

HEAD:

Probing Network Characteristics

DEK

*A distributed network
performance framework*

Bio

*Michael is a principal software engineer
at Proficient Networks. He can be reached
at mike@lrlart.com.*

Byline

Michael Larson

**NOTE TO AUTHORS:
PLEASE CHECK ALL URLs AND
SPELLINGS OF NAMES CAREFULLY.
THANK YOU.**

Pullquotes

*A key aspect of the
PerfScout design is
its flexibility*

It is one thing to blindly push packets onto the Internet, and quite another to monitor, record, and act on packet performance. For instance, say that the available network bandwidth is of interest and you would like to act on a sudden constriction of a connection's available bandwidth. Identifying a sustained drop in bandwidth would allow you to terminate lower priority streaming video connections or transfer the connection to a different server. In this article, I present a simple framework that can support this behavior and more. It is flexible enough in utility that the framework is analogous to a network Swiss army knife. It can fit into your network to aid in the diagnosing and resolution of network events as they occur.

This framework, which I call "PerfScout," is based on a message-queuing system where messages are dispatched between different blocks of responsibility. Each block operates independently and dispatches messages in a fire and forget manner. What's more, PerfScout is fairly OS agnostic and can be recompiled to a large number of supported operating systems with minimal work (over 32 different flavors including Mac OS X, most UNIX variants, Windows, plus others). The complete source code for PerfScout is available electronically; see "Resource Center," page 5.

Knowing Your Performance Needs

Say that your digital video processing site has certain expectations on the quality of your network connection to specific destinations. And suppose that bandwidth per connection is an important quantity for customers trying to upload videos to your web site for processing. Low bandwidth would tend to indicate that a customer would give up trying to upload data—which translates to a lost opportunity. That is one scenario; other important network performance scenarios could include Voice over IP and latency, streaming video and the available bandwidth, file copying and the packet loss, and router congestion and low-priority data transmission.

In situations such as these, you would want to monitor network performance and act on poorly performing connections. Actions that might be appropriate for poor connections would be to move the client to less loaded servers, limit the number of connections to servers, delay in sending data, notifying network engineers or automated route control systems, or simply record your network's performance. Heck, recording your network's performance may even be a legal requirement (or you may also want to independently verify your IPS's records).

PerfScout allows for extensions in three

areas:

- Sources. How data destinations of interest are to be identified.
- Tools. What network performance is measured.
- Actions. How to respond to network events.

PerfScout is built on a framework that provides for concurrent processing at each stage of the pipeline with these capabilities. The provided implementation contains two types of network performance tests: network bandwidth estimator and a latency tool. Other sources, tools, and actions can easily be plugged in to provide a more complete picture of your network's connections.

Efficient Processing of Results

A key aspect of the PerfScout design is its flexibility. This model can be split into a distributed set of components. PerfScout encapsulates communication between each block of work and allows the engineer to focus on just two details:

- How to efficiently manage work that is requested of me.
- How to perform the work that is being requested in a timely manner.

Figure 1 shows a processing block layout of PerfScout, in which there are three basic blocks in this design. The "glue" between each block is the messaging framework that provides the conduit for the dispatching and reception of messages. This glue can be upgraded for various distributed implementations (such as TCP/UDP communication or interprocess communication), and is highly portable (through the use of the ACE library; see the accompanying text box entitled "The Adaptive Communication Environment Library"). It certainly is possible to mix transport mechanisms between these blocks. The relative order of processing is from left to right.

A fully distributed PerfScout would be pretty neat. Potential uses could be:

- Coordination of multiple streaming servers at various locations.
- Notifications of users accessing streams.
- Shutdown of streams if bandwidth limitations are reached.

Figure 2 shows a distributed PerfScout ensconced in a hypothetical network where the web server dispatches destinations that are of interest (in this case newly connected clients) to be measured. Finally, these results are recorded in a database, and an action request (such as dropping low-priority clients when bandwidth drops) sent back

to the web server.

- PerfScout pipelines messages through
- each stage of processing (ownership of a
- message is the responsibility of the input
- queue's owner). Each stage must manage
- its input queue of work intelligently and
- in a manner that avoids queue overflow.
- Relying on this messaging interface clean-
- ly encapsulates the specific processing de-
- tails for each component, and concretely
- defines the input/output relationship for
- each processing block. Correct manage-
- ment of each block's input queue results
- in a highly scalable framework. It is worth
- noting that this type of design is used in
- several commercial products, such as Mi-
- crosoft's Message Queue Center (MSMQ)
- or IBM's MQSeries. For our needs, these
- canned messaging systems are a bit of an
- overkill. For the task at hand, what is
- needed is a thin streamlined implementa-
- tion that provides a highly flexible and ef-
- ficient implementation that is portable
- across a wide variety of platforms.

- The job of software designers within
- this framework is to optimize the pro-
- cessing of messages within each process-
- ing block. This work is generally algorithm
- specific and encapsulated within the com-
- ponent, but involves specific workflow
- rules. This design scales well and lets you
- add more processing requirements at any
- step, provided that queue sizes are man-
- aged in a timely manner (so that queue
- overflow is avoided). Finally, PerfScout
- can support additional components with-
- out requiring a recompilation of code
- (with some small modifications to the cur-
- rent design—perhaps a dynamically load-
- ing processing block).

PerfScout Design

- Figure 3 shows the object diagram for Perf-
- Scout (the Source, Tool, and Action blocks
- are shown here in relation to Figure 1).
- The Tool and Action blocks require a man-
- ager object to effectively handle incoming
- messages. These messages are subse-
- quently dispatched internally for process-
- ing (the Source processing block has no
- input queue and, therefore, no manager—
- sources operate independently with no co-
- ordination). Operations within the Tool
- and Action blocks are coordinated through
- their respective Managers. These blocks
- contain processing clients that perform the
- real work within PerfScout. These clients
- derive from their respective base classes
- (*ToolBase* and *ActionBase*). These client
- base classes provide a common interface
- for initialization and communication.

- Each entity that communicates using a
- message queue within PerfScout derives
- from the *Task* base class (Listing One).
- *Task*, in turn, derives from the ACE base
- class *Task_Base*. The *Task* class is a sim-
- ple wrapper around the ACE library that

- lets PerfScout support a message notifi-
- cation system. *Task* provides two basic ser-
- vices: a worker thread and a message
- queue. *Task::put()*, *Task::get()*, and *Task*
- *::getAll()* are used to access the message
- queue, while the *run()* method services
- the worker thread. *put()*, *get()*, and
- *getAll()* are mutexed to prevent concu-
- rent access, and *get()* and *getAll()* block
- when the message queue is empty. Exit-
- ing *Task::run()* terminates execution of
- the thread, which should not occur unless
- a *Task*-derived object is to be restarted or
- shutdown. *Task* is templated on the mes-
- sage type, which, for PerfScout is always
- the *Test* object. Derived classes are required
- to implement their own *run()* method.
- Pretty straightforward stuff. Any modifica-
- tions specific to the message queue be-
- havior, or transport of messages, needs
- only to be done within the *Task* base class.

- The common communication token dis-
- patched between objects that derive from
- the *Task* base class is the *Test* object (List-
- ing Two). The *Test* object contains data
- used to identify target destination, re-
- quested tests, current active test, time of
- test request and collection, and finally the
- test results. The current active test, *m_cur-*
- *Test*, is required so that test types can be
- performed sequentially on a *Test* object
- within the Tool Manager. The accessor
- *getActiveTest()* returns the value of *m_*
- *curTest*. Encapsulating the current test
- means that the Tool Manager doesn't need
- to track progress of individual *Tests*. When
- all tests are complete, the *Test* object will
- return *kEnd* via *Test::getActiveTest()* sig-
- naling the completion of processing for a
- specific *Test* object. The *Test* object wraps
- an additional object—*Result*. The *Result*
- object maps tests to their results. The *Re-*
- *sult* object is contained as a map collec-
- tion within the *Test* object.

- There are many ways to handle input
- queue processing. The initial implemen-
- tation of PerfScout uses a rather straight-
- forward (or simple) approach where the
- maximum size of the queue is defined at
- instantiation—any puts into queues that
- exceed this limit are silently dropped.
- Therefore, all objects that derive from *Task*
- need to manage their queues in a quick
- and efficient manner, ensuring that mes-
- sages do not back up. The Tool Manager
- and Action Manager dispatch messages to
- their clients quickly; therefore, the real
- work at the client level is to efficiently
- process these tests within the queue of
- each client.

- There are two source implementations
- included within PerfScout: *TestSource*
- reads a parsed XML file at five minute in-
- tervals and contains the location, and types
- of tests to be performed, and *External-*
- *Source* contains a TCP connection that lets
- requests be remotely dispatched to this

4.1 component via an XML request. Other
 - sources can obviously be built and in-
 - serted at this level, such as integration with
 - a web server, router, route- control device,
 4.5 and so on.

- The second processing block, known
 - as the “tool block,” contains the *Tool-*
 - *Manager* (Listing Three) that is derived
 - from the *Task* base class. Incoming *Test*
 4.10 requests are received by the *ToolManag-*
 - *er* and dispatched to the clients contained
 - within the *Tool* block. These clients de-
 - rive from *ToolBase*, which provides a ba-
 - sic framework to perform specific network
 4.15 tests. The *run()* method in *ToolManager*
 - receives messages and dispatches these
 - requests to the corresponding tool. There
 - is a completion thread as part of the *Tool-*
 - *Manager* encapsulated within *ToolMan-*
 4.20 *ager* (called the *CompleteTest* class). The
 - *CompleteTest::run()* method receives com-
 - pleted tests and either circulates the *Test*
 - to the next test or pushes out the finished
 - tests to the next processing block (the Ac-
 4.25 tion Block). Again, the determination of
 - when a *Test* has finished processing with-
 - in this block is determined by the *Test* ob-
 - ject itself. For tool clients, it might make
 - sense to allow for processing of multiple
 4.30 tests to occur simultaneously across mul-
 - tiple worker threads.

- Now to the final processing block, the
 - Action block, which contains the *Action-*
 - *Manager*. Test results are interpreted here
 4.35 and actions can be performed based on
 - these results. The *ActionManager* (Listing
 - Three) receives a single *Test* message con-
 - taining test results for a specific destina-
 - tion. In this block, there again can be mul-
 4.40 tiple clients as in the Tool block. Each
 - client within the ActionManager block re-
 - ceives a copy of the *Test* object as in
 - method *ActionManager::run()*. In Perf-
 - Scout, there are two client implementa-
 4.45 tions: One is a simple *Test* result to HTML
 - conversion, and the other is an interface
 - to writing the data to a SQL database. All
 - kinds of useful action modules are possi-
 - ble at this level, such as a module that
 4.50 shutdowns a stream if the bandwidth
 - reaches a certain lower limit, or an e-mail
 - notification system on certain network
 - events, or a recording/reporting module
 - that lets you capture network health over
 4.55 a long period of time, and so on.

- **Bandwidth Detection and Latency**

- Two tool clients are provided within Perf-
 - Scout—a bandwidth detection tool and
 4.60 a latency tool. Latency is a fairly straight-
 - forward quantity. Latency measures the
 - roundtrip delay in the transport of the
 - packet. There are several ways to capture
 - latency. This tool uses UDP probes to
 - compute the latency. This approach means
 - that latency can only be captured from
 4.67 hosts that respond to ping (other tech-

4.68 niques such as http SYN/ACK or TTL lim-
 - ited probes can avoid this limitation, but
 - have other limitations in their use). La-
 - tency captures the round-trip packet trav-
 - el time. Latency is important to applica-
 - tions that are sensitive to delays such as
 - voice over IP.

4.75 It is also worth noting that passively col-
 - lected data such as an http SYN/ACK la-
 - tency retrieved from a web page with a 1
 - pixel gif would mean that the Source pro-
 - cessing block would effectively collect the
 4.80 data, which would then push the *Test* data
 - directly to the Action processing block,
 - bypassing the Tools block.

- Available bandwidth is not a direct
 - quantity that can be measured but must
 - be inferred from at least two different la-
 4.85 tency measurements. The algorithm used
 - in this implementation is a rather simple
 - algorithm used to compute bandwidth—
 - remember, it is a simple matter to plug in
 4.90 different versions as they are developed.

- Bandwidth is inferred from latency val-
 - ues received for packets of differing
 - sizes—the theory is that a larger packet
 - will encounter more latency due to longer
 4.95 send/receive queues at the Layer 2 level
 - (Ethernet)—where the original data pack-
 - et is broken down into smaller chunks for
 - transmission over the wire. If you were to
 - draw a linear relationship between two
 4.100 measurements of differing packet sizes,
 - then this equation can be used: *Band-*
 - *width=(lat2-lat1)/(bytes2-bytes1)* (*bytes-*
 - *ms*), where *lat2* and *lat1* are the results of
 - two latency measurements, and *bytes2* and
 4.105 *bytes1* are the results of respective byte
 - counts of packets used in the latency mea-
 - surements. For the PerfScout implementa-
 - tion, the first packet is 56 bytes and the
 - second packet is 4096 bytes in size. The
 4.110 assumption of a linear relationship between
 - latency and bandwidth is not always true
 - for various reasons such as router input
 - queues, congestion, and so on. Therefore,
 - multiple tests help identify and remove
 4.115 outliers in computing this value.

- **Conclusion**

- PerfScout can be customized and ex-
 - panded in many directions, as suits the
 4.120 needs of your network. New tools,
 - sources, and actions can be added and
 - processing blocks can be redistributed
 - across your network. A flexible monitor-
 - ing tool is a must have for performance
 4.125 critical networks. PerfScout can fulfill this
 - role by providing a customizable frame-
 - work that eases integration into a network
 - environment. Once integrated PerfScout
 - can monitor, record, and act on events
 4.130 within your network.

DDJ

(Listings begin on page xx.)