Don't Just Talk About the Weather—Manage It! A System for Measuring, Monitoring, and Managing Internet Performance and Connectivity

Cindy Bickerstaff, Ken True, Charles Smothers, Tod Oace, and Jeff Sedayao
Intel Corporation

Clinton Wong
@Home Corporation

Don't Just Talk About The Weather - Manage it! A System for Measuring, Monitoring, and Managing Internet Performance and Connectivity

Cindy Bickerstaff, Ken True, Charles Smothers, Tod Oace, Jeff Sedayao
Intel Corporation
Clinton Wong
@Home Corporation

Abstract

In an environment where Internet access is mission-critical, Intel has created the Internet Measurement and Control System (IMCS) with three objectives: 1) Devise quantitative measures of Internet performance; 2) Monitor those metrics to detect performance problems before customers and employees start calling; and 3) Enable first line support in the Network Operations Center (NOC) to handle as many problems as possible without having to escalate to network engineering staff. Intel implements IMCS by measuring key statistics of ping measurements, HTTP GETs, and router accounting tables. Boundary conditions are set up for the key statistics, and alerts are sent if those conditions are exceeded. The NOC personnel that receive the alerts use predefined scripts for each kind of alert. To make IMCS accessible to all and very usable, IMCS presents all of its information on the Web. Even network debugging tools like ping and traceroute are accessible through web interfaces. IMCS has proven successful in detecting problems and changes in the Internet infrastructure, although problems have been encountered because of IMCS's active measurement techniques. Future improvements to IMCS include fixing the configuration format of boundary condition definitions, adding more services to be monitored, increasing the use of passive measurements, and improving how alerts are reported.

1. Introduction

Internet Performance is like the weather - people talk about it but feel helpless to do anything about it. But you can do something about it! Like many large multinational corporations, Intel has many thousands of employees and contractors who use the Internet through Internet gateways dispersed through the US and the world. Despite a massive investment in Internet Connectivity, Intel had no quantitative measurement of whether its Internet connections were performing well or poorly. In addition, Intel has a Network Operations Center manned 24x7 but only a limited number of engineers working on Internet Connectivity issues. In this environment of heavy use, no quantitative performance data, and small amount of staff, Intel created the Internet Measurement and Control System (IMCS) with the following objectives:

- Devise quantitative measures of Internet performance
- Automatically monitor those metrics to detect performance problems before customers and employees start calling
- Enable first line support in the Network Operations Center to handle as many problems as possible without having to escalate to the Internet network engineering staff

This paper describes how Intel built IMCS to meet these goals and how well IMCS can help in managing Internet performance. The first section details the challenges that lead us to create IMCS. It talks about Intel's Internet Connectivity environment and the key goals of IMCS. The second section goes over the IMCS approach - the techniques used to quantitatively measure Internet performance and our strategy to present that information and make it usable. The third section details how we implemented IMCS. It talks about specifically about our performance measurements, measurement architecture, and user interface design and implementation. The next section talks about Intel's experiences with IMCS, and the last section talks about our plans to enhance IMCS.

2. The Challenge of Internet Performance Management

Prior to the initial deployment of the IMCS in mid-1997, Intel's Internet use focussed on the use of Intel's corporate presence server (CPS) by external customers and on the use by employees for business use and reasonable personal use. The availability of the Internet had become a critical part of standard business procedures much like a fax or a telephone. Since IMCS has been deployed, the criticality of the Internet has entered the Intel business processes for more than a billion dollars per month at our order placement eCommerce site. [1] These business procedures extend across all of Intel's customers, groups and divisions around the world. Multiple geographically diverse gateways have been deployed across Intel. Consistent access policies to these gateways' routers, servers and bastion hosts have been implemented. A tiered and scripted support approach was developed to address the requirements for 24x7x365 coverage, restricted secure access and limited Internet Connectivity staff. Skilled NOC staff were already available 24x7x365 for Intel network business needs and were the designated resources for first level support for Intel's Internet connectivity.

Now that the Internet has become a critical business tool, the following IMCS goals were established to meet the challenge of managing its performance:

- Define quantitative measures of Internet performance that can be used to manage connected ISP performance and track/act on levels of service delivered to external and internal customers.

- Detect performance problems before customers start calling to complain of performance problems

- Enable first line support to consistently and securely resolve as many problems as possible.

3. The IMCS Approach

Intel's approach to managing Internet performance is fairly straightforward. First, we want to quantitatively measure Internet performance. To do that, we define quantitative performance metrics corresponding to real performance issues and then consistently measure them. Once we have defined metrics and measure them, we monitor Intel's Internet gateways for when the measured values of the metrics violate some boundary condition for good performance. When a boundary violation is detected, the Intel NOC goes through carefully scripted actions in order to find the problem and contact our ISPs if necessary.

It's worth discussing this quantitative approach in a little more detail. The core of this approach is to obtain performance data, process it with an algorithm that includes comparison to usual performance and act when unusual performance is detected. Action takes place once the data demonstrates unusual behavior. Note that all parts of this process are critical. Actions are based on quantitative data. The data should be meaningful, reflecting in some way real Internet conditions. Comparison to usual performance or "limits" is important. We need a defined trigger for taking actions when a performance variable exceeds a limit. Otherwise, we may take actions when conditions do not warrant it (false positive) or not do anything when conditions demand it (false negative). Finally, actions are important. There is no point in measuring something if you are not going to act on the data.

The second part of Intel's approach involves the way IMCS presents information. All of IMCS is accessible from the Web by everyone at Intel. We do not hide our performance information since Internet performance affects almost everyone at Intel. All of the debugging tools like traceroute and ping and the troubleshooting scripts are made available to Intel's NOC (and everyone else) via the Web. Again, we do not hide performance behind some management console available only to a few people. This gives us incentive to do a good job with Internet performance. It also can be a convenient time saver. When there are questions about Internet performance, we have the ability to answer those questions with a URL showing the actual performance.

As part of the web display of performance, we graph our performance variables. This makes its easier for people to understand, rather than presenting raw data. There are a number of advantages to making our Internet debugging tools web based. Since they are web-based, our NOC and other interested parties do not need log in access to routers in order to debug problems. Also, the complexity of running certain debugging tools is reduced by allow users to execute the tools by filling in web forms and hitting a button.

4. Implementing IMCS

Now that we have described our approach to Internet performance monitoring, we will go over how we implemented it with IMCS. The first part of this section covers how we defined summary statistics for performance, what algorithms we used to determine whether boundary conditions were violated, and our statistical treatment of network performance data. The second part describes how we implement the user interface to IMCS – the alerting and web interfaces. The last part goes over some other implementation highlights such as implementation languages used and configuration interface design.

4.1. Creating Quantitative Measures of Internet Performance and Setting Boundary Conditions

As we mentioned in the discussion of our approach, IMCS measures various network performance metrics and trigger alarms when the data exceeds some threshold. But what metrics does it look for and why? How are boundary conditions triggered? What values of the data set off the triggers? In this section, we answer these questions about the IMCS implementation. First we talk about the types of measurements that we take. After that, we talk about how we summarize the data we collect and how determine if performance has slipped out of our boundaries for acceptable performance. We will briefly discuss the use of robust statistics in the network performance space and then discuss how all of the data collection, analysis, and out of bounds checking modules work together.

4.1.1. Types of Network Metrics

IMCS has three distinct modules it uses to measure network performance: *Timeit*, *IP Stats*, and *Imeter*. In this section, we discuss these collection agents that we use. For each module, we will talk about how we collect network performance data with it, why we use this module, and what metrics we derive with it.

Before moving on to what we measure, we should mention how the IMCS measurement systems are positioned. A system within each firewall complex does the network performance measurements and displays the results on the web. As a result, IMCS monitoring at each gateway is independent from another, and the system keeps functioning if one or more Internet gateways are unavailable.

4.1.1.1. The Timeit Module

Timeit [5] is the name of a Perl script which fetches web pages from a specified list of URLs and then provides summary information about the transfers. Timeit is also the name of the IMCS module that uses the timeit script to determine performance of web connectivity between Intel and the rest of the world. Since a vast amount of Internet use is web use, measuring HTTP Get operations gives us some idea of how the Internet functions for users of our web sites and for our internal users use of the Internet.

Many pieces of information are collected from each set of URLs fetched. For each URL, we measure the length of time it takes to perform a DNS lookup of the site we will access and the time it takes to setup a TCP connection to that site. And once we've connected, we measure the rate at which we receive the web page in bytes per second. For the times that we cannot get the web page, we record the failure rate. When we retrieve a URL, we only get the initial page and not any graphics referred to by the page. This is so we can keep the URL fetching process uniform between different web sites.

We look at each of the results for a reason. DNS lookup time can be a significant factor in web performance. We measure it in order to track whether it becomes a problem. Connect time is often association with network delays. Measuring rate as opposed to download time is a way to make comparisons between large pages and smaller pages fairer. Finally, error percentage tells us how much the HTTP gets were successful. The download rate must always be looked at together with the error rate. A high download rate might seem good but it isn't good if accompanied by a high error rate.

From the URL set results, timeit reports DNS lookup times at the 50th and 75th percentile values. This shows the central tendency of DNS lookup performance as well as slower lookup times. Timeit reports connect times in the same way, at the 50th and 75th percentiles. Transfer rates are reported using the 25th, 50th, and 75th percentile values. The number of URLs that timeit attempts to fetch and the percentage of failed requests are also reported as metrics. We have an estimated user experience metric which ranges from "poor" with a value of 1 to "good" with a value 5. This statistic is based on the connect time.

Statistics must be based on meaningful input in order to produce meaningful results. In the case of timeit, each set of URLs must be large enough so that variances in one fetch do not paint an incorrect picture of the performance whatever part of the world we are measuring. Each set needs a large number of URLs, picked from a wide variety of sources. We measure 48 sets of urls with between 1 and 78 urls in each set. The set with a single URL is a special URL to the local host. This measurement gives us some idea of the loading of the IMCS measurement system.

In addition to measuring the web performance from our customers and partners to us, we care about performance of web access from our employees to the

Internet. At Intel, web proxy servers are part of our Internet firewall strategy. While having timeit monitor web performance to many parts of the world is useful for troubleshooting employee Internet access problems, it doesn't show what performance is like through the proxy servers. Thus we have timeit do measurements through the proxy servers.

Four of the 48 sets of urls that we have timeit monitor are proxy urls. In each of our Internet gateways that have proxy servers, we monitor performance through the local proxy to 42 sites within North America. We also measure performance to the same sites without going through a proxy. This allows us to pinpoint whether proxies are behaving slowly or the Internet in general is not being very responsive. If we see a drop in rate for proxy downloads while we see no drop in direct downloads, we know that there is something wrong with the proxies.

One of the main criticisms of using HTTP Gets, like timeit does, for network monitoring is that server loading becomes a factor in the numbers you can obtain. We get around this by measuring the same sets of URLs from multiple locations in roughly the same time period. Doing this washes out the local effects of a server. If a server is slow, all of the Timeit fetches should reflect that slowness. If some are slow and others are not, the slowness should be attributable to network conditions.

4.1.1.2. IP Stats

How much traffic is passing through our firewall in a single day? Who is using it and for what? How much traffic went through in the last 15 minutes? Are we running out of capacity? The *IP stats* module helps answer these questions.

The IP stats module collects data from the IP accounting table of each firewall complex’s outer firewall routers. It gathers byte and packet counts of IP source and destination address pairs. We collect this data by doing the following:

- A cron job runs every 5 minutes and runs an Expect script that does a "show ip accounting" and a "clear ip accounting." This generates a list of source and destination IP addresses with the corresponding byte and packet count between the two.

- Every 15 minutes, this data is aggregated and reported via the web interface.

- Results are also copied into a daily log file.

Because the accounting data contains source and destination information, byte, and packet counts but not port information, we must infer what type of traffic was passed. We maintain a list of specific servers and a list of special networks within the company.

By categorizing the traffic based on these lists, we can begin to understand the who and the what parts of our earlier question. Noticing whether the Intel address appeared as the source or the destination tells us whether the traffic was inbound or outbound.

The IP stats module does the following categorizations:

- **Determines whether traffic is inbound or outbound**. By examining whether the source address or the destination address was that of a host inside the firewall, we can see the direction of traffic. This indicates whether the byte count should apply to the inbound or outbound statistics.

- **Categorizes the type of traffic**. As previously stated, we only have source and destination information, not destination port number. But because we tend to keep certain types of traffic on specific hosts, we can assume that traffic caused by any given host was of a specific type. We don't know exactly which packets were passed with a destination port of 80, but because the IP address was that of a web proxy server, it was probably due to web traffic. Similarly, we can count and categorize traffic as mail, DNS, USENET News, or something else.

- **Categorizes the site**. In addition to the type of traffic, we want to know which sites within Intel are generating traffic and if so, how much. Changes in routing can cause one site’s traffic to flow in one Internet gateway and out another. By monitoring for this condition, IMCS can warn us of broken routing conditions. If we see lots of Santa Clara traffic entering through our Chandler Arizona gateway, there is probably something wrong.

Once the daily logs from each IMCS system are collected and aggregated, further calculations can be made. This information is useful when determining which facility needs additional Internet bandwidth, or perhaps which facility needs its own Internet connection.

In order to make all these assumptions, we must maintain a list of which networks within the company are in use at which facility, and which servers are used for what purposes. Sometimes, instead of categorizing based on a specific host address, we categorize based on the subnet address. For example, the people running our www.intel.com servers install and replace their servers at a rapid pace. Those servers have their own subnet, so any traffic due to servers on that entire network are thrown into the www.intel.com bucket. This keeps us from having to update our server list at the rate that they make changes.

This brings us to the topic of a very important traffic category, the amount of traffic that can be attributed to www.intel.com. There is a steady stream of traffic through our firewall due to people on the Internet visiting our www.intel.com servers. It is a corporate objective that the quality of this web site be maintained at a very high level. If the quantity of traffic due to the web servers falls below a threshold, it might mean that there is something wrong with out Internet connection. Outages for the www.intel.com web site are high visibility problems, and quick action is required. In addition, we use these statistics to determine when we need to upgrade the circuits which connect www.intel.com to the Internet.

4.1.1.3. Imeter

Imeter (first known as Lachesis [7]) is intended as a general measure of the quality of Internet connectivity. It works by sending out ICMP echo-requests to key Internet landmark sites and measuring delay and packet loss (ping). We include in our list of key Internet landmarks sites that Intel employees access a lot as well as key parts of the Internet infrastructure such as the root name servers. Clearly Internet application performance will suffer without good access to the root name servers.

The metrics that we derive from Imeter are *percentage of loss packets* and *median packet delay*. Like HTTP download rate and error rate in the Timeit Module, these metrics are complementary and must be studied together. Low packet delay must not come at the price of high loss. One solution to loss in a network is to make packet queues longer. This results in more delay.

4.1.2. Data Summarization and Boundary Analysis Algorithms

Now that we know how IMCS measures different aspects of network performance, what do we do with the data that we generate? There can be a tremendous amount of raw data associated with each IMCS metric. A single metric might have many data points taken in a single collection interval. To handle all of this data, IMCS aggregates raw data into summary statistics. We often use summary statistics of metrics, making use of the seventy-fifth percentile, median (50th percentile), or fractional representation of collected data. These statistics, known as Response Summary Statistics (RSS), are then run through an Out of Bounds (OOB) checker to determine whether a boundary condition has been violated.

The OOB checker reads a configuration file to apply one or more bounds checking algorithms on an RSS. Currently, there are three algorithms: *Static Value*, *Delta Fraction*, and *Delta Shift*. Each algorithm compares the current RSS value with the previous RSS value, using a *shift* value, *A/B* value, and *last_result* flag to determine if the current RSS value is “out of bounds” with respect to historical data. We will explain all of how these variables interact with each algorithm.

The *shift value*, in the context of the Static Value algorithm, indicates the maximum or minimum value that an RSS can have without triggering an out of bounds condition. With the Delta Fraction algorithm, the shift value specifies how much of a fractional difference can occur between two consecutive data points without triggering. Another way to think of this algorithm is that the shift value is the percentage that two data points can differ by without triggering an alert. With the delta shift algorithm, the shift value is maximum amount that two consecutive RSS data points can differ by without trigger an out of bounds condition. A shift value indicates how much an RSS is allowed to change relative to the previous data point. The shift boundaries are absolute limits as with the Static Value algorithm. The shift value is a fixed number with the Delta Shift algorithm. It is a fraction with the Delta Fraction algorithm.

The *A/B* value indicates if the shift the specified shift value applies towards downward or upward trends in the data. Finally, a last_result flag is initially cleared, and set when an out of bounds condition occurs. It is used by the OOB checker to ensure that an alert doesn’t happen unless the trend happens over more than two samples, avoiding frivolous alerting on one-time exception data.

We create the different RSS boundary checking algorithms to deal with a variety of situations. If we felt that a particular RSS metric was well controlled and understood, then we would set up hard boundaries with the Static Value algorithm. A state table for the Static Value algorithm is shown in Table 1.

Table 1: State table for the Static Value algorithm

If A/b is	And RSS is	And last_res is	Action is
A	> shift value	0	last_res = 1
A	< shift value	1	last_res =0
A	> shift value	1	Send alert
A	< shift value	0	No action
B	< shift value	0	Set last_res=1
B	> shift value	1	Set last_res=0
B	< shift value	1	Send alert
B	> shift value	0	No action

If an RSS metric was not as well understood, we would use the Delta Shift algorithm. Table 2 contains the state table for the Delta Shift Algorithm.

Table 2: State table for the Delta Shift algorithm

A/B	If RSS(t) is	Last_dat +	Last_res is	Action is
A	<	Shift value	0	Last_dat= RSS(t)
A	<	Shift value	1	last_dat = RSS(t) last_res = 0
A	>	Shift value	0	Last_res= 1
A	>	Shift value	1	Send alert
B	>	-Shift value	0	Last_dat= RSS(t)
B	>	-Shift value	1	Last_dat= RSS(t) Last res=0
B	<	-Shift value	0	Last_res= 1
B	<	-Shift value	1	Send alert

If we did not have a good understanding of the behavior of the metric, then we would tolerate much wider swings by using the Delta Fraction algorithm. Tables 3, contains the state table for this algorithms.

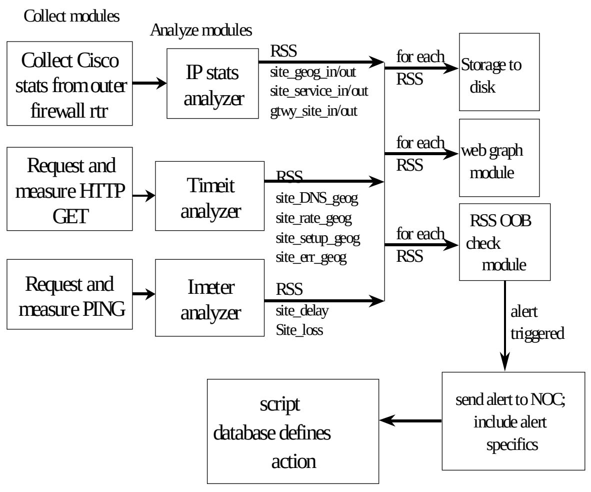
Table 3: State table for the Delta Fraction algorithm

A / B	Multiply RSS by	If column 2 product is	Last_dat multiply by:	And last_res is	Action is
A	(1 –shift value)	<	(1 +shift value)	0	Last_dat= RSS(t)
A	(1 –shift value)	>	(1 +shift value)	1	Last_dat= RSS(t) Last_res= 0
A	(1 –shift value)	>	(1 +shift value)	0	Last_res= 1
A	(1 –shift value)	>	(1 +shift value)	1	Send alert
B	(1 +shift value)	>	(1 –shift value)	0	Last_dat= RSS(t)
B	(1 +shift value)	>	(1 –shift value)	1	Last_dat= RSS(t) Last_res= 0
B	(1 +shift value)	<	(1 –shift value)	0	Last_res= 1
B	(1 +shift value)	<	(1 –shift value)	1	Send alert

4.1.3. Robust Statistical Treatment of Data

You may have noticed that we have been using percentile statistics and have not used terms like *average* and *standard deviation*. It has been well established in the network measurement literature [2,3,4] that Gaussian (normal) or Poisson distributions do not represent network performance data well enough for planning or predicting purposes. Network data are characterized as “self-similar” [2]. Willinger’s paper clearly demonstrates that the Central Limit Theorem does not apply to network traffic data. As such, the traditional process control summary statistics of mean (for central tendency) and standard deviation (for variability) are not valid when applied to network data. Thus, for the IMCS the central tendency is summarized by the median or 50th% of the aggregated data. The spread between the 75th and 25th% of the aggregated data quantifies the variability. The spread between the 75th and 25th% is called the inter-quartile range (IQR). The 25th, 50th and 75th percentiles are plotted on one graph to minimize the number of graphs that need to be presented and to enhance understanding the customer experience. Traditional statistical process control methodology uses mean and standard deviation or X-bar and R charts for graphically presenting summary results of central tendency and variability. As most network engineering staff are less familiar with quantifying variability, IMCS plots the 25th and 75th percentiles instead of the IQR.

Figure 1: IMCS Data Flow



4.1.4. Data Flow between Modules

Earlier, we talked about how Imeter, Timeit, and IP Stats generated data with separate data collection modules. We also mentioned how data was fed into the OOB checker to determine out of bounds conditions. How does all the data flow between IMCS modules. We show the data flow and key metrics in Figure 1. The raw data generated from the collection modules are transferred to analyzer modules, which compute data values RSS. The RSS data values are stored to disk, sent to the graphing module for display, and piped into the OOB checker for alerting. If an out of bounds condition is detected, then an alert is sent to the Intel Network Operations Center (NOC).

4.2. Monitoring IMCS Metrics and Responding to IMCS Alerts

Figure 1 shows that when an out of bounds condition is detected, an alert is sent to Intel's NOC. What happens when that alert is sent? How does the NOC know what to do? This section describes how we implemented the user interface to IMCS and how alerts are processed.

Once IMCS generates an alert, an alphanumeric page is sent to the Intel Network Operations Center describing the RSS that is out of bounds. If you look at figure 1, you will notice that each RSS metric contains the site name and the type of metric (loss, delay, HTTP rate, etc.). NOC personnel execute a script based on the type of metric. For example, if a delay alert is generated, there is a specific script that must be done that is different from say, an HTTP rate alert. The site information lets the NOC know where the problem lies and who to dispatch to fix the problem.

The main interface to IMCS is a web page that we call "The Big Picture." The Big Picture is a snapshot of all of Intel's Internet connectivity in one screen. This web page is a large table where each row is a different type of metric (e.g. packet delay) and each column represent a firewall complex. In each cell of the table is a thumbnail graph of the appropriate metric for each firewall are shown. The thumbnail shows a rolling 24 hour graph for that metric. This allows someone looking at the big picture page to see a comparative view of the metric from each firewall. At the start of on each row of the Big Picture is a script that describes how to handle alerts for the metric.

Across the top of each column is general information about that particular firewall complex. There is a web link to the history of alerts for each gateway, as well to as IMCS System Status for each gateway. If a gateway IMCS system is having a problem then its System Status turns red. Otherwise it is green.

Each miniature "thumbnail" graph on the Big Picture is linked to a web page containing more information about that gateway's metric. This makes it easy to quickly drill down and get more information about a problem or look in detail at a metric. On these more detailed pages, we display graphs of rolling 24 hours, the current week, the last week, and the past 20 weeks for that particular metric. We can also get the actual data points for recent values of the metric. This is useful when you need real data values for troubleshooting a problem or for tuning the alert threshold values.

IMCS scripts usually have the NOC personnel check if an out of bounds condition is local or Internet wide and change their response accordingly. On each metric row, there is a pointer to the proper script that is to be executed when an alert for that metric is received.

The modular construction of IMCS and its geographically distributed monitoring systems lends itself well to using the web as the primary reporting and access mechanism. In order to remain somewhat browser and HTTP server independent, all of the IMCS display and query functions are built on standard HTML pages, with frames and JavaScript used sparingly to provide easy control, and common CGI Perl programs to remotely generate HTML pages based on GET-mode queries. Debug programs such as ping, traceroute, and whois are implemented as CGI programs. The use of GET-mode CGI queries allows you to copy an URL to a mail message, send it, and have someone else see the results of the query too. It also enables us to embed actual pings or traceroutes into a script HTML page. We can have a link such as “ping the gateway interface” point to a URL that actually pings the gateway interface and return the results.

While each IMCS system creates HTML pages and graphs/thumbnails for each metric on its own document tree, a special setup was required to produce the “Big Picture” that would accommodate the potential outages of a remote IMCS system, and display the overall health of the firewalls. The page containing the Big Picture is not on several servers – only one. We discovered that browsers don't always flush graphics from the browser cache (even with a Pragma: no-cache, and all the other standard tricks), unless the web-server also emits an Expires: header for the graphic. Our previous web servers did not have an Expires option so we use the Apache server, which does have the Expires option. A cron job was constructed to run every three minutes to grab the latest thumbnails for a Big Picture display. This job refreshes the graphics for display and reports possible problems with the measurement process.

4.3. Other implementation highlights

IMCS is implemented with PERL, C, and Expect. The analyzers, OOB checker, pager, and graph generator are done in PERL. The PERL graphing code makes use of *GNUplot* and *pbmtools* to generate GIF files. Packet delay and loss data are collected by *fping*, a C program. Router statistics are collected by an Expect script.

Individual modules are configured with text files on the file system of the IMCS monitor. Only one configuration file exists for a particular module – it is edited and stored in one central location and pushed to IMCS machines through our Make driven update process [8]. Processes on each machine go through a rule set to see which configurations and directives apply to the current process.

As we mentioned above, detail pages are created and updated for every metric that IMCS measures. This includes metrics displayed on the "Big Picture" as well as many more metrics which are not on the "Big Picture". Most of the additional metrics target specific countries or regions of the world. These metrics are used to determine the network performance of our electronic commerce activity and allows us to monitor and improve that performance.

We mentioned above that recent data is available in text format via a "View recent data" web link at the top of each detail page. Incidentally, the simple text format is how IMCS stores its data, and is what the graph_rss tool reads to generate its graphs. Each data point is represented by a line of text with two whitespace separated fields. The first field is a UNIX time value, and the second is the value of the metric at that point in time. New data in this format is easily appended, and graph_rss uses a binary search to quickly find the start of data it needs.

In addition to graphical and textual view of data, we have a web tool that lets us do side-by-side comparisons of full sized graphs. We also have a "My Big Picture" tool to construct custom display of thumbnail graphs. These tools are extremely useful when we need to compare IMCS metrics not on the "Big Picture".

5. Experiences with IMCS

What are Intel's experiences with IMCS? Our experiences have been generally very good. We have found that IMCS is effective in finding problems and extremely extendable. We did encounter a number of other issues, though. In this section, we will detail our experiences with IMCS, starting with ability to find problems, describing its extensibility, and talking about other issues that we ran into.

5.1 Discovering Problems

IMCS is pretty good at finding problems. Imeter delay and loss usually were the first indicators of line down or congestion conditions. This is because Imeter's role in monitoring general conditions and its relatively frequent 10 minute polling interval. Figure 2 shows periods of high delay. In this particular case, we noticed that there we began to experience high periods of delay after one maintenance window.

Figure 2: Delay problems after network "maintenance"

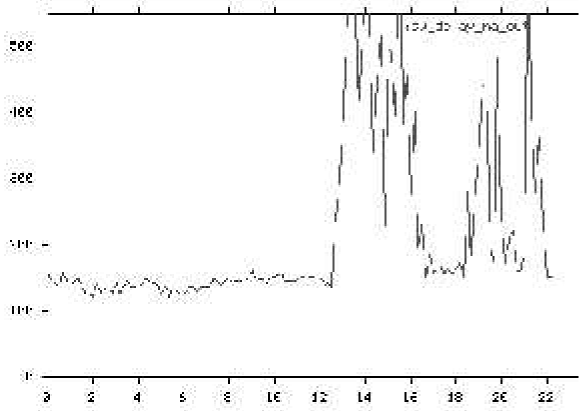


Figure 3: Proxy HTTP Error Rate During Network Problem

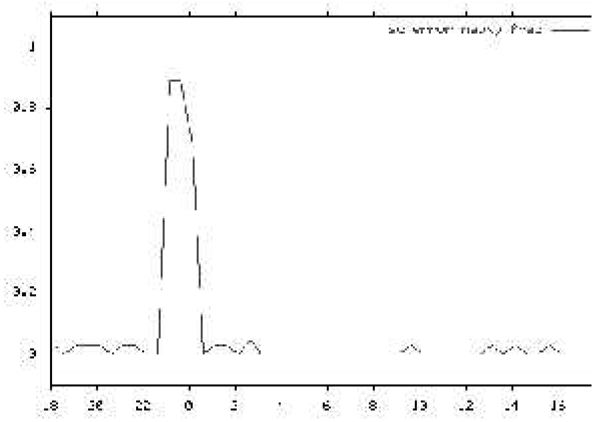


Figure 3 shows a high error rate for proxy HTTP get requests. This particular incident occurred during a problem with a DMZ network segment. The graph shows the start and end of the problems as errors increase during the episode and decrease after the segment was fixed.

As previously noted, timeit is run against URL sets that represent geographic diversity within a country and industries relevant to Intel business foci within the same country. Prior to deploying significant business applications in a country, IMCS results are evaluated against criteria known to result in relatively good performance. If the country's performance is expected to result in poorer performance for Intel's business application customers then effort is applied to find and fix possible sources of problems. Such an effort was undertaken with the main regional ISPs for the country shown in Figs 4 and 5.

Figure 4: Performance to City 1 Before and After Corrective Effort

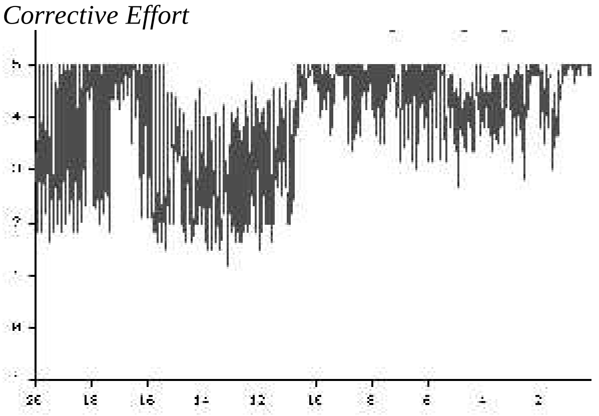
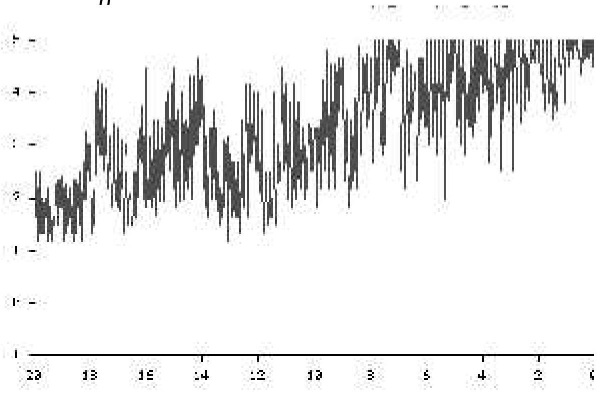


Figure 5: Performance to City2 before and after Corrective Efforts



By early December 1998, one region showed a substantial improvement in the time of day variability of performance as measured by the narrowing of the daily highs and lows in each of the 25th/50th/75th percentiles for the transfer rate and the decrease in connect times. The other region for the country experienced gradual improvement over a four week period primarily in the connect times. Since all three of the 25th/50th/75th percentiles were impacted at the same time it is very unlikely that all 18-20 target sites improved simultaneously. This suggests network infrastructure improvement. The Intel timeit measurement tool measured other countries at the same time. No other countries showed improvements simultaneously with this country's improvements. This suggests the changes were not due to Intel measurement or network systems. A request has been made to identify the infrastructure changes implemented by the country's ISPs but as of this writing they are unknown.

5.2 Extensibility

The modular construction of IMCS makes it easy to add a new metric to the system. A recent example of this was the addition of a User Experience metric (*usrexper*) which is based on a categorization of the timeit connect metric to yield a number from 1 to 5 indicating the probable user's experience (1=poor, 5=execellent). To add the metric to timeit processing to the IMCS systems we did the following:

- Modified the timeit_rss program to compute the new usrexper metric <gateway>_usrexper_<region>_estim
- Modified the graph_rss program to handle graphing of the usrexper metric.
- Modified the timeit_pretty program to generate the HTML index page for timeit to include the new usrexper pages.

Since graph_rss generates both the full-size and thumbnail size graphs of each metric, and the HTML pages, the results were available at once. To fill in the historical data for usrexper, the <gateway>_connect_<region>_[50|75|iqr] files were processed (one time) to yield the usrexper streaming files on each IMCS system.

By adding a new CGI script on the IMCS system, you can make available new debugging tools. The best example of this was the addition of the Looking Glass functionality (based on the Perl scripts from Digex). This script allows a query to the router to be performed without requiring the use of the router's access password(s) by the end-client. The passwords are provided by two scripts (aetna and lemnos) which are not executable directly via CGI, but are invoked by the CGI programs as part of the Looking Glass functions.

A modification to the Looking Glass program allowed us to write a router-based Imeter (where delay and stats are gathered from PING on the ISP router). The results were processed by the Imeter RSS code to create new streaming files and graphics. Router based Imeter allows us to compare performance between ISPs at the same gateway.

One of the more interesting extensions of IMCS is letting IMCS systems monitor their own performance statistics. Our IMCS boxes currently track their own load averages and CPU idle time and make that data available on the web.

When the 'My Bigpicture' function was created, a new framed HTML page set was created, with JavaScript enabled selection functions to invoke the new my_bigpicture.pl CGI script. The support CGI script (available_metrics) was deployed to the IMCS systems, and a get_available_metrics_list.pl program is run (when the metrics are added/deleted) to create a control file (available_metrics.ini) for use by the my_bigpicture.pl program.

5.3 Issues Encountered

We ran into a number of problems with IMCS. The first problem is really part of any kind of alert system based on limits. It is difficult to pick limits that are effective. The only way that you can achieve good boundary conditions is to guess at some limits, and see if you get false positive alarms or see false negatives situations. If you get false positives, you need to make your limits less stringent. We wanted to avoid too many false alarms because this results in the NOC personnel no longer taking IMCS seriously. If we see false negatives, then we need to make the boundary limits more stringent. False negatives mean that the tool is not doing its job.

As conditions change, the metric limits need to change too. The most common condition we encountered with this is with Imeter delay limits. As Internet usage increases at a site, the site's line to the Internet becomes increasingly congested. We see this as time of day swings in delay. Because there is usually lead time in getting bandwidth upgrades, we have had to change the metric limits to reduce the number of alerts generated. Congestion caused alerts are not useful after we have learned that the line is congested and needs upgrading. After the line upgrade and congestion goes away, we go in and make the delay limits more stringent.

The way we implemented the trigger limits continues to cause us problems. The trigger limits for a metric and the last datapoint for the metric are stored in the same file. This makes it impossible to centrally manage and distribute trigger configuration files because the contents of the file containing the limits is constantly changing.

A final problem we encountered is with active measurements. To measure performance, we generate live network traffic. Some sites turn off ICMP access in fear of ICMP based attacks like SMURF [9]. This reduces the number of landmarks that an ICMP echo based tool can monitor. ISPs also have the ability to affect the priority of ICMP packets in their networks. Given that the typical action would be to reduce the priority of ICMP packets, Imeter would be less reflective of actual user traffic in this such networks.

To eliminate the effect of web server performance, we measure performance to identical web sites at roughly the same time from all of our Internet gateways. As a result, each measured web site gets multiple hits from Intel measurement system. We have received a few complaints from web sites that we are measuring. IMCS measurement traffic can be an issue if the web sites have fees based on the volume of traffic they generate. In such cases, we usually remove the web sites from our URL lists.

6. Future Improvements

We are planning a number of additions and improvements to IMCS. These include monitoring additional services, finding new ways to measure performance, and modifying its alerting functionality.

Other Internet services are promising targets for IMCS monitoring. In particularly, we are looking at focusing IMCS on SMTP relaying and DNS services. For SMTP, we want to monitor queue depth, propagation delay through a server, message traffic, and message size. Coupled with monitoring system performance, looking at these will give us a thorough view of how well our SMTP relays are functioning. For DNS, we want to monitor response time for response to general queries and for queries of the zones that our DNS servers serve.

Since active measurements have a number of drawbacks, we want to start focusing more closely on passive measurements. SMTP relay logs, web server logs, and proxy server logs contain a wealth of information that can be mined for performance information without generating live network traffic. We also want to start looking at other forms of passive data such as netflow export data from cisco routers.

Finally, we want to improve the way that we do alerts. Paging has its problems as we have mentioned, and we want to find other ways of notifying NOC staff and others, such as screen color changes and e-mail.

References

[1] “Intel Sets Commerce Record.” *InternetWorld*. November 23, 1998.

[2] Leland, Will E., Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *Proceedings ACM SIGCOMM 93*, 13-17 September 1993

[3] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Transactions on Networking*, 2(4), pp. 316-336, August 1994.

[4] Paxson, V., G. Almes, J. Mahdavi, M. Mathis. “Framework for IP Performance Metrics.” RFC 2330, May 1998

[5] Lidl, K, ftp://ftp.va.pubnix.com/pub/uunet/timeit-2.1.tar.gz, October, 1996.

[6] Libes, Don. *Exploring Expect*. O’Reilly and Associates, Inc., Sebastopol, CA 1995.

[7] Sedayao, J. and K. Akita. “LACHESIS: A Tool for benchmarking Internet Service Providers,” *Proceedings of the Ninth System Administration Conference (LISA IX)*, Monterey, California 1995.

[8] Hambridge, S., C. Smothers, T. Oace, J. Sedayao. “Just Type Make! - Managing Internet Firewalls Using Make and other Publicly Available Utilities.” *Proceedings of 1st USENIX conference on Network Administration*. Santa Clara, California, April 1999. 7] Sedayao, J. and K. Akita. “LACHESIS: A Tool for benchmarking Internet Service Providers,” *Proceedings of the Ninth System Administration Conference (LISA IX)*, Monterey, California 1995.

[9] CERT Advisory CA-98.01. "'smurf ' IP denial of service attack". Originally posted January 5, 1998. URL: http://www.cert.org/advisories/CA-98.01.smurf.html.