TOPOLOGY DISCOVERY USING AN ADDRESS PREFIX BASED STOPPING RULE

Benoit Donnet
Timur Friedman
Laboratoire LiP6/CNRS
Université Pierre & Marie Curie
{benoit.donnet, timur.friedman}@lip6.fr

Abstract Recently, a first step towards a highly distributed IP-level topology discovery tool has been made with the introduction of the Doubletree algorithm. Doubletree is an efficient cooperative algorithm that allows the discovery of a large portion of nodes and links in the network while strongly reducing probing redundancy on nodes and destinations as well as the amount of probes sent. In this paper, we propose to reduce more strongly the load on destinations and, more essentially, the communication cost required for the cooperation by introducing a probing stopping rule based on CIDR address prefixes.

Keywords: Topology discovery, Cooperative algorithm, Doubletree, CIDR

1. Introduction

This is a time when highly distributed applications are in full expansion. Among others, we can cite *SETI@home* [Anderson et al., 2002] (probably the first one and the most famous), *FOLDING@home* [Larson et al., 2002] and the *Human Proteome Folding Project* [Bonneau et al., 2004].

The network measurement community is not an exception to this fashion. Some measurement tools have already been released as daemons or screen savers. In particular, in France, we have *Grenouille* [A. Schmitt et al., vice], a monitoring tool for broadband networks. More recently, we saw the introduction of *NETI@home* [Simpson and Riley, 2004], an application collecting network performance statistics from end-systems.

Tools allowing for topology discovery at the IP level, based on *traceroute* [Jacobsen, 1989], are becoming more distributed. There is a number of well known systems, such as *skitter* [Huffaker et al., 2002], *RIPE NCC TTM* [Georgatos et al., 2001] or *NLANR AMP* [McGregor et al., 2000], skitter being probably the most extensive one as it considers a set of between 20 and 30 monitors tracing towards a million destinations. The two others, TTM and AMP, con-

sider a larger number of monitors (on the order of one or two hundreds) but they trace in full mesh, avoiding to probe outside their own network. However, the need to increase the number of traceroute sources in order to obtain more complete topology measurement is felt [Clauset and Moore, 2004; Lakhina et al., 2003].

The idea of placing a tracerouting tool inside a screen saver, an idea first suggested by Jørg Nonnenmacher as reported by Cheswick et al. in [Cheswick et al., 2000] should allow one to quickly obtain a structure of a considerable size. Following this idea, a publicly downloadable measurement tool within a daemon, DIMES [Shavitt and Shir, 2005], has been released in September 2004. At the time of writing this paper, DIMES counts 4644 agents distributed across 78 countries.

Such a large structure has, however, inherent scaling problems. For instance, if all the monitors trace towards the same destination, it could easily appear as a distributed denial of service (DDoS) attack. Furthermore, such a system must avoid consuming undue network resources. However, before the development of the *Doubletree* algorithm [Donnet et al., 2005b], little consideration had been given to how to perform large-scale topology discovery efficiently and in a network-friendly manner.

Based on the tree-like structure of routes in the internet, Doubletree acts to avoid retracing the same routes through these structures. The key to Doubletree is that monitors share information regarding the paths that they have explored. If one monitor has already probed a given path to a destination then another monitor should avoid that path. We have found that probing in this manner can significantly reduce load on routers and destinations while maintaining high node and link coverage.

In this paper, we aim to improve Doubletree in order to more strongly reduce the impact on destinations. We propose to replace a stopping rule based on destination addresses with a stopping rule based on the *CIDR address prefixes* [Fuller et al., 1993] of destinations. The idea is to aggregate the destinations set into subnetworks, i.e. we filter each destination address and associate them to a subnetwork with the use of the CIDR address prefixes. Each monitor will probe all the destinations in each subnetwork. Futher, this proposal should also allow to reduce the amount of communication required by Doubletree. Indeed, instead of shareing a set of $(\mathrm{interface}, \mathrm{destination})$ pairs, monitors will share a set of $(\mathrm{interface}, \mathrm{prefix\_destination})$ pairs.

The rest of the paper is organized as follow: in Sec. 2, we introduce our prior work on the Doubletree algorithm. In Sec. 3, we present our methodology and our results. In Sec. 4, we present related work. Finally, in Sec. 5, we conclude and discuss further works.

2. Prior Work

Our prior work [Donnet et al., 2005b] described the inefficiency of the classic topology probing technique of tracing routes hop by hop outwards from a set of monitors towards a set of destinations. It also introduced Doubletree, an improved probing algorithm. Data for our prior work, and also for this paper, were produced by 24 skitter monitors on August 1st through 3rd, 2004. Of the 971,080 destinations towards which all of these monitors traced routes on those days, we randomly selected a manageable 50,000 for each of our experiments.

Considering first the inefficiency, we note that only 10.4% of the probes from a typical monitor serve to discover an interface that the monitor has not previously seen. An additional 2.0% of the probes return invalid addresses or do not result in a response. The remaining 87.6% of probes are redundant, visiting interfaces that the monitor has already discovered. Such redundancy for a single monitor, termed *intra-monitor redundancy*, is much higher close to the monitor, as can be expected given the tree-like structure of routes emanating from a single source. In addition, most interfaces, especially those close to destinations, are visited by all monitors. This redundancy from multiple monitors is termed *inter-monitor redundancy*.

While this inefficiency is of little consequence to skitter itself, it poses an obstacle to scaling far beyond skitter's current 24 monitors. In particular, inter-monitor redundancy, which grows in proportion to the number of monitors, is the greater threat. Reducing it requires coordination among monitors.

Doubletree is the key component of a coordinated probing system that significantly reduces both kinds of redundancy while discovering nearly the same set of nodes and links. It takes advantage of the tree-like structure of routes in the internet. Routes leading out from a monitor towards multiple destinations form a tree-like structure rooted at the monitor. Similarly, routes converging towards a destination from multiple monitors form a tree-like structure, but rooted at the destination. A monitor probes hop by hop so long as it encounters previously unknown interfaces. However, once it encounters a known interface, it stops, assuming that it has touched a tree and the rest of the path to the root is also known.

Both backwards and forwards probing use stop sets. The one for backwards probing, called the *local stop set*, consists of all interfaces already seen by that monitor. Forwards probing uses the *global stop set* of (interface, destination) pairs accumulated from all monitors. A pair enters the stop set if a monitor visited the interface while sending probes with the corresponding destination address.

A monitor that implements Doubletree starts probing for a destination at some number of hops h from itself. It will probe forwards at $h + 1$, $h + 2$, etc., adding to the global stop set at each hop, until it encounters either the

destination or a member of the global stop set. It will then probe backwards at $h-1$, $h-2$, etc., adding to both the local and global stop sets at each hop, until it either has reached a distance of one hop or it encounters a member of the local stop set. It then proceeds to probe for the next destination. When it has completed probing for all destinations, the global stop set is communicated to the next monitor.

The choice of initial probing distance h is crucial. Too close, and intra-monitor redundancy will approach the high levels seen by classic forward probing techniques. Too far, and there will be high inter-monitor redundancy on destinations. The choice must be guided primarily by this latter consideration to avoid having probing look like a DDoS attack.

While Doubletree largely limits redundancy on destinations once hop-by-hop probing is underway, its global stop set cannot prevent the initial probe from reaching a destination if h is set too high. Therefore, we recommend that each monitor set its own value for h in terms of the probability p that a probe sent h hops towards a randomly selected destination will actually hit that destination. Fig. 1 shows the cumulative mass function for this probability for skitter monitor `apan-jp`. For example, in order to restrict hits on destinations to just 10% of initial probes, this monitor should start probing at $h = 10$ hops. This distance can easily be estimated by sending a small number of probes to randomly chosen destinations.

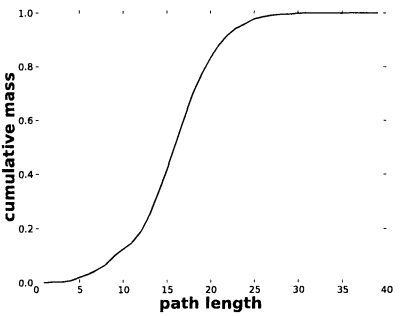


Figure 1. Cumulative mass plot of path lengths from skitter monitor `apan-jp`

For a range of p values, compared to classic probing, Doubletree is able to reduce measurement load by approximately 70% while maintaining interface and link coverage above 90%.

One possible obstacle to successful deployment of Doubletree concerns the communication overhead from sharing the global stop set among monitors. Tracing from 24 monitors to just 50,000 destinations with $p = 0.05$ produces a set of 2.7 million $(\text{interface}, \text{destination})$ pairs. As pairs of IPv4 addresses

are 64 bits long, an uncompressed stop set based on these parameters requires 20.6MB.

A way to reduce this communication overhead is to use *Bloom filters* [Bloom, 1970] to implement the global stop set. A Bloom filter summarizes information concerning a set in a bit vector that can then be tested for set membership. An empty Bloom filter is a vector of all zeroes. A key is registered in the filter by hashing it to a position in the vector and setting the bit at that position to one. Multiple hash functions may be used, setting several bits set to one. Membership of a key in the filter is tested by checking if all hash positions are set to one. A Bloom filter will never falsely return a negative result for set membership. It might, however, return a false positive. For a given number of keys, the larger the Bloom filter, the less likely is a false positive. The number of hash functions also plays a role.

In [Donnet et al., 2005a], we show that, when $p = 0.05$, using a bit vector of size 10^7 and five hash functions allow nearly the same coverage level as a list implementation of the global stop set while reducing only slightly the redundancy on both destinations and internal interfaces and yielding a compression factor of 17.3.

To reduce the load on destinations, we already investigated the concepts of *capping* and *clustering* [Donnet et al., 2005a]. The capping aims to impose an explicit limit on the number of monitors that target a destination. The clustering may be seen as a specialization of the capping by dividing the monitors into clusters, each cluster focusing on a different destination list. The real problem in theses mechanisms is how to assign monitors to destinations. In [Donnet et al., 2005a], we chose to work randomly but future work might reveal that a topologically informed approach provides better yield.

3. Doubletree with CIDR

3.1 Methodology

Skitter data from the beginning of August 2004 serves as the basis of our work. This data set is composed of traceroutes gathered from 24 monitors scattered around the world: United States, Canada, the United Kingdom, France, Sweden, the Netherlands, Japan and New Zealand. All the monitors share a common destination list of 971,080 IPv4 addresses. Each destination is probed in turn by each monitor. To cycle through the destination list, it can takes a few days, usually three. For our studies, in order to reduce computing time and hard disk space to a manageable level, we decided to work on a limited destination subset of 50,000 items randomly chosen amongst the whole set.

We conduct simulations based on the skitter data, applying Doubletree, as described in [Donnet et al., 2005b]. A single experiment uses traceroutes from all 24 monitors to a common set of 50,000 destinations chosen at random. Each

data point represents the average value over fifteen runs of the experiment, each run using a different set of 50,000 destinations. No destination is used more than once over the fifteen runs. We determine 95% confidence intervals for the mean based, since the sample size is relatively small, on the Student t distribution. These intervals are typically, though not in all cases, too tight to appear on the plots.

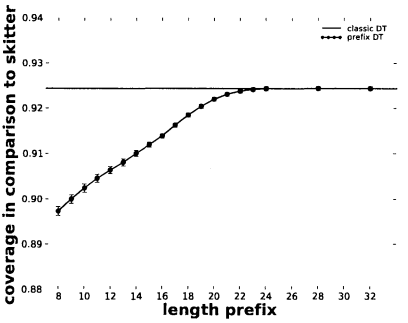
We use $p = 0.05$, which is a value that belongs to the range of p values that our previous work identified as providing a good compromise between coverage accuracy and redundancy reduction. We test all prefixes length from /8 to /24, as well as lengths /28 and /32 (i.e. full IPv4 addresses).

Each monitor probes each destination and records in the global stop set $(\text{interface}, \text{destination\_prefix})$ pairs instead of $(\text{interface}, \text{destination})$ pairs. Compared to classic Doubletree, we only change the global stop set stop rule. Each result considered is compared, in Sec. 3.2, with classic Doubletree and skitter.

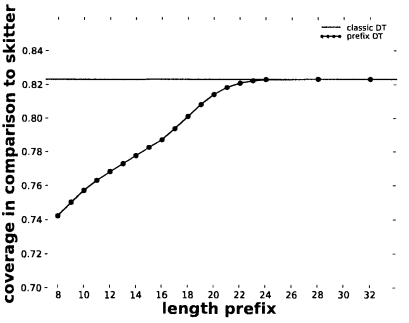
3.2 Results

Fig. 2 shows the main performance metric for a probing system: its coverage of the nodes and links in the network. It illustrates how the nodes and links coverage vary in function of the prefix length. A value of 1.0 (not shown here) would mean that application of Doubletree with the given prefix length had discovered exactly the same set of nodes and links as skitter. As already pointed out in our previous work, the use of Doubletree implies a small accuracy loss in the link and node coverage compared to skitter. The lowest level of performance is reached for the /8 prefix. In our data set, on average, there are thirteen /8 subnetworks. As these subnetworks are quite large, monitors are stopped early in their probing. The loss of accuracy, however, is not so dramatic. The link coverage is 0,742 instead of 0,823 and node coverage is 0,897 instead of 0,924. We believe that the coverage level is still high due to the way exploration is performed by the first monitor to probe the network. Indeed, this first monitor uses an empty stop set (by definition of Doubletree) and is thus never stopped in its exploration. We further note that performance improves with prefix length until reaching nearly the same accuracy as classic Doubletree with /24 prefixes.

We believe that the loss of accuracy, compared to classic Doubletree, is essentially located within the subnetworks containing destinations but also inside the core of the network, where duplicated links (and the associated nodes) are missed due to the prefix based stopping rule. Typically, probes reach a very few number of destinations in each subnetwork but, in general, they are stopped at the ingress routers. We miss thus essentially the vast majority of destinations located in a given subnetwork. However more nodes and links may be missed



(a) Nodes

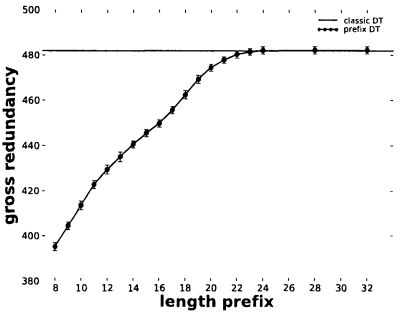


(b) Links

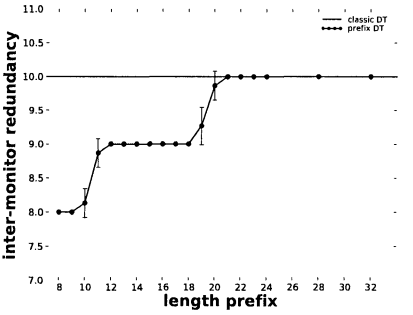
Figure 2. Coverage when using prefixes

if the network structure of the subnetwork is more complex, i.e. the subnetwork is not only composed of an ingress router that connects destinations with the rest of the network.

Doubletree aims also to reduce the load on routers. It would be a concern if the redundancy were to increase when introducing a prefix based stopping rule. The ordinates in Fig. 3(a) specify the gross redundancy on router interfaces, i.e. the total number of visits. The ordinates in Fig. 3(b) represent the inter-monitor redundancy. Inter-monitor redundancy, as defined in [Donnet et al., 2005b], is the number of monitors that visit a given interface. The maximum inter-monitor redundancy on destinations, not shown here, is 24. As the extreme values are the most worrisome, we consider redundancy on the 95^{th} percentile interface.



(a) Internal interfaces: gross



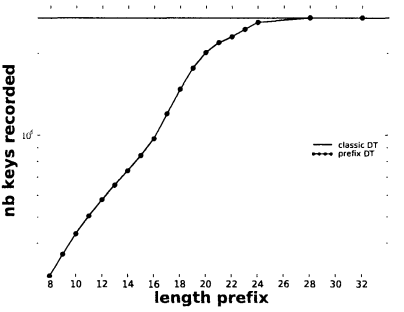
(b) Destinations: inter-monitor

Figure 3. Redundancy on 95th percentile interfaces when using prefixes

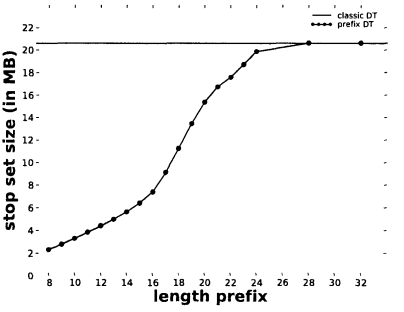
As shown by Fig. 3, the redundancy is not increased when using prefix based stopping rule. Further, for low prefix lengths, the redundancy is reduced for both destinations and routers.

Fig. 4 compares global stop set size. Fig. 4(a) shows the number of keys recorded in the global stop set (in log-scale) as a function of CIDR block prefixes. Fig. 4(b) shows the global stop set size in megabytes.

We can see that there is a strong reduction for low prefixes. For instance, if we consider a /8 prefix, the global stop set will only contain, in average, 302,854 keys. As each key is recorded as a 64 bit value, it corresponds to a stop set of around 2.31MB. Compared to the classic Doubletree, there is a compression factor of 8.9.



(a) Number of keys



(b) Megabytes

Figure 4. Global stop set size when using prefixes

In addition to the mechanism presented in this paper, we could also implement the global stop set as a Bloom filter without losing much coverage accuracy [Donnet et al., 2005a, Sec. 3].

	/8	/16	/24	/32
Prefix DT	2.31	7.40	19.87	20.61
Prefix DT with BF	0.361	0.689	1.192	1.192
Classic DT		20.61		
Classic DT with BF		1.192		

Table 1. Global stop set size comparison (in MB)

Table 1 compared the global stop set implemented as a set of pairs and as a Bloom filter. It also compares classic Doubletree with the mechanism presented in this paper. Concerning Bloom filters, we follow Fan et al's suggestions [Fan et al., 1998, Sec. V.D] for tuning the vector size and the number of hash functions to use.

We see that coupling the prefix based stop rule with a Bloom filter implementation of the global stop set introduces a very strong reduction in the global stop set size. For instance, using a /8 prefix stop rule gives, compared to classic Doubletree, a compression factor of 57.1.

4. Related Work

Very little work has been conducted on efficient measurement of the overall internet topology. This is in contrast to the number of papers on efficient monitoring of networks that are in a single administrative domain (see for instance, Bejerano and Rastogi's work [Bejerano and Rastogi, 2003]). The two problems are extremely different. An administrator knows their entire network topology in advance, and can freely choose where to place their monitors. Neither of these assumptions hold for monitoring the internet with end-host based software. Since the existing literature is based upon these assumptions, we need to look elsewhere for solutions.

Govindan and Tangmunarunkit [Govindan and Tangmunarunkit, 2000] have proposed the idea of starting traceroutes far from the source. Using a probing strategy based upon IP address prefixes, the *Mercator* system conducts a check before probing the path to a new address that has a prefix P. If paths to an address in P already exist in its database, Mercator starts probing at the highest hop count for a responding router seen on those paths. No results have been published on the performance of this heuristic, though it seems to us an entirely reasonable approach in light of our data.

The Mercator heuristic requires that a guess be made about the relevant prefix length for an address. That guess is based upon the class that the address would have had before the advent of CIDR.

Finally, Some authors [Siamwalla et al., 1998; Burch and Cheswick, 1999] have already suggested the idea of guiding topology discovery at IP level according to BGP information. They use BGP backbone routing tables in order to determine the destinations of traceroutes. For each prefix in the tables, they repeatedly generate a random IP address within that prefix. With the traceroute results to these destinations, they build a router adjacency graph. By probing only one destination per prefix, this technique may miss several nodes and links.

5. Conclusion

In this paper, we present an improvement to the Doubletree probing algorithm. By using stop rules based on address prefixes, we show that we are able to reduce load on destinations while maintaining an acceptable level of coverage accuracy. Further, if we use this simple mechanism with a global stop set implemented as a Bloom filter, we still reduce the global stop set size to very low proportions.

The next prudent step for future work would be to test the algorithms that we describe here on an infrastructure of intermediate size, on the order of hundreds of monitors. We have developed a tool called *traceroute@home* that we plan to deploy in this manner.

We also aim to improve Doubletree in order to guide probing with a higher level information. We plan to develop and experiment algorithms allowing Doubletree to realise more accurate exploration through the use of AS level topology and AS path information.

Acknowledgments

The authors are member of the traceroute@home project. This work was supported by: the RNRT's Metropolis project, NSF grants ANI-9986397 and CCR-0325701, the e-NEXT European Network of Excellence, and LiP6 2004 project funds. Mr. Donnet's work is supported by a SATIN European Doctoral Research Foundation grant.

Without the skitter data provided by kc claffy and her team at CAIDA, this research would not have been possible. They also furnished much useful feedback. Marc Giusti and his team at the Centre de Calcul MEDICIS, Laboratoire STIX, Ecole Polytechnique, offered us access to their computing cluster, allowing faster and easier simulations. Finally, we are indebted to our colleagues in the Networks and Performance Analysis group at LiP6, headed by Serge Fdida, and to our partners in the traceroute@home project, Mark Crovella, Jose Ignacio Alvarez-Hamelin, Alain Barrat, Matthieu Latapy, Philippe Raoult and Alessandro Vespignani, for their support and advice.

References

A. Schmitt et al. (ongoing service). La mæteo du net.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61.

Bejerano, Y. and Rastogi, R. (2003). Robust monitoring of link delays and faults in IP networks. In *Proc. IEEE Infocom*.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Bonneau, R., Baliga, N. S., Deutsch, E. W., Shannon, P., and Hood, L. (2004). Comprehensive de novo structure prediction in a systems-biology context for the archaea halobacterium. *Genome Biology*.

Burch, H. and Cheswick, B. (1999). Mapping the internet. *IEEE Computer*, 32(4):97-98.

Cheswick, B., Burch, H., and Branigan, S. (2000). Mapping and visualizing the internet. In *Proc. 2000 USENIX Annual Technical Conference*, San Diego, California, USA.

Clauset, A. and Moore, C. (2004). Traceroute sampling makes random graphs appear to have power law degree distributions. Technical Report arXiv:cond-mat/0312674 v3, University of New Mexico.

Donnet, B., Friedman, T., and Crovella, M. (2005a). Improved algorithms for network topology discovery. In *Proc. PAM 2005*, Boston, USA.

Donnet, B., Raoult, P., Friedman, T., and Crovella, M. (2005b). Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS 2005*, Banff, Canada.

Fan, L., Cao, P., Almeida, J., and Broder, A. Z. (1998). Summary cache: A scalable wide-area web cache sharing protocol. In *Proc. ACM SIGCOMM*.

Fuller, V., Li, T., Yu, J., and Varadhan, K. (1993). Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. RFC 1519, Internet Engineering Task Force.

Georgatos, F., Gruber, F., Karrenberg, D., Santcroos, M., Susanj, A., Uijterwaal, H., and Wilhelm, R. (2001). Providing active measurements as a regular service for ISPs. In *Proc. Passive and Active Measurement Workshop (PAM)*.

Govindan, R. and Tangmunarunkit, H. (2000). Heuristics for internet map discovery. In *Proc. IEEE Infocom*.

Huffaker, B., Plummer, D., Moore, D., and Claffy, k (2002). Topology discovery by active probing. In *Symposium on Applications and the Internet*, Nara City, Japan.

Jacobsen, V. (1989). traceroute.

Lakhina, A., Byers, J., Crovella, M., and Xie, P. (2003). Sampling biases in IP topology measurements. In *Proc. IEEE Infocom '03*.

Larson, S. M., Snow, C. D., Shirts, M., and Pande, V. S. (2002). FOLDING@home and GENOME@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*.

McGregor, A., Braun, H.-W., and Brown, J. (2000). The NLANR network analysis infrastructure. *IEEE Communications Magazine*, 38(5):122–128.

Shavitt, Y. and Shir, E. (2005). DIMES: Let the internet measure itself. cs.NI 050699, arXiv.

Siamwalla, R., Sharma, R., and Keshav, S. (1998). Discovering internet topology. Technical report, Cornell University, Ithaca, NY 14853.

Simpson, Jr., C. R. and Riley, G. F. (2004). NETI@home: A distributed approach to collecting end-to-end network performance measurements. In *Proc. Passive and Active Measurement Workshop (PAM)*.