Discovering Topologies at Router Level: Part II

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*—Measurement and monitoring of network topologies are essential tasks in current network scenarios. Indeed, due to their utility in planning, management, security, and reliability of network infrastructures, effective and efficient approaches and tools for discovering large topologies are gaining more and more attention from both Application Service Providers and network administrators. In this paper we propose a hybrid methodology and its implementation in a software platform we called *Hynetd*. We present the architecture, some novel algorithms and methods adopted in the discovery chain, and a performance evaluation over two network scenarios: a small scale test-bed and a large scale MAN in the heart of Napoli (Italy). We provide experimental results confirming and improving those previously obtained by a prototype of *Hynetd*. In addition a comparison with a commercial tool is presented. Achieved results, in terms of accuracy, discovery time, and traffic injected, are very encouraging in both considered scenarios.**

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 planning, management, security, and reliability, as well as to obtain substantial advantages in fault management, performance analysis and service allocation. Moreover, as the automatic generation of realistic topologies is a difficult task [1], such knowledge proves to be very useful also to perform accurate simulations. 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 with as few probing packets as possible and within the minimum time.

In the past years, several techniques and tools have been proposed. We have classified them in *active*, *passive*, and *hybrid*. The first ones are based on tools such as Ping and Traceroute, and infer topology information from network behavior, the second use SNMP to obtain information from devices, whereas the *hybrid* approaches use both methodologies. When using an *active* methodology, two problems must be solved: (i) recognizing the interfaces belonging to the same network device, referred to as *alias resolution*; (ii) reconstructing correct subnet addresses and net-masks. The use of SNMP instead, often requires particular privileges to be granted by the network administrator. Using both methodologies together, allows to mitigate such problems.

Since 1993 many works dealing with topology discovery have been published. They differ in terms of methodology, number of employed probes, prerequisites and explored protocol layers. Many works using an *active* methodology have been presented in literature [4] [8] [12] [13] [16] [18] [20] [17]. They heavily use Traceroute, Ping, and some techniques to perform the *alias resolution*. As regards works based on a *passive* methodology in [6] [10] [11] [14] [19] different SNMP-based architectures have been presented. To the best of our knowledge, few works adopt a *hybrid* methodology [5] [7]. Finally, several proprietary and commercial tools exist (e.g. HPs OpenView [21], IBMs Tivoli [22], RocketSoftware NetCure [23], NetworkView [26]). Due to space constraints we can not provide the details of the cited works, for a more careful analysis refer to [2].

Analyzing the literature we have found a lack of tools which are flexible, robust, and publicly available. To this end, we started the design of an application aimed at maintaining minimal prerequisites, being usable on every IP-based network, and exploiting all the available information sources to discover the topology. In [3] a preliminary version (namely 0.1) of *Hynetd* (Hybrid Network Topology Discovery) has been presented. Despite its quite good performance, to improve the discovery process we have completely revised its architecture. Here we present version 0.2 of *Hynetd* with a new and more efficient architecture aimed to improve discovery efficiency and performance by means of: (i) multi-threaded activities; (ii) a new algorithm named Backtrace to efficiently execute many concurrent Traceroutes; (iii) a novel approach for *alias resolution* using *Ping with Record Route* option; (iv) some rules reducing the number of IP addresses involved in the Ally algorithm; (v) an heuristic to recognize serial links. *Hynetd* is released at [24] under General Public License.

To illustrate the performance improvement, we provide results of a careful comparison amongst both the first version of *Hynetd* and NetworkView 3.5 [26], a commercial topology discovery software. The performance indicators we consider are 5 accuracy parameters (described in Section III), probing traffic amount, and discovery time. The experimental evaluation has been carried out over two network scenarios: a small scale controlled test-bed and a large scale MAN.

The rest of the paper is organized as follows. Section II briefly explains *Hynetd* architecture and the proposed innovations. In Section III we present small and large scale experimental analysis. Finally, Section IV ends the paper with conclusions and issues for research.

II. *Hynetd* Architecture

In this section we provide an overview of the new architecture and some information regarding the innovations we introduced with respect to [3]. Details on design, planning, implementation and testing are reported in [2]. The activity

[0]This work has been partially supported by PRIN RECIPE and CONTENT EU NoE, OneLab and NETQOS EU projects.

1930-529X/07/$25.00 © 2007 IEEE

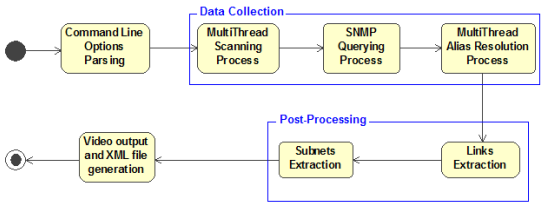

Fig. 1. Activity diagram.

diagram of Fig. 1 shows an high level view of the *Hynetd* algorithm highlighting two functional macro-blocks: data collection and post-processing. These blocks are sequentially executed because the second one requires as much data as possible to accurately reconstruct the topology. The first phase involves multi-threaded activities to obtain overlapping among I/O operations.

Data collection is carried out in three steps: *scanning*, *interrogation* and *alias resolution*. The *scanning* starts by sending ICMP *echo requests* toward the IP addresses provided by the user. When an *echo reply* is received, an SNMP capability test is conducted. If passed, the address is stored in the *SNMP list*. Otherwise, ICMP mask request, Ping with Record Route option (PingRR), and Backtrace (see Section II-A for more details on this algorithm) are sequentially executed toward this destination, storing the results in the IcmpTable. After *scanning* is completed, all nodes from *SNMP list* are *interrogated* storing the results directly in the final topology structure. The last step analyzes IcmpTable to perform the *alias resolution*. Firstly, the Source Address (see [4] for a description) and PingRR (see Section II-B) techniques are applied. Then, a pair-wise test is performed by using Ally algorithm. Such test is skipped for the pairs classifiable by Ally prevention rules (see Section II-C). During the last two steps, discovered routers composing the final topology are determined.

Data post-processing reconstructs the other elements of the topology by analyzing the collected information. First of all, serial links (i.e. associated to a '/30' subnet) are identified by using the collected SNMP data and the IcmpTable entries. In the latter case, two methods can be selected: the first one uses Traceroute and PingRR discovered paths; the second one, applied pairwise, uses some heuristics based on IP address properties (see Section II-D). Afterward, subnets are reconstructed by sequentially using three information sources: interfaces of SNMP-enabled nodes, serial links, and IcmpTable. For subnets reconstructed by using IcmpTable, if no hosts responded to ICMP mask request, their mask is calculated by using the *subnet guessing from a cluster of addresses* [7] heuristic.

A. Backtrace Algorithm

Backtrace is a novel algorithm, designed to reduce the number of packets needed for tracing the routes 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 [17], 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. Two different methods are introduced to calculate such distance. The first one uses the TTL value contained in IP headers carried back by ICMP *port unreachable* packets. The second one uses an heuristic combining the TTL value of received ICMP *echo reply* packets with some well known TTL default values ([15]). 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. 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). In this way it is possible to trace the routers filtering one of the two protocols without restarting the whole tracing process, as required by other Traceroute implementations.

TABLE I
PINGRR ANSWERS USEFUL FOR *alias resolution*.

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

B. Alias resolution using PingRR

During *Hynetd* development, several tests have been conducted. Analyzing their results we have discovered that the behavior of destination nodes, when receiving a packet with Record Route option, is strongly dependent on the IP stack implementation. Analyzing such differences, we have devised a technique to perform the *alias resolution*. In details, from all the possible types of PingRR answers we identified three particular cases useful for this aim. Tab. I contains, for such cases, the addresses inserted by the destination host into Record Route option field. As we can see, they can be one or two depending on the implementation. The first row of Tab. I (case 1) refers to a destination host which inserted the address toward which we sent the packet (DEST), together with the outgoing interface (OUT) it uses to forward the echo_reply packet. If different, they reveal two alias interfaces. In the other two cases (2 and 3) the FIX address represents the default used by the destination node, for ICMP error packets. This can be different from the address toward which we sent the probe packets, thus revealing two alias interfaces. 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 the 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 which requires 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 issue, we introduce a set of rules preventing the execution of Ally algorithm to the address pairs for which any of them applies. Such rules are similar to those from *nec* [18] but, in contrast, they do not 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. Moreover, the Ally algorithm is not applied to the addresses already associated to a node by means of other techniques.

D. Serial link Heuristic

Exploiting some properties of the IP addresses of nodes connected through 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 such hosts have consecutive addresses which are part of a "/30" subnet. Therefore, by analyzing all address pairs from IcmpTable, we consider as connected through serial links, two hosts having all the following properties: (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 assume the 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

The Ally algorithm sends UDP packets to obtain ICMP port unreachable replies. Because most routers feature a limited rate when generating ICMP error packets, a multi-threaded 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-threaded 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 *subnet guessing from a cluster of addresses* heuristic fails in some 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 net-mask. The correct one should be 255.255.255.0 because broadcast addresses can not be assigned to physical interfaces. We solve this problem by widening the net-mask of 1 bit when the network id or the broadcast address of the subnet are part of the cluster.

TABLE II
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 only on 2 routers and DNS inverse look-up disabled

III. EXPERIMENTAL ANALYSIS

We compared the performance of versions 0.1 and 0.2 on a small scale test-bed. Moreover, we conducted some experimentations with the version 0.2 on the *University of Napoli "Federico II"* MAN, comparing the performance also with a commercial tool. 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 number of discovered and total entities; (ii) *accuracy of interfaces* and *net-masks* as the ratio between number of correct and total entities with respect to discovered routers and subnets respectively. The experimentations are aimed to evaluate: i) the accuracy, discovery time, and generated traffic in different operating conditions; ii) the impact of the network segment from which to perform the tests on the considered parameters; iii) the trend of the discovery time and generated traffic as a function of the number and dimension of the analyzed subnets; iv) the effectiveness of the introduced Ally prevention rules.

A. Small Scale Analysis

In order to evaluate *Hynetd* performance, we first used a controlled test-bed in order to have a complete knowledge of the topology to be discovered and of the cross traffic relying on the network. The test-bed was composed of 7 *software routers* (3.4 GHz P4, 2 GB RAM, and 3 Fast Ethernet interfaces) with Linux and a notebook (Acer Travelmate 2502 LMI, 3.0 GHz P4, and 512 MB RAM) with Linux and running *Hynetd*. The tests were executed on two particular topologies (see Fig. 2 and 3) on which the discovery process could be difficult (the motivations for such difficulties are provided in the related section). Moreover, they were executed by using different number of threads and retries as well as four different network conditions which are listed and explained in Tab. II. 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. 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, as a consequence, it can not be detected. Tab. III shows the accuracy obtained by versions 0.1 and 0.2 in all four network conditions. *Hynetd* version 0.2 attains optimal results in all considered conditions.

1930-529X/07/$25.00 © 2007 IEEE

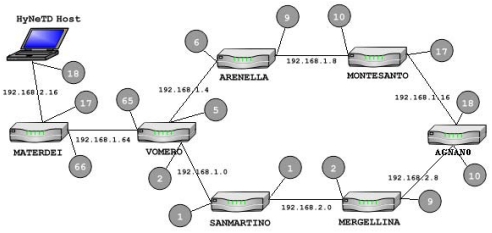

Fig. 2. Ring topology

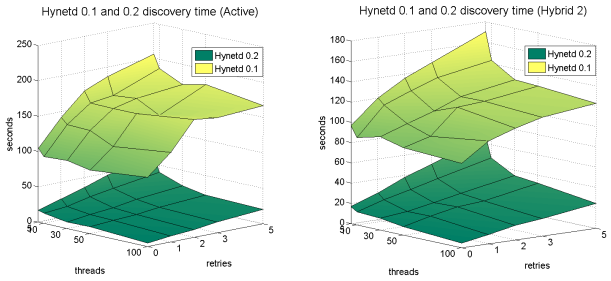

Fig. 4. Discovery time for ring (left) and backup (right) topologies.

TABLE III
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%

TABLE IV
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%

Instead, version 0.1 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 the approach implemented in *Hynetd* 0.2 can be really effective on network topologies that comprise a loop. Fig. 4 (left) shows the discovery time as a function of the number of threads and retries in the *active* condition. 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 overlapping featured by such version. Fig. 5 sketches the traffic generated by the discovery process as a function of the number of retries for all considered network conditions. As shown, such traffic varies with network conditions. Despite this, it almost linearly increases with the retry value in every condition. The comparison highlights how the introduction of the 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 one of the seven routers, it depends on the routing configuration) not forwarding packets in normal conditions. For this reason, recognizing such host as a router is not simple. Tab. IV shows the accuracy obtained by the two versions in all network conditions. Again, *Hynetd* 0.2 attains the best results in all conditions when compared to the older version. Indeed, version 0.1 needs the information collected with the *passive* methodology to achieve the same results. Moreover, the serial link heuristic allows to discover the link between Vomero and SanMartino even if it is not traversed by probe packets. These results confirm that the *Hynetd* approach is capable to effectively discover backup paths. Fig. 4 (right) reports the discovery time in Hybrid 2, as a function of the number of threads and retries. As we can see, such time increases with the retries, while it decreases with the thread number. For these results the same considerations of the previous topology apply. In Fig. 6 we report the traffic generated by the discovery process, in all network conditions, as a function of the retries value. Again, traffic increases with the retries value in every condition, but the comparison highlights that *Hynetd* version 0.2 generates less traffic in every network condition.

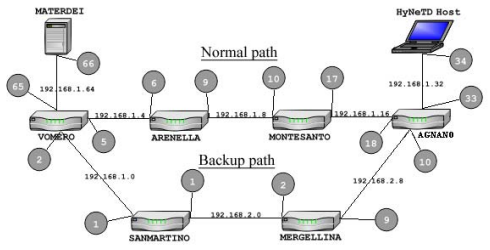

Fig. 3. Backup topology

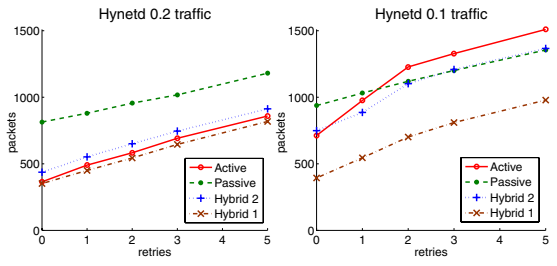

Fig. 5. Ring topology: Traffic generated by discovery process.

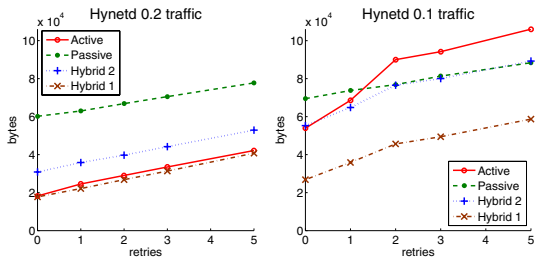

Fig. 6. Backup topology: Traffic generated by discovery process.

TABLE V
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

B. Large Scale Analysis

To evaluate *Hynetd* performance on a real and large scale network on which also real traffic is present, we used the Metropolitan Area Network of the University of Napoli "Federico II" (named UniNa in the following), reported in Fig. 7, whose properties are listed in Tab. V. These experiments were performed by using the class B address of such network. The tests were conducted from three different locations, referred to as 'A', 'B' and 'C' in the map of Fig. 7, corresponding to the main cross-connect and two terminal nodes respectively. Each experiment was performed by using Active and Hybrid methodologies. In Tab. VI we report mean (μ) and standard deviation (σ) of each considered parameter averaged on the three locations. Results show that *Hynetd* takes, in average, about 53 minutes to discover the network topology. As a first consideration, we expected that by using Hybrid methodology the discovery time would be lower. This was not the case because of the 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 fewer but larger packets. The average byte rate is about 6 Kb/sec, which it is negligible when compared to network

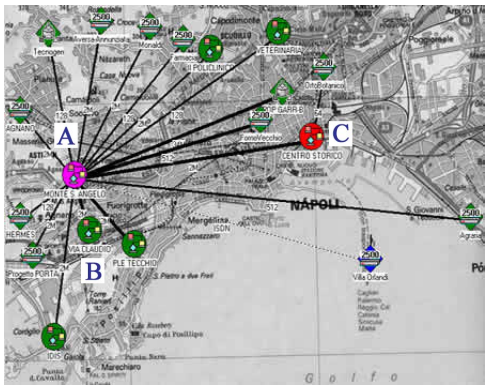

Fig. 7. High level view of UniNa network topology.

TABLE VI
AVERAGE RESULTS OBTAINED ON UNINA NETWORK

	Active		Hybrid	
	μ	σ% [%]	μ	σ% [%]
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

bandwidth (i.e. $\in [10Mbps, 2.5Gbps]$). As for the accuracy, changing the source point had some effect only on subnets and net-masks, in overall it is preserved even if the tool is run from a terminal node. Moreover, using Hybrid configuration the accuracy is always higher.

C. Further Investigations on Hynetd performance

During previous analyses we detected some factors that affect the discovery time and generated traffic. The most important is the number of active addresses in the scanned range. Starting from this observation, we performed an analysis aimed to investigate the influence of the number of active addresses on the discovery time. To this aim, 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* on an increasing number of subnets. Fig. 8 shows the discovery time as a function of the number of scanned subnets (left) and of the number of active hosts per subnet (right). These are two different ways to display the same results, each of which evidences a peculiar aspect. The former shows 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. 8 (right), instead, 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. The same trend is evidenced in Fig. 9 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). As for the effectiveness of Ally prevention rules, Fig. 10 shows that the percentage of total address pairs involved in the Ally algorithm is small and decreases when

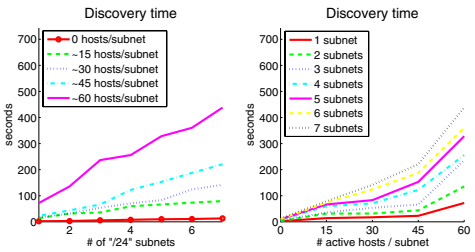

Fig. 8. Analysis of the discovery time.

TABLE VII
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

increasing the address space dimension. These results confirm that the Ally prevention rules are actually able to avoid the number of pairs from exponentially increasing.

D. Comparison with a commercial and proprietary tool

We compared our tool with NetworkView 3.5 [26], 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* 0.2, 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. 7. NetworkView does not apply any *alias resolution* technique, so it should generate less traffic in less time. As shown in Tab. VII, the results are completely opposite: *Hynetd* 0.2 is faster, more efficient, and scales better. Moreover, we observed that the topology discovered by NetworkView is not accurate because it seems to recognize as routers only the hosts responding to SNMP requests.

IV. CONCLUSIONS

In this paper we proposed a platform for discovery network topologies at router level. We provided details on both design and implementation issues, and we described the new algorithms proposed to enhance the discovering phase. We showed how, using a hybrid and parallel approach, our proposal is able to outperform similar platforms. For the comparison we used both a previous version of our tool and a commercial tool. We provided evidences that thanks to the introduction of the Backtrace algorithm, Ally prevention rules, and the serial link heuristic, *Hynetd* 0.2 is able to correctly discover the topology also when other tools fail (ring and backup topologies). More precisely, we reported results - in terms of accuracy, discovery time, and traffic injected (over both small and large networks) - that outperform the older version and the selected commercial tool. In the case of a large network we showed how the position of the discovery probe does not heavily affect the final results. Our ongoing work is concerned with the development of a module able to merge the information derived from different instances of *Hynetd* running at different locations.

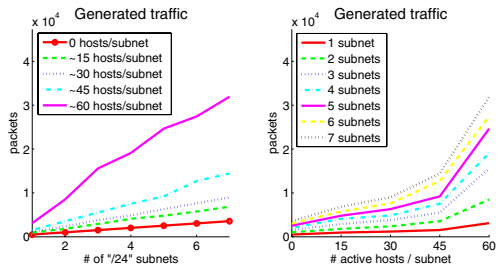

Fig. 9. Analysis of the generated traffic.

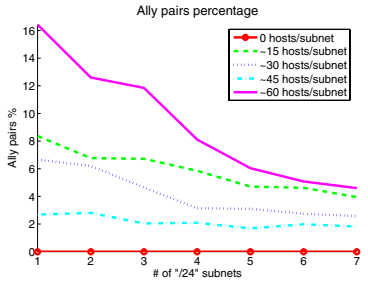

Fig. 10. Percentage of IP addresses pair involved in Ally algorithm.

This will allow to compare the results with other multi-source topology discovery tools.

REFERENCES

[1] V. Paxon, S. Floyd, *"Why we don't know how to simulate the Internet"*, Winter Simulation Conference, pp 1037-1044, 1997.
[2] A. Botta, W. de Donato, A. Pescapé, G. Ventre, *"Design, Implementation, and Testing of a Hybrid Tool for Network Topology Discovery"*, TR-DIS-TD-3-2007, available at http://www.grid.unina.it/software/TD/.
[3] D. Emma, A. Pescapé, G. Ventre, *"Discovering topologies at router level"*, 5th IEEE IPOM 2005, pp. 118-129, Oct.'05, Barcelona, Spain.
[4] J. Schnwlder, H. Langendrfer, *"How to Keep Track of Your Network Configuration"*, TU Braunschweig, Germany, 1993.
[5] Wood et al., *"Fremont: A System for Discovering Network Characteristics and Problems"*, Winter USENIX Conference, Jan. 1993.
[6] 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.
[7] R. Siamwalla, R. Sharma, S. Keshav, *"Discovering Internet Topology"*, Department of Computer Science, Cornell University, 1999
[8] B. Huffaker, D. Plummer, D. Moore, K. Claffy, *"Topology Discovery by active probing"*, CAIDA, University of California, 1998
[9] G. Malkin, *"Traceroute Using an IP Option"*, RFC1393, 1993
[10] Y. Breitbart et al., *"Topology Discovery in Heterogeneus IP Networks"*, *IEEE INFOCOM'2000*, Tel Aviv, Israel, 2000
[11] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, A. Silbershatz, *"Topology Discovery in Heterogeneus IP Networks: The NetInventory System"*, IEEE/ACM Trans. on Net., Vol. 12, n. 3, '04
[12] R. Govindam, H. Tangmunarunkit, *"Heuristics for Internet Map Discovery"*, USC/Information Sciences Institute, 2000
[13] P. Barford et al., *"The Marginal Utility of Network Topology Measurements"*, ACM/SIGCOMM Internet Measurement Workshop, Nov.'01.
[14] P. Dinda et al., *"The Architecture of the Remos System,"*, 10th IEEE Symp. on High-Perf. Dist. Comp. (HPDC01), Aug. 2001.
[15] http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html
[16] R. Teixeira et al., *"In Search of Path Diversity in ISP Networks"*, Computer Science and Engineering, University of California, 2003.
[17] B. Donnet, T. Friedman, M. Crovella, *"Improved algorithms for Network Topology Discovery"*, *Passive and Active Measurement Workshop '05*.
[18] D. Magoni, M. Hoerdt, *"Internet core topology mapping and analysis"*, in *Computer Communications*, vol. 28, no. 5, pp. 494-506, March 2005.
[19] F. Nazir et al., *"An Efficient Approach Towards IP Network Topology Discovery for Large Multi-subnet Networks"*, *11th IEEE Symposium on Computers and Communications*, 2006.
[20] M. Gunes, K. Sarac, *"Analytical IP Alias Resolution"*, Department of Computer Science, University of Texas at Dallas, 2006
[21] http://www.openview.hp.com
[22] http://www-306.ibm.com/software/tivoli/
[23] http://www.rocketsoftware.com/portfolio/netcure
[24] http://www.grid.unina.it/software/TD/
[25] N. Spring, R. Mahajan, D. Wetherall, *"Measuring ISP Topologies with Rocketfuel"*, Computer Science and Engineering, University of Washington, 2002.
[26] http://www.networkview.com/