Efficient Routing using *NIx–Vectors*

George F. Riley, Mostafa H. Ammar, Ellen W. Zegura
College of Computing Georgia Institute of Technology Atlanta, GA 30332
{riley,ammar,ewz}@cc.gatech.edu

Abstract

We introduce the concept of NIx–Vector *routing which gives efficient and consistent routing of packets in Internet routers. A* NIx–Vector *is a compact representation of a routing path, which is small enough to be included in a packet header. We show how, by including the routing information in the packet header, routing decisions can be made in* $O(1)$ *time at each router, with no caching or state in the routers (other than the existing routing tables). The creation of a* NIx–Vector *for a source and destination pair requires one round trip time, but once the* NIx–Vector *is known, it can be retained by the source and reused indefinitely with no further setup time required (or until it becomes no longer valid, which is easily detected).*

In addition to $O(1)$ *routing decisions, the use of* NIx–Vectors *to record and specify routes has other benefits.* NIx–Vectors *provide for* route pinning, *which is beneficial for reservation protocols and mitigates the effect of routing flaps on long lived flows. A variation of* NIx–Vectors *can insure symmetrical routes from a source to a destination and back to the source, which is also beneficial to some reservation protocols.*

We give a complete description of how NIx–Vectors *are created and used in Internet routers, and give empirical data showing the number of bits required to represent* NIx–Vectors *for typical Internet paths.*

1 Introduction

The routing of packets through the Internet consists of a series of independent routing decisions made at each router along the path between any source and destination. When many packets travel between the same source and destination pair, the same routing decisions are made repeatedly and independently, without knowledge of previous decisions. With this in mind, it has long been recognized that including the desired path in packet headers can reduce the burden on routers. For example, the specification for the Internet Protocol in RFC791[1] allows for the *Strict Source Routing* option, which includes a list of IP addresses in packet headers to indicate the route each packet should take. However, the functionality of this routing option is severely limited given the bounded size of the options field available in the Internet Protocol. In fact, as specified in RFC791, only nine forwarding addresses can be specified, which clearly is not sufficient for today's Internet routing paths[2]. The basis of our proposed *NIx–Vector* routing (which is short for *Neighbor–Index Vector*), is the observation that the specification of the IP address of each next hop in a routing path is an overkill of information. With our method, the same information can be specified in just a few bits per hop, and thus much longer routing paths can be specified in the same amount of space. *NIx–Vectors* make source-specified routing a feasible option within the Internet context, in which flows typically have paths much longer than nine hops. In addition, having a feasible, low-overhead source routing mechanism, such as *NIx–Vectors*, allows for the incorporation of important functionality within the Internet. This includes pinning of routes for Quality of Service (QOS) related protocols, and packet tracing for determining the source of denial of service attacks.

The term *NIx–Vector* is short for *Neighbor–Index Vector*. Routers choose the next hop for packets from an ordered set of *routing neighbors*, and the ordinal of a given neighbor is the *neighbor–index*, or *NIx*. For example, if a router has three immediate routing neighbors, these would be numbered 0, 1, and 2. The concatenation of all *NIx* values chosen on the path from a source to a destination constitutes the *NIx–Vector*.

NIx–Vector routing consists of two distinct phases. First a *NIx–Vector* is created as a packet flows from a source to a destination. During the creation phase, routers make normal routing decisions on packets, and record those decisions in the packet

header. After creation, the *NIx–Vector* is included in future packets from that source to the same destination, and is used by the routers to efficiently route the packets. We expect that the creation of *NIx–Vectors* will be somewhat infrequent, with the majority of packets using previously created and retained *NIx–Vectors*.

To create a *NIx–Vector*, each router along a routing path (including the source) simply appends the neighbor–index value of the next hop routing neighbor to the *NIx–Vector* that was received. The next hop routing neighbor for any given packet is determined by whatever routing mechanism already in place at each router (ie. normal destination lookup). The number of bits appended to the *NIx–Vector* at a given hop is exactly the number of bits needed to represent the largest neighbor index possible at that hop. It is important to note that the number of bits needed is not a function of packet content or length, but rather is a function of the number of next-hop neighbors defined at each router. Thus the number of bits needed to record the neighbor–index at each router is a constant (for that router) and is expected to be reasonably small and remain somewhat static over time. For routers that expect to be adding next-hop neighbors over time, simply using some extra bits will allow for a multiplicative increase in the number of routing neighbors that can be recorded. For example, if a router presently has 12 routing neighbors, but expects that up to 63 neighbors may be present in the future, that router should record 6 bits rather than the 4 needed to record the existing neighbors.

Figure 1 shows a simple network as an example. Each of the routers have varying numbers of neighbors, and each router numbers those neighbors sequentially as shown. Suppose the path for a packet is 0–1–3–6–9–12. The resulting *NIx–Vector* for this path is `0 10 10 011 1`. Note that the spaces shown in the 9 bit value above are for ease of reading only, the actual *NIx–Vector* is `010100111`.

Once a *NIx–Vector* from a source to a destination is created and becomes known to the source, then it can be subsequently used for efficient $O(1)$ routing decisions. Each router (and the original source) extracts the appropriate number of bits from the received *NIx–Vector*, which specifies the correct neighbor–index for the next hop. This neighbor–index is used to index a table of next hop IP address, interface number, and (optionally) layer 2 address. The packet is then forwarded to this neighbor with no further processing. As previously mentioned, the number of bits extracted is a constant at each router, and will be the same as the number of bits recorded when the *NIx–Vector* was created. Returning to our example, the extracted *NIx* values at each hop between 0 and 12 would be 0, 2, 2, 3, and 1; resulting in the correct path 0–1–3–6–9–12. To specify the same information using *Strict Source Routing*, the IP address of each next hop must be specified in the packet header. In this example, the packet header would contain the IP address list 1–3–6–9–12, which is 160 bits of information.

The remainder of this paper is organized as follows. In section 2 we give a detailed description of *NIx–Vectors*with a detailed example. In section 3 we describe how we estimated the length of the *NIx–Vectors* for typical routing paths in the Internet. Finally, in section 4 we give some conclusions and discuss future directions of our work.

2 *NIx–Vectors*

Now we describe more formally the methodology for describing, creating, and using *NIx–Vectors* for routing decisions. We start with the following definitions.

The path of a packet from a source S to a destination D consists of a series of *routing decision points*, which we denote P_i. For a path that consists of v hops, there are $v+1$ decision points; the source is P_0, the first hop router is P_1, the last hop router before the destination is P_{v-1}, and the destination is P_v. While there is no routing decision needed at a destination, we include it in the definition for completeness.

At each P_i, the next hop for a packet is selected from an ordered set of *routing neighbors*, which we denote $\mathbf{N}_i$. The *routing neighbor count* is the cardinality of set $\mathbf{N}_i$, which we denote C_i. The individual members of set $\mathbf{N}_i$ are denoted $\mathrm{N}_{i,j}$, where $0 \leq j < \mathrm{C}_i$. The position j of neighbor $\mathrm{N}_{i,j}$ in set $\mathbf{N}_i$ is called the *neighbor–index* or *NIx*. The number of bits needed to represent a neighbor–index at P_i is $\lceil log_2 \mathrm{C}_i \rceil$, which we denote the *neighbor index bit count*, or B_i. If we consider the network to be a directed graph of vertices and edges, the set $\mathbf{N}_i$ can be considered to be the set of all vertices for which an edge exists from vertex i, and C_i is the outdegree of vertex i. Finally, a *NIx–Vector* is a sequence of n bits, where $n = \sum_{i=0}^{v-1} \mathrm{B}_i$. The first B_0 bits indicate the neighbor chosen at P_0, the next B_1 bits indicate the neighbor chosen at P_1, and so

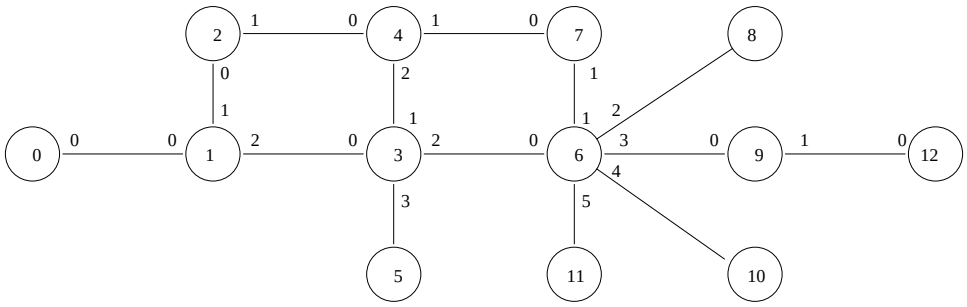

Figure 1. Simple Routing Path

on. Finally, we define *NIx–Vector used* (*NVU*) to be actual length of a *NIx–Vector* at any point in time and *NIx–Vector length* (*NVL*) to be the maximum allowable value that *NVU* can attain. The semantics for *NVU* and *NVL* vary slightly depending on whether a *NIx–Vector* is being created or used for routing, as described below.

With these definitions, we can now describe how *NIx–Vectors* are created and used for routing decisions.

2.1 Creating the *NIx–Vector*

At each decision point, the router (and the original source) will determine B_i, the number of bits that will be needed to record the neighbor–index for a packet to be routed. This value is a constant at each decision point, and not dependent on any information in the packet. The B_i value can be administratively chosen larger than actually needed to allow for multiplicative growth in the number of routing neighbors over time. The router then chooses the appropriate neighbor–index for routing this packet, by using normal packet routing methods. It records the selected neighbor–index by appending B_i bits to the *NIx–Vector* in the packet header and advancing *NVU* by B_i. It then forwards the packet to this neighbor normally. Thus when the packet reaches the final destination, the complete routing history of the packet is recorded in the *NIx–Vector*, using the number of bits specified by *NVU*. In [3] we outline our proposed Record–NIx–Vector IPv4 option which allows for 254 bits in the *NIx–Vector*. We show later that this is sufficient for routes between most source and destination pairs.

We illustrate the creation of a *NIx–Vector* with an example. Suppose that a total of 9 hops are used between a given source and destination pair, as shown in Table 1. In the table, the column labeled C_i is the number of neighbors available at each hop, the column labeled B_i is $\lceil log_2 C_i \rceil$, which is the number of bits needed to retain the routing decision, and the column labeled *NIx* is the neighbor neighbor–index selected for routing the packet. The *NVU* and *NIx–Vector* columns show the used counter and the *NIx–Vector after* the routing decision has been recorded at each hop. In this example, the routing path for a 9 hop route is logged using only 27 bits.

2.2 Routing with the *NIx–Vector*

Once a *NIx–Vector* from a source S to a destination D has been determined, and the *NIx–Vector* has been returned to S, then S can simply use that *NIx–Vector* in any future packet to be sent to D. The *NIx–Vector* is copied to the packet header, the *NIx–Vector length* (*NVL*) is set to the *NIx–Vector used* (*NVU*) value at the final destination when the *NIx–Vector* was created, and the new *NVU* is set to zero.

At each routing decision point, when a packet with a *NIx–Vector* is received, the router simply extracts the next B_i bits from the *NIx–Vector* and advances the *NVU* value by that number of bits. The extracted value is the neighbor–index recorded when the *NIx–Vector* was created, and the packet can be unconditionally forwarded to that neighbor. As long as the route from S to D that was recorded when the *NIx–Vector* was created remains valid, then all packets from S with the same *NIx–Vector* will follow exactly the same path to D as the original *NIx–Vector* creation packet.

When *NIx–Vectors* are used, routers can forward packets in constant time, regardless of the size of the routing tables. All of the operations listed above for router actions are simple addition, bit extraction and table indexing, with no looping or searching

Hop	C_i	B_i	*NIx*	*NVU*	*NIx–Vector* (binary)
0	2	1	1	1	1
1	3	2	2	3	1 10
2	6	3	5	6	1 10 101
3	13	4	4	10	1 10 101 0100
4	27	5	20	15	1 10 101 0100 10100
5	5	3	3	18	1 10 101 0100 10100 011
6	14	4	7	22	1 10 101 0100 10100 011 0111
7	5	3	1	25	1 10 101 0100 10100 011 0111 101
8	2	1	0	26	1 10 101 0100 10100 011 0111 101 0
9	1	1	0	27	1 10 101 0100 10100 011 0111 101 0 0

Table 1. *NIx–Vector* **Creation Example**

needed. This discussion assumes full deployment of *NIx–Vector* capabilities on all routers, which of course is not reasonable. A discussion of incremental deployment is given in [3]. Also in [3] is a discussion of an end-to-end protocol for the creation and management of *NIx–Vectors*, and a proposed addition to the *sockets API* to allow application control over *NIx–Vector* usage.

3 Estimating the Length of *NIx–Vectors*

It is clear that we need a good estimate of how many bits will be needed to store *NIx–Vectors* in real Internet routing paths. We set out to answer this question in the following way.

1. Obtain *traceroute* information for a large number of destinations, with hop by hop addresses of the routing decision points.
2. Determine the number of routing neighbors for each node found. Whenever the traceroute shows a hop from router A to router B, note that B is a routing neighbor of A, and A is a routing neighbor of B.
3. Compute the number of bits needed for the neighbor–index at each node found.
4. Follow each route in the traceroute list to compute the length of the *NIx–Vector* and the total hop count for each.
5. Compute the average and maximum *NIx–Vector* length.
6. Prepare a histogram of *NIx–Vector* length distribution.
7. Prepare a histogram of the hop count distribution.
8. Prepare a histogram of the number of routing neighbors distribution.

We are grateful to Cheswich and Burch [4] for making available their data from the *Internet Mapping Project*. At the time we obtained it, their database contained a total of 308,807 traceroutes, finding a total of 96,586 distinct nodes.

With this data, we were able to compute the statistics outlined above. Figure 2 shows the neighbor count distribution, and Figure 3 shows a magnified version of the same information with only the larger values shown. Almost all have a neighbor count less than 20, with a few in the range of 20 to 80, and a few outliers with larger counts. The largest neighbor count we found was 151, at router 204.70.9.138, corerouter1.westorange.cw.net. The average neighbor count was 3.98 (not including the leaf nodes with only one neighbor). Figure 4 shows the hop count distribution. This shows the majority of routes have a hop count in the range of 9 to 29, with an average of 16.68 hops. Finally Figure 5 shows a distribution of the *NIx–Vector* lengths. The results show almost all *NIx–Vectors* that we measured can be represented with less than 150 bits, with a few outliers up to 158, and one at 178. The average *NIx–Vector* length was 82.87. This is well within the maximum *NIx–Vector* length of 254 in our proposed IPv4 option.

We acknowledge and address the following potential shortcomings of this measurement approach.

1. We have no guarantee that the 96,586 nodes sampled is a representative sample of the com-

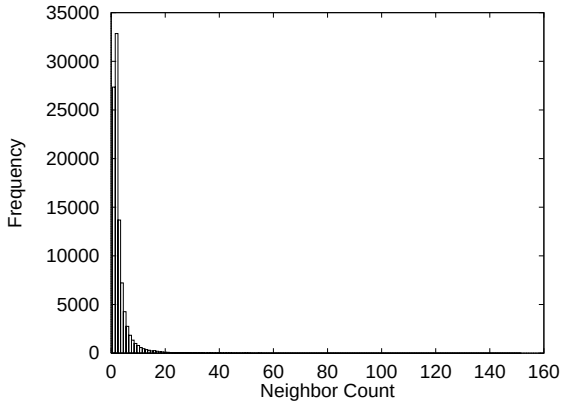

Figure 2. Neighbor Count Distribution

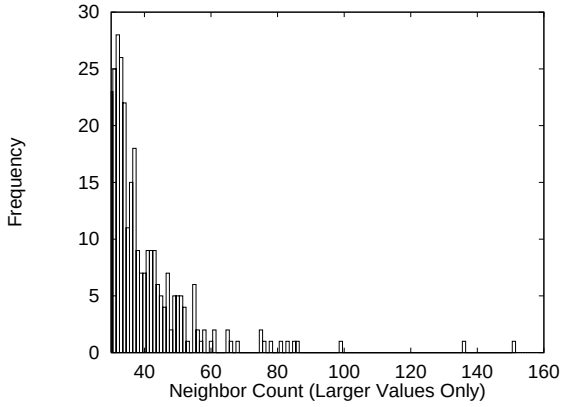

Figure 3. Neighbor Count Distribution (Larger Values Only)

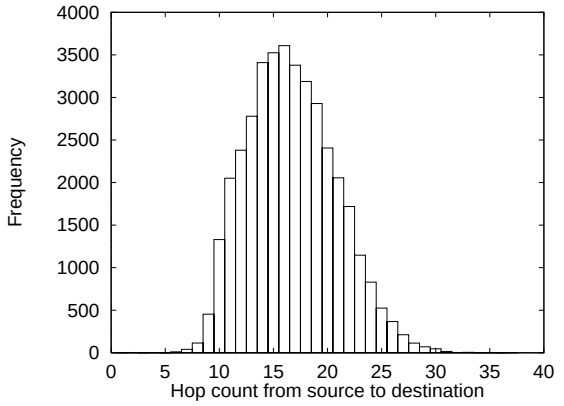

Figure 4. Hop Count Distribution

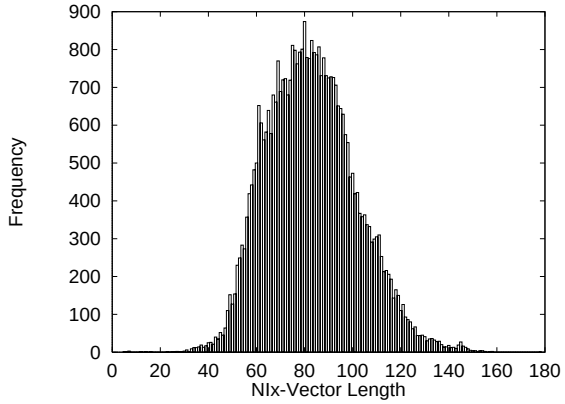

Figure 5. *NIx–Vector* **Length Distribution**

plete Internet, and in fact is only a small fraction of the total number of systems. While this is certainly true, Theilmann[2] has shown, using a similar method but using different sources and destinations, that the average hop count for Internet routing paths is 16 hops, which very closely matches our observations.

2. The routes discovered were all initiated from the same location, and thus may not be representative of routes from other locations. Almeroth[5] has shown that the random placement of a source for Internet traffic has little affect on the distance from a random destination. Also, again referring to Theilmann as above, the strong similarity between the hop count measurements made in [2] (measurements taken from a European location) and our observations indicates that the source location of the measurements has little affect.

3. It is unlikely that we discovered every Internet router, even with this large number of traces. While this is certainly true, the size of a *NIx–Vector* is dominated primarily by the number of routers in the path with a large number of routing neighbors. We argue that routers with large number of routing neighbors are more likely to be discovered by our approach.

4. It is unlikely that we discovered every neighbor of each router found. To adjust for this uncertainty, and to give a generous estimate of *NIx–Vector* lengths, we doubled the observed count of routing neighbors when computing the *NIx–Vector* size for the various routing paths. While we still can't be sure that doubling the count is gives a reasonable bound on the actual number of neighbors, we point

out that the neighbor–index technique allows for an exponential growth of the number of routing neighbors for a linear growth in the number of bits used. Thus, even if one of our measurements is off by an order of magnitude in the count of neighbors at a given router, it can be accounted for with just three or four more bits at that hop.

5. This data represents only a snapshot of the topology information in an environment that is changing over time. Faloutsos[6] gives an empirical analysis of the growth of the Internet topology. In that work, the *Effective Diameter* of a network is defined as bound on the maximum hop count between any two nodes (with high probability). Empirical measurements are given that suggest the effective diameter of the Internet is in fact growing, but at a relatively show rate. Since the average hop count between any two nodes is a factor in determining the size of the *NIx–Vector*, this work suggests that average *NIx–Vector* length will grow at a much slower rate than the overall size of the Internet.

We are confident that *NIx–Vectors* are a viable approach for efficient routing. When using an IPv4 option to carry the *NIx–Vector* information as we propose in [3], our data shows that there at least 100 bits left over in almost all of the routes we analyzed, and about 175 left over in the average case. Our data would have to be off by a very large margin for *NIx–Vectors* to overflow the alloted space in an IPv4 option on the routes we analyzed. For routes that we did not analyze, there would have to be much longer hop counts, and they would have to traverse routers with large neighbor counts to cause the *NIx–Vector* to overflow.

4 Conclusions and Future Work

A *NIx–Vector* is an extremely compact representation of a complete Internet routing path. Empirical measurements show that paths can be recorded in an average of about 83 bits, with a maximum of about 180 bits. Including the extra overhead for an IPv4 option, the total packet header overhead required for *NIx–Vector* routing is 147 bits on average. Assuming an average packet size of 12,000 bits, this overhead amounts to only about 1.3%.

Using the *NIx–Vector* representation for routing paths and including this information in a packet header has several advantages. Routers can make routing decisions in O(1) time, regardless of the size of the routing table. Sources can be guaranteed consistent route to a given destination, and can request that a symmetrical return route be used. Packet reordering due to redundant links between routing neighbors is eliminated.

For future work, we plan to implement the end to end protocol for creating and managing *NIx–Vectors* into the Linux protocol stack, to demonstrate the viability of our approach. We will also implement the processing of the various Internet Protocol Options proposed into the Linux routing daemon to test the overall performance of our approach. Finally, we plan to explore a method probabilistic packet marking technique using *NIx–Vectors* as a methodology for path reconstruction; which will allow for tracing of packets used during denial of service security attacks.

References

[1] J. Postel, "Internet RFC791: Internet protocol." Network Working Group, Sep 1981.

[2] W. Theilmann and K. Rothermel, "Dynamic distance maps of the internet," in *Proceedings of IEEE Infocom 2000*, March 2000.

[3] G. F. Riley, E. W. Zegura, and M. H. Ammar, "Efficient routing using nix-vectors (long version)," Mar 2000. Technical Report GIT-CC-00-27.

[4] B. Cheswick and H. Burch, "The Internet mapping project." http://cm.bell-labs.com/who/ches/map/index.html, 1999. Bell Labs and Carnegie Mellon University.

[5] K. Almeroth and M. Ammar, "Multicast group behavior in the internet's multicast backbone," in *IEEE Communications*, June 1997.

[6] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of the ACM SIGCOMM*, 1999.