A Technique for Counting NATted Hosts

Steven M. Bellovin
smb@research.att.com
AT&T Labs Research

Abstract— **There have been many attempts to measure how many hosts are on the Internet. Many of those endpoints, however, are NAT boxes (Network Address Translators), and actually represent several different computers. We describe a technique for detecting NATs and counting the number of active hosts behind them. The technique is based on the observation that on many operating systems, the IP header's ID field is a simple counter. By suitable processing of trace data, packets emanating from individual machines can be isolated, and the number of machines determined. Our implementation, tested on aggregated local trace data, demonstrates the feasibility (and limitations) of the scheme.**

I. Introduction

For a number of reasons, including the shortage of IPv4 addresses, many locations are connected to the Internet by means of *NAT* (Network Adress Translator) [1] boxes. NATs pose many challenges to protocols and to the Internet architecture [2], but they also make life difficult for people who want to count the number of hosts on the Internet. A NAT box will use a very small number of IP addresses—perhaps just one—but can act as a relay for many different hosts behind it. Until now, there has been no way to count how many distinct hosts are hidden behind NATs. We propose a technique that can count the number of hosts accessing the Internet from behind small NAT boxes (i.e., those that handle a small number of clients).

Our technique is based on the observation—well known in some circles, but apparently not documented anywhere—that the "id" field in the IP [3] header (hereinafter referred to as `IPid`) is generally implemented as a simple counter. (The same observation is used for different purposes in [4].) As a consequence, consecutive packets emitted by a host will carry sequential `IPid` fields. Strings of consecutive `IPid`'s represent strings of consecutive packets from a given host; by counting the number of strings coming from a given IP address, we can determine how many hosts are really represented by that address.

Naturally, life is not that simple, for quite a variety of reasons. The most obvious is that not all packets from a machine are sent to the Internet. Some may stay on a LAN; indeed, some are sent to the "loopback" address, and stay within the host. Our algorithm must therefore handle gaps and irregular spacing.

A second complication is that not all hosts use simple counters. Some use byte-swapped counters. That is, since there are no defined semantics to the `IPid` field, and hence no reason to make it a counter, some operating systems running on "little-endian" hardware don't bother putting the counter into "network" (big-endian) byte order.

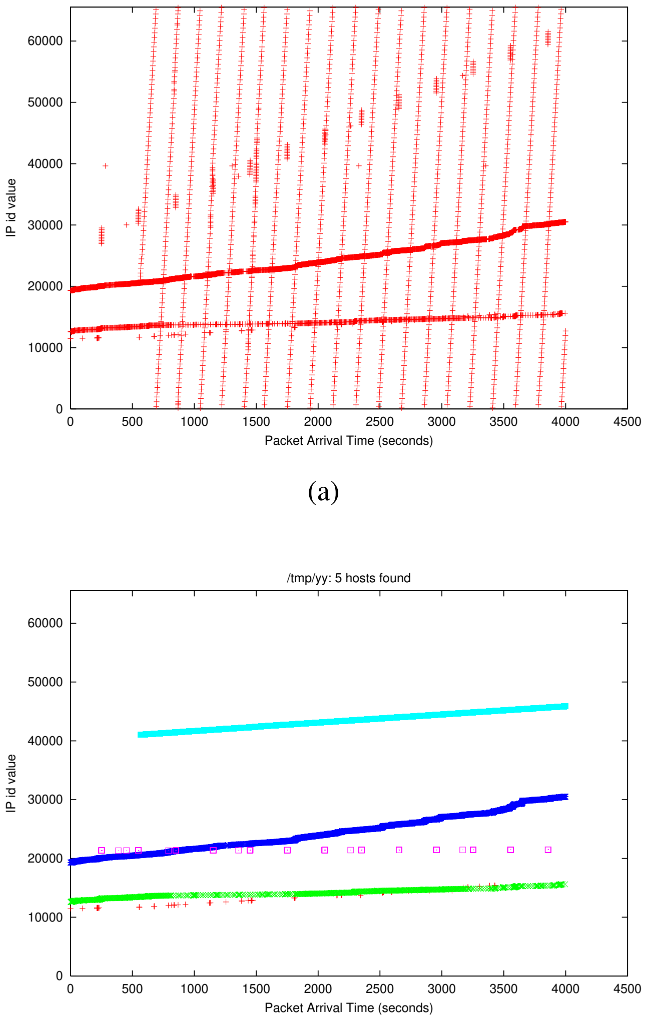
Some hosts take evasive measures. Since the `IPid` field is used only for fragment reassembly (see below), some Linux kernels use a constant 0 when emitting Path MTU discovery [5] packets, since they cannot be fragmented. Recent versions of OpenBSD and some versions of FreeBSD use a pseudo-random number generator for the `IPid` field. Some versions of Solaris use separate sequence number spaces for each ⟨source, destination, protocol⟩ triple, to avoid fragment collisions from busy hosts. All of these complicate (and to some extent block) the analysis.

These difficulties notwithstanding, we have implemented this scheme and tested it using synthetic NAT data (see Section III) derived from real packet traces. The full algorithm is described in Section II. Our observations and conclusions are described in Section III. How to block our analytic technique—which turns out to be the behavior required for correct functioning of NAT boxes—is described in Section IV. A brief survey of the behavior of some commercial devices is described in Section V. Future work is outlined in Section VI.

A. *Intended Uses of the `IPid` Field*

As noted, the intended purpose of the `IPid` field is for use in fragment reassembly. RFC 791 [3] describes its role as follows:

> The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an



(a)

(b)

Fig. 1. (a) A view of some raw data. (b) The same file after processing. All of the graphs show `IPid` as a function of time.

> internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

In other words, the `IPid` field is just a bit string used to make packets unique. There is no requirement that it be a counter. Furthermore, there are other fields that contribute to the uniqueness, one of which—the destination address—is likely to differ significantly in different streams.

If there is no fragmentation, there is no need for the `IPid` field. On the other hand, fragmentation can, in theory, happen anywhere. The `IPid` field must be unique in all "live" packets that have the same protocol number and source and destination IP addresses. If we assume that packets can survive for up to 10 seconds (which is long, but not preposterously so, for a worst-case check), and if we assume that the average packet is 150 bytes long, simple arithmetic shows that the maximum sending rate is limited to about 7.8M bps. By today's standards, this is not very fast. If we assume that packets being sent that fast represent bulk data transfer, packets may be 1500 bytes long. That bounds transmission speed to 78 Mbps, which is still quite reachable. (Two machines on the author's desk can communicate at that speed.) In other words, fragmentation during high-speed transmission may be problematic. Note, too, that this transmission rate is per ⟨source, destination, protocol⟩ triplet if and only if the `IPid` space is different for different triplets.

TABLE I

VALUES OF PARAMETERS USED IN THE CURRENT VERSION OF THE ALGORITHM.

Parameter	Value
timelim	300
gaplim	64
timefac	5
gapfac	70
fsize	50

II. ALGORITHM

Figure 1a shows a plot of a raw data file. Apart from our desire to produce a single scalar value—the number of hosts behind the NAT—it is clear that some post-processing is needed. The more-or-less horizontal lines are individual hosts; there appear to be at least two, possibly three or four. The near-vertical lines are hosts with byte-swapped `IPid` fields; there is at least one, and possibly two. Processing the data (Figure 1b) shows that there are five hosts, two of which used byte-swapped counters. There may have been a sixth host—the sequencing algorithm discarded 61 packets—but if so, it sent too few packets to be detected.

The basic algorithm is simple. We build up a set of `IPid` sequences. When a new packet (i.e., a new `IPid`) is received, we scan the set of sequences, seeing which one is the "best" match. If we find one, we append the new `IPid` to that sequence; otherwise, we create a new sequence of length 1.

We define "best" according to the following algorithm:

- If the new `IPid` is more than *timelim* seconds older or newer than the last received packet in the sequence, it is deemed not to match, regardless of the other criteria.

- If the new `IPid` is exactly one higher than the last received packet in a sequence, it is a *Perfect* match. (All comparisons are done modulo 2^{16}.)
- If the `IPid` is within *gaplim* of the last packet received, but has not been seen before, it is labeled *OutOfOrder*. We do not currently distinguish between higher-numbered `IPid`, which would indicate a small gap, and an `IPid` lower than the last one received, which indicates out-of-order reception.
- Finally, an `IPid` that is close enough but has been seen before is labeled *Dup*.

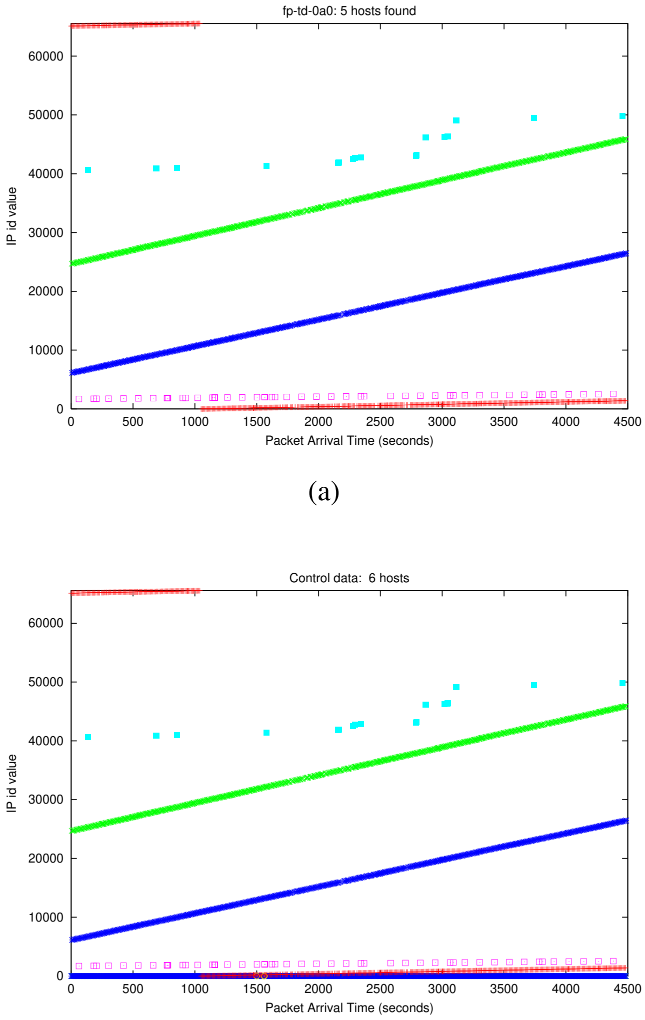
The parameter values shown in Table I were determined empirically, and may need to change for different environments.

After all of the sequences are collected, adjacent sequences may be *coalesced*. That is, sequences whose ends are close enough (defined as `IPid`'s being within *gapfac*·*gaplim* of each other, and received within *timefac*·*timelim* seconds of each other) are merged. This accounts for hosts that send a moderate amount of local traffic between sequences of Internet activity. We considered using average packet rate as an additional discriminator, but experience shows that that can vary too much.

Finally, sequences that are composed of less than *fsize* packets are simply discarded. While they may represent hosts that do not talk much to the Internet (and indeed, some of those showed up in our tests), they can also represent failed guesses at sequences.

Our algorithm must take into account some of the complications described in Section I. Packets with a 0 `IPid` are the easiest: they're dropped. We never let them match any real sequence; otherwise, they can cause confusion if they are temporally interspaced with a real sequence that is wrapping around modulo 2^{16}. (A future version might assign them to a special non-matching sequence; this would allow determination of their temporal length.)

Swapped-byte sequences are more troublesome. We maintain two different collections of sequences, one for normally-incrmented `IPid` fields, and one for byte-swapped `IPid` fields. When a new packet is received, we match it against both collections. It is added to the sequence it matches best, regardless of which collection the sequence is in. If it has an equal score on one sequence in each collection, or if it does not match any sequence, it is added to both collections. This is the primary cause of failed sequences. (The current implementation does not deal with close matches to two or more sequences in the same collection, nor does it use as a differentiator how close the `IPid` is to the expected value, save for the *Perfect* case described above.)



(a)

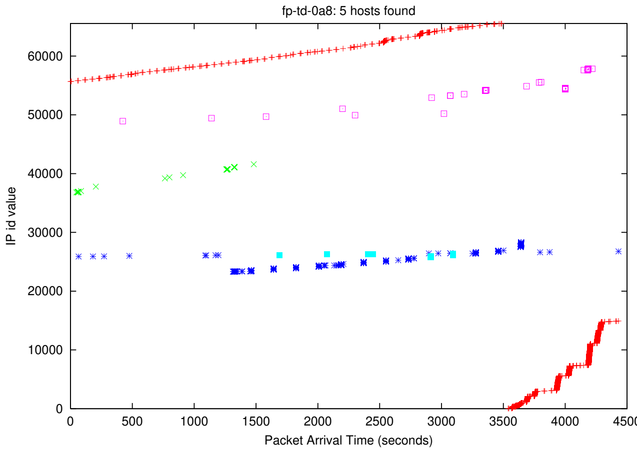
(b)

Fig. 2. Analytic and control graphs. (a) is the output from our algorithm; (b) is an `IPid` plot of the actual input data, based on IP address.

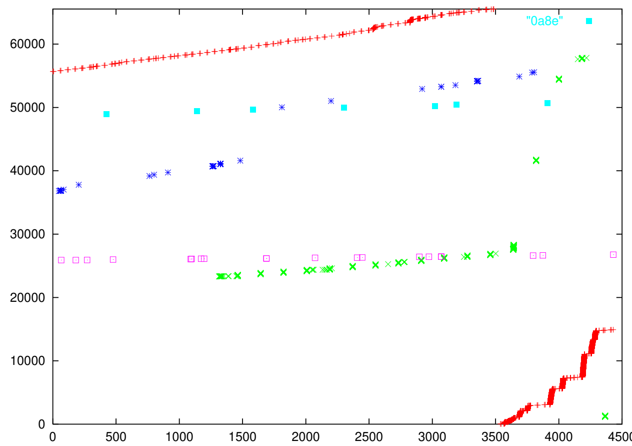
We do not currently attempt to deal with the randomized `IPid` generator used by OpenBSD and FreeBSD. Cryptanalyzing the generator may be infeasible in any event. It should be possible to detect a random background to other, linear sources; the current version of the code does not do that.

III. Observations and Limitations

We tested the program using "synthetic" NAT data. That is, we did not pursue the obvious course of obtaining packet header traces from locations likely to harbor NATs, such as the head end of cable modem systems or the local POP of a DSL provider. Apart from the difficulty of obtaining such traces—the monitoring point needs to be close



(a)



(b)

Fig. 3. An example of `IPid`-space collision. (a) is the analytic output; (b) is the real data stream.

to the outside of the NAT box, to avoid miscounts because too many packets are going in different directions—such data provides no easy way to validate the results. That is, we can build our algorithm and look at the resulting graphs and tables, but we would not know if we have miscounted. In fact, our early experiments showed that it was very easy to miss a lot of data.

Instead, we took real trace data and arbitrarily grouped together various sets of machines as if they were behind a NAT. That is, we analyzed the `IPid` sequence from a set of machines, without looking at the actual IP addresses, precisely as if they were all behind some NAT. (The particular subnet chosen was our organization's wireless LAN, since it had only client machines, rather than servers.) We then compared the results of our analysis to the actual IP address data. The results were very good: in no case were

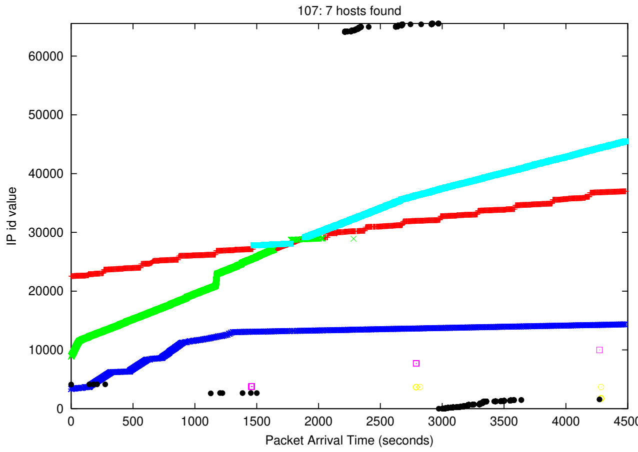


Fig. 4. Because of an `IPid` collision, two hosts are interpreted as three.

we off by more than one, and in each of those cases the omitted machines were very light users of the Internet, and fell beneath our threshholds.

The packet trace data was from the Internet edge of a facility housing several hundred researchers. This preserved the Intranet/Internet distinction; the monitor did not see any Intranet-bound packets. We collected only the first 16 bytes of each outbound IP header, omitting the destination IP address and higher-level headers. This preserved privacy: we learned nothing about any individual host other than the rate at which it sent data to the Internet. We did not learn the destination. We selected a client-only subnet—our algorithms and data structures probably cannot cope with the `IPid` change rate of a server, since there would be too many different destinations intermingled—and divided it up into smaller subnets of 16 addresses apiece. We then analyzed separately the data for each such subnet.

A sample pair of graphs is shown in Figure 2. It is quite clear that the two are almost identical. In fact, the only notable difference is the sequence of some 0 `IPid` packets in the control data. Investigation showed that those packets came from a router that used a 0 `IPid` for link-local OSPF packets, and a sequential `IPid` for TCP.

A second difference is that the analysis missed one host. Examination of the source data showed that the missing host sent only five packets, all with 0 `IPid`'s. (If you are viewing a soft copy of this paper, you can see the points on the control graph by zooming in on the X-axis at $t = 1500$.)

Collisions in `IPid` space can prove troublesome. Figure 3 shows the analytic and control graphs of another subnet. In the sequence starting with `IPid` of about 26000, a second host starts sending at about 1300 seconds into the trace. The analysis missed that, and assumed that those packets were from the first host. It did recognize the second host at about $t = 1600$, but mismatched both streams until the second host went silent at $t = 3750$.

A more serious collision is shown in Figure 4, where what is obviously (and actually) two hosts are detected as three.

In general, large numbers of packets (or even of hosts) do not confuse the analysis engine. A test on 66,563 packets from 26 hosts produced a plausible (though not absolutely correct) graph, with errors mostly from `IPid` collisions. But large gaps in the space (presumably caused by Intranet activity) cause considerable trouble. In one case, activity from a busy file server produced a random-looking distribution, enough so that we analyzed its behavior at a quiet time of day.

We conclude that this technique is primarily suitable for analyzing NATs serving networks with comparatively little Intranet traffic. This would describe most home NATs and virtually all hotel NATs. It does not describe the majority of businesses, which tend to run their own file and email servers.

IV. Privacy and Correctness Issues

Many organizations (and individuals) do not wish any disclosure of the number of machines behind their NATs. A properly designed NAT can block information leakage; however, a great deal of care is necessary to avoid violating the defined semantics of the `IPid` field and IP fragment reassembly. There are three possible cases.

In all packets emitted by the NAT, the source IP address will be changed to either a single value or a value drawn from a very small set. Furthermore, most packets use TCP. This means that uniqueness must be maintained at least on a per-destination address basis, if fragmentation is possible.

The simplest case, though, is if the *DF* (Don't Fragment) bit is set in the IP header of the incoming packet. This is, in fact, reasonably common, since the DF flag is used for Path MTU Discovery [5]. In such situations, the NAT box can rewrite the `IPid` field freely, since there will never be any reassembly. Setting it to 0, as Linux does, is one possibility; as discussed below, in a NAT situation this can leak information, and hence is probably undesirable.

The second case is ordinary packets with DF not set. Such packets may be fragmented by any router along the path to the destination, and hence must carry unique `IPid` fields. Accordingly, a NAT should rewrite all such `IPid` fields, to ensure uniqueness. (Setting the DF bit is generally inadvisable, since that would place the burden of Path MTU Discovery on the NAT.)

Some hosts never use Path MTU Discovery; some use it only for TCP. A NAT that treated DF packets differently than non-DF packets for the same protocol would thus leak the fact that at least two different policies exist behind it. Therefore, to preserve privacy the NAT should do the same thing—send a unique `IPid` field—on all packets.

The third case, though, is more complex, and puts an extra burden on NATs: where already-fragmented packets arrive at the NAT, it must preserve uniqueness while also ensuring that all fragments of the same datagram carry the same new `IPid` field. (This is noted as a problem in [1], [6], though those documents do not suggest any solution.) Doing so requires that it keep state for each fragment of each packet, until it determines that it has sent all fragments of it. As [7] notes, it is unclear how often this is done.

Using a random `IPid`field has its own challenges to uniqueness. While linear congruential generators have a maximal cycle length, such generators are easily cryptanalyzed [8], [9]. A keyed generator, as is used in OpenBSD and FreeBSD, provides some protection, but one needs to be careful to avoid duplication if the generator is rekeyed periodically.

V. Behavior of Some Commercial NAT Devices

We performed some tests on a number of commercial home NAT devices. In addition, we used the NAT component of a version of Darren Reed's IP Filter package.

We tried sending small and large (i.e., intentionally fragmented) ICMP "ping" packets, and TCP packets with and without the "Don't Fragment" bit set. We then monitored the packets on both sides of the NAT. The results were the same in all cases: the `IPid` field was not rewritten, the correctness arguments discussed in Section IV notwithstanding.

A further test would be to deliberately generate `IPid` conflicts. This could be done by using a special system that would watch for other machines' outbound packets, and deliberately emit its own packets with the same `IPid`. We have not yet done that.

VI. Future Work

There are two major paths for future work: improving the techniques described in this paper, and using the mechanisms to analyze the prevalance of NATs.

There is little more that can be said at this time about the latter. Doing it depends on the availability of suitable data. As noted, the scheme works best for smaller NATs, which suggest that it is best applied to data collected from ISPs providing broadband access to home and small business users.

There are a number of clear directions for extending this work. The first is better sequence detection algorithms. There is at least a superficial similarity to some image processing problems; it may be possible to adapt algorithms from that field.

Another tack would be to use other packet header information to improve the analysis. For example, the TCP 4-tuple (source address, source port, destination address, destination port) strongly identifies a connection (by definition!); such packets can be linked into a sequence without inspection of the `IPid` field. Among other things, this can help disambiguate `IPid` space collisions, as shown in Figure 3.

Other protocols may contain similar sequence information. IPsec [10] packets generally carry sequence numbers; RTP [11] packets contain timestamps, etc.

Finally—and most intrusively—passive fingerprinting [12] can be used to detect types of hosts behind a NAT. The scheme may not be able to tell how many hosts of each type exist, but it should be able to tell what types of hosts are there.

VII. Acknowledgments

Steve North suggested the scatter plot as a visualization technique. The original idea for the detection algorithm came from John Denker. Randy Bush made a number of useful comments on the paper itself.

References

[1] P. Srisuresh and K. Egevang, "Traditional IP network address translator (traditional NAT)," RFC 3022, Internet Engineering Task Force, Jan. 2001.

[2] T. Hain, "Architectural implications of NAT," RFC 2993, Internet Engineering Task Force, Nov. 2000.

[3] J. Postel, "Internet protocol," RFC 791, Internet Engineering Task Force, Sept. 1981.

[4] Ratul Mahajan, Neil T. Spring, and David Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proceedings of SIGCOMM 2002*, 2002, to appear.

[5] J. C. Mogul and S. E. Deering, "Path MTU discovery," RFC 1191, Internet Engineering Task Force, Nov. 1990.

[6] M. Holdrege and P. Srisuresh, "Protocol complications with the IP network address translator," RFC 3027, Internet Engineering Task Force, Jan. 2001.

[7] D. Senie, "Network address translator (nat)-friendly application design guidelines," RFC 3235, Internet Engineering Task Force, Jan. 2002.

[8] Jim Reeds, ""Cracking" a random number generator," *Cryptologia*, vol. 1, no. 1, January 1977.

[9] Jacques Stern, "Secret linear congruential generators are not cryptographically secure," in *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1987.

[10] S. Kent and R. Atkinson, "Security architecture for the internet protocol," RFC 2401, Internet Engineering Task Force, Nov. 1998.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," RFC 1889, Internet Engineering Task Force, Jan. 1996.

[12] Honeynet Project, "Know your enemy: Passive fingerprinting," March 2002, `http://project.honeynet.org/papers/finger`.