Design, Implementation, and Testing of a Hybrid Tool for Network Topology Discovery

Alessio Botta, Walter de Donato, Antonio Pescapé, and Giorgio Ventre
University of Napoli "Federico II", Italy
{a.botta,walter.dedonato,pescape,giorgio}@unina.it

Abstract

In this Technical Report we report our activities on the Design, Implementation, and Testing of a Hybrid Tool for Network Topology Discovery we called *Hynetd*.

I. Introduction

Computer networks are becoming ever more ubiquitous and, as a consequence, more and more complex. The knowledge of the topology of a network allows to improve its management and to execute more accurate simulations. In the first case, substantial advantages in fault management, performance analysis and service allocation are obtainable. In the second one, there is a benefit because of the automatic generation of realistic topologies is a difficult task [1]. In addition, due to dynamic behavior and large size of real network topologies, the discovery process has to be necessarily performed in an automatic fashion and it should supply complete and correct results as soon as possible, generating as few traffic as possible.

In the past years, several techniques and tools have been proposed. We have classified them in *Active*, *Passive*, and *Hybrid*. The first is based on tools such as Ping and Traceroute, inferring topology information from network behavior, and the second one uses SNMP and to obtain information from devices. Using active methodology introduces two additional problems: (i) recognizing the interfaces belonging to the same network device, problem known as *alias resolution*; (ii) reconstructing correct subnet ids and net-masks. Finally, we refer as *Hybrid* a tool that uses both methodologies.

In our previous work [2] we proposed a preliminary version of an algorithm implemented in *Hynetd* (v. 0.1), a hybrid tool that assumes minimum prerequisites on the network (i.e. one or more address ranges to be scanned). In this technical report we present the new *Hynetd* version (v. 0.2) having a new and a more efficient architecture and containing some techniques to improve discovery efficiency and performance. More precisely, we introduce new approaches and modified the first version in order to: (i) perform the steps followed by the algorithm in a parallel fashion; (ii) reduce the redundancy of the information collected and, therefore, (iii) to reduce the traffic overhead; (iv) improve the alias resolution phase; (v) increase the accuracy of the link reconstruction phase. To demonstrate the improved performance of the novel approach, we provide results of a careful comparison with both the first version and with NetworkView 3.5 [25], a commercial topology discovery software. The experimental evaluation

has been carried out in terms of accuracy, traffic overhead, and discovery time over two network scenarios (small and large scale topologies).

The rest of the paper is organized as follows. Section 2 presents a review of the most relevant works about topology discovery. In Section 3 the design of *Hynetd* 0.2 is shown. Section 4 explains its architecture, innovations, and improvements. In Section 5 we present small and large scale experimental analysis. Finally, Section 6 ends the paper with conclusions and issues for research.

II. Reviewing the literature

Since 1993 many works dealing with Topology Discovery have been published. They differ in terms of methodology used, number of employed probes, prerequisites and explored protocol layers. In this section we revise such works according to the classification of the methodologies we proposed in Section I.

First, we report the works using an active methodology.

The authors of [3] propose an algorithm named *Ined* that uses Ping and Traceroute from a single source point. They introduce, for the first time, two techniques to resolve the *alias resolution* problem which are based on DNS inverse look-up and the source address of ICMP port unreachable packets (known as Source Address technique) respectively.

The Skitter project [7] faces the problem of Internet topology discovery at two different layers, that are IP and Autonomous Systems (AS). One of its components, namely *iffinder*,performs the alias resolution by using a combination of the Source Address and the Ping with Record Route option (PingRR in the following). Moreover this is the only project supporting the Traceroute IP option [8], even if it is not widely supported by operational devices.

Mercator project [11] tries to discover the Internet topology using active only methodologies. It is the first to exploit source routing in order to increase the number of discovered interfaces and to reveal transversal links.

Barford et al. in [12], by using the Traceroute tool, study the utility of adding information sources when performing wide-area measurements in the context of Internet topology discovery. They show that the utility of additional measurement sites rapidly declines even after the first two sites.

The Rocketfuel project [15] is aimed to discover ISP topologies and presents a centralized architecture that employs many probes. It introduces a new technique to resolve aliases, named Ally algorithm, that exploits ‘id’ field of IP header extracted from ICMP port unreachable error packets.

In [17] Magoni and Hoerdt present Nec (Network Cartographer) whose aim is to map the heart of the Internet as fast as possible with the highest attainable accuracy. Their approach focuses on routers and layer 3 links and introduces some heuristics to minimize the number of IP address pairs involved in alias resolution.

In [19] Gunes e Sarac, after an analysis of all alias resolution techniques proposed in literature, introduce a new method that exploits the partial symmetry of routes from a source to a destination. This approach does not generate additional traffic overhead because it only uses Traceroute discovered paths.

Traceroute@home [16] introduces the *Doubletree* algorithm whose efficiency has been proved to reduce traffic

overhead generated by many probes using Traceroute toward many destinations. To the best of our knowledge it is the first project released under GPL license and freely available.

Second, we analyze the works using a passive methodology.

Mansfield et al. [5] propose a framework aimed to map Internet topology using only SNMP. This approach has been proved to be not effective because of limited access permissions in a wide and heterogeneous network such as Internet.

Bejerano et al. [9], [10] propose several algorithms exploiting SNMP to infer topologies at both layers 2 and 3. These algorithms have been partially implemented in BindView's NETinventory software. Their approach is very efficient, but practically usable only in single administrative domains.

Remos [13] is an architecture useful to provide shared resource information to distributed applications. To retrieve data from different networks and hosts, it utilizes several collectors using different technologies, such as SNMP or benchmarking. Anyway, Remos is tailored to distributed applications, such for example grid computing, therefore it can not be used for general purpose network topology discovery.

Nazir et al. [18] propose an SNMP based algorithm that tries to overcome the limitations related to the support of such protocol on network devices. It also features a visualization module that progressively shows the resulting topology during the discovery process.

Third, a review of the works using a hybrid methodology follows.

The Fremont system [4] is the first example of hybrid architecture, composed of several discovery modules each of them exploiting different protocols and information sources.

In [6], Kashav et al. introduce a base algorithm, then specialized using the combination of different techniques such as SNMP, routing information and Traceroute. They also present some heuristic methods to obtain subnet associated net-masks the most useful of which is named *subnet guessing from a cluster of addresses*.

As for proprietary solutions, SNMP-based tools for automatic discovery of network topology are included in many commercial network management systems (i.e. HPs OpenView [20], IBMs Tivoli [21] and RocketSoftware NetCure [22]). These tools assume that SNMP is widely deployed, but they also send ICMP messages toward not SNMP-capable hosts and routers. Details of their discovery algorithms are proprietary and not available to the authors of this work. Among these tools, in order to perform a comparison with Hynetd, we have selected NetworkView [25] because there is a 30 days trial version available for download. It is a tool which discovers both network and application layer topologies exploiting ICMP,SNMP, NetBIOS e TCP (port scanning). Moreover it shows results, in graphical format, during the scanning process.

Our approach is aimed at maintaining minimal prerequisites, being usable on every IP based network, and exploiting all the available information sources to discover the topology. For this purpose, *Hynetd* architecture has been conceived as hybrid and multi-threaded.

Hynetd introduces some innovations: (i) an algorithm named Backtrace, that efficiently implements the execution of many concurrent Traceroutes; (ii) a novel approach for IP alias resolution using Ping with Record Route option; (iii) some rules that reduce the number of IP addresses pairs involved in the Ally algorithm and (iv) an heuristic

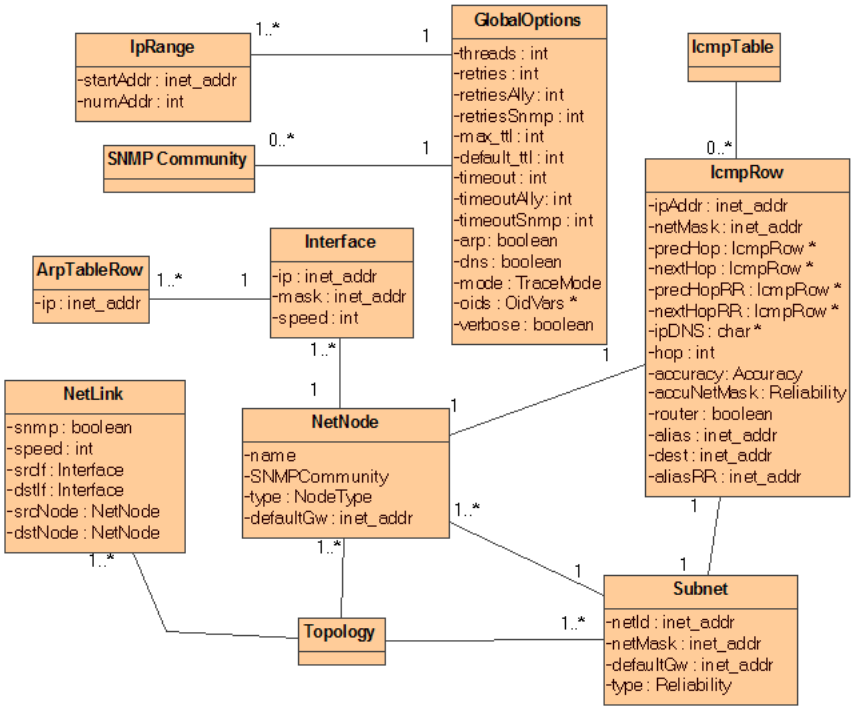

Fig. 1. Business type model.

to recognize serial links in some specific conditions. In addition, it is maintained under GPL license and freely available on the Internet [23]. This permits to compare it with other tools and over other networks.

III. Design

Hynetd has been designed by using UML (Unified Modeling Language) and following the guidelines from software engineering for object oriented programming. Fig. 1 shows the business type model that highlights the classes belonging to application domain. The discovered topology is made of 3 lists of objects: *NetLinks*, *NetNodes* and *Subnets*. *NetNode* has a related list of *Interfaces*, which, in turn, have an *ArpTableRows* list associated. *Subnet* has a list of associated *NetNodes* with at least one interface associated. The *GlobalOptions* object stores all the information needed to perform the discovery, such as *IpRanges* list and *SNMP community* list. Finally, *IcmpTable* rows (named *IcmpRow*) are filled up by active methodologies during the data collection phase. Its structure has been designed to allow the storage of all obtained information and the to quickly reconstruct both Traceroute and PingRR discovered paths in both source-destination and destination-source directions. Such aspect is important because it allowed us to design and implement efficient algorithms for data post-processing.

The activity diagram in Fig. 2 shows an high level view of the *Hynetd* algorithm. Such diagram highlights two functional macro-blocks that respectively perform the data collection and its post-processing needed to reconstruct the final topology. These blocks are sequentially executed because the first interacts with the network and its nodes while the second one works off-line on acquired data. Only the first phase involves multi-thread activities to obtain

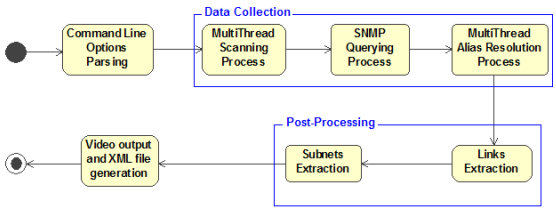

Fig. 2. High level architecture activity diagram.

overlapping among I/O operations.

During the development phase, several tests were performed on all *Hynetd* components by using mock-objects technique, to progressively verify, correct, and improve their operation.

IV. *Hynetd* Architecture and Modules

As remarked in Section III, *Hynetd* operates in two main phases: data collection and post-processing. In this section we will explain in details the operations performed in such phases.

Data collection is carried out in three steps: *scanning*, *interrogation* and *alias resolution*. As shown in Fig. 3, first of all a scan of all IP ranges provided by the user is performed. For this aim, each thread extracts an IP address from a range and sends an *echo request* to it. Then, if *echo reply* is received, the same thread verifies whether SNMP is enabled on this host. If yes, the address is stored in a list named *SNMP list* that will be later used. Otherwise, ICMP mask request, PingRR and Backtrace (see Section IV-A for more details on this algorithm) are executed in sequence toward this destination, storing the results in the IcmpTable. Afterward, all nodes from *SNMP list* are interrogated storing the results directly in the final topology structure. Last step analyzes IcmpTable to perform alias resolution. It first applies the Source Address and PingRR (see section IV-B) techniques, exploiting previously collected information stored in the table. Then a pair-wise test is performed, with multiple threads, by

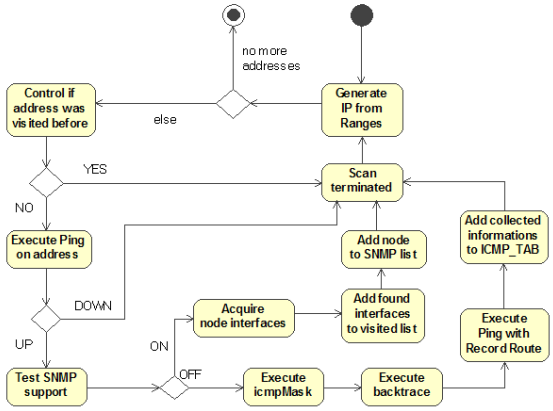

Fig. 3. Scanning thread activity diagram.

using Ally algorithm only for those pairs not classifiable by Ally prevention rules (see section IV-C). At the end of this phase all discovered routers have been created and added to the final topology.

In the second phase the final topology is reconstructed by analyzing all the collected information.

First of all, serial links (i.e. associated to a '/30' subnet) are extracted - in order - from SNMP-enabled nodes and from IcmpTable. In the latter case two different methods can be optionally selected: the first uses the paths recorded using Traceroute and PingRR; the second is applied pairwise and uses some heuristics based on IP address properties. Afterward, subnets are reconstructed by using, sequntially, three information sources: interfaces of SNMP-enabled nodes, serial links and, then, the IcmpTable. In the latter case, addresses are grouped looking at the preceding hop obtained by PingRR. For such subnets, if no hosts responded to ICMP mask request, the net-mask is calculated by using the *subnet guessing from a cluster of addresses* [6] heuristic.

A. Backtrace Algorithm

Backtrace is a novel algorithm designed to reduce the number of packets needed when tracing the route from one host toward many destinations. It is also designed to be effective even in presence of routers configured to avoid their traceability.

As observed by the authors of [16], the information obtained by using Traceroute from a single source toward many destinations can be well represented by a tree structure. This property highlights that intermediate nodes are common to many traced routes. Exploiting this information, the Backtrace algorithm operates in reverse direction with respect to standard Traceroute. In practice, it sends packets with decreasing values of TTL-field which starts from the destination hop distance and end when a known host replies. Fig. 4 shows the activity diagram of the whole algorithm, follows an explanation of how such distance is calculated.

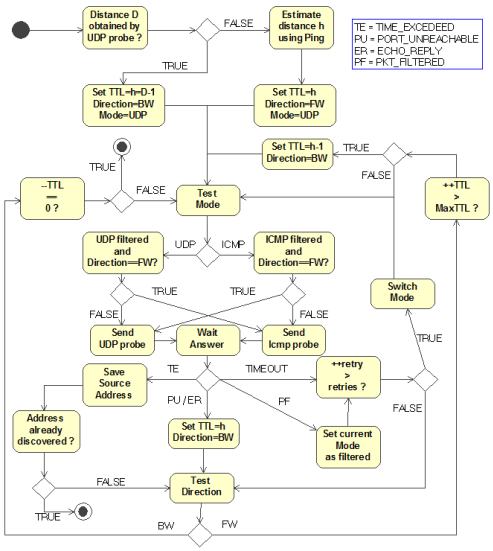

Fig. 4. Backtrace activity diagram.

TABLE I

USEFUL CASES FROM PINGRR.

Case	Addresses inserted
1	DEST - OUT
2	FIX - FIX
3	FIX

Two different methods are introduced to calculate the exact hop distance of each destination. The first uses the TTL value contained in IP headers carried back by ICMP port unreachable packets. The second uses an heuristic combining the TTL value of received ICMP echo reply packets with some well known TTL default values ([14]). As this heuristic method can underestimate the distance, the Backtrace algorithm features a preliminary stage in which the TTL is increased (starting from the heuristic value) until destination is reached. Using one of the above methods, the exact hop distance is calculated and the Backtrace algorithm can effectively perform its task.

An additional feature of this algorithm is related to the protocol it uses to send the probing packets. In details, when a timeout is detected (i.e. the answer from a router is not received), the probe packet is retransmitted by using another protocol (it alternates between ICMP and UDP). This way it is possible to trace the routers filtering one of the two protocols without restarting the whole trace process, as required by other Traceroute implementations.

B. Alias resolution using PingRR

As remarked in Section III, several tests and analyses have been conducted during *Hynetd* development. Analyzing the results of the tests we have observed that the behavior of destination nodes, when receiving a packet with Record Route option, is strongly dependent on the IP stack implementation. Exploiting such differences we have devised a technique able to perform the alias resolution. In details, from all the possible types of PingRR answers we identifies three particular cases useful for this aim.

Tab. I contains, for such cases, the addresses inserted by destination host into Record Route option field. As we can see, they can be one or two depending on the implementation. In the first row of Tab. I, DEST means that such destination host inserted the address toward which we are sending the packets, together with its outgoing interface address (OUT). They can be different thus revealing two alias interfaces. In the other two cases (second and third rows) the FIX address represents the default one used by the destination node for ICMP error packets. It can be different from destination address we used to send the probe packets thus revealing two alias interfaces.

This technique requires the address/addresses inserted by the destination host in IP header option fields. However, all along the path from a source to destination, intermediate nodes may add other addresses to such header. Therefore, in order to extract only the information inserted by the destination host, different tests exploiting previously collected data (hop distance, subnet mask, ...) are performed.

As for the efficiency, this method is comparable to the Source Address technique. The only limitation is that of being applicable only to the destinations with a maximum distance of 7 hops from source.

C. Ally prevention rules

To obtain an accurate alias resolution it is often necessary to execute many instances of the Ally algorithm. Applying such algorithm to a set of N addresses, results in $\binom{N}{2}$ executions, each of them requiring at least two packets to be sent. Therefore, the number of pairs increases exponentially with N, and the overall process becomes very expensive in terms of time and traffic.

To cope with this issues, we introduce a set of rules to be applied to each pair preventing the execution of Ally algorithm when at least one of them applies. Such rules are similar those from *nec*[17]. However, in contrast with our rules, *nec* require many sources in order to be applied.

The Ally prevention rules we apply to all address pairs are the following: (i) addresses resolving to the same domain name through DNS inverse look-up are alias; (ii) addresses having hop distances from source which differ by more than 1 hop are not alias; (iii) addresses belonging to the same loop-free path obtained with Backtrace are not alias; (iv) addresses belonging to the same path obtained with PingRR are not alias; (v) addresses having the same hop distance from source and belonging to paths, toward the same destination, obtained with Backtrace or PingRR are alias;

Along with these rules, another one is utilized: if two addresses are already associated to a node, their aliasing is skipped. In this way, pairs already aliased using Source Address and Record Route method are not processed by Ally.

D. Serial link's Heuristic

Exploiting some properties of the IP addresses of hosts connected though a serial link, this heuristic allows to identify such links. It can also discover links not traversed by probe packets. The basic idea is that serial links are characterized by consecutive addresses part of a "/30" subnet. Therefore, by analyzing all addresses pairs from IcmpTable, we consider as connected through serial links two hosts verifying this set of rules: (i) addresses are numerically consecutive; (ii) hop distance from source differs by 1; (iii) addresses do not end with "'00"' or "'11"' bits; (iv) broadcast and network addresses of the related subnet are not active; (v) broadcast and network addresses of adjacent subnets are not active.

These rules assumes that the adopted routing algorithm obtains the minimum distance between each pair of subnets. When this condition is not satisfied, the second rule may produce false negatives.

E. Further optimizations

Ally algorithm sends UDP packets to obtain ICMP port unreachable replies. Because most routers feature a rate limit when generating ICMP error packets, a multi-thread execution of such algorithm may cause the loss of some replies. To overcome this problem, our implementation of the Ally algorithm includes a packet retransmission mechanism that allows a more effective multi-thread execution. At the same time, this modified version of the algorithm is able to reduce the number of packets injected into the network.

Moreover, we found that the heuristic "subnet guessing from a cluster of addresses"' fails in some dummy cases. As an example, if the cluster contains only 192.168.1.3 and 192.168.1.127, the heuristic returns 255.255.255.128 as

TABLE II

ROUTER HARDWARE DESCRIPTION.

CPU	Pentium 4 3.4 GHz
RAM	2 GB DDR PC3200
Hard Disk	300 MB Serial ATA
Network interfaces	Realtek 8139 Fast-Ethernet OnBoard Davicom Fast-Ethernet 10/100 PCI Davicom Fast-Ethernet 10/100 PCI

net-mask. The correct one should obviously be 255.255.255.0, because broadcast addresses can not be assigned to physical interfaces. To solve this problem, we simply introduced a control on the result of this heuristic. If network id or broadcast address of the subnet are part of the cluster, then the net-mask is widen of a bit.

F. Hynetd modules

Hynetd has been implemented in C language under Linux OS and it is made of several modules. The most relevant are: TYPES which defines all the types of the application domain and implements their comparison functions; SCANNER implementing the data collection and all the algorithms needed for this phase; POST_PROCESSING which implements the off-line reconstruction of the final topology; SNMP implementing functions to query snmp-enabled nodes; OUTPUT which implements the console and file output generation.

V. EXPERIMENTAL ANALYSIS

In this section we show the results obtained by a number of experimental tests we performed using *Hynetd*. As a first step, we compared the performance of 0.1 and 0.2 versions on a small test-bed network. After, we conducted some analyses of the 0.2 version on the MAN of the University of Napoli Federico II.

A. Small Scale Analysis

In order to evaluate *Hynetd* performance we first used a controlled test-bed. In this way, we had a complete knowledge of the topology to be discovered and of the cross traffic relying on the test-bed.

The test-bed was composed of 7 routers (Personal Computers with hardware configuration described in Tab. II and running Kubuntu Linux 5.10 with kernel 2.6.12) and a notebook (Acer Travelmate 2502 LMI, 3.0 GHz P4, and 512 MB RAM) running Kubuntu Linux 6.05 we used to execute *Hynetd*.

The tests were executed on two particular topologies (see Fig. 5(left) and Fig.5(right)) on which discovery process could be difficult (the motivations at the base of such difficulties are provided, for each topology, in the related section). The parameters we evaluated are: traffic generated (both in-going and out-going), discovery time and a set of accuracy parameters. In details we define: (i) *accuracy of routers*, *subnets* and *links* as the ratio between discovered entities and total entities; (ii) *accuracy of interfaces* and *net-masks* as ratio between correct entities and total entities with respect to discovered routers and subnets.

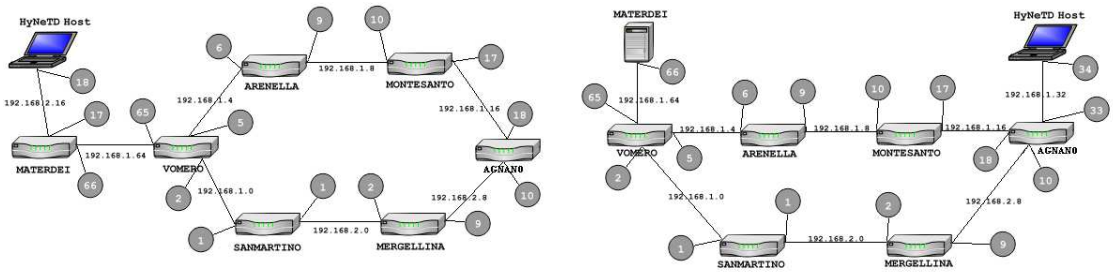

Fig. 5. Ring (left) and Backup (right) topology

Moreover, the tests were executed by using a different number of thread and retry as well as four different network conditions which are listed in Tab. III. Each test was repeated three times reporting, in the following, the average results.

1) Ring Topology: it is characterized by a loop and its discovery may fail when adopting active methodologies from a single source point. Indeed, in such configuration, two routers will not forward packets, therefore they should not be correctly recognized. Moreover, the link between them is never traversed by probe packets, as a consequence, it can not be detected.

Tab. IV shows the accuracy obtained by 0.1 and 0.2 versions in all four network conditions. The newest *Hynetd* version attains the best results in all conditions. Instead, 0.1 version obtains the same results only when using the passive methodology. Moreover, the serial link heuristic allows to discover the link between Mergellina and Agnano even if it is not traversed by probe packets. These results confirm that *Hynetd* approach can be really effective in most cases on network topologies which comprises a loop.

Fig. 6 (left) shows the discovery time as a function of the number of threads and retries in the condition we called Active (see Tab. III). The tests have been performed also in the other conditions and similar results have been obtained. As we can see, the discovery time decreases when the number of threads increases while it increases with the number of retries. However, the discovery time taken by version 0.2 is significantly lower and its trend is more regular. This behavior mostly depends on the higher pipelining featured by such version.

Fig. 6 (right) sketches, for the all considered network conditions, the traffic generated by discovery process as a function of the number of retries. As shown, such traffic is different for different network conditions. Despite

TABLE III

NETWORK CONDITIONS.

Name	Description
Passive	SNMP available on all routers and DNS inverse look-up enabled
Active	SNMP not available on all routers and DNS inverse look-up disabled
Hybrid 1	SNMP not available on all routers and DNS inverse look-up enabled
Hybrid 2	SNMP available on some routers and DNS inverse look-up disabled

TABLE IV

ACCURACY IN THE CASE OF RING TOPOLOGY.

Hynetd version	Network condition	Accuracy				
		Routers	Links	Subnets	Interfaces	Net-masks
0.2	*passive*	100%	100%	100%	100%	100%
	active	100%	100%	100%	100%	100%
	hybrid 1	100%	100%	100%	100%	100%
	hybrid 2	100%	100%	100%	100%	100%
0.1	*passive*	100%	100%	100%	100%	100%
	active	71%	86%	62%	100%	93%
	hybrid 1	71%	86%	58%	100%	93%
	hybrid 2	87%	100%	100%	100%	89%

this, it increases with retry value in every condition. The comparison highlights how the introduction of Backtrace algorithm and Ally prevention rules impacts on the traffic generation.

2) Backup Topology: it is characterized by the presence of a backup path. This path should not be recognized when using the active methodologies because, in normal conditions, it is never traversed by probe packets. Indeed, there is a router (which of the seven depends on the routing configuration) that does not forward packets in normal condition. For this reason, also recognizing such host as a router is not that simple.

Tab. V shows the accuracy obtained by the two versions in all 4 network conditions (see Tab. III). Again, *Hynetd* 0.2 attains the best results in all conditions when compared to the older version. Indeed, version 0.1 needs the passively collected information to attain the same results. Moreover, the serial link heuristic allows to discover also the link between Vomero and SanMartino even if it is not traversed by probe packets. These results confirm that *Hynetd* approach is capable to effectively discover backup paths.

In Fig. **??** (right) we report the traffic generated by the discovery process, in all network conditions, as a function of the retries value. Again, traffic increases with retries value in every condition but the comparison highlights that *Hynetd* version 0.2 generates less traffic.

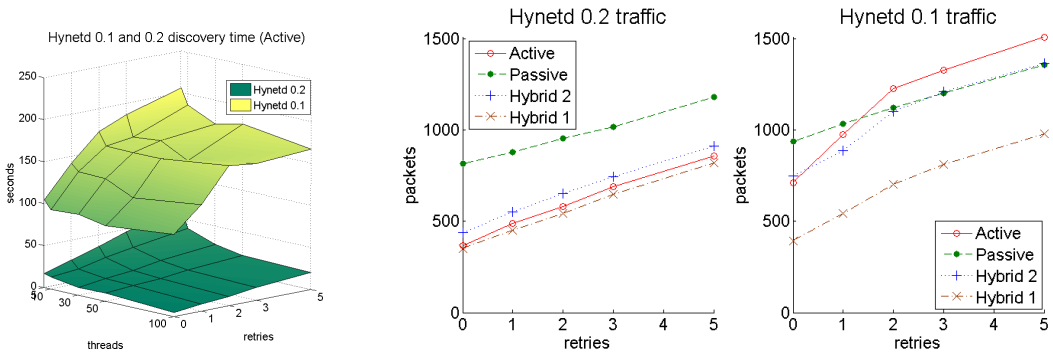

Fig. 6. Ring topology: Discovery time in Active condition (left), Traffic generated by discovery process (right)

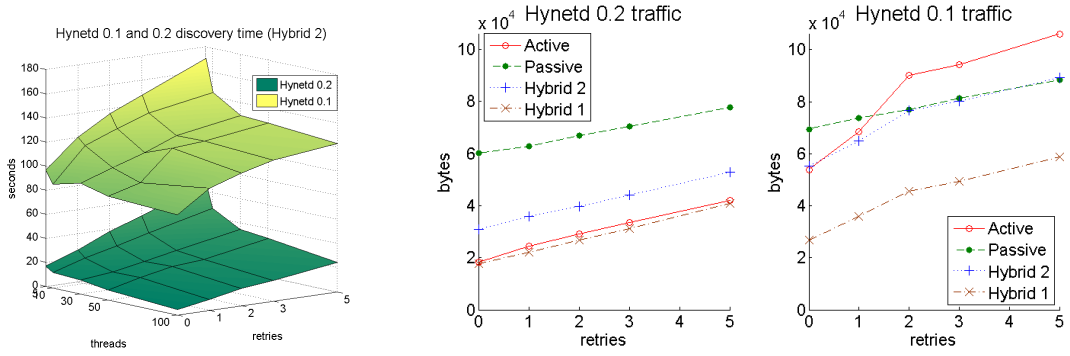

Fig. 7. Backup topology: Discovery time in Hybrid 2 condition (left) and Traffic generated by discovery process (right).

Fig. 7 (left) reports the discovery time in Hybrid 2 as a function of the number of threads and retries. As we can see, such time is in direct proportion to the retries while in inverse to the thread number. For these results the same considerations of the previous topology apply.

B. Large Scale Analysis

To evaluate *Hynetd* performance on a real and large network with real traffic, we used the metropolitan area network of the University of Napoli Federico II (named UniNa in the following), the properties of which are listed in Tab. VI. These experiments were performed by using the entire class B address of such network.

The tests were conducted from three different locations (referred as ‘A’, ‘B’ and ‘C’ in the map of Fig. 8) in order to evaluate the influence of the source position inside the network on the results. Each experiment was performed by using Active and Hybrid methodologies. In Tab. X we report mean (μ) and standard deviation (σ) of each considered parameter averaged on the three location while in Tab. VII, VIII, and IX the results for each location are presented.

Results show that *Hynetd* takes about one hour to discover all the network. As a first consideration, we expected that by using Hybrid methodology the discovery time would be lower, but this was not the case because of the

TABLE V

ACCURACY IN THE CASE OF BACKUP TOPOLOGY.

Hynetd version	Network condition	Accuracy				
		Routers	Links	Subnets	Interfaces	Net-masks
0.2	*passive*	100%	100%	100%	100%	100%
	active	100%	100%	100%	100%	100%
	hybrid 1	100%	100%	100%	100%	100%
	hybrid 2	100%	100%	100%	100%	100%
0.1	*passive*	100%	100%	100%	100%	100%
	active	83%	83%	50%	100%	100%
	hybrid 1	83%	83%	37%	100%	100%
	hybrid 2	83%	100%	100%	100%	100%

TABLE VI

UNINA NETWORK.

Number of sites	19	Average number of active addresses	4808
Physical diameter	$\approx$ 8 Km	Average number of active hosts	2774
Network diameter	9 hops	Number of links	209
Number of routers	180	Number of subnets	649

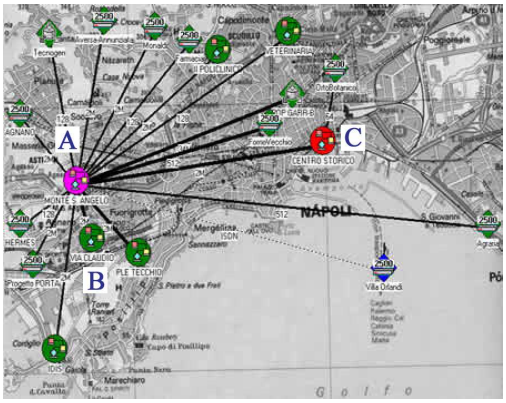

Fig. 8. High level view of UniNa network topology.

TABLE VII

RESULTS OBTAINED FROM LOCATION A

	Active		Hybrid	
	μ	σ [%]	μ	σ [%]
Time [s]	3'808	11	3'929	11
Traffic [packets]	413882	11	415'863	6
Traffic [bytes]	17'388'600	11	19'116'124	7
Routers Accuracy [%]	100	0	100%	0
Subnets Accuracy [%]	43	6	53	5
Links Accuracy [%]	75	0	86%	0
Interfaces Accuracy [%]	73	1	75	1
Net-masks Accuracy [%]	56	3	66	1

TABLE VIII

RESULTS OBTAINED FROM LOCATION B

	Active		Hybrid	
	μ	σ [%]	μ	σ [%]
Time [s]	2'708	5	2'690	3
Traffic [packets]	435'863	2	404'649	2
Traffic [bytes]	18'624'244	3	18'859'759	2
Routers Accuracy [%]	100	0	100	0
Subnets Accuracy [%]	46	5	50	4
Links Accuracy [%]	75	0	86	0
Interfaces Accuracy [%]	74	1	76	1
Net-masks Accuracy [%]	58	4	69	2

TABLE IX

RESULTS OBTAINED FROM LOCATION C

	Active		Hybrid	
	μ	σ [%]	μ	σ [%]
Time [s]	3'017	13	3'205	13
Traffic [packets]	451'780	5	456'377	1
Traffic [bytes]	19'510'104	5	21'432'617	1
Routers Accuracy [%]	100	0	100	0
Subnets Accuracy [%]	49	5	56	4
Links Accuracy [%]	75	0	86	0
Interfaces Accuracy [%]	73	1	75	1
Net-masks Accuracy [%]	53	4	69	2

TABLE X

AVERAGE RESULTS OBTAINED ON UNINA NETWORK

	Active		Hybrid	
	μ	$\sigma\%$ [%]	μ	$\sigma\%$ [%]
Time [s]	3'177	18	3'331	19
Traffic [packets]	433'840	4	425'629	6
Traffic [bytes]	18'507'649	6	19'802'833	7
Routers Accuracy [%]	100	0	100	0
Subnets Accuracy [%]	47	6	53	6
Links Accuracy [%]	75	0	86	0
Interfaces Accuracy [%]	74	1	76	1
Net-masks Accuracy [%]	56	4	68	2

presence of many SNMP-enabled network printers that slowly replied (involving several timeouts) to queries. Also, the traffic generated during the discovery process has a little variation and highlights that SNMP queries generate less but bigger packets. The average traffic byte rate results to be about 6 Kb/sec, which it is negligible when compared to network bandwidth (i.e. $\in [10Mbps, 32Gbpbs]$). Changing the source point had substantial effect only on subnets and net-masks accuracy. Moreover, using Hybrid configuration the accuracy is always higher. Finally, comparing the results obtained from the three locations (that are the main cross-connect and two terminal nodes), we can state that the accuracy is preserved even if the tool is run from a terminal node.

C. Further Investigations

1) Scalability Analysis: During previous analyses we detected some factors that affect discovery time and generated traffic. The most important is the number of active addresses (hard to predict) in the scanned range. Starting from this observation, we performed this analysis in order to investigate the influence of the number of active addresses on the discovery time.

To perform that we called "scalability analysis" we decided to use 5 classes of '/24' subnets which differ in terms of number of active addresses (that are 0, 15, 30, 45, and 60 hosts/subnet). We then selected 7 subnets, for

each class, executing *Hynetd* progressively on one subnet, two, three, and so on.

Fig. 9 shows the discovery time as a function of the number of subnets scanned (left) and of the number of active hosts per subnet (right). These are two ways to display the same results. The former evidences that the discovery time increases linearly with address space dimension (i.e. the number of subnets to be scanned) even if the slope depends on the number of active hosts (i.e. on the subnet class). This is an important result useful to evaluate an upper bound for the time taken by *Hynetd* to discover a topology of whichever network. Fig. 9 (right) allows to verify that the discovery time of fixed number of subnets increases more than linearly with the percentage of active addresses of such subnets.

This same trends is evidenced in Fig. 10 (left) where we report the traffic generated by the discovery tool as a function of the number of subnets (left) and the number of active hosts per subnet (right).

Finally, Fig. 10 (right) shows that the percentage of total addresses pairs involved in Ally algorithm is small and decreases when the address space dimension increases. These results confirm the effectiveness of Ally prevention rules. Indeed, if such rules were not introduced, the number of pairs would exponentially increase.

2) Comparison with a commercial tool: In this analysis we compared our tool with NetworkView 3.5 [25], a commercial topology discovery software freely available in a 30 days trial version. This software uses many techniques and protocols to achieve also application level discovery. To perform a fair comparison with *Hynetd*, we enabled only ICMP and SNMP features and chose the same retries and timeout values for both tools. The comparison was made on a portion of the UniNa network, running the software at location 'C' in Fig. 8. NetworkView does not apply any alias resolution technique, so we expected it to generate less traffic in less time. As shown in Tab. XI, the results are completely opposite: *Hynetd* is faster, more efficient and scales better. Moreover, we observed that the topology discovered by NetworkView is not accurate because it seem to recognize as routers only those responding to SNMP requests.

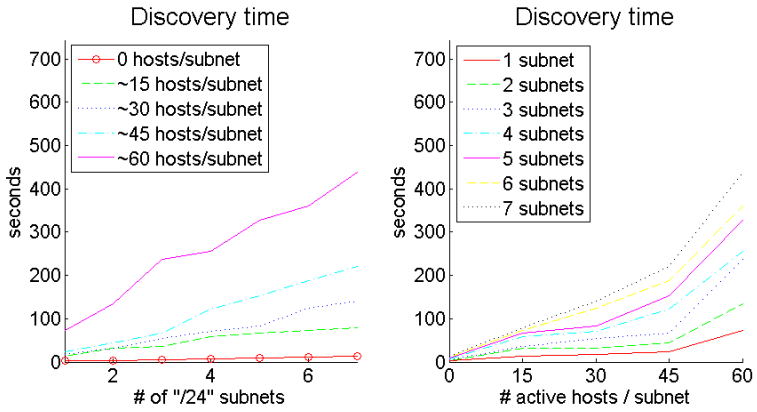

Fig. 9. Discovery time scalability.

TABLE XI

Hynetd 0.2 VS NETWORKVIEW 3.5.

	Hynetd			NetworkView		
Range	Time	Pkts	Bytes	Time	Pkts	Bytes
'/24'	49 sec	14248	1268513	1202 sec	22196	2015897
'/23'	98 sec	18427	1549027	1510 sec	28484	2571034

REFERENCES

[1] V. Paxon, S. Floyd, *"Why we don't know how to simulate the Internet"*, Winter Simulation Conference, pp 1037-1044, 1997.

[2] D. Emma, A. Pescapé, G. Ventre, *"Discovering topologies at router level"*, 5th IEEE IPOM 2005 LNCS 3751, pp. 118129, October 2005, Barcelona, Spain.

[3] J. Schnwlder, H. Langendrfer, *"How to Keep Track of Your Network Configuration"*, TU Braunschweig, Germany, 1993.

[4] Wood et al., *"Fremont: A System for Discovering Network Characteristics and Problems"*, Winter USENIX Conference, Jan. 1993.

[5] G. Mansfield et al., *"Techniques for automated Network Map Generation using SNMP"*, Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Mar. 1996.

[6] R. Siamwalla, R. Sharma, S. Keshav, *"Discovering Internet Topology"*, Department of Computer Science, Cornell University, 1999

[7] B. Huffaker, D. Plummer, D. Moore, K. Claffy, *"Topology Discovery by active probing"*, CAIDA, University of California, 1998

[8] G. Malkin, *"Traceroute Using an IP Option"*, RFC1393, 1993

[9] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Sashadri, A. Silbershatz, *"Topology Discovery in Heterogeneus IP Networks"*, *IEEE INFOCOM'2000*, Tel Aviv, Israel, 2000

[10] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, A. Silbershatz, *"Topology Discovery in Heterogeneus IP Networks: The NetInventory System"*, IEEE/ACM Transactions on Networking, Vol. 12, no. 3, 2004

[11] R. Govindam, H. Tangmunarunkit, *"Heuristics for Internet Map Discovery"*, USC/Information Sciences Institute, 2000

[12] P. Barford et al., *"The Marginal Utility of Network Topology Measurements"*, ACM/SIGCOMM Internet Measurement Workshop, Nov. 2001.

[13] P. Dinda et al., *"The Architecture of the Remos System,"*, 10th IEEE Symp. on High-Perf. Dist. Comp. (HPDC01), Aug. 2001.

[14] `http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html`

[15] R. Teixeira, K. Marzullo, S. Savage, G. M. Voelker, *"In Search of Path Diversity in ISP Networks"*, Computer Science and Engineering, University of California, 2003.

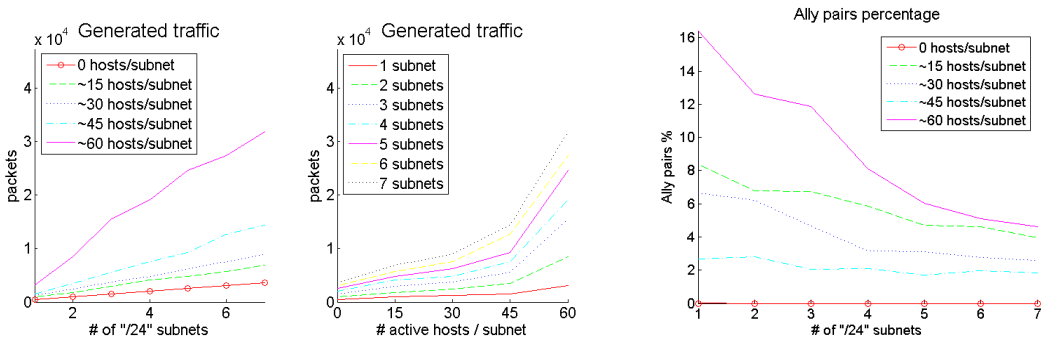

Fig. 10. Generated traffic scalability (left) and Ally pairs percentage scalability (right).

[16] B. Donnet, T. Friedman, M. Crovella, *"Improved algorithms for Network Topology Discovery"*, *Passive and Active Measurement Workshop (PAM)*, 2005.

[17] D. Magoni, M. Hoerdt, *"Internet core topology mapping and analysis"*, in *Computer Communications*, vol. 28, no. 5, pp. 494-506, March 2005.

[18] F. Nazir, M. Jameel, T. H. Tarar, H. A. Burki, H. F. Ahmad, A. Ali, H. Suguri, *"An Efficient Approach Towards IP Network Topology Discovery for Large Multi-subnet Networks"*, *11th IEEE Symposium on Computers and Communications*, 2006.

[19] M. Gunes, K. Sarac, *"Analytical IP Alias Resolution"*, Department of Computer Science, University of Texas at Dallas, 2006

[20] `http://www.openview.hp.com`

[21] `http://www-306.ibm.com/software/tivoli/`

[22] `http://www.rocketsoftware.com/portfolio/netcure`

[23] `http://www.grid.unina.it/software/TD/`

[24] N. Spring, R. Mahajan, D. Wetherall, *"Measuring ISP Topologies with Rocketfuel"*, Computer Science and Engineering, University of Washington, 2002.

[25] `http://www.networkview.com/`