Automated Event Detection for Active Measurement Systems

McGregor A.J. and Braun H-W

Abstract— **The utility of active measurement systems, like NLANR's AMP, is limited because users must poll the system. Most potential users of the data collected by measurement systems are too busy to regularly check the measurement system for interesting behaviour. This paper proposes automatic event detection as a solution to this problem. Algorithms for detecting changes in RTT, jitter and loss are described and experience with am implementation is presented, including performance data. The methodology presented has been refined to the point where laboratory tests of automatic event detection appear to be bearing fruit.**

Keywords— **Network performance measurement, monitors, active measurement, event detection.**

I. Introduction

It is easy to collect large volumes of data through active measurement. The NLANR AMP system [1][2][3], for example, consists of around 120 active monitors operating in a full mesh. Each monitor measures the round trip time to each other monitor every minute and the route to each other monitor every 10 minutes. The data collected is transfered to a central site where it is made available to users as raw data and as web based performance graphs.

A set of graphs is available for each of the approximately 14,000 paths that are measured. While these graphs have proved useful when a change in the network has been detected they do not often aid in the *detection* of interesting network events. There are two reasons for this. First, there are just too many of them. No human could examine all the graphs often enough to detect network events in a timely way. Second, even though most sites have a single AMP machine and only need to examine the data for their site, the shortage of networking personnel means that most network administrators are event driven; they do not poll for work.

The over arching goal of the AMP project, and most other active measurement projects, is to increase the performance of the network for its users. Supporting the diagnosis of network problems after they are detected is a useful function in this respect. However, the data AMP collections contain enough information to aid in the discovery of network problems before the users of the network report them.

A.J. McGregor is with The University of Waikato, Hamilton, New Zealand and the National Laboratory of Applied Network Research (NLANR), San Diego Supercomputer Center, University of California, San Diego, La Jolla, USA. Email: tonym@cs.waikato.ac.nz

H-W Braun is with the National Laboratory of Applied Network Research (NLANR), San Diego Supercomputer Center, University of California, San Diego, La Jolla, USA Email:hwb@nlanr.net

This work is funded, in part, by NSF Cooperative Agreement No. ANI-9807479. The U.S. government has certain rights in this material.

Making use of the latent capacity of the data to discover network problems, in a practical way is not trivial. Some of the difficulties that must be overcome are:

- There are large volumes of data involved. AMP collects over 1Gb of data a day and has an incoming data stream of about 1.5Mbps. Any processing must be able to match this data rate, and fit in the memory available on inexpensive equipment. Meeting this need is compounded by the large number of paths. In some cases all these paths must be monitored for events of interest.
- There is a lot of natural variation in the data. Paths differ in their normal RTT and the amount of variability in this data. On one path a 10% increase in RTT might be quite normal, while on another it would indicate an unusual event.
- When a network event occurs many elements are often affected at the same time. Naive notification of every event could swamp the system, the network and the support staff with error notifications at the very time they are all likely to be under stress.

This paper describes a system for atomic detection of events in data collected by active measurement. The system is self-adjusting to the fit the data being monitored but is simple enough to be implemented within practical constraints. A post-processor sends notifications to users who have requested them. This notification processor limits the number of notifications using a modified exponential backoff to group successive notifications into a single message.

The heart of the system is divided into three algorithms; each algorithm is responsible for detecting one class of network event. The first of these, the "plateau detector," detects a change in the base RTT between a pair of monitors, even in the presence of (or absence of) jitter. Figure 10, below, contains some examples of events detected by the plateau algorithm. The second algorithm detects a change in the RTT jitter over a path. The third algorithm detects excessive packet loss.

The algorithms are based around two "windows" that advance in time as each new data item is received. The first of these (which is the oldest in time) is used to characterise the normal state of the path. The second window indicates the current state of the path. Simple statistics are calculated on these windows and compared to give the basis of the event detection. An additional buffer is used as a filter; it removes transient outliers from the data so that they do not cause false triggers.

The rest of the paper is organised as follows. Section II, Active Measurement and section III, The Visualisation Alternative, present background information about active measurement systems which provides the context for the

event detector. Section IV, Architecture, gives an overview of the event detection system and the way it interacts with other components of AMP. The algorithms that make up the core of the system are described in detail in section V, Algorithms. This abstract description of the algorithms is supported with a practical example in section VI. As described above, meeting the performance goals is an integral part of the problem. Section VII, Performance, describes an ad hoc performance test which give an indication of the system resources required to run the event detector. The paper concludes with section VIII, Implementation Status, and section IX, Conclusions.

II. The AMP System

The event detection system described in this paper was designed to meet the needs of event detection in the NLANR AMP system. AMP is NLANR's active measurement project and is part of the NAI, network analysis infrastructure[4][5]. AMP's focus is making site to site measurements of round trip time (RTT), packet loss, topology and throughput across the National Science Foundation (NSF) approved HPC networks. At the time of writing around 120 monitors are deployed at NSF HPC awardee sites. This number is currently increasing by a few each month.

Types of Measurement

Each of these monitors sends a single, 64 byte, ICMP packet to each of the others every minute and records the time until a reply is received, or a loss if no reply is received. In addition, every 10 minutes, the route to each other monitor is recorded using traceroute. These round trip times (RTTs) are the core of the system and the focus of event detection. Throughput tests can also be run between any pair of the monitors using a web based throughput test request. (Throughput tests are only run on demand because of the high cost, in terms of network traffic, of running these tests.) The following throughput tests are available:

- Bulk TCP data transfer
- Bulk UDP data transfer
- ping -F
- treno

AMP Architecture

AMP machines independently make measurements to one another. The data collected is sent to the central AMP site, at the San Diego Supercomputer Center, for processing and publication via the WWW. To improve robustness, two central machines are used. The data is sent to each central machine independently by each monitors. This arrangement is shown in figure 1. The two central machines have a common domain name (`watt.nlanr.net`) and share the web serving and analysis workload.

The Active Mirror

Because AMP measurements are continuous there is no natural end time at which to send the collected data from

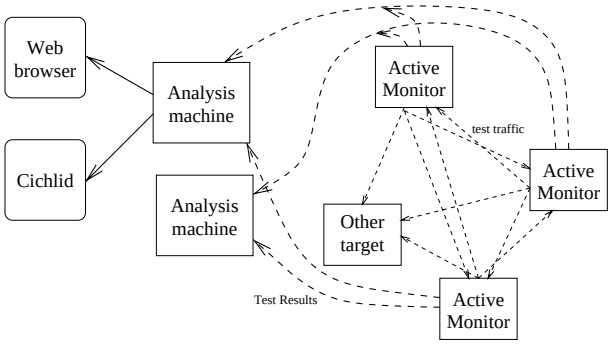

Fig. 1. Amp Architecture

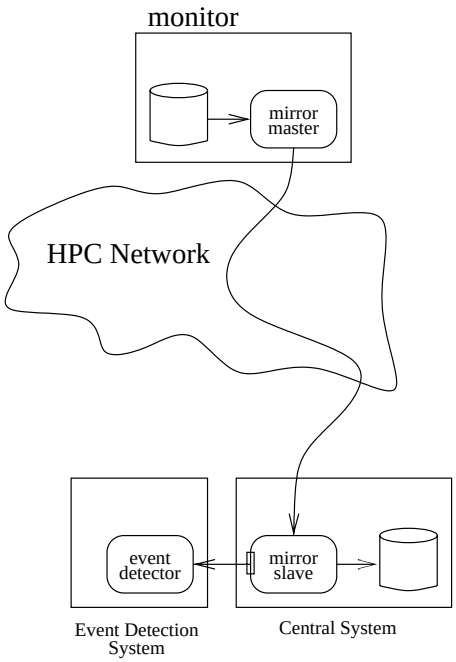

Fig. 2. Active Mirror

the measurement machines to the central servers. In addition we want the system to have a near to real time nature with the data in the web pages for today being current to within a few minutes. This is important if the system is to be used as a diagnosis tool. To achieve these ends we have developed an active mirror system. See figure 2. This operates much like the daily mirror used on many FTP sites except that file changes are reflected on the mirror site more quickly. When a monitor is started it opens a TCP connection to each of the central machines. It then watches the last-modified date on the files in its directory tree where the measurement results are stored. When a file is updated the changes to the file are sent to each of the central machines. The process is fault tolerant so that if a central machine or a monitor fails, when it recovers all machines will be brought up to date. In addition to keeping the central sites current, this approach avoids a peaky transfer load that could overwhelm the central servers or disturb the measurements being taken by the monitors.

As part of the event detection project a real time data

interface was added to the data mirroring process. The slave process (on the central machine) listens for TCP connections on a known port number. If a connection is established to this port a copy of the data received from the monitors is made and, after formatting, is sent to this *real time data port*. This mechanism allows the event detector (or other user of the real time data) to be on the same, or a different machine, to the active mirror, depending on its processing requirements. Currently the two reside together, although we expect to move the event detection to a separate machine if event triggers become popular with our users.

The data on the central web servers can be accessed from a web page that lists the monitor sites as hyperlinks. When a link is selected a table of the RTT and loss from that site to all the other sites is supplied. Again the site names are hyperlinks. If a site from the table is selected, RTT and loss data for that pair of sites is displayed as a year-to-date graph and a set of weekly graphs for all weeks this year. Further hyperlinks allow selection of a detailed display of any day, including the RTT by time of day and as a frequency distribution. The route data can be displayed in a tabular form (like the output of traceroute) or as a graphical plot using the Otter tool from CAIDA[6]. These displays are best understood by visiting the AMP web site, which is linked off the NLANR home page[3].

III. The Visualisation Alternative

There are a large number of AMP monitors and consequently a very large number of pairs of monitors. Data is collected on the path between every pair of monitors and there are web pages for each pair. As described earlier this creates problems for people looking for interesting events in the data. While this paper focuses on event detection the Cichlid visualisation tool[2] can be used as a different approach this problem. The Cichlid tool (named after the species of fish with the same name, some of which change colour) was developed at NLANR to allow visualisation of a wide range of network data. In brief, Cichlid is a distributed, animated, display tool for bar charts and 'vertex/edge' graphs. It is known for its colourful moving images. We have used Cichlid to produce a number of visualisations, including the network 'terrain' shown in figure 3. Again, this is best viewed interactively at the Cichlid web site[7].

IV. Architecture

The event detection system is shown in figure IV. The following sections describe this diagram.

Real Time Data Interface

As mentioned in section II a real time data feed has been added to the data transfer mechanism between the monitors and the central AMP servers. The data the feed is formatted as shown in figure 5.

The timestamp is a Unix style timestamp; seconds from midnight UTC 1 Jan 1970. The source and destination addresses are null terminated strings. The data type is a

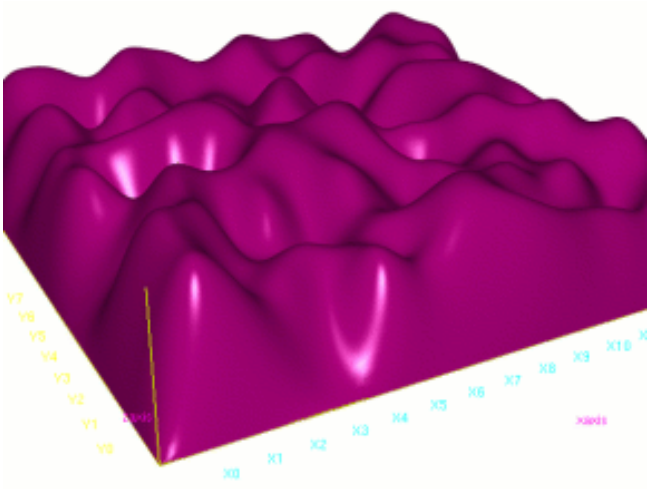

Fig. 3. Cichlid Network "Terrain"

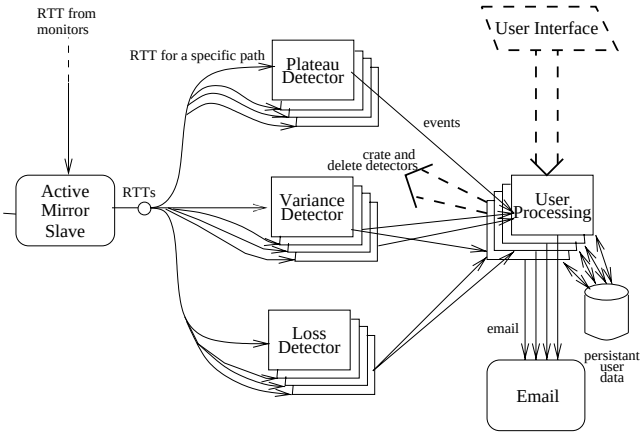

Fig. 4. System Architecture

short integer, specifying the type of data being transfered. Currently only 0 (for RTT data) and 1 (for path data) are defined. The final field is a 16 bit integer giving the number of milliseconds RTT in the case of RTT data.

There are about 20 million RTT records transfered each day. Consideration was given to reducing the data transfered, and making the transfer record a fixed sized, by encoding the source and destination values as short integers. This would save around 300Mb of data transfer per day. However, it was decided to use the string format because the central machines (that are doing the data mirroring) are heavily used and the CPU cycles and memory required to convert the data from the string format to an index are likely to exceed those required to transfer the data. The additional network resources required are unlikely to cause problems because the machines are physically close and can

Unix Time	Source monitor	Destination monitor	Data Type	Value

Fig. 5. Real Time Data Interface

Count	Accumulation Period
1	Immediate
2	5 minutes
3	15 minutes
4	30 minutes
5	1 hour
6	2 hours
7	4 hours
8	8 hours
≥ 8	1 day

TABLE I
BACK OFF TIMES

have a dedicated LAN.

Event Detectors

The system contains three types of event detector: plateau, jitter and loss. There is an instance of each detector for each path for which a user has requested that type of watch be maintained.

Detectors have parameters that modify their function. The plateau detector, for example, includes a sensitivity parameter that changes the level of change at which a trigger will be generated. Although there are defaults, users may set these parameters to change the level of notification they receive from the system.

We expect there to be thousands of instances of the detectors because most users are interested in many paths. Some users may be interested in all the paths to a particular site. If two users request a watch on the same path, with the same parameters, they share a detector instance. If the parameters vary, different detector instances are created. We plan to include parameter profiles into the user interface to encourage reuse of the same parameters, and thereby reduce the number of instances.

User Processing

The main role of the user processing component is to limit notification emails. If every event detected generated a notification, some network events, especially those that affect all paths to a site, would cause hundreds or thousands of notifications to a single user. Apart from the inconvenience this would cause it adds additional load to the network, and personnel who may have the responsibility of fixing the network, at a time when they are already overloaded. To reduce this effect the user processor uses a modified exponential back off. After the first message, further triggers are accumulated for 5 minutes and delivered in a single email message. If messages keep arriving they are accumulated for longer times, as shown in table I.

The user processor also has a maintenance role. Each instance of the user processor maintains a persistent data structure of information about users, including the paths that they are watching. When required new detector instances are created by the user processor. When users no longer watch a path, the appropriate detector is informed and it will, if necessary, deallocate itself.

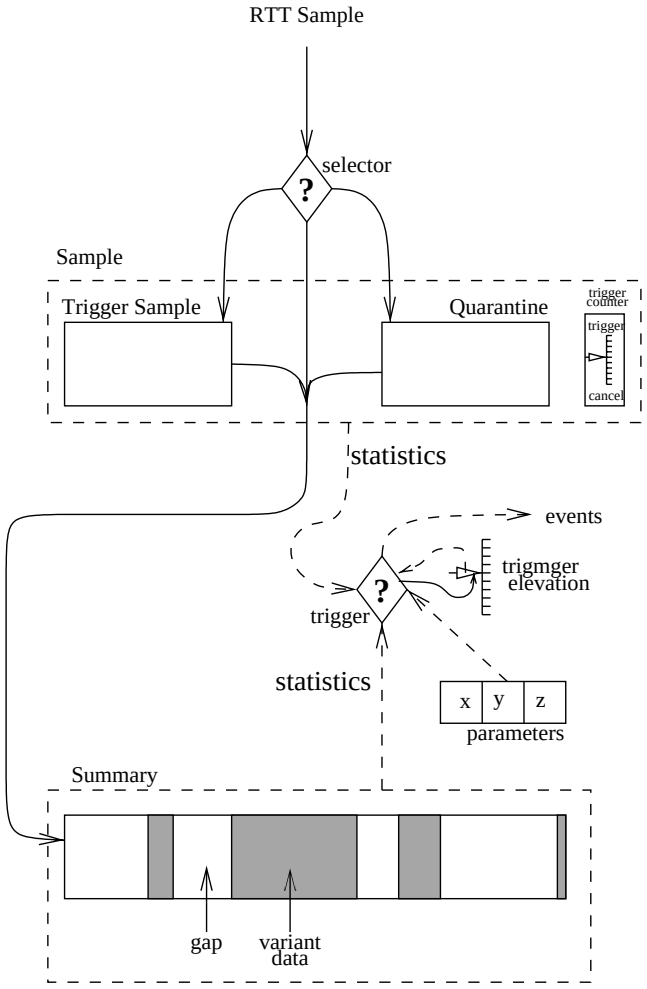

Fig. 6. Plateau Detector

V. ALGORITHMS

The core of the system is the event detection algorithms and the main algorithm is the plateau detector. The plateau detector watches a path for *significant* changes in the *base RTT*. By base RTT we mean a change in most of the RTT values measured. Transient spikes do not contribute to a change in the base RTT unless they occur frequently.

By significant we mean a change that is large enough to be of interest. Significance is, in part, determined by the nature of the data. Changes which occur frequently are not of interest. Significance is also determined by the user of the event detection system who chooses parameters that control the degree of variation from normal that is needed to cause a trigger.

A. Plateau Detector

The plateau detector gets its name from a common example of a significant change in the base RTT where a step in the RTT occurs at a particular time. All values after that

time are based on the new, higher, level. Figure 10 has an example of a short plateau at about midnight 8/9 January. A plateau does not have to return to the original value to be detected by the plateau detector. Further, although the detector takes its name from this common example, a well defined step is not necessary for an event to be detected. Any significant change in the RTT is satisfactory. For example the change that occurs on Saturday 13th January in figure 10 is also detected by the plateau detector.

The plateau detector algorithm has several mechanisms that interact. A precise discussion of all these mechanisms simultaneously would be unnecessarily clumsy. Instead we begin with an overview of the main elements, omitting some of the details for clarity. These details will be added as refinements later in the discussion. Of necessity, some of the descriptions at the beginning of this discussion are simplifications and may not reveal the complete situation.

The plateau detector is based around two windows, the *summary window* and the *sample window*. The summary window is used to characterise the normal state of the path. Samples for the previous period are stored in the summary window. The number of samples stored in the summary window is determined by a user set parameter. A small number of days is common. The mean and variance of the summary window are used by the *trigger selector* to decide how an incoming sample should be treated.

As samples arrive they are passed into the summary window unless it appears that the sample might form part of a trigger. In this case, the sample is stored in the sample window. The decision if a sample potentially forms part of a trigger is made by the *trigger selector*. The selector makes this decision on the value of the sample, the mean and variance of the summary window, and on parameters that the user has chosen. To be chosen as a potential trigger a sample must exceed the mean plus the variance times the user selected sensitivity.

$$RTT > sample_{mean} + (sample_{variance} * sensitivty) \quad (1)$$

As we noted above, a plateau trigger requires a change in the base RTT; a single measurement is not sufficient to cause a trigger. Instead, a trigger is generated when a number of samples that meet equation (1) are received. The required number of samples is another user parameter. Each time a sample that fulfils (1) is received a counter is incremented. If this counter is non-zero and a sample that does not meet (1) is received the counter is decremented. If the counter reaches the user parameter *trigger duration* a trigger is generated. If the counter returns to zero the trigger is aborted. In either case, data in the sample window is passed to the summary window.

Missing Samples

Sometimes a RTT can't be measured. This could be because there is no functioning path between the monitor, or because the monitor is not operating. This creates gaps in the data set like the one on Saturday 20 Jan in the dataset in figure 10.

When a gap occurs the algorithm compresses time so that the gap is covered over. In effect that period of time is ignored and processing continues as if it did not exist.

Outliers

In summary, as we have described it so far, a trigger is generated when there are enough samples that exceed statistics generated from the summary window. This approach works well in many situations but some sequences of input cause undesirable triggers, or cause desired trigger to be missed. We introduce refinements to deal with these in the following paragraphs.

The mean and variance are susceptible to extreme outliers. If a very large sample arrives it contributes a lot of weight to the mean and variance of the summary window, inflating these values and causing triggers to be less likely.

One solution to this problem is to use a different statistic than the mean. In particular, the median is much more stable in the presence of modest numbers of outliers. We have not adopted this approach for two reasons. Firstly, calculating the median requires significantly more machine resources than the mean because a sorted list of the samples must be maintained. Secondly, there is no equivalent statistic for the variance.

Given the nature of plateau detection it is appropriate to discard outliers. Outliers are identified as samples that exceed *twice* the trigger value contained in (1).

$$RTT > sample_{mean} + (sample_{variance} * sensitivty * 2) \quad (2)$$

The algorithm can not simply discard samples that meet (2) because they might be the beginning of a new, much higher base RTT. To deal with outliers, the sample window contains two sub-components which buffer different types of trigger samples. The first buffer (the *trigger sample* buffer) stores measurements that meet (1) but not (2). These samples will eventually be passed to the summary window. The second buffer (the *quarantine*) stores samples that meet (2). These are counted as part of the trigger but will only be passed to the summary buffer if the trigger is successful. If the trigger fails samples stored in the quarantine are discarded.

Low Variation Prohibition

HPC networks are intended to give good performance. To this end, their capacity normally exceeds the load they carry by a large margin. This is a design feature (not an indication of under use) and the implication can be see clearly in delay plots like that in figure 7. Indeed, some paths show less that 1ms RTT variation for days at a time.

Long periods of stable behaviour intermixed with occasional variation in RTT introduces a new, but similar problem to outliers in the data. After a long stable period even large variations in the RTT have little impact on the mean and variance. To avoid this problem only samples that vary significantly from the current mean contribute to the new mean and variance.

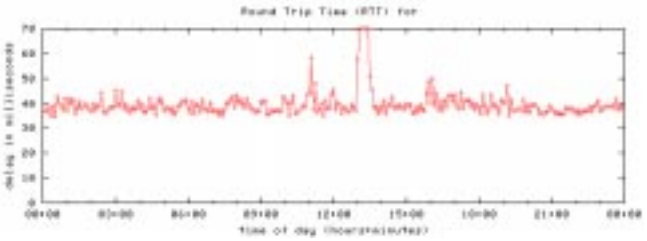

Fig. 7. Friday, 12 Jan

As each sample is inserted into the summary window it is checked to see if it lies within ± 20% of the mean. If it does it is not included into the summary statistics. Unlike samples missing from the original data set, the sample still occupies a space within the window and is passed along with the rest of the data, eventually being removed when it has been in the summary window for the time set by the user as the window size.

When a trigger is active (i.e. when the sample window is not empty) this mechanism is inhibited, and all samples are passed to one or other of the summary window buffers.

So, in summary, new data is compared against the mean and variance of data (in the summary window) that shows variation from the norm. We believe that a human looking at the RTT graphs does much the same thing, subconsciously ignoring the stable periods and focusing on those that vary from the norm. Without this refinement the detector triggers much more frequently in situations that appear to be little different to the surrounding data.

Minimum Trigger Level

If the mean is very stable, and small (say $< 20ms$), it is possible that a trigger could represent just a few milliseconds variation. The low variation prohibition, described above does not prohibit these events because it is based on a percentage variation from the mean. When the mean is small, small variations are permitted.

Such events are unlikely to be of interest and we filter them out, after the main trigger mechanism has occurred.

Trigger Elevation

Once an event has been triggered the data that caused the event is transfered from the sample window buffers to the summary window. However, the summary window is normally 3 orders of magnitude bigger than the sample window and this new data has little immediate effect. If the change in base RTT is a long term one, eventually the summary window will adjust to the new data. In the mean time new triggers will be repeatedly generated.

This is an undesirable behaviour, because the user has already been notified of this new RTT level. To inhibit these subsequent triggers the trigger threshold is temporarily raised to the largest value in the samples that caused the current trigger, plus 20%. Although there might be an outliers in the trigger it is not of concern here. It is expected that the user will investigate the path that caused the trigger and will be aware of the RTTs, including the maximum, even if it is much larger than the other RTTs that caused the trigger.

The raised trigger lasts for one full summary window duration after which time it is removed. Further triggers during this time (which can only result from the base RTT increasing beyond the raised trigger threshold) raise the trigger threshold further and start a new timeout period.

Estimated Running Statistics

Although the mean and variance do not require a lot of resources to calculate for a particular data set doing so over a moving window requires storage of the data for the duration of the window. The AMP system collects more than 1Gb of new data a day. Roughly half of that is RTT data. If many paths are watched a lot of memory will be used. We expect some users to be interested in the overall health of the system. Such a user might select all paths but with a low sensitivity and long summary window (say 7 days) so that only most unusual events are reported. Storing all the samples in memory would require several gigabytes of RAM, for this one user. Because this memory is divided into many small data sets, and each data set is changed each minute as new samples arrive and old ones are removed, these RAM requires are not easily mutable into disk storage.

To reduce memory usage the data that makes up the windows is not stored. Instead estimates of the window's mean and variance are used. These estimates are:

Mean:.

$$\begin{array}{ll} if(n < windowSize) & n = n + 1 \\ else & \sum^{\approx} x = \sum^{\approx} x - \frac{\sum^{\approx} x}{n} \\ \sum^{\approx} x = \sum^{\approx} + x & \end{array} \qquad (3)$$

$$\bar{x}^{\approx} = \frac{total}{n}$$

Variance:.

$$\begin{array}{ll} if(n < windowSize) & n = n + 1 \\ else & \sum^{\approx} x = \sum^{\approx} x - \frac{\sum^{\approx} x}{n} \\ & \sum^{\approx} x^2 = \frac{\sum^{\approx} x^2}{n} \\ \sum^{\approx} x^2 = \sum^{\approx} x^2 + x^2 & \end{array} \qquad (4)$$

$$\sigma^{2\approx} = \frac{\{sum^{\approx}\}^2}{x} - \sum^{\approx 2} n(n-1)$$

Calculation of these estimates is complicated by the presence of omitted samples as described in V-A. When we described the sample window earlier we described it conceptually with samples arriving at one end and being eliminated from the other. Some of these samples are omitted from the mean and variance. When they enter and leave the summary window there is no change to the statistics or the components used to calculate the statistics. This is easy to emulate with the approximations described above when a sample enters the window. However, there is no way to tell when an omitted sample leaves the window.

The information needed to determine when an omitted sample leaves the summary window is maintained in a doubly linked list of tuples. The first member of the tuple indicates whether this tuple describes a set of omitted samples or samples that contributed to the mean. The other member is a count that gives the number of entries of that type at this point in the sequence.

When a new sample is inserted into the summary window the tuple at the head of the list is checked. If its type ('omitted' or 'included') matches that of the new sample the tuple's counter is incremented. If it doesn't match a new tuple is created and linked into the list.

The reverse process occurs when a sample is to be removed from the window. The tuple at the tail of the list is checked. If the type is 'included' the summary statistics are updated. If the type is 'omitted' then the summary statistics are not updated. In either case the counter is decremented and, if it becomes zero, the tuple is removed from the list.

The need to keep a list of this type works against the motivation for introducing estimated running statistics, i.e. to reduce the memory usage of the system. However, the worst case memory usage of the algorithm described here is about the same as keeping all summary data (one slightly larger node per sample). In most cases there are long runs of omitted or included sample and the memory use of the list will be quite small.

A.1 Parameters

There are three parameters that the user can select. These are:

1. Sensitivity: The variance multiplier.
2. Trigger Duration: The number of samples that must exceed the threshold.
3. Adaptability: The size of the summary window.

In the following paragraphs we describe the impact each of these has on the operation of the algorithm and give some hints for setting the parameter.

A.1.a Sensitivity. Sensitivity sets how big the change has to be for a trigger to occur. The sensitivity is multiplied by the variance (the square of the standard deviation) and added to the mean to generate the basic trigger threshold. A small number (say 1 or 2) is a good place to start with this parameter.

A.1.b Trigger Duration. Trigger duration controls how long the RTT needs to have changed before a trigger is signalled. Larger values are more conservative about reporting an event. Shorter values report events more quickly, but may be affected by noise. Because samples that don't exceed the threshold reduce this count quite small values can be used. $5 - 10$ samples is a good starting place.

A.1.c Adaptability. There are two opposing factors in choosing the size of the sample window. A very short sample window will mean that regular variations (such as the diurnal variations that are common on commodity Internet connections) will cause frequent triggers. On the other hand a very long trigger will take a long time to adjust to a change in the base RTT.

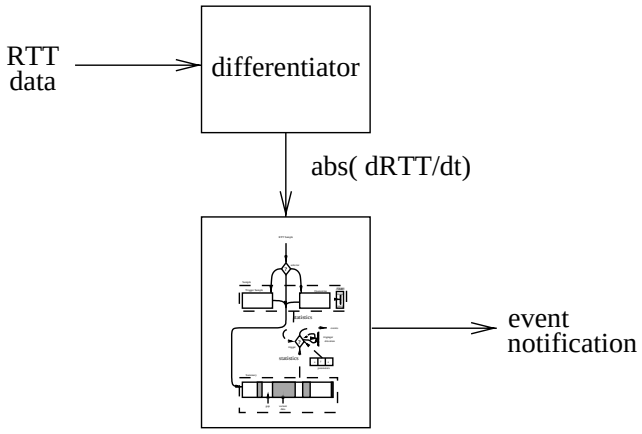

Fig. 8. Example Path

If the data being monitored exhibits diurnal characteristics then at least $1\frac{1}{2}$ days should be used. If the weekend/weekday behaviour is different at least 3 days is recommended. It is possible that values in excess of 7 days could be useful if the data shows high loads on particular days of the week as might be the case if there are weekly backups, although we have not experienced this. For connections with no noticeable diurnal behaviour smaller values of adaptability will allow the system to adjust more quickly.

B. Jitter Detector

The jitter detector is based on the plateau detector with a pre-processor that numerically differentiates the data before passing it to the plateau detector. After the data has been differentiated the sign is discarded and the resulting absolute values are passed into the plateau detector. The procedure is shown in figure 8. An example of the different stages of transformation is is shown in figure 11.

C. Loss Detector

The loss detector is simpler that the other detectors because the acceptable state for loss varies less than for RTT and jitter. The architecture of the loss detector is shown in figure 9. Data indicating the RTT or a loss is feed into the system. This is converted into 0 for a successful RTT measurement and 1 for a loss. An estimated running mean, similar to equation (3), is maintained as an estimate of the loss. When the loss exceeds the threshold indicated by the user a trigger occurs and the trigger level is elevated, as described in the plateau algorithm.

VI. Example

This section shows an example of the plateau algorithm in operation. It also gives an example of how the data is modified by the variance algorithm before being passed into the plateau detector.

Figure 10 shows the RTT for the path between the Startap and Virginia Polytechnic Institute AMP monitors dur-

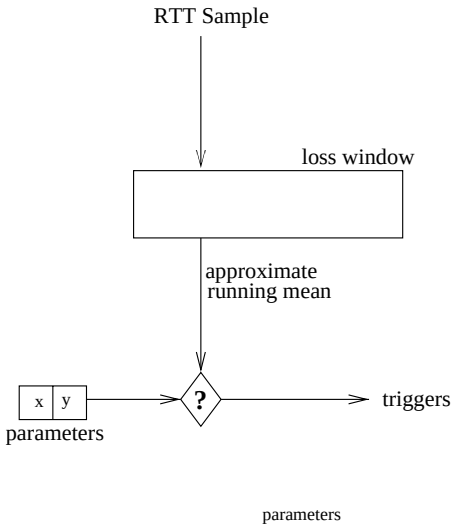

Fig. 9. Loss Architecture

ing January 2001. This data was chosen for the example because it contains a number of interesting characteristics that demonstrate features of the event detection system, not because it is typical of an HPC path. It is actually very unusual; most HPC paths show very little variation. Routing errors account for some of the variation in this data.

The results of running the plateau detector on this data with the following parameters are included below.

- Sensitivity = 1 variance
- Trigger Duration = 10 samples
- Adaptability 3 days

The events that were detected are marked with arrows in figure 10 and are also shown in tabular form below.

Trigger Number	Time
1	Jan 4 14:21:03
2	Jan 8 23:44:02
3	Jan 13 13:02:08
4	Jan 13 17:59:09
5	Jan 14 17:05:08
6	Jan 14 18:55:15
7	Jan 18 19:55:12
8	Jan 22 16:25:08
9	Jan 26 12:13:02
10	Jan 30 19:41:03

Note that the same plateau algorithm parameters are used through out the example. The first and second tigers differ greatly. The first trigger has picked out an event that is only a little greater than the surrounding variation while the second is vastly different. The first could be suppressed by decreasing the sensitivity (increasing the number of variances). The second is likely to be detected whatever the value of sensitivity and adaptability. A very long trigger duration could suppress it because it is a relatively short event.

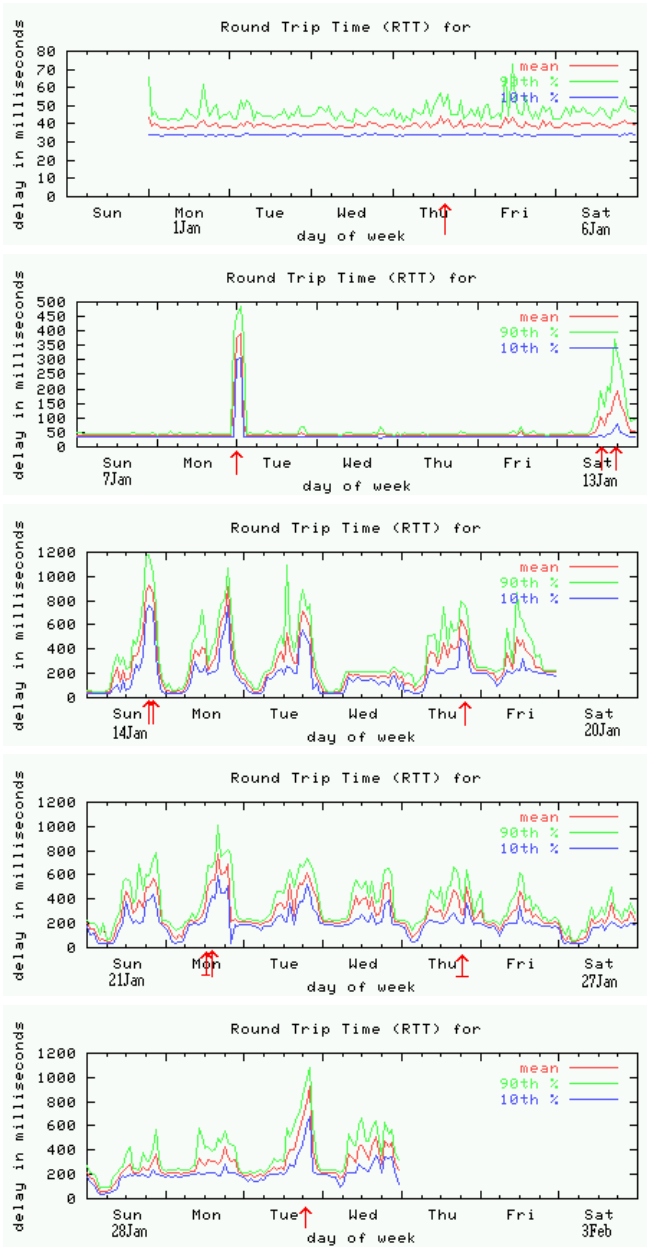

Fig. 10. Example Path

Event 3 occurs at a much lower RTT than event 2, demonstrating the adaption of the algorithm. The second trigger on Saturday 13 Jan (event 4) occurs because the base RTT continues to grow beyond the raised threshold created when the first event that day (trigger 3) occurred. This is appropriate because the user needs to be notified that the RTT has increased significantly since the previous notification.

Events 5 and 6 are similar to 3 and 4 and event 7 is similar to event 3. Event 8 mirrors event 1.

The second to last event (event 9 on Friday 26 Jan) is unusual. It appears to be lower than the peaks of the previous day. In part this is because of the summary nature of the graphs. Careful examination of the data revealed that the peaks on Jan 25 are indeed higher but they are

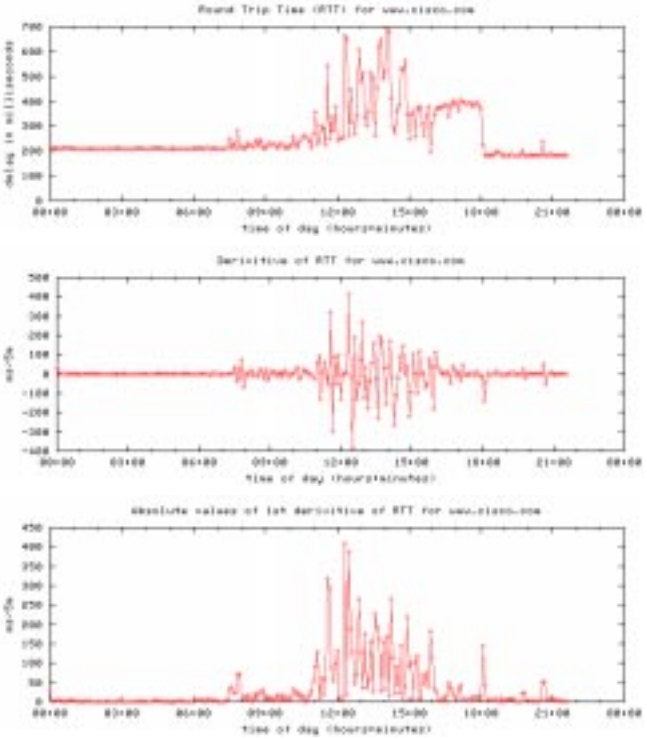

Fig. 11. Jitter Processing

short spikes, while those on Jan 26 are longer. The Jan 25 peaks are not quite long enough to cause a trigger and are treated as transients. While lower sensitivity would remove this trigger it is interesting to reflect on the binary nature of the trigger mechanism. When the data is such that the trigger is near it operation point the presence or absence of a trigger is arbitrary. This is inherent in the nature of notification. It would be possible to generate a trigger strength that indicating how close to the trigger operating point the trigger was generated. Trigger 1 would have a small value while trigger 2 would have a large one. However, eventually a user must either be notified or not notified.

Jitter

Figure 11 shows the transformations on data in the variance detector before it is fed to the embedded plateau detector. Note that periods of little change have been transformed to low values while periods of change have higher values. For a particularly salient example look at the elevated RTT period between 16:00 and 18:00 which becomes close to zero after the transformation because it is a period of little change.

VII. Performance

To test the performance of our initial implementation we created a system with 14400 event detectors and fed it with 5 days data for one path (Jan 1 to Jan 5 of the Startap/Virginia Polytechnic Institute trace shown in figure 10. The data was read from a file and, within the test program, was replicated and fed to all 14400 detectors.

The parameters were set to the same values as the example above, i.e.

- Sensitivity = 1 variance
- Trigger Duration = 10 samples
- Adaptability 3 days

This creates a situation similar to monitoring all paths with a single set of parameters. The program took 2m:28s to run. Memory use stabilised after 3 days data was processed at 209Mb. These results are pleasing and suggest that while high memory use is an issue CPU use is not. It is hard to say if the memory required might become a limit in a practical system. A suitable machine might have 2Gb of memory allowing 10 complete sets of detectors on the data. This seems likely to be adequate, although spending effort reducing the memory usage might be valuable.

VIII. Implementation Status

Achieving automatic event detection has proved to be an elusive goal in practice. The system described in this paper has been arrived at through experimentation and is based, in part, on four previous implementations all of which had limitations. These implementations were based on differentiation, process charts and a similar approach to the current system.

Versions of all the components of the system described here have existed in at lease one of these systems. The current implementation is still under construction and, at the time of writing, consisted of the plateau detector (which was used to produce the results described in sections VI and VII) and most of the user processing module, but not the external user interface (the persistent data store is implemented). The real time data interface is also complete.

The plateau detector is the most complex part of the system and its operation is hard to understand without an implementation. Having completed this component we do not expect any major problems implementing the rest of the system.

IX. Conclusions

A practical automatic event detection would greatly enhance active measurement systems by allowing them to change their mode of operation from user polling to measurement system generated alerts.

After several unsuccessful attempts we have refined our methodology to the point where laboratory tests of automatic event detection appear to be bearing fruit. While the system has a design philosophy, based around comparing the statistics in a sample and a summary window, refinements were required to make the system robust. These were developed, in part, through experimentation with real data sets which highlighted limitations.

Although there is still implementation work remaining, we are reasonably confident that this will not present any unexpected challenges. It seems much more likely that, once the system is deployed, practical use of it will raise new issues.

In this paper have focused on the AMP active measurement system but it is likely that other active measurement

systems would also benefit from a similar system. Before deploying a similar event detector on an active measurement system thought needs to be given to the load generated by the system. The critical parameters are the number of paths and the likely profile of paths being selected for event detection because these influence memory usage. The performance of the existing implementation suggests that total data set size is of less importance. As far as we know, AMP is the largest system of its type so having developed the detector for AMP it seems likely that it will handle the workload of other active measurement systems.

Acknowledgements

The authors would like to acknowledge the work of Pieter van Dijk, an undergraduate student at the University of Waikato, who implemented a previous algorithm for event detection, including the window concept.

References

[1] A.J. McGregor and H-W. Braun, "Balancing cost and utility in active monitoring: The AMP example.," *INET 2000*, July 2000, *also at http://byerley.cs.waikato.ac.nz/ tonym/papers/inet2000.*

[2] J.A. Brown, A.J. McGregor, and H-W. Braun, "Network performance visualisation: Insight through animation," *PAM2000 Passive and Active Measurement Workshop*, pp. 33–41, Apr. 2000, *also at http://byerley.cs.waikato.ac.nz/ tonym/papers/pam2000-cichlid.PS.*

[3] http://moat.nlanr.net/AMP/.

[4] A.J. McGregor, H-W. Braun, and J.A. Brown, "The NLANR network analysis infrastructure," *IEEE Communications Magazine special issue on network measurement*, pp. 122–128, May 2000, *also at http://byerley.cs.waikato.ac.nz/ tonym/papers/ieee-comms.PS.*

[5] http://moat.nlanr.net/.

[6] http://www.caida.org/.

[7] http://moat.nlanr.net/Software/Cichlid.