possible cases for each $D_i$.

- $N > 1$: This is the case where more than one TCP sequence is associated with the same IPID sequence. This IPID sequence must be a global counter. We consider such IPID sequence to represent one host which could be Windows or FreeBSD. To pinpoint exact OS of this host, further analysis in step 3 is needed.
- $N = 1$: In this case, there is only one TCP sequence associated with a sequence $D_i$. This could be because there is only one TCP connection during the observed duration or it could be a system which uses per-flow IPID. Possible operating systems in this case are Windows, Linux 2.6, or FreeBSD. Further analysis in step 2 could narrow down the answer.
- $N = 0$: It is possible that there is no TCP sequence associated with this IPID sequence. This might be because this host carries no TCP traffic. In such case, we will count this IPID sequence as one host with unknown operating systems. (There is no further processing.)

*2) Observing patterns of TCP sequence starting number:* From the first step, if we find only one TCP sequence for each

IPID sequence, we need to determine whether this one TCP connection corresponds to a global IPID or a per-flow IPID. We examine the pattern of the TCP sequence starting number to distinguish a Linux host from Windows and FreeBSD hosts. As shown in Table I, Windows and FreeBSD systems do not have any pattern of the first TCP sequence number, while Linux shows an increasing order of the first TCP sequence number for each connection. Therefore, in this step, we take $c_{j,1}$ from all sequences $C_j \in \hat{\mathbb{C}}_i$ and try to construct a new monotonic sequence $\acute{C}$ with a conditions below where we experimentally set $timelim_{StartSeq}$ to 60 seconds, and $gaplim_{StartSeq}$ to 200,000,000.

$$\begin{aligned} \acute{C} = (c_{1,1}, c_{2,1}, ..., c_{n,1}, ...) : \\ c_{n,1}(t) - c_{n-1,1}(t) \leq timelim_{StartSeq} \text{ and} \\ c_{n,1} - c_{n-1,1} \leq gaplim_{StartSeq} \end{aligned} \quad (3)$$

If $|\acute{C}| \geq 1$, we assume that number of Linux hosts equals to $|\acute{C}|$. Otherwise, if $|\acute{C}| = 0$, that IPID sequence will represent one host, either Windows or FreeBSD. Further processing in step 3 is required to distinguish between Windows and FreeBSD hosts.

*3) Observing TCP source port behaviors:* This step handles the case where a TCP sequence cannot be associated with any IPID sequence which could be due to the effect of random IPID implementations. Moreover, this step can distinguish between Windows and FreeBSD from which steps 1 and 2 fail to identify. We see from Table I that a source port pattern can distinguish a MacOS host from an OpenBSD host, as well as differentiate a Windows host from a FreeBSD host. Therefore, we try to discover any pattern within a set of observed source ports. Let $\mathbb{S}$ be a set of all monotonic increasing sequences, $S_i$ that satisfy conditions below. We experimentally set $timelim_{SrcPort}$ to 30 seconds and $gaplim_{SrcPort}$ to 64.

$$\begin{aligned} \mathbb{S} = \{S_i = (s_{1,1}, s_{2,1}, ..., s_{n,1}, ...)| \\ s_{n,1}(t) - s_{n-1,1}(t) \leq timelim_{SrcPort} \text{ and} \\ s_{n,1} - s_{n-1,1} \leq gaplim_{SrcPort}\} \end{aligned} \quad (4)$$

If $|\mathbb{S}| = 0$, this means we cannot find any source-port pattern. In the case that this TCP sequence is not associateed with any IPID sequence (i.e., random IPID), we conclude that this TCP sequence belongs to an OpenBSD host. In the case that this TCP connection is associated with an IPID sequence (i.e., global IPID), we conclude that this host is FreeBSD. On the other hand, if $|\mathbb{S}| > 0$, each sequence $S_i \in \mathbb{S}$ will represent a MacOS host in case of random IPID, or a Windows host in case of global IPID.

## V. Experiments and Results

We evaluate the proposed method using two trace files: a synthetic NAT traffic trace and a real NAT traffic trace obtained from a wireless router. We compare number of hosts detected by our method with that of Bellovins method. We also evaluate accuracy of OS classification against the ground truth. In cases that we do not have the ground truth, we compare it with the result of p0f, a popular passive OS fingerprinting tool [3].

### A. Synthetic NAT Traffic

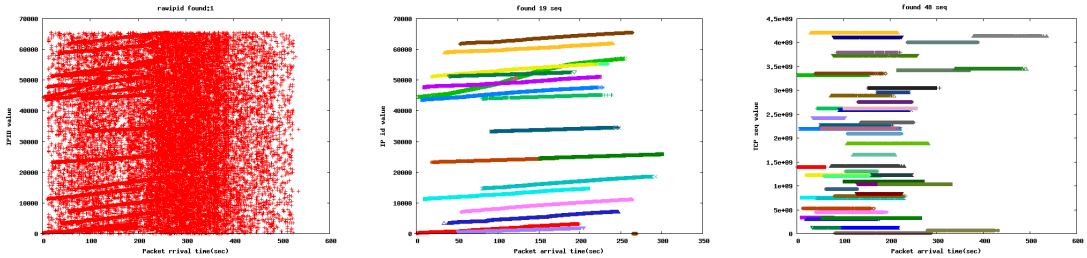
We create synthetic NAT traffic by capturing traffic from 16 hosts and craft their IP header such that all hosts have the same IP address. Operating systems of the 16 hosts are 16 hosts which are ten Windows XP, two MacOS 10.6 , two Linux 2.6, one FreeBSD 8.1 and one OpenBSD 4.8. We make each host download 3 files from a web server.Each download starts after the previous one within a random interval: 10, 20, 30, 40 or 50 seconds. We replay download traffic from all hosts together as if all hosts are behind the same NAT domain. We analyze the replay traffic. Patterns of IPID and TCP sequence number are shown in Figure 1.

We can see many obvious lines of IPID sequence from Figure 1(a). The scattered dots are from random IPIDs. After we clean up short sequences and combine adjacent sequences, we find 19 IPID sequences as shown in Figure 1(b). However, there are actually 17 IPID sequences (ten from ten windows hosts, one from a FreeBSD host, and six from three Linux hosts). The two extra sequences are short sequences formed by chance. Completing the host classification phase, we obtain 18 hosts as shown in Table III. The two extra hosts we detected are due to the two erroneous sequences. Note that we incorrectly conclude that seven TCP sequences belong to an OpenBSD machine when only three sequences actually belong to it. As a result, we yield 15 true positives and 3 false positives. If we only used the IPID information like Bellovin, we would have detected 19 hosts, with 14 true positives and 5 false positives.

TABLE III
Result of Synthetic NAT Trace

| Quantity | Classified OS | Actual OS | Detected sequences of IPID:TCP |
|---|---|---|---|
| 10 | Windows | Windows XP | 1:3 |
| 1 | FreeBSD | FreeBSD 8.1 | 1:3 |
| 1 | Linux | Linux 2.6 | 3:3 |
| 1 | Linux | Linux 2.6 | 2:2 |
| 1 | Unknown | Linux 2.6 | 1:1 |
| 2 | Unknown | - | 1:0 |
| 1 | MacOS | MacOS 10.6 | 0:2 |
| 1 | OpenBSD | MacOS 10.6 | 0:7 |

Next we compare results of our method to those of Bellovin's under different scenarios listed in Table IV. The first scenario is from the previous result. The traces of the other five scenarios were created in the same way as that of 16 hosts. The results show that our technique can detect Windows, FreeBSD, and MacOS hosts accurately, but it fails to detect more than one OpenBSD host. In other words, no matter how many OpenBSD hosts there are, our algorithm will count them as one host. This is because our algorithm assumes all TCP sequences that it cannot classify belong to an OpenBSD host. If these TCP sequences are formed by chance, then we always get at least one false positive. If these TCP sequences are from $k$ OpenBSD machines, where $k > 1$, we obtain $k - 1$ false negatives.



(a) RAW IPID (b) IPID seq (c) TCP seq

Fig. 1. Result of synthetic NAT

TABLE IV
COMPARISON OF THE PROPOSED TECHNIQUE WITH BELLOVINS ALGORITHM

| Scenarios | # Hosts Estimated | | True Positives | | False Positives | |
|---|---|---|---|---|---|---|
| | Ours | Bellovin | Ours | Bellovin | Ours | Bellovin |
| 16 Mixed OS | 18 | 19 | 15 | 14 | 3 | 5 |
| 10 Windows | 10 | 10 | 10 | 10 | 0 | 0 |
| 3 Linux | 2 | 9 | 2 | 3 | 0 | 6 |
| 3 MacOS | 3 | 0 | 3 | 0 | 0 | 0 |
| 3 FreeBSD | 3 | 3 | 3 | 3 | 0 | 0 |
| 3 OpenBSD | 1 | 0 | 1 | 0 | 0 | 0 |

### B. Real NAT Traffic

We capture the real NAT traffic trace from a wireless router which acts as a NAT gateway. At time of experiments, there were seven hosts connecting to this wireless router as verified by the ARP command. However, we do not know the actual OS breakdown. We capture 15 minutes of traffic at the outbound interface of the wireless router. We find four IPID sequences and 109 TCP sequences. If we only used the IPID sequence information to estimate the number hosts, we would have detected only four out of seven hosts.

After the host classification phase, our algorithm detects eight hosts: three Windows, three MacOS, one Linux and one OpenBSD. Details are given in Table V. We know for sure that the detection of OpenBSD host is wrong because there is no OpenBSD usage in an office that this wireless router serves. Since we do not know for sure the real operating systems of other seven hosts, we compare with the results of p0f.

TABLE V
RESULT OF REAL NAT TRACE

| Quantity | Classified OS | OS by p0f | Detected sequences of IPID:TCP |
|---|---|---|---|
| 1 | WindowsXP | Windows:2000 SP2,XP | 1:12 |
| 1 | Linux | Windows:XP/2000 | 1:44 |
| 1 | WindowsXP | Windows:2000 SP2,XP | 1:3 |
| 1 | Windows | Unknown | 1:1 |
| 1 | MacOS | Unknown | 0:33 |
| 1 | MacOS | Unknown | 0:2 |
| 1 | MacOS | Unknown | 0:12 |
| 1 | OpenBSD | Unknown | 0:2 |

## VI. LONG-TERM ANALYSIS OF NAT USAGE

In order to study a long-term trend of NAT usage, we apply our algorithm to a collection of longitudinal traffic traces called MAWI traces[2] that archive daily packet traces collected at a trans-Pacific transit link between Japan and the US for more than 10 years. Each trace file is composed of pcap format packets starting from 14:00 to 14:15. We select trace files from the first and second day of every month in 2001-2010. The IP addresses in the traces were anonymized, but this does not affect the performance evaluation of our method. The average number of the unique IP addresses appeared in a trace increases from 27K in 2001 to 202K in 2010. Our main focus is on the estimation of the average number of NATted hosts and the average number of hosts accessed via a NATted host over years. We experimentally set the parameters shown in Table II based on the results in [1], [4].

Figure 2 displays the percentage of the average number of the estimated NATted IP addresses in MAWI trace from 2001 to 2010. The errorbars in the figure indicate the standard deviations. We can confirm that the percentage is relatively low (less than 2%) and stable over years. High variability in 2002 is due to rapid increase of DNS traffic volume. On the other hand, a lower percentage in 2003-2004 is the results of the worm propagations (i.e., Blaster and Sasser), characterized by a huge number of hosts with a few packets.

Moreover, the average number of identified hosts behind a NATted IP address is estimated to 5 stably over years (see Figure 3). This result suggests a less probability of the existence of a large-size NAT in the dataset.

The MAWI traces are also used to compare our method to Bellovin's in term of processing time. The first and second day traces of every month in 2001 are selected. The average number of packets per trace is 2.9M packets. On average, our method takes 13 minutes, and Bellovin's takes 10 minutes. The 30% overhead of our method is due to additional processing of TCP sequence number and TCP source port. The main factors that impact the processing time of both methods are the number of packets and the number of flows in traffic trace.

## VII. DISCUSSIONS

### A. Accuracy of the estimation

We discuss some issues that could have potential impacts on the accuracy of our method, though the proposed method outperformed the original one.

First of all, the IPID value assigned by a source host can be changed at middle boxes. Some NAT device change IPID value or set it to constant value, such as zero, before passing it

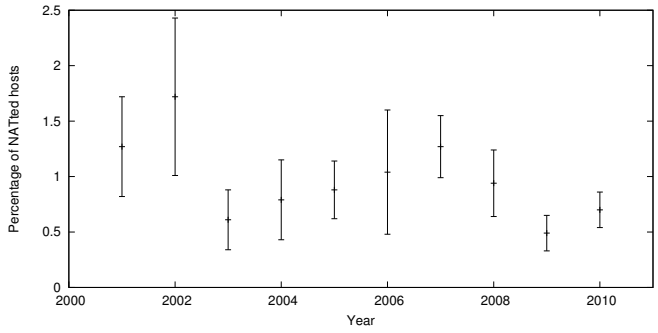



Fig. 2. Percentage of NATted IP address

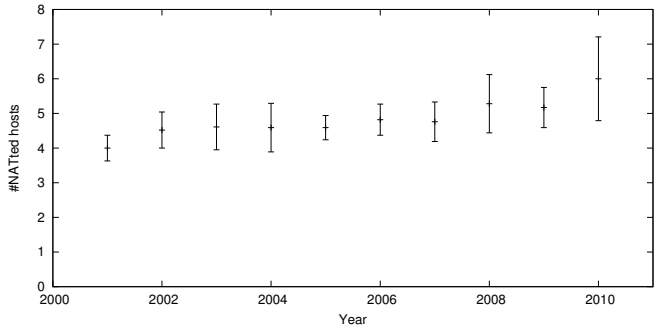



Fig. 3. Average number of NATted hosts per IP address

to another network. In this case, such hosts cannot be detected by our method. Similarly, VPN hosts affect the performance of our method since we cannot distinguish between VPN hosts and NATted hosts, though we expect few VPN connections in the trans-Pacific link.

Next, using TCP sequence and TCP source port helps in distinguishing among global IPID, per-flow IPID and random IPID host. However, if there is no TCP connection or only one TCP connection per IPID sequence, the method only identifies Windows, Linux or FreeBSD hosts. TCP source port pattern is a way to identify a random IPID host, such as MacOS. However, NAT functionality can be implemented in many ways. Some NAT device could randomly map a source port of an original packet. In this case, the method hardly obtains an appropriate result from using the pattern of TCP source port for random IPID host. However, considering the population of OpenBSD hosts in MAWI traces [9], we expect that the effect of this to the estimated NAT percentage is small.

Lastly, the detection accuracy depends on the parameter setting corresponding to the network environment (e.g., size, design and usage of network). One of our future works will be to investigate an automatic tuning method of an appropriate parameter setting.

In summary, there are still some limitations to more accurately estimate the number of NATted hosts in wild. However, the proposed algorithm likely estimate at least a lower-bound of the percentage of NATted hosts in the longitudinal traffic traces.

### B. Long-time trend of NAT usage

We showed that the average number of NATted hosts is relatively small and stable over years, thus the rapid and wide deployment of the NAT could not be confirmed over time at least in the transit link traces. One of the reasons might be the use of the global IPv4 addresses in universities with large address blocks. Such universities have less motivation to use NAT, unlike commercial ISPs with small address blocks. Also, the number of global IPv4 addresses assigned to hosts increased before the IPv4 address exhaustion was widely recognized. So, in near future, there is a possibility to observe more NATted hosts even in the transit link.

## VIII. Conclusions

The technique for estimating number of NATted hosts using IPID sequence information is an easy and powerful method. However, IPID sequence alone is not enough to give accurate results because some operating systems implement IPID as per-flow counter or random number. We propose a technique to improve accuracy of this method by incorporating other TCP information such as TCP sequence number and TCP source port. Grouping IPID sequences according to TCP sequences helps identify hosts with per-flow IPID (e.g., Linux 2.6). Moreover, patterns of TCP source port can help identify MacOS which implements random IPID. However, our method fails to detect OpenBSD hosts because they implement all of IPID, TCP sequence number and TCP source port in a random manner. Other limitations include failure to distinguish between NATted hosts and VPN hosts and failure to detect the case of port address translation, for example. Finally, we apply our algorithm to a collection of longitudinal traffic traces and find relatively low percentage of NAT usage.

## Acknowledgment

The authors would like to thank the NII Internship Program for providing financial support for this work.

## References

[1] S. M. Bellovin, "A Technique for Counting NATted Hosts," in *ACM SIGCOMM Internet Measurement Workshop (IMW2002)*, 2002, pp. 267–272.
[2] MAWI Traffic Archive. [Online]. Available: http://mawi.wide.ad.jp/
[3] M. Zalewski. Passive OS fingerprinting tool. [Online]. Available: http://lcamtuf.coredump.cx/p0f.shtml
[4] R. Beverly, "A Robust Classifier for Passive TCP/IP Fingerprinting," in *Passive and Active Measurement Conference (PAM2004)*, 2004.
[5] T. Kohno, A. Broido, and K. Claffy, "Remote Physical Device Fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.
[6] L. Rui, Z. Hongliang, X. Yang, L. Shoushan, Y. Yixian, and W. Cong, "Passive NATted Hosts Detect Algorithm Based on Directed Acyclic Graph Support Vector Machine," in *International Conference on Multimedia Information Networking and Security (MINES '09)*, 2009, pp. 474–477.
[7] G. Maier, F. Schneider, and A. Feldmann, "NAT usage in residential broadband networks," in *Passive and Active Measurement Conference (PAM2011)*, 2011, pp. 32–41.
[8] M. West and S. McCanne, "TCP/IP Field Behavior," RFC 4413, March 2006.
[9] K. Fukuda, "An Analysis of Longitudinal TCP Passive Measurements," in *Traffic Monitoring and Analysis Workshop (TMA2011)*, 2011, pp. 29–36.