ANEMOS: An Autonomous NEtwork MOnitoring System

Antonios Danalis
Computer & Information Sciences
University of Delaware
Newark DE, 19716
Email: danalis@eecis.udel.edu

Constantinos Dovrolis
College of Computing
Georgia Tech
Atlanta GA, 30332
Email: dovrolis@cc.gatech.edu

***Abstract*— We describe the design and implementation of *ANEMOS*, an *Autonomous NEtwork MOnitoring System*. ANEMOS allows network operators and end-users to schedule, perform, and analyze active measurements on several network paths through a Web-based GUI. The measurements can be performed with "off-the-shelf" tools, such as *Ping*. The current prototype measures end-to-end available bandwidth with *Pathload*, and round-trip delays and losses with a UDP-based configurable variation of *Ping*. The measurements are archived using the *MySQL* database, and they can be visualized using *MRTG*. A major feature of ANEMOS is that it supports rules, post-processing, and alarm detection. Specifically, the user can form processing rules, correlating measurements of different metrics, paths, and time intervals. These rules are then used to detect changing or alarming conditions in the performance of one or more paths, issuing informative alarms. We illustrate ANEMOS with measurements that resulted from monitoring a few paths in US and Europe.**

I. Introduction

Active measurements of delays, losses, or available bandwidth are being widely used to monitor the end-to-end performance of a network path. Such measurements are often performed through rather primitive text-based tools, such as *Ping*, making the analysis, archiving, and visualization of the gathered data cumbersome. Our motivation in this work is that a network operator or end-user should have the capability to schedule measurements of different metrics in several network paths through a flexible and simple graphical interface. The collection of underlying measurement tools should be extensible, allowing the user to "plug-in" additional tools as they become available, or configure the existing ones. The results of the measurements should be archived in a relational database that allows sophisticated queries, post-processing, and interactive visualization. Furthermore, we should not expect the user to constantly monitor the measurements, watching for sudden changes in the performance of a network path. Instead, the measurement system should be able to automatically analyze the collected data, based on user-specified rules, issuing alarms whenever the conditions of a rule are satisfied. Such conditions, for instance, can be a sudden decrease in the available bandwidth of a path, or a significant increase in its Round-Trip Time (RTT). The alarms issued by the system can be then used for detecting congestion, anomalies, attacks, or flash crowds, they can trigger changes in the configuration of overlay networks [1], [2], or, in a more advanced version of the system, they can lead to changes in the configuration of the path routers.

Driven by the previous motivation, we designed and implemented the *Autonomous NEtwork MOnitoring System*, or *ANEMOS*. In this paper, we describe the salient features of the system, focusing on its system architecture and key implementation aspects. ANEMOS has some similarities with other network monitoring tools or architectures, such as Pinger [3], Surveyor [4], or the Network Weather Service [5]. One major difference, however, is that ANEMOS provides rules and alarms. Specifically, the system evaluates user-specified rules on the collected data while the measurements are in progress, issuing alarms when the rule conditions are satisfied. Another difference is that ANEMOS has been designed for modularity and extensibility, allowing the user to plug-in and use any text-based measurement tool with minimal modifications in the ANEMOS software. Also, the user can request the measurements to be performed either in real-time, or to be scheduled as a batch process. All the interactions with the system are through a Web-based GUI. For portability reasons, ANEMOS is written in Java.

The measurement tools supported by the current implementation are a UDP-based configurable variation of *Ping* for RTTs and packet losses, and *Pathload* [6] for end-to-end available bandwidth. Each *measurement task* is a periodic iteration of a number of measurements. The period as well as the start and end time of a measurement task are selected by the user. The user can request an arbitrary number of measurement tasks, possibly over different network paths, as long as the *Worker* component of the system is installed at the two end-hosts of a path. The measurement data are pulled by the ANEMOS *Coordinator*, which is the central process in the system, and stored there using the *MySQL* [7] database. Upon the user's request, the data of one or more measurement tasks are retrieved from the database and visualized with *MRTG* [8].

The rest of the paper is organized as follows. Section 2 mentions the most relevant tools and projects. Section 3

This work was supported by the SciDAC program of the US Department of Energy (award # DE-FC02-01ER25467) and by an equipment donation from Intel Corporation.

describes the ANEMOS system architecture, while Sections 4 and 5 focus on key implementation issues. Section 6 focuses on the specification of processing rules. Section 7 illustrates the use of ANEMOS, monitoring a few paths in US and Europe. Some ideas for future extensions of the system are given in Section 8.

II. Related systems

There are several open-source and commercial network monitoring tools and architectures. In the following, we mention the most popular systems that are related to ANEMOS.

Among the open-source tools, the most closely related to ANEMOS are *PingER* [3], *Surveyor* [4], the *National Internet Measurement Infrastructure (NIMI)* [9], and the *Network Weather Service* [5].

PingER uses *Ping* to measure RTTs and loss rates to hundreds of hosts around the world. PingER provides performance information and long-term trends about many different geographical areas of the Internet. *Surveyor* is a measurement infrastructure that is being deployed at several sites around the world. Surveyor measures the performance of Internet paths using the IETF IPPM standardized metrics [10], [11]. The project is also developing methodologies and tools to analyze the performance data off-line. The *National Internet Measurement Infrastructure (NIMI)* [9] is a measurement architecture in which a collection of measurement hosts cooperatively measure the properties of Internet paths and clouds by exchanging test traffic among themselves. The key emphasis of the architecture is on scalability, authentication, security, and administrative control. The architecture supports several measurement tools. The *Network Weather Service* [5] is a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. Currently, the system includes sensors for end-to-end TCP throughput and RTT. NWS also allows the archiving and visualization of the measurements.

The main similarity between these systems and ANEMOS is that they all monitor end-to-end paths through active measurements. The main difference is that these systems do not support real-time processing, analysis, and visualization of the measurements. A major characteristic of ANEMOS, on the other hand, is that it can evaluate measurements as they are being collected, and take rule-based actions without user intervention.

Among the commercial monitoring platforms, *HP's Openview* [12], *IBM's Netview* [13], and *SUN's Solstice* [14] are the most popular. These products have several differences with each other, but they all basically provide *decentralized, local, and passive* monitoring services. This means that these systems are commonly based on SNMP to monitor the behavior of individual network devices, rather than end-to-end paths. The main similarity to ANEMOS is that they also provide rule-based alarms and real-time processing of the collected measurements. The main difference with ANEMOS is that they do passive measurements of local network devices, rather than active monitoring of an entire end-to-end path.

III. System architecture

The architecture of ANEMOS is made up of three types of components. The *Client* module constitutes the user interface to the system, the *Coordinator* is the central component that is responsible for the scheduling of measurement tasks and for data collection and analysis, while the *Worker* modules are the components that interact with the measurement tools at the end-hosts of the monitored paths. In the following, we describe each component in more detail.

A. Clients

An ANEMOS *Client* is a web applet. It is written and compiled for JVM 1.1, and so it can run on both browsers relying on the Java Plug-in, and on browsers that have a build-in JVM. The *Client* provides two login options: one for regular users and one for the system administrator.

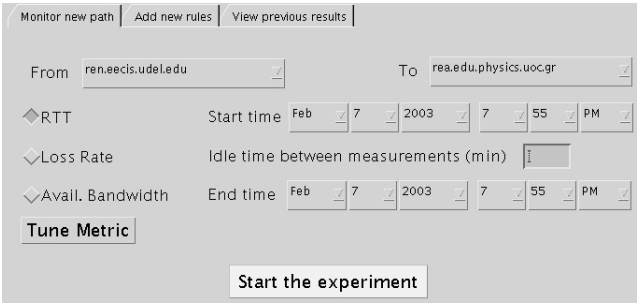

Fig. 1. Monitor new path

In 'Regular User' mode, the interface provides three main operations. First, with the Monitor new path tab (shown in Figure 1) the user can specify a monitored path by selecting two end-hosts that run ANEMOS *Workers*. The user also chooses the measurement tool, the start time, the end time, and the time interval between iterations.

Second, with the Add new rules tab the user can form data analysis rules, and specify the corresponding alarms. The rules' syntax is described in detail in Section 5.

Third, with the View previous results tab (shown in Figures 2 and 3) the user can see all previous and ongoing measurement tasks that she initiated, and also visualize their results. The graphs are generated with the use of `rateup`, a component of MRTG. ANEMOS allows the user to plot data collected at a previous date, with a specified granularity and graph width. In this way, both long-term and short-term traffic trends and patterns can be detected.

In 'Administrator' mode, the interface provides the user with mainly two operations. First, the ability to review, and terminate if necessary, ongoing measurement tasks. Second, to add or remove a *Worker* process from a remote end-host. In future versions of the system, the administrator will also be able to edit ongoing measurement tasks, rules, and the list of available measurement tools. If, for example, a measurement is proven to be too intrusive, it will be possible to schedule it less frequently.

The *Client* modules communicate with the *Coordinator* through a strongly typed protocol. Consequently, ANEMOS

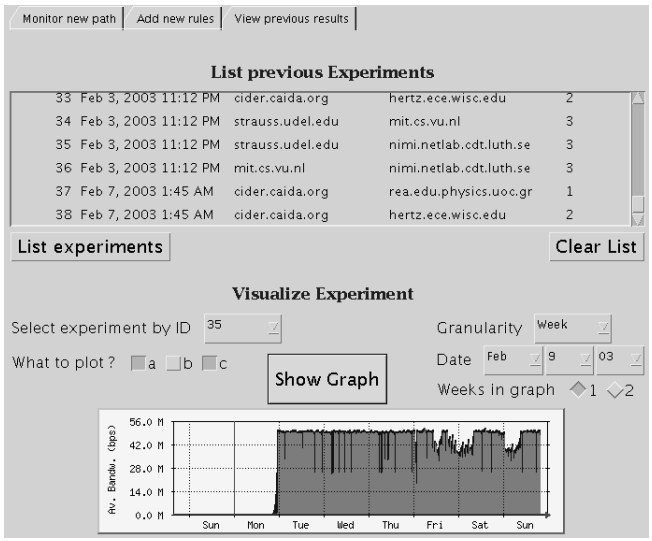

Fig. 2. View previous results

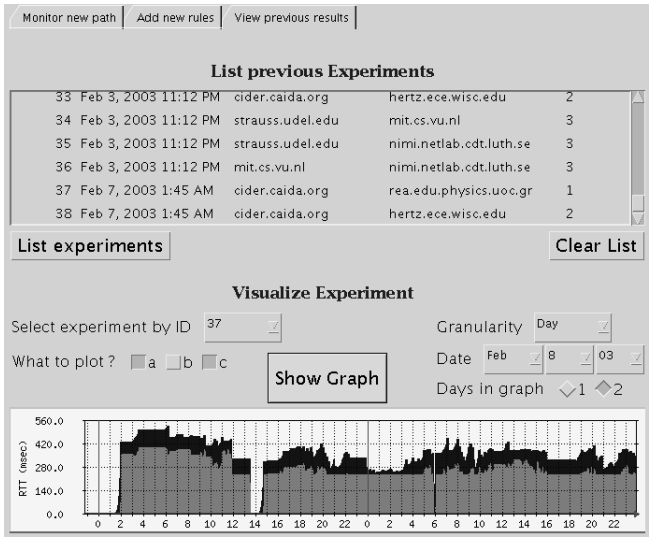

Fig. 3. View previous results

users that want to modify the *Client* interface, or users that want to integrate ANEMOS with their front-end tools, can do so rather easily.

B. Workers

The ANEMOS *Workers* are in charge of the following operations at the end-hosts of the monitored paths. First, they execute the appropriate tool for each measurement task. Second, a *Worker* module at the source host of a path communicates with the *Worker* module running at the destination host to coordinate the measurement task[1]. Third, *Workers* read the measurement tools' output to gather the final results, and send this data to the *Coordinator* for storage and analysis.

[1]Note that, in general, an active measurement requires the cooperation of both the source and destination hosts of a path.

To make the *Workers* tool-independent, ANEMOS assumes that all measurement tools return three values, referred to here as A, B, and C. So, at least in principle, the measurements can be performed by any tool, as long as there is a wrapper script for that tool that will perform the appropriate parsing of the tool's output, and feed the *Worker* with just three numbers. For instance, in the case of our UDP-based *Ping* implementation, the user can modify the corresponding wrapper script to change the semantics of the three collected values. So, the three values returned to the *Worker* can be the minimum, the median, the maximum, or any percentile of the measured RTTs.

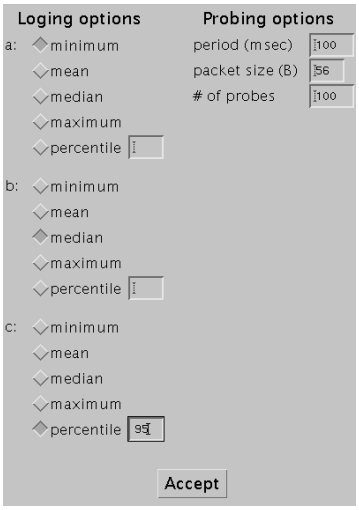

Fig. 4. Tunning of *Ping* measurements.

In the case of *Pathload*, variables A and C are the lower and higher bounds of the available bandwidth variation range respectively, while the variable B is equal to $(A + C)/2$.

C. Coordinator

The central component of ANEMOS is the *Coordinator*. This module manages the *Workers*, responds to user requests, maintains a scheduling queue for all active measurement tasks, archives the resulting data, and also, analyzes the measurements evaluating user-specified rules and issuing the corresponding alarms.

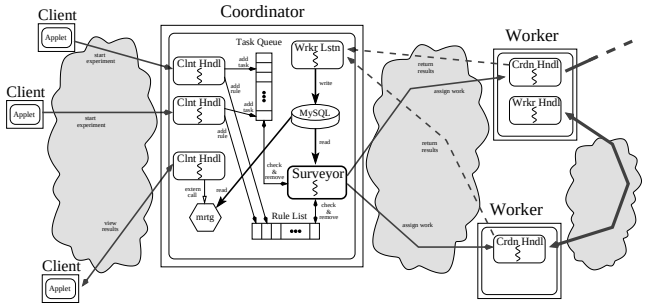

Fig. 5. System architecture

The *Coordinator* process uses several threads. A `Client Handler` thread is responsible for interacting with each user, adding new `tasks` or `rules` in the corresponding queues, or calling MRTG to create a visual representation of some measurements. A `Worker Listener` thread is responsible for collecting the results of each measurement task from the corresponding remote *Worker*, and for storing the arriving measurements into the *MySQL* database. An important thread in the *Coordinator* is the `Surveyor`. This thread maintains the scheduling queue for all measurement tasks, initiates and aborts experiments on remote *Workers*, and also evaluates each rule on the gathered measurements.

The *Coordinator* is a centralized entity in ANEMOS, and so it may seem to contradict the distributed nature of our system. Nevertheless, its role is mostly to coordinate the experiments by exchanging short messages with the *Workers*, rather than performing CPU-intensive or network-intensive operations. For this reason, we believe that the centralized nature of the *Coordinator* will not impede the scalability of ANEMOS. Furthermore, having all the results stored in a single database ensures that rule evaluation and data plotting for visualization purposes will not produce additional network traffic. Future releases of the system can include a secondary *Coordinator*, running at a remote host, to act as backup in case the primary *Coordinator* fails.

IV. Measurement tools

To make ANEMOS extensible and open to new measurement techniques, the actual measurement tools are not embedded in the system. Instead, it should be relatively easy to plug-in new tools in the system. In the current implementation, ANEMOS supports two measurement tools: a variation of *Ping* to measure RTTs and loss rates, and *Pathload* to measure end-to-end available bandwidth.

To measure a path's RTT, as well as the forward-path packet loss rate, we wrote a UDP-based variant of *Ping*. In contrary to the well-known *Ping* utility, however, our tool requires cooperation from both end-hosts of the path. The tool uses UDP for both the forward and reverse path packets, and allows the user to specify the number of packets to be sent, the time interval between packets (down to 50msec), as well as the packet payload (down to 12 bytes) (see Figure 4).

To measure the available bandwidth of a path, we use `Pathload` [15]. The available bandwidth of a network path $\mathcal{P}$ is the maximum throughput that $\mathcal{P}$ can provide to a flow, given the aggregate rate of cross traffic in $\mathcal{P}$. *Pathload* sends periodic short UDP packet streams from the source to the destination. The basic idea in *Pathload* is that the one-way delays of a periodic packet stream show increasing trend, when the stream rate is larger than the available bandwidth. The tool uses an iterative algorithm, through which the rate of probing streams converges to an interval (called "grey-region") in which the available bandwidth varies.

V. Security and authentication

ANEMOS supports multiple users performing measurements or visualizing results at the same time. Consequently, the system should provide authentication. Also, the system should use encryption for the communication of passwords between the *Clients* and the *Coordinator*.

In the current implementation, we use the `logi.crypto` Java package, developed by *logi* [16], to provide encryption. When a *Client* wants to establish a connection with the *Coordinator* it starts a *CipherStream*. This initializes a Diffie-Hellman key exchange using Triple-DES, and it then ensures that all data sent through it will be encrypted and decrypted automatically.

To provide authentication, the communication protocol between the *Clients* and the *Coordinator*, requires that all messages include the username and the password of the requesting user.

VI. Rules and alarms

A significant feature of ANEMOS is that it allows the user to specify rules for analyzing the collected measurements. Since the data are stored in a relational database, they can be retrieved and processed in real-time through sophisticated SQL queries.

For example, a network operator can specify a rule such as: *check if the average RTT in the last 5 minutes is more than 20 msec PLUS the 5-minute average RTT 24 hours ago*. When a rule evaluates as true, ANEMOS issues an alarm. The alarm can be an email message to the network operator, a call to her beeper, or simply a message written to a special file. In the current implementation we only support the latter type of alarm.

Another use of rules is to correlate measurements from different paths. By comparing the RTTs or available bandwidth of diverse paths, one can choose optimal routes in an overlay network, or detect congestion in the overlap between different paths. For example, if two disjoint paths, which merge at the access link of a Web-farm, exhibit a simultaneous and significant loss rate increase, we can assume that that access link is congested. Similar correlations in diverse paths can provide clues for the occurrence of a distributed denial-of-service attack or flash-crowd.

ANEMOS does not assume that users know how to write SQL queries. Instead, users specify the rules that they want to evaluate using a powerful Web-based form. Then, ANEMOS translates this form into an SQL query that is issued to the *MySQL* database by the *Coordinator*.

To create a rule, the user first defines an arbitrary number of `Date` events relative to the evaluation of the rule. The `Date` field is determined based on the evaluation time of a rule, rather than based on absolute time. For instance, `0` refers to the time of the evaluation, while `14 Days` translates to two weeks before the time of the evaluation.

A `Variable` can be the MINIMUM, AVERAGE, or MAXIMUM of one of the logged variables `A`, `B`, or `C` of the

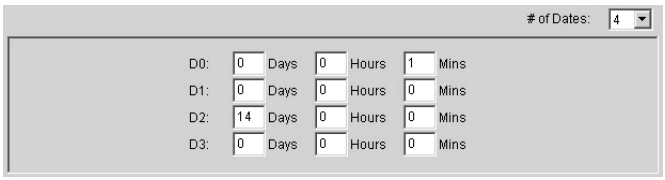

Fig. 6. Defining `Date` events

specified task, over a certain time interval (defined by two `Date` events).

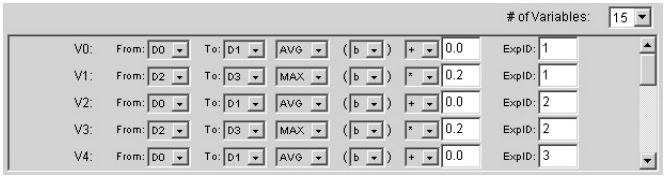

Fig. 7. Defining `Variables`

The form allows for an arbitrary number of `Variables` to be combined using the arithmetic operators "+", "-", "*" and "/" in `Variable Combinations`. Additionally, in the definition of the i'th `Variable Combination`, any previous `Variable Combinations`, from 0 to $i-1$ can be used. This feature provides the semantics of parentheses.

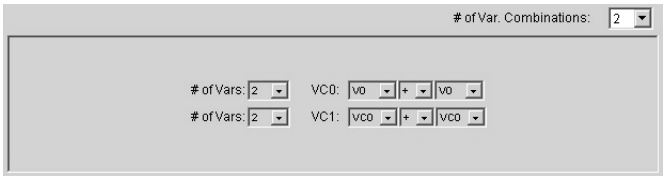

Fig. 8. Defining `Variable Combinations`

A `Condition` is a boolean entity and is defined to be the comparison of any two `Variables`, or `Variable Combinations`.

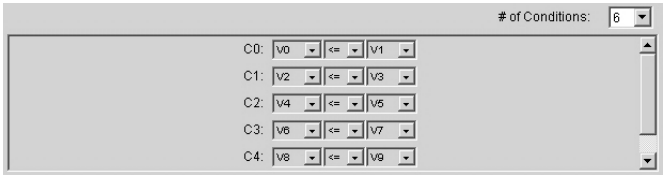

Fig. 9. Defining `Conditions`

To provide the semantics of boolean expressions, the form has support for `Condition Combinations`. A `Condition Combination` is constructed by an arbitrary number of `Conditions`, combined using the logical operators "&&" (AND) and "||" (OR). Again, the semantics of parentheses are provided by the use of previously defined `Condition Combinations` in the definition of the current `Condition Combination`.

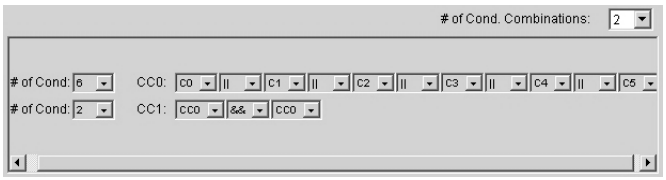

Fig. 10. Defining `Condition Combinations`

Finally, a `Rule` is one of the `Condition Combinations`.

Fig. 11. Defining a `Rule`

To demonstrate the use and expressive power of the ANEMOS rules, we provide the following example. Suppose that a corporation has four local area networks interconnected in a fully-connected switched topology. Tasks 1 through 6 measure the available bandwidth of the six interconnecting links. We want an alarm to be raised if any of the links becomes more than 80% utilized, i.e. if the available bandwidth that Pathload measures drops below 20% of the capacity of the link. The following expressions would implement the corresponding rule.

D0: 1 Mins
D1: 0 Mins
D2: 14 Days
D3: 0 Mins
V0: From D0 To D1 AVG(b) + 0.00 ExpID: 1
V1: From D2 To D3 MAX(b) * 0.20 ExpID: 1
V2: From D0 To D1 AVG(b) + 0.00 ExpID: 2
V3: From D2 To D3 MAX(b) * 0.20 ExpID: 2
V4: From D0 To D1 AVG(b) + 0.00 ExpID: 3
V5: From D2 To D3 MAX(b) * 0.20 ExpID: 3
V6: From D0 To D1 AVG(b) + 0.00 ExpID: 4
V7: From D2 To D3 MAX(b) * 0.20 ExpID: 4
V8: From D0 To D1 AVG(b) + 0.00 ExpID: 5
V9: From D2 To D3 MAX(b) * 0.20 ExpID: 5
V10: From D0 To D1 AVG(b) + 0.00 ExpID: 6
V11: From D2 To D3 MAX(b) * 0.20 ExpID: 6
C0: V0 <= V1
C1: V2 <= V3
C2: V4 <= V5
C3: V6 <= V7
C4: V8 <= V9
C5: V10 <= V11
CC0: C0 || C1 || C2 || C3 || C4 || C5
Rule: CC0

Note than the previous rule assumes that the maximum value of available bandwidth in any path over the period of 14 last days will be equal to the capacity of that path. So, this rule does not have to be updated if the network capacities are upgraded or changed in any way. If this assumption does not

hold, the user can directly assign the actual values of the link capacities to the appropriate variables.

VII. Illustrative results

To test and debug the system, we have installed *Workers* on several hosts in the United States and Europe, including hosts at U-Delaware (`udel.edu`), CAIDA (`caida.org`), U-Wisconsin (`wisc.edu`), U-Vrije (`vu.nl`), U-Luleå (`luth.se`), and U-Crete (`uoc.gr`), with the *Coordinator* running at the University of Delaware. Although the number of *Workers* currently is only about ten, the insignificant load imposed on the *Coordinator* and on the *MySQL* database (which are the centralized components of the system) gives us a positive indication that the system is scalable to a large number of *Workers*. Additionally, some network or host related problems during the measurements have shown that the system can recover from outages and failures. In particular, as shown in Figure 3, at around 14:00 on Feb 7th no measurements were performed for about an hour due to a network outage. Nevertheless, when the connection was reestablished the system started taking measurements again without any user intervention.

To illustrate the use of rules and alarms, Figure 13 shows RTT variations in a 33-hour time period in three paths. The three graphs, from top to bottom, are the RTTs between U-Delaware and U-Crete, between U-Delaware and U-Vrije, and between U-Vrije and U-Crete. The rule that we specified is: *check if the RTT between U-Delaware and U-Crete is greater than the RTT between U-Delaware and U-Vrije PLUS the RTT between U-Vrije and U-Crete*. Such a rule, for instance, can determine the optimal routing between U-Delaware and U-Crete in a three-site overlay network that connects U-Delaware, U-Crete, and U-Vrije (see Figure 12).

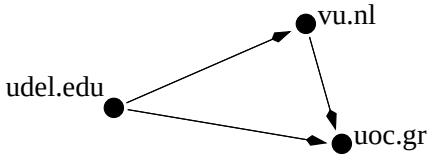

Fig. 12. Topology of a possible “detour” at an overlay network

ANEMOS identified two time periods in which the previous rule was satisfied, issuing two alarms at around 7:30am and 9:30am EST at the right side of the graphs. The corresponding three RTTs that the alarm reported were: 297msec (U-Delaware, U-Crete), 141msec (U-Delaware, U-Vrije), and 147msec (U-Vrije, U-Crete) at the 7:30am event, and 302msec (U-Delaware, U-Crete), 141msec (U-Delaware, U-Vrije), and 132msec (U-Vrije, U-Crete) at the 9:30am event.

Another interesting case is the one shown in Figure 14. Here the rule was monitoring the value of the available bandwidth, as well as the width of its variation range (“grey-region”). The rule raised an alarm several times during the time periods that appear highly variable in the graph, as well as in the four events (12:00,17:00,9:30,17:00) that the available bandwidth variation range was between zero and the capacity.

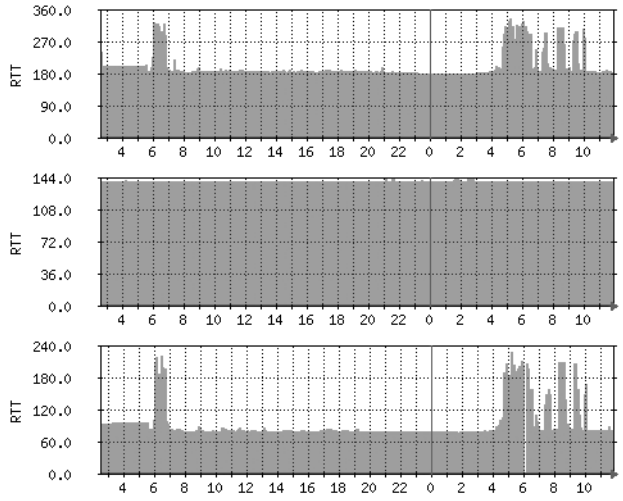

Fig. 13. RTT measurements

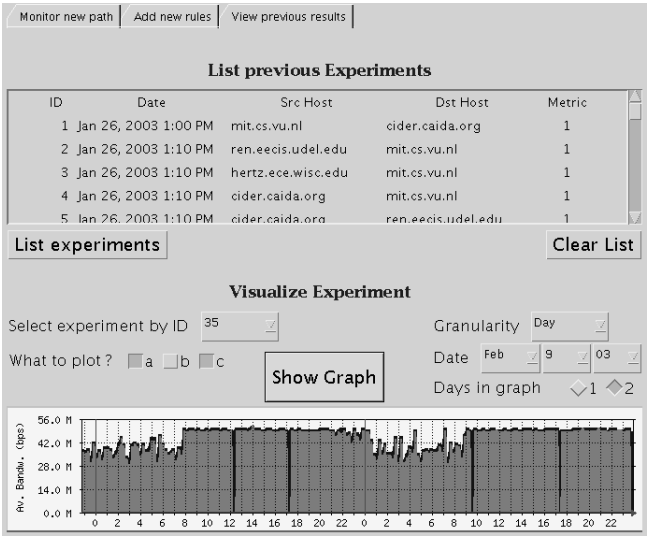

Fig. 14. Available bandwidth measurements

VIII. Future work

There are several features that we plan to add to the system. First, an area that could be improved is authorization. In a wide-scale installation of ANEMOS, there should be additional control on the measurement hosts or tools that a particular user can access.

The extensibility of the tool can be also improved. Currently, it is easy to replace one of the external tools used for the measurements by another tool, by just replacing the tool’s executable and the corresponding wrapper. To add a new tool however (keeping the existing ones), a user should edit the main code. In future releases, we plan to make the configuration of external tools as easy as adding and removing lines from the system’s configuration file.

We plan to release the source code for ANEMOS in April 2003.

REFERENCES

[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. of the 18th ACM Symposium on Operating Systems Principles SOSP 2001*, Canada, Oct. 2001.

[2] J. Touch and S. Hotz, "The X-Bone," in *Proc. of Third Global Internet Mini-Conference at Globecom '98*, Australia, Nov. 1998, pp. 75–83.

[3] W. Mathews and L. Cottrell, "The PingER project: Active internet performance monitoring for the HENP community," *IEEE Commun. Mag.*, vol. 38, no. 5, pp. 130–136, May 2000.

[4] S. Kalidindi and M. Zekauskas, "Surveyor: An infrastructure for internet performance measurements," in *Proc. INET'99*, 1999.

[5] R. Wolski, N. Spring, and C. Peterson, "Implementing a performance forecasting system for metacomputing: the network weather service," in *Proc. of Supercomputing*, 1997.

[6] BW-meter tools: Pathrate and pathload. [Online]. Available: http://www.pathrate.org/

[7] MySQL AB. MySQL. [Online]. Available: http://www.mysql.com/

[8] T. Oetiker and D. Rand. Multi router traffic grapher. [Online]. Available: http://www.mrtg.org/

[9] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, "An architecture for large-scale internet measurement," *IEEE Commun. Mag.*, vol. 36, no. 8, pp. 48–54, Aug. 1998.

[10] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way delay metric for IPPM," RFC2679, 1999.

[11] G.Almes, S. Kalidindi, and M. Zekauskas, "A one-way loss metric for IPPM," RFC2680, 1999.

[12] Hewlett Packard. Hp openview. [Online]. Available: http://www.openview.hp.com/

[13] IBM. Netview for AIX. [Online]. Available: http://www.tivoli.com/products/index/netview/

[14] SUN. SUN solstice. [Online]. Available: http://www.sun.com/products-n-solutions/sw/solstice/

[15] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Rela tion with TCP Throughput," in *Proceedings of ACM SIGCOMM*, Aug. 2002.

[16] L. Ragnarsson. The logi.crypto java package. [Online]. Available: http://www.logi.org/logi.crypto/stable/