PathCrawler: Automatic Harvesting Web Infra-Structure

Cesar Marcondes, M.Y. Sanadidi and Mario Gerla
Computer Science Dept.
University of California, Los Angeles
Los Angeles, USA 90050
Email: {cesar,medy,gerla}@cs.ucla.edu

Ramon S. Schwartz, Raphael O. Santos and Magnos Martinello
Computer Science Dept.
Universidade Federal do Espírito Santo
Vitória, ES, Brazil 29060-900
Email: {ramon,santos,magnos}@inf.ufes.br

***Abstract*—As network topologies have grown in size and complexity, it is becoming a daunting task for network administrators to keep track the capacity dimensioning of newly installed web-servers within a single or multiple providers. In fact, monitor capacity dimensioning is not a trivial activity since network state changes rather frequently, in particular, in academic environments. In this paper, we describe estimation algorithms and the software architecture of an efficient network management suite to automatically mine path capacity and minimum delays from a venture point to a set of observed web servers. The principle of the suite is based on packet dispersion techniques and repetitive non-intrusive measurements. We provide analytical insights, simulation results and some real case studies where we argument about the correctness, accuracy and usefulness of the suite in the context of management and operation of complex IP based networks.**

Keywords : Network Management , Measurements and Monitoring , Performance Evaluation, Inline TCP Estimators, Webcrawler.

I. Introduction

The today's ISP (Internet Service Provider) competitive market turns mandatory that popular web-servers be placed in strategic positions of great peer connectivity and large capacities. Thus, such strategic placement decision depends upon how updated are topological maps of the network resource infra-structure. In other words, the paths that carry users requests from the network core to web server must be properly dimensioned in terms of capacity, shortest distance and so on. However, given the dynamic nature of today's IP networks, it is excessively demanding to perform such activity by hand. This is particularly true in academic networks, where there is freedom to deploy new servers, at will, without prior consultation. As these servers become available through search engines, the underlying network infra-structure plays an important role on accommodating more users requests.

These basic observations regarding the utility of mining information on network topology motivated us to implement a network management suite that automatically harvest the underlying capacity information from a local network and keep an up-to-date graph-based map of the delay-capacity infra-structure from a venture point to a set of web-servers. This suite is based on prior insights from packet pair methods [1] implemented in an instrumented Linux kernel [2] and is built upon an open software architecture comprised of web-crawler [3], statistical analysis [4] and topology visualization [5].

One of the main differences of our discovery suite, which is called PathCrawler, when compared with others tools that build full physical maps through SNMP (Simple Network Management Protocol), such as NetInventory [6], is that our suite has a totally indirect and statistical approach to gather information. Hence, instead of relying on management traffic (SNMP), usually prone to errors due to MIB incompatibility, filtered ports and turned off daemons, we use delay and path capacity inference from inside regular TCP connections made by PathCrawler. In particular, our approach can be considered extremely low profile and non-intrusive since it could be piggybacked with normal network management operations such as internal web indexing or caching. Additionally, the Robots Exclusion Standard [7] is used in order to respect web server policies.

In what follows, we present, in Section II, a detailed description of the estimator algorithms incorporated in PathCrawler. Such algorithms are analyzed mathematically in terms of estimation convergence probability, and they are further studied through simulations about the convergence time. Section III describes the overall architecture of the system, including the integration with a webcrawler, a description of the modified kernel TCP implementation, and a description of the statistical data processing and graph plotting. Finally, on Section IV shows several case studies conducted at Federal University of Espirito Santo (Brazil) based on measurement results, where one can see the accuracy of the tool, its' robustness to high loads of cross-traffic and possible usages in network policy identification. We finalize the paper with a literature review in Section V and conclude the paper with some outline of our future work in Section VI.

II. Estimators

The development of effective algorithmic solutions for the estimation of path characteristics, such as capacity and propagation delay, in a reliable and timely way, is a tough challenge. The algorithms have to be (*i*) accurate, (*ii*) computationally inexpensive in order to scale and run in real-time, and (*iii*)

978-1-4244-2066-7/08/$25.00 ©2008 IEEE

non-intrusive since they cannot consume excessive disk space or large amounts of bandwidth, and above all, (*iv*) they cannot assume that the web server will cooperate.

A. Capacity Estimator

Given such restrictions, the path capacity estimation method (aka. bottleneck link estimation) of our choice is based on packet pair dispersion techniques, specifically on CapProbe [1] method. This method permits the discovery of the slowest link capacity from the measurement point to the web server. The key insight is that if a packet pair passes through the bottleneck back to back without being impacted by queueing, the dispersion t_b between them will reveal the bottleneck (Fig. 1). Therefore, the original CapProbe idea is to send a group of packet pairs out-of-band onto the network link. Each of the packet pairs obtains one possible sample of the network capacity. Moreover, by monitoring the end-to-end delay of the packet pairs, the method can find the ones that experience minimum delay, hence CapProbe can filter out incorrect capacity estimations that may have queued somewhere along the path. In the case of both probing packets of size MSS do not suffer any queueing delay (also called minimum RTT sum), the estimated capacity can be calculated as follows:

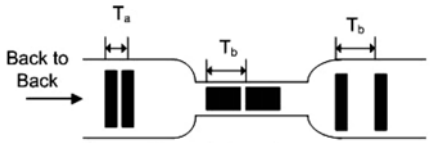

Fig. 1. Packet pair dispersion overview

$$\hat{C} = \frac{MSS}{T_b} \tag{1}$$

For our purpose, Capprobe alone is not sufficient, since we would like to embed the path capacity estimation mechanism inside TCP because TCP is the most pervasive type of traffic in the internet, and it is usually not filtered. Additionally, using TCP unlike CapProbe, it is not necessary to generate out-of-band traffic, TCP naturally generates packet pair samples during the *slow start* phase. In this phase, the TCP sender increases its *congestion window* by one segment size for every acknowledgment packet received and, therefore, it triggers the transmission of a packet pair sent back-to-back to the receiver. Based on these micro-bursts, our method consists on implementing delay monitors inside the TCP receiver [2] (the capacity agent component, Section III) and controlling the packet pair transmissions to find the bottleneck such as to resemble CapProbe.

B. Likelihood of Non-Queuing

Critics of packet pair dispersion techniques usually claim that packet pairs that experience no queueing delay is a rare event. However, it is important to differentiate "no queueing" delay probability (or probability of finding the bottleneck buffer instantaneously empty) from "zero utilization" of the bottleneck. A non-queued packet pair can happen quite easily depending on the cross-traffic type conditions coming to the bottleneck, even if the overall load is high (i.e. 80%).

In CapProbe, the authors claim that based on "no-queuing" probabilities for several cross-traffic types, like Poisson, CBR (Constant Bit Rate) and Pareto On/Off, can be obtained rather easily if enough capacity samples are obtained. Our approach relies on analytical modeling in order to obtain the probability of no-queueing depending on several cross-traffic types. We summarize their results on equations (2),(3),(4). Note that, λ represents the arrival rate coming to the bottleneck queue, μ the bottleneck service rate, τ is the dispersion interval of individual packet pair, tx is the deterministic transmission time of a cross-traffic in CBR mode, and finally $\hat{t}$ is the mean ON/OFF time periods of Pareto sources.

$$p_{link}^{poisson} = (1 - \frac{\lambda}{\mu})e^{-\lambda\tau} \tag{2}$$

$$p_{link}^{cbr} = max(0, 1 - \lambda(tx + \tau)) \tag{3}$$

$$p_{link}^{pareto(on/off)} = 1 - \frac{\lambda}{\mu} - \frac{\tau}{2\hat{t}} \tag{4}$$

Thus, if we want to express the expected number of packet pairs necessary to obtain a non-queued packet pair sample (also called "good sample") that passes without being queued along the path. We can calculate this number N assuming independent trials as follows:

$$N = \sum_{k=1}^{\infty} k.p_{link}.(1 - p_{link})^{k-1} = \frac{1}{p_{link}} \tag{5}$$

In the following, we provide one example on the number of samples necessary to find a "good sample", or converge to capacity. Suppose the traffic-type is Poisson, and suppose the bottleneck link speed is 40Mbps while the webserver machine is 100Mbps. Using an hypothetical 5000 bytes packet size (to ease calculations), clearly, the dispersion induced by the bottleneck rate is 1 msec. On the other hand, the packet pair dispersion starting from the webserver point will be 0.4 msec.

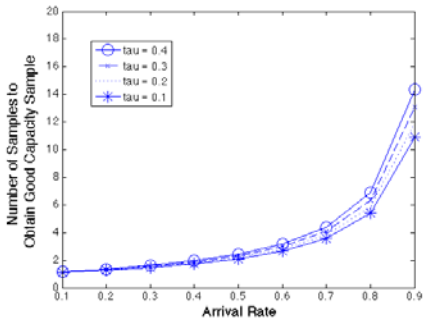

Fig. 2. Expected Number of Samples based on Poisson as a function of τ

As we change the arrival rate, and thus the load by fixing the service rate, N increases from one packet pair sample to almost 15 samples (Fig. 2). This provides a numerical support to obtain a good sample. In fact, as the difference between

bottleneck and webserver link speed increases (μ = 1 ms and τ varies from 0.4 to 0.1). Therefore, in the case of μ = 1ms and τ = 0.1ms, the bottleneck is slower by one order of magnitude, which implies that the pair samples necessary drops to 10. This way, even a small web transaction of about 20KB would be sufficient to obtain a good sample. Similar results, up to 20 samples, for other traffic types such CBR and Pareto On/Off, can be found in [1].

C. Embedded Capacity Estimation in Slow Start

Although abundant in the *slow start* phase, the number of actual good packet pair samples might be affected by the exponential growth of the *congestion window* which occurs at each RTT cycle. Assuming no cross-traffic, all the packets that belong to one RTT cycle will be sent in trains. These packets, except for the first and second ones, will be clearly queued, a behavior we refer here as **packet train self-interference**. In order to clarify this point, let us define the train of packets that is generated during the slow start phase cycle k as:

$$cwnd\_train\_dispersion(k) = 2^k \times \tau \quad (6)$$

$$where\ \ k = k + 1, every\ \ RTT\_cycle \quad (7)$$

$$\tau = \frac{MSS}{Capacity} \quad (8)$$

As the congestion window increases, the dispersion between the beginning and the end of the train increases exponentially. When the train reaches the bottleneck queue, each individual packet pair will be impacted by each other unless they are separated in time. Thus, it becomes mandatory that the capacity estimation method solves this problem by using pacing techniques. Usually pacing techniques are applied to TCP connections, when it is important to space the transmission of packets evenly according to a determined rate in order to avoid shallow buffers like the ones found in satellite links. In our case, since our capacity estimation algorithm has to be implemented within a webcrawler, more specifically, inside a TCP receiver, this control is carried out by delaying the transmission of acknowledgment packets (Fig. 3). Hence, effectively pacing in time the transmission of packet pairs coming from the sender.

Despite this pacing solution, one legitimate question raises about what would be the **right interval to space out ACKs** such that capacity estimation is not compromised by train self-interference and the download can still happen in a reasonable period?

D. Pacing Based on Analysis of Queue Empty Interval

If one can choose the best interval to send packet pairs, it should be on average the same as the length of time between empty instants of the bottleneck queue. Otherwise, if packets pairs are sent too close, the packet pair will likely find the queue occupied. Thus, we conducted a study using simulation of the time between empty queues in a more realistic Internet scenario.

In order to more closely mimic real life Internet traffic, we model cross traffic using Pareto aggregated sources. The Pareto

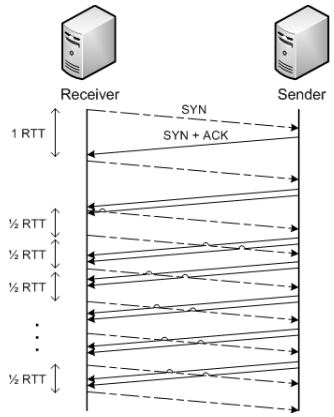

Fig. 3. Pacing algorithm

distribution is a heavy-tailed distribution that has properties that are qualitatively different to commonly used (memory-less) distributions such as the exponential, normal or Poisson distribution. It has been shown in [8] that aggregate Pareto distribution is well suited model for Internet traffic in real world scenarios, since only a small percent of TCP Internet traffic is long-lived, while the rest consists of short, bursty traffic.

Our simulation scenario was executed using the popular ns-2 simulator [9]. The topology consists of 1,2 and 4 Pareto sources sending to receivers (Fig. 4) for 300 seconds. The bottleneck is 1Mbps link with one-way delay of 26 msecs, the access links are 1Gbps with 1 msec delay, and the buffer size is set according to the bandwidth-delay product. The set up uses the specific parameters as shown in Table I. The Pareto traffic parameters are taken from [10] but the sending rate is adjusted according to our topology. In the real world, as a rule of thumb, Internet service providers (ISPs) often impose an operational limit on utilization of the bottleneck link to half of the capacity before deciding to upgrade. Thus, the amount of Pareto traffic is slightly less than half that of the capacity of the bottleneck link so as not to overwhelm the narrow link.

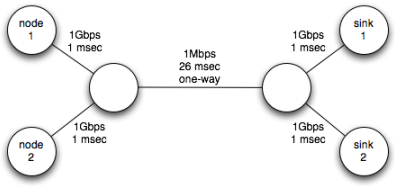

Fig. 4. Simulation Topology Used to Study Time Between Empty Bottleneck

We monitor the queue size and time between empty queues in this scenario. We obtain the distribution of the interval between clear queues in each case and plot out the cumulative distribution function (CDF) as in Figure 5. We observe from the CDF that in the case of two or four flows, 30 ms is sufficient to clear the queue 90% of the time. Note that one-way delay in this case is 26 msecs. On the other hand, in the case of one flow, different from aggregate Pareto sources, 30 ms clears the queue more than half of the time. Thus, we

Number of flows	1,2 and 4
Burst time	500ms
Idle time	1sec
Aggregate Tx rate	300 Kbps
Packet size	200 bytes
Shape	1.4

TABLE I
SIMULATION SETUP FOR PARETO CROSS TRAFFIC

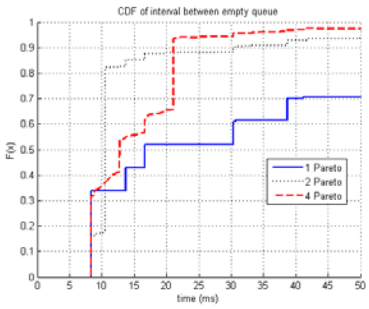

Fig. 5. Cumulative distribution function of the length of time between empty bottleneck queues for 1, 2, and 4 Pareto flows

conclude that 30 ms (about half RTT) is a reasonable value to use as our pacing interval in this network scenario.

We further validated the TCP capacity estimation and the benefits of pacing the pairs using simulation. The results present good accuracy and fast compared to without pacing. Unfortunately though, we omit them due to space constrains and we refer to [2] for more realistic results.

In addition, there are other reasons to set the sending rate to be one acknowledgment packet (ACK) per half RTT. Since the RTT/2 value represents the *one-way delay* of the connection, theoretically, at the moment an ACK leaves the receiver, the transmission of a packet pair triggered by a previous ACK would have occurred, and yet it will not induce timeout events at the sender. Reviewing Figure 3, we can see that our pacing algorithm starts with an initial RTT value estimated from the 3-way-handshake process. In the sequence, time gaps of RTT/2 are inserted between each acknowledgment packet transmission in such a way the corresponding packet pairs arrive separately from each other.

In order to conclude the estimation section, we also assume that once we identify one bottleneck using packet pairs with minimum delays, the *propagation delay* of this path is just a by-product of the capacity estimation itself. Besides, whenever we build topological maps, minimum RTT information can be augmented with ICMP probes (traceroute) to uncover all the intermediary routers and their distances (if allowed).

III. IMPLEMENTATION

We have implemented the path characteristics estimation algorithms presented in the prior section within the context of the **PathCrawler** system, which comprises a flexible module to be incorporated in any network management suite. In this section, we describe the developed system architecture in more detail. We divide the rest of this section to describe: the low level module, directly related to several changes in the Linux kernel, and the high level module that groups several applications to collect path information.

We start by describing the key features of **PathCrawler** as follows:

- Automatically obtain a set of web servers addresses, described by a domain, using search engines like Google API, usually ordered by popularity.
- Automatically obtain path capacity (narrow link) and minimum response time estimation from a central point at ISP core network to a group of N web-servers by fetching pages using a robot crawler.
- Post-statistical analysis of the experiments, in order to summarize the findings.
- Topological map augmented with the infra-structure information from the venture point (measurement point) to a group of N web-servers.

All these features are mapped in the main system modules: Low Level and High Level modules. The low level is responsible to infer characteristics, capacity and delays, from within normal TCP connections and pass this information back to the crawler application by regular *getsockopt*. On the other hand, the high level module use this information to prepare detailed statistical and topological maps related to the web-servers analyzed.

A. Low Level Module - TCP Kernel Modifications

One advantage of the low level implementation, i.e. modifying the Linux kernel to perform packet pair capacity estimation, is that it allows to be done inside the TCP code. Thus, instead of just a socket abstraction, the kernel space code actively allows the estimation algorithms to work with the highest precision and packet delivery control.

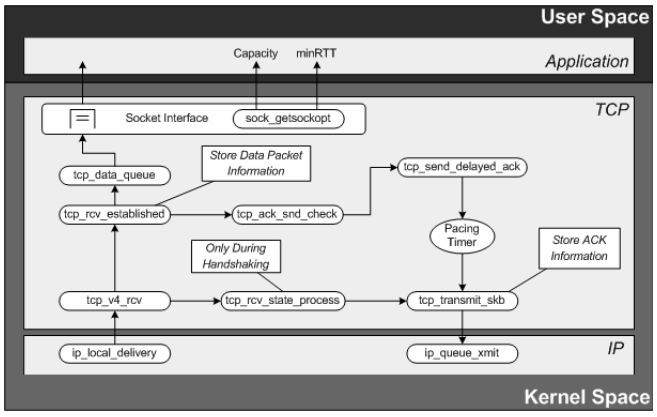

Fig. 6. Kernel Modifications

In Figure 6 we can see that the main modifications of the TCP Linux kernel stack are concentrated in the functions responsible for the TCP input process. In a regular connection, the packets are delivered to the TCP stack through the function *ip_local_delivery*. Before the connection is established, i.e., during the 3-way-handshake algorithm, these packets pass by *tcp_v4_rcv* and are handled by *tcp_rcv_state_process*,

which we use to perform our first RTT estimate by subtracting the SYN-ACK packet arrival time from the SYN packet delivery time. The following packets are received by *tcp_rcv_established*, where all the necessary information for our capacity estimation is stored in a constrained memory space. Meanwhile, the function *tcp_ack_snd_check* is called to verify whether or not an ACK packet should be sent as a response to the data packet. Normally, this would lead to a prompt reaction of TCP. Instead, in order to obtain better results in our collection we call the *tcp_send_delayed_ack*, so that a pacing interval is set and ACKs are sent smoothly. Specifically, this is done by setting the *ato* value (Acknowledgment timeout) to RTT/2. Guaranteeing that the packet pairs are not sent back to back. Whenever the pacing timer is up, an ACK packet is prepared and also its information is stored in our structure. In the sequence, the packet is sent by means of *tcp_transmit_skb* directly to the IP stack.

Another difficulty is to perform an accurate RTT estimation for each packet received. From the receiver side perspective, this means to correctly associate each ACK sent, with its respective data packet pair coming from the sender. Furthermore, TCP has been designed to concede the main congestion and flow control at the sender, which occasionally difficult packet association at the receiver. To address such issues, we carry out the RTT estimation of packet pairs by associating ACK with data packets by means of the TCP timestamp option [11]. The timestamp option, currently a default option in many TCP implementations (including Linux and Windows) and created for both hosts, is an alternative of measuring RTT.

We further sophisticate the use of TCP timestamp by means of an ACK-Packet Pair association algorithm (Fig. 7). During the 3-way-handshake phase, the receiver initiates the connection sending the current timestamp clock value in the *TSval* field [11]. Whenever the sender agrees to it, the timestamp sent previously is echoed and the receiver prepares another packet with a new timestamp. At this moment, the current timestamp clock should be once again the next *TSval*, thus, in order to provide precision and correct association measurements, we replace the timestamp by a *BaseTS* which we increment by the constant one at each ACK transmission. Therefore, we avoid dependence on timestamp granularity determined by the operating system and instead we use our own instruction counter for high precision delay measurements. Moreover, the increment i, in $BaseTS + i$, will match the exact position in our constrained vector where all necessary information for RTT and capacity estimation is stored. Thereafter, every packet pair transferred by the sender will carry the timestamp value of the corresponding ACK responsible for triggering it.

Despite the considerable modification of timestamp option values, the primitive RTT estimation of the TCP receiver is not actually affected. As soon as each packet arrives, the timestamp $BaseTS + i$ echoed from the sender is used for our packet association and then immediately replaced by the original value TCP expected to receive.

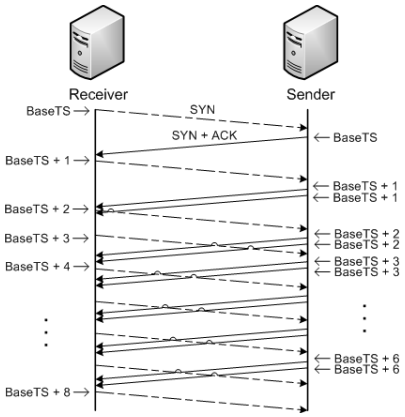

Fig. 7. Timestamp association

B. High Level Module - Web Analysis Application

This module consists on the proper network management application and is executed at user level. The main objective is to perform the infra-structure evaluation. Therefore, it uses several integrated sub-components, as described in Figure 8.

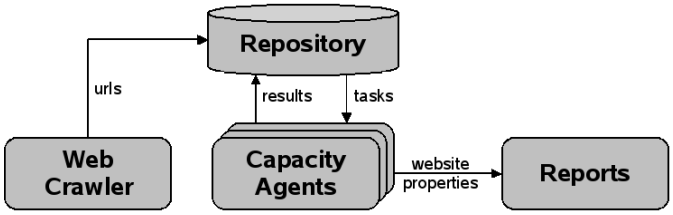

Fig. 8. Application Architecture

As it can be seen in Figure 8, the whole process is composed by several phases: (*i*) search the list of desired web servers to measure, and fetch of pages from their URLs; (*ii*) path evaluation on a per server basis; (*iii*) overall report for all servers selected. Additionally, the first two phases are done in parallel, thus without creating a system bottleneck. The phases are accomplished by the following four components:

1) Web Crawler : This component was developed as an extension of HarvestMan [3], a popular python webcrawler. It has a set of functions to parse an initial start-up URL and then as it fetches more pages, the webcrawler parses more URLs in a repetitive manner. Note that it finds out new elements (servers) based only on fetched URL, as opposed to traditional SNMP discovery approach which relies on remote agents.

2) Repository : Every URL found by the webcrawler is redirected to this component. All the links are then classified according to the domain name and web server IP address. This repository is responsible to pre-organize the tasks of capacity estimation to be performed by the ”Capacity Agents”.

3) Capacity Agents : This component performs downloads from the specified web server (fetching pages, images, files, etc), thus recording at the end of each transaction, the capacity and minimum RTT obtained by the instrumented kernel (Low Level Module). Since, the estimation method is bounded by

the network itself, we cannot induce application interference by opening parallel connections. Thus, the component order the transactions in sequence and consider the task completed as it reaches a number of packet pairs that allow ”good sample” convergence. In addition, for statistical robustness several measurement are repeated according to a repeat setting. Once the process is complete, the data about the web-server is made available to the Report component.

4) Reports : **PathCrawler** creates a sequence of reports to analyze every set of measurements. The first report aggregate all the servers and their respective estimations per sample. The second report is the result of a post-processing script using Octave [4], and it produces statistical graphs that show the average and other moments of the samples behavior on path capacity and minimum delay. Finally, the last report consolidates the set of measurements by creating a visual network-graph that show the central point and the connectivity (in terms of intermediate nodes, capacity and RTT) to the desired web-servers. Such visual graph is create by using the tool Otter [5]. We will present these reports when we show our case study in the next section.

IV. Experiments for harvesting web infra-structure

In this section, we describe an extensive “web infra-structure harvesting” campaign using the developed PathCrawler. All the experiments were conducted from July to September 2007 for the usual period of activity (daytime) as well as nighttime. The central measurement point was chosen to be located at POP-ES (National Research Network Point of Presence). In fact, the choice of the measurement point is justified by the fact that it is the best connection to the Internet from the Federal University of Espírito Santo (UFES).

Measurements were collected exploring web path diversity to a set of local and wide area network. The set of local measurements was conducted to estimate the capacities in the university network. These measurements were important not only to evaluate the connectivity of the departments in a complex IP based network, but also to validate our estimation techniques since it is possible to check each estimated capacity with the real value. We have done several internal experiments within UFES. In more than 98% of the experiments the error was below 2% from the real physical capacity. We have tested successfully to some webservers of different departments with capacities varying from 5Mbps to 100Mbps.

For the experimental environment, we installed the PathCrawler integrated on a modified Linux 2.6.18 kernel in one simple host, a Pentium III 1.0GHz processor, 512MB RAM, 100Mbps Fast-Ethernet. The machine was connected directly to a Cisco3745 with multiple 100Mbps interfaces, one of the main access switches to the UFES 100Mbps backbone.

A. Local experiments and its topology

The local experiments starts with the UFES network case, in terms of path capacity and propagation delay. Figure 9 presents such information. From Figure 9(a), we can observe that 15% of the measured sites had capacity [1] less or equal to 10 Mbps and 85% had narrow link capacities equal to 100Mbps. In terms of delay (Fig. 9(b)), 85% of the measures sites had end-to-end delays of less than 2 msec, therefore connected to the same switch, or very close to it (less than 6 msec). Although these results are clearly simple, recall that the aim was to infere UFES infra-structure checking each estimated capacity with the real value. Indeed, the measurements allowed to obtain the topology map presented in Figure 10. It is important to note that the obtained capacity measures corresponded exactly to the real capacity value. A particular remark is related to the site with capacity of around 5 Mbps (the thinnest line in Figure 10). This value of capacity is explained by a rate limit parameter setting at the respective interface of the router (a management policy used to regulate traffic due to P2P abuse).

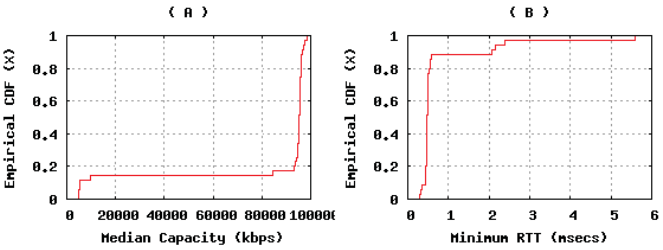

Fig. 9. Network path capacities and delay at UFES

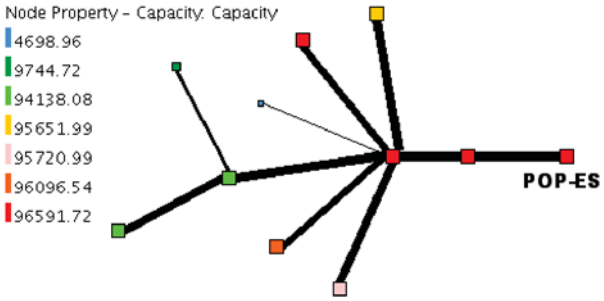

Fig. 10. UFES Topology obtained from PathCrawler

B. Large-scale experiments and its topology

For the large scale measurements, our focus was oriented on the domain ’.org.br’ because it is a domain often well connected to the academic network which is particularly important to know in the context of the Brazilian National Research Network (RNP), equivalent to the Internet2 in US. The connectivity of the UFES backbone with academic networks, the case of all sites in the domain ’org.br’, goes through an access link to RNP of 34Mbps which obviously allows only to measure slower link speeds.

The measurement bulk consisted of 100 estimations per evaluated web server and each complete experiment for the domain .org.br took 7 hours. Table II presents some extra care on these measurements. A web server supports either persistent connections or not persistent. In the case of non-persistent the minimum content length is 20K bytes because

[1]The median is used as the descriptor that is not sensitive to the outliers

we were interested in the capacity estimation over the first 10 packet pairs (as soon as possible). Otherwise, the acceptable content length is at least 2K bytes just to fill one packet pair, completing 10 pairs through downloading the content over and over in the same persistent connection. Thus, we filtered a subset of sites that provide the desired features for accuracy, even though the suite can work in more general mode.

Total IPs	Web sites with timestamp option	Web sites with persistent connections	Number of evaluated sites
805	463	431	388

TABLE II
DESCRIPTION OF THE MEASUREMENTS FOR THE .ORG.BR DOMAIN

Our analysis of the collected data starts by presenting the heterogeneity of the sites belonging to the domain .org.br focused on path capacity and propagation delay. Fig. 11 presents such information. Fig. 11(a) shows that around 25% (97 sites) of the measured sites had narrow link capacities less or equal to 10Mbps. The rest 75% had capacities around 34 Mbps clearly limited by the measure point capacity. In terms of delay, from Figure 11(b) can be seen that 65% of the websites (252) had an end-to-end delays of less than 50 msecs, therefore geographically located close to the measure point. The rest 35% (136) had a end-to-end delay longer than 50 msecs delay (until 200 msecs), capturing servers geographically far from the measure point.

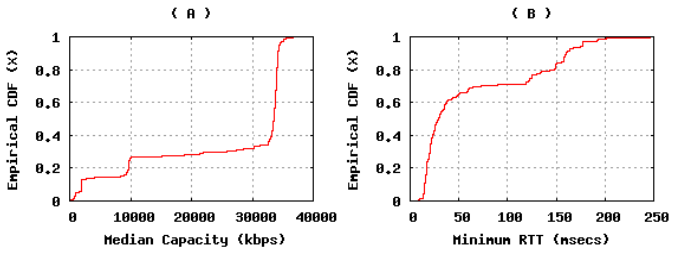

Fig. 11. The domain org.br capacity and end-to-end delay

A result that deserves special attention is the topology map of the domain .org.br. In fact, we are interested in providing a large scale view in order to determine e.g. regions with shared bottenecks, substantial differences on capacities at the same router, etc. Note that shared bottlenecks seen from the backbone point of view are not necessary known by the web site designers. Fig. 12 shows the org.br topology obtained from the measure point view. The cooler is the color, the lower is the capacity. Such analysis allows to map regions (clusters with common capacities) such that anomalies can be automatically detected (e.g. router configuration problems).

Although in an initial analysis, it seems to be complicated to mine some information, a subset of the domain has been taken in order to extract more usefull information. Fig. 13 shows such a subset in which we can point out (by inference) the points where capacities are distinct. For instance, the router $R1$ has two distinct capacities, 253 Kbps (DSL) to the site www.asclaras.org.br and a capacity of 2 Mbps to the site www.imsear.org.br, but the end-to-end delay for both sites are basically the same. Still on the same branch, a shared common capacity of 2 Mbps is observed to the sites www.imsear.org.br and www.oab.org.br, but in this case the location of the shared common capacity is either on $R2$ or $R3$.

Taking the case of router $R4$, the end-to-end delays are quite similar to the sites www.faet.org.br and www.rits.org.br, but the difference between capacities (www.faet.org.br 875 Kbps, and www.rits.org.br equal or higher than 34 Mbps) are more substantial compared to $R1$. These results lead us to claim some possibilities to explain such a significant capacity differences. First, the web site is aware about its connectivity seen from the backbone perspective because it has a router interface limited or due to a physical link limitation. Second, there are routers with different interfaces capacities to sites effectively close each other. Third, assuming that there is no limitation at both sides, there may be a configuration problem in the router interface. In this last case, it can be a rate limit configuration either at the router interface (as mentioned previously in the case of UFES network) or a negotiated SLA (Service Level Agreement).

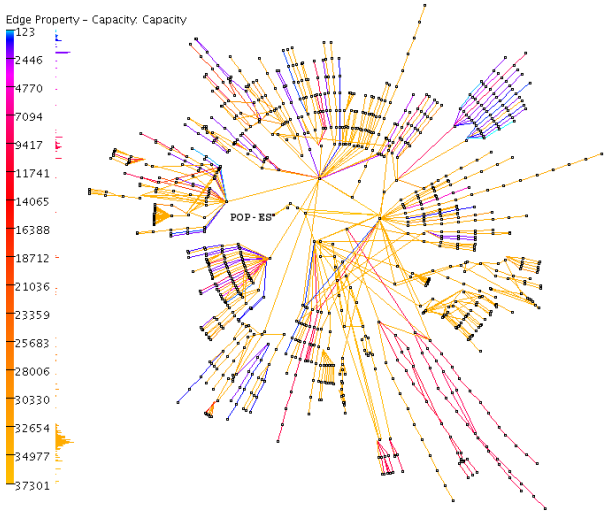

Fig. 12. Org.br topology

C. Measurements accuracy and robustness

The objetive of the following results is to show that high loads of the network link does not affect the accuracy of the obtained capacities. Fig. 14 shows three experiments of the domain .org.br, each one took on average 7 hours (total of 21 hours) and were executed for distinct periods of time. In Figure 14, the empirical CDF is presented for capacity and RTT for each experiment. The experiments were subjected to different loads as it can be seen in the figure since we align the CDFs with the respective time ranges of the MRTG graph providing the link utilization.

A first observation is that the measurements are not intrusive, i.e., a measurement campaign does not increase the usual load of the link (experiment 1 and respective capacity/RTT CDF left alignment). A second important remark is that even

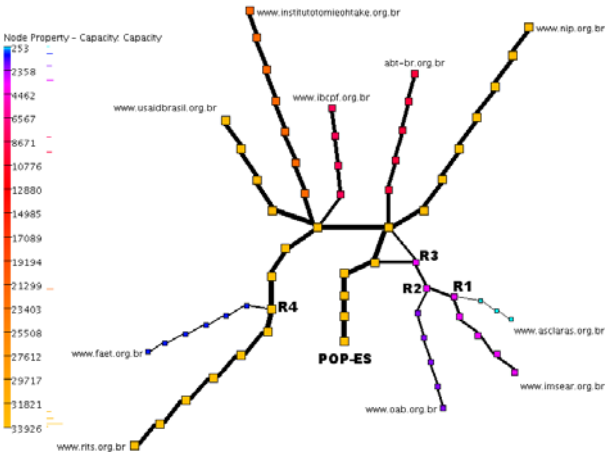

Fig. 13. A subset of the org.br topology with capacity details

if the load has been high, experiments 2 and 3 as shown in fig. 14, all the capacity CDFs obtained at distinct periods of the day can be seen as being very similar. Moreover, the capacity measurements are competing with the inbound traffic (green in the figure) since it is executed at the downstream flow. This result reinforces the fact that the proposed estimation method keeps its robustness even in the presence of high loads.

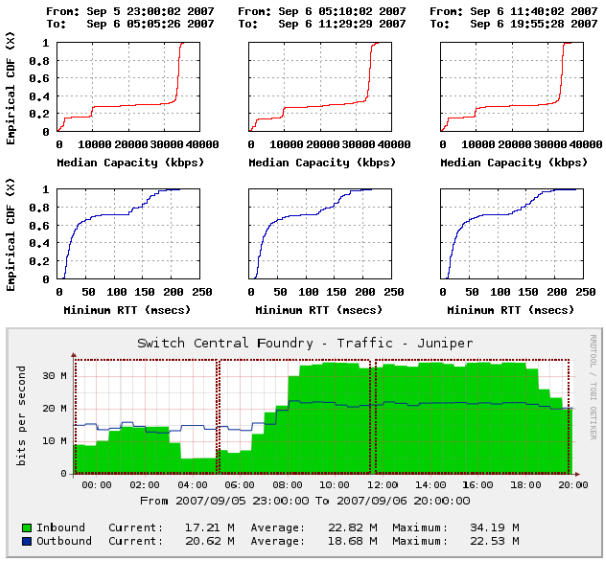

Fig. 14. Link utilization and Empirical capacity CDFs

V. Related Work

Network-layer discover algorithms are present in many network management systems such as HP OpenView and IBM's Tivoli. However, these rely heavily on SNMP that, in general, has interoperability problems due to vendor-specific extensions on MIBs. Thus, the advantage of estimation provide a robust response to heterogeneous network environments where SNMP solutions are not very useful. A first step on providing estimations to infer path characteristics come from PathChar [12], but it is a rather outdated technique and it does not have a software management infrastructure like ours. Other related work focus on discovering the Internet infrastructure using route and BGP information like Skitter [13] or a combination of RTT and SNMP like NetInventory [6] and traceroute based like Netdimes [14]. Therefore, as far as the literature review concern, there is no management tool that can provide facilities to harvest capacity estimation, in such a methodic fashion, as PathCrawler can do.

VI. Conclusions and Future Work

In this paper, we present a management suite that can harvest path capacities from a central point (POP-ES) to a desired set of webservers. The underlying algorithms, such as the capacity estimation, and embedded estimation in TCP, are shown in detail and validated analytically and through simulations. Some tests cases are discussed where we provide arguments in favor of the correctness, accuracy and usefulness of PathCrawler. In the future, we intend to study capacity update dynamics in a large network like RNP.

VII. Acknolegments

This work has been partially funded by FAPES/MCT/CNPq/CT-INFRA #36316008/2007. Magnos Martinello is supported by a scholarship from CNPq/Brazil (PDJ) #151606/2007-2. Raphael O. Santos supported by a master scholarship from CAPES/Brazil. Cesar Marcondes is supported by CAPES/Brazil #1283-02-2.

References

[1] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "Capprobe: a simple and accurate capacity estimation technique," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 67–78, 2004.

[2] C. Marcondes, M. Gerla, M. Y. Sanadidi, M. Martinello, and R. Schwartz, "Exploring embedded path capacity estimation in tcp receiver," *IEEE Workshop on End-to-End Monitoring Techniques and Services*, 2007.

[3] A. B. Pillai, "The harvestman web crawler," 2007. [Online]. Available: http://www.harvestmanontheweb.com/

[4] J. W. Eaton, "Gnu octave project," 1993-2006. [Online]. Available: http://www.gnu.org/software/octave/

[5] B. Huffaker, N. Evi, and C. K., "Otter: a general-purpose network visualization tool," in *Internet Society*, June 1999.

[6] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz, "Topology discovery in heterogeneous ip networks: The netinventory system," *IEEE ACM Transactions on Networking*, vol. 12, no. 3, pp. 401–414, June 2004.

[7] "Robots exclusion standard," 2008. [Online]. Available: http://en.wikipedia.org/wiki/Robots.txt

[8] M. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 835–846, 1997.

[9] "The network simulator ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[10] S.-G. Liu, P.-J. Wang, and L.-J. Qu, "Modeling and simulation of self-similar data traffic," *International Conference on Machine Learning and Cybernetics*, vol. 7, no. 3921-3925, pp. 18–21, Aug 2005.

[11] V. Jacobson, R. Braden, and D. Borman, "Rfc 1323 - tcp extensions for high performance," *IETF*, 1992.

[12] A. B. Downey, "Using pathchar to estimate internet link characteristics," 1999.

[13] B. Huffaker, D. Plummer, D. Moore, and K. Claffy, "Topology discovery by active probing," *Symposium on Applications and the Internet (SAINT)*, no. 90-96, 2002.

[14] [Online]. Available: www.netdimes.org