MySQL Performance Analysis with Percona Toolkit and TCP/IP Network Traffic

A Percona White Paper

By Baron Schwartz

February 2012

Abstract

The TCP network traffic conversation between a client and a MySQL server is a rich source of data for many types of analysis. In this paper, we show how the simplest type of traffic—packet headers with no payload—can provide a surprising variety of information about the system's performance. The techniques that we demonstrate are quick, require only readily available open source software, and permit "black box" analysis of a system without any a priori knowledge.

TCP traffic is an ideal place to begin when you want to learn how well a networked system is performing, without a large investment of time or money. A quick investigation of the system's performance via TCP analysis can help you prioritize. Just as importantly, it might help you avoid a lengthy and expensive analysis effort that is destined to produce no results.

It is not necessary to capture and analyze the full contents of the network traffic between the client and the server. Instead, it is enough to capture only the packet headers in the TCP/IP protocol. This data is compact (only 384 bytes per packet), is usually not regarded as privileged even in very secure environments, and is nonintrusive to capture. In addition, you can collect the data from a variety of places in the network. For example, you can capture it from the server, from a client machine, or from a network device in the middle. There is also no requirement to analyze the data on the machine from which you gather it. You can move it elsewhere and process it at your leisure.

Fundamental Performance Metrics

TCP/IP packet headers are sufficient for in-depth analysis because they contain the fundamental metrics of performance from which many other metrics can be derived. These metrics are the timestamp of the query's arrival at the server, and the timestamp of completion. In addition, it is necessary to be able to correlate arrivals and completions, and recognize them as belonging to the same query.

The TCP/IP packet headers contain the source and destination IP addresses and TCP ports. These are sufficient to identify packets that belong to a single query. The program you use to capture the TCP traffic, on the other hand, measures the arrival and completion timestamps. In this white paper, we will use the readily available open source *tcpdump* program.

The analysis techniques that we will demonstrate assume that certain properties hold true of the TCP/IP traffic. Chief among them is that the application-layer protocol is half duplex. Protocols that have pipelining or full duplex behavior are not as easy to analyze, and are beyond the scope of this paper. However, many protocols are half duplex, including MySQL's.

Derived Performance Metrics

As mentioned previously, it is possible to derive a great deal of additional metrics about the traffic, and therefore about the underlying system, given that we know how to identify packets that belong to a single query, and the query's arrival timestamp and completion timestamp.

For the purposes of this paper, we assume that a query's arrival and completion are synonymous with the beginning and end of query execution, respectively. In reality, these are not exactly the same, due to factors such as delays between the packet arriving and being de-

livered to the application. However, such delays are typically very small, and do not appear to affect the validity of the results.

The first derived metric is quite obvious: by subtracting the arrival timestamp from the completion, we obtain the query's response time. Given that information, we can choose arbitrary intervals of time, and over each interval, we can compute metrics such as the following:

- Throughput, or queries per second
- The sum of execution times observed within the interval
- Busy time, or the total duration during which at least one query was executing
- The server's utilization
- Averages, such as the average concurrency and response time during the interval

Of course, it is also quite simple to compute many types of standard statistical metrics over time intervals. For example, we can compute the 95th percentile and standard deviations of response times.

Capturing the TCP/IP Traffic

Let us see how to capture traffic from a system and save it for later analysis. The analysis we will demonstrate in this white paper uses standard UNIX utilities and shell scripts, so we will work with the data in a textual format, one line per packet.

To capture the data, we will use the *tcpdump* program. We execute the program as follows:

```
tcpdump -s 384 -i any -c 250000 port 3306 -w out.tcp
```

This command captures the first 384 bytes—the headers only—from each packet that travels through any network interface in the machine from which the command is run. It instructs *tcpdump* to filter on the TCP port, capturing only packets whose source or destination port is 3306, which is the standard MySQL TCP port. It writes the data to a file called `out.tcp`, and quits after capturing 250,000 packets.

When collecting TCP data, it is wise to ensure that no packets are dropped. Dropped packets can make it difficult to reconstruct the conversation accurately using text-based tools such as those demonstrated in this paper. When it exits, *tcpdump* prints out a message indicating whether any packets were dropped. You may need to change the `-s` option or set the `-B` option to help avoid dropped packets, and operating system-specific TCP configuration is also possible if necessary.

After collecting the data, we will move it to another machine for analysis. The next step is to run *tcpdump* against the saved file and convert it from binary format to text. At the same time, we will apply another filter to discard packets that represent low-level operations

such as acknowledgements or connection handshakes, and save the output into a file called `tcp-file.txt`:

```
tcpdump -r out.tcp -nnq -tttt \
  'tcp port 3306 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)' \
  > tcp-file.txt
```

The resulting data will be similar to the following sample:

```
2012-02-10 10:30:57.818202 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.818440 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
2012-02-10 10:30:57.819916 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 246
2012-02-10 10:30:57.820229 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 2896
2012-02-10 10:30:57.820239 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 1168
2012-02-10 10:30:57.822832 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.823071 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
```

This is the basic input that is required for the analysis we will demonstrate in this paper. Each line represents a single packet, with its timestamp and the origin and destination IP address and TCP port. The final field in each line is the size of the complete packet, but we did not capture those bytes, as previously stated.

The first line represents a packet traveling from an application server, with the origin port 56520. Its destination is the MySQL server, which is listening on port 3306. The second line is a response from the server back to the same client. That represents the lifetime of a single query to MySQL. The query response time is 238 microseconds.

Transforming Packets Into Query Performance Metrics

Now that we have captured the TCP packet data and represented it in plain text, we can use a set of open-source tools to transform it into a representation of queries instead of packets. We will use *pt-tcp-model*, which is part of Percona Toolkit, a set of productivity power tools for MySQL users. It is freely available from http://www.percona.com/software/. We execute the tool as follows:

```
pt-tcp-model tcp-file.txt > requests.txt
```

The resulting file consists of one line per query, instead of one line per packet. The first three lines follow:

```
     7 1328887857.818202 1328887857.818440  0.000238 10.124.62.89:56520
    10 1328887857.819916 1328887857.820229  0.000313 10.124.62.89:56520
    14 1328887857.822832 1328887857.823071  0.000239 10.124.62.89:56520
```

Readers who wish to reproduce this paper's results are invited to download the full file from www.percona.com/files/tcp-analysis-sample-data.zip.

The fields in each line are as follows: a sequence number that identifies the query's arrival order at the server, the arrival timestamp, the completion timestamp, the response time, and the host and port from which the query originated. The first line in this sample corresponds to the first two lines in the previous sample. The queries are printed in completion-time order.

By default, the tool counts a request's duration from the last inbound packet until the first outbound packet. You can configure this, however, and you may wish to do so in case you want to see the effects of large responses, such as a query that returns many rows of data.

It is already possible to perform a variety of powerful analysis techniques on the data in this format, but some of the techniques we will demonstrate require queries to be aggregated together over time intervals. The *pt-tcp-model* tool supports another mode of operation that performs this aggregation, but it requires that they first be sorted in arrival-time order. We will sort the data, then aggregate it into intervals of 200ms:

```
sort -n -k1,1 requests.txt > sorted.txt
pt-tcp-model --type=requests --run-time=.2 sorted.txt > sliced.txt
```

We will explain how we chose 200ms later in this paper.

Black-Box Performance Analysis

We are finally ready to begin analyzing this data and discover what it can tell us about the server. Before we begin, we describe the data we will use. The sample was captured from a production MySQL database server running a popular e-commerce website. The server is very powerful, and only lightly loaded. The query workload that executes on the server is generally simple, but there are occasional complex queries and batch jobs. The sample is about 110 seconds long. Although we need only the TCP/IP packet headers for this white paper, in fact we captured the first 4096 bytes of each packet, so that we could extract the queries from them when we found something interesting and wanted to investigate more deeply.

It is a good idea to become familiar with a purpose-built plotting program, and to be able to plot files of data quickly. Spreadsheets are often a poor tool for this task, because they require importing the data into their own format, changing a graph in a spreadsheet is tedious, and they typically do not perform well on very large data sets. We will demonstrate an open-source UNIX plotting tool called *gnuplot*. Another good choice is the *R* statistics package.

The first analysis technique we will demonstrate is simultaneously one of the simplest and most useful: a plot of query response times, as shown in Figure 1:

```
gnuplot> plot "requests.txt" using 3:4
```


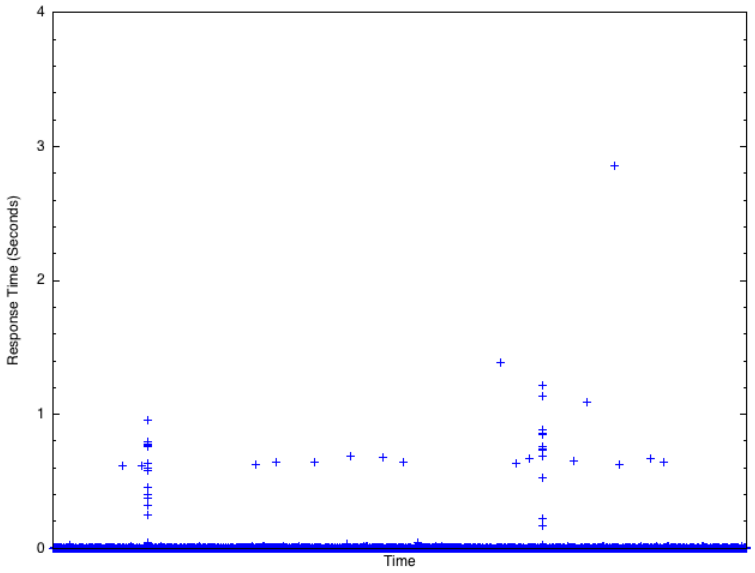

Figure 1: A time-series plot of query response times

Figure 1 shows most queries executing very quickly, packed so densely at the bottom of the plot that they create a solid blue line. However, several anomalies are apparent. The first is the presence of two vertical lines of long-running queries that are clustered close together. The second is a horizontal line of queries that appear to execute much more slowly than normal, but seem fairly consistent each time. Finally, some queries appear to be outliers, not fitting into any pattern.

Gaining this insight about the server's performance might require a large amount of work by other means. This is why plotting response times is a valuable tool for detecting performance anomalies that may merit further investigation.

How can we explain the anomalies? Results will vary, but in the case presented here, the vertical lines represent momentary lock dependencies among queries. Because the queries are plotted in completion-time order, queries that are blocked and then complete soon after being unblocked will be plotted in a relatively tight cluster, one after another, and the first query in the cluster probably can be blamed for blocking the others. Completion-time analysis is a powerful method for detecting lock contention.

In this case, the blocking between the queries is relatively short-lived, so a plot in arrival-time order will also show the vertical spikes of queries, with a more apparent lean to the left. It is also interesting to look at the vertical separation amongst the queries in each spike. That probably represents the query's service time after it acquires the lock, so these queries might be slow even without the blocking.

The group of queries clearly experiences much longer total response time than they would without the lock, so this problem is well worth some effort and expense to solve. A detailed investigation into the contents of the TCP packets revealed that the vertical lines in this example were caused by `SELECT FOR UPDATE` queries.

The horizontal line, on the other hand, is caused by a query that runs more slowly than is typical, but this problem is not as serious, because it does not cause several queries to wait. Still, every slow query is worthwhile to examine, preferably with a profiling tool such as Percona Toolkit's *pt-query-digest*, which can help prioritize queries by their contribution to total response time.

Generalized Stall Detection

In the previous section we demonstrated the power of completion-time analysis for revealing dependencies between queries. A natural follow-on question is whether it is fruitful to compare arrival times and completion times. Perhaps this can cause some types of patterns to emerge more clearly than would be visible in a simple response-time plot?

It turns out that this is indeed the case. The easiest way to demonstrate this is to count the number of arrivals and completions in each time interval, and plot the difference between them. Using the sample of data that we aggregated into 200-ms time slices, we can create Figure 2 by subtracting the fourth column from the fifth:

```
gnuplot> plot "sliced.txt" using ($5-$4) with lines
```

The characteristic pattern on the resulting plot is caused when queries arrive in one interval of time, but do not complete until the next interval, thereby causing a dip in the first interval followed by a spike in the second. Depending on the distribution of response times and the aggregation interval selected, spikes will almost certainly appear at some point. The question is how closely one wishes to look, and whether the spikes represent a problem that needs to be solved.

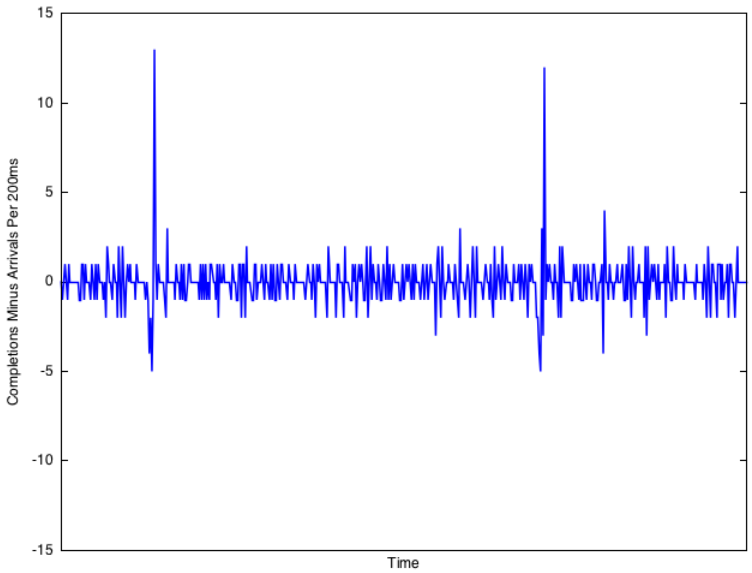

Figure 2: Completions versus arrivals in 200ms intervals

In Figure 2, the spikes indicate the same stalls that appeared on the response-time plot. However, this is not always the case. In many instances, a group of queries is blocked while waiting for a resource, but then can continue to completion as a group, rather than holding and releasing the resource one by one. Such a group of stalled queries might not be visible on a response-time plot, but an arrivals-versus-completions plot can show it very clearly.

Choosing an Aggregation Interval

Figure 2 showed the data aggregated into 200-millisecond intervals, or two-tenths of a second. Why did we choose this interval, and how can you select a good value in general?

There are two approaches to selecting the interval, depending on the question you would like to answer with the stall analysis:

- Are you interested in whether stalls are long-lived enough to cause an impact on the application? If so, then choose an aggregation interval that is approximately similar to, or slightly less than, your maximum acceptable stall length.
- Are you interested in how severe stalls become, in the common case? Then choose an interval that is small enough to capture short-lived stalls, based on the typical response time of the queries.

The reasoning behind these two approaches is as follows. When considering a stall's duration, it is necessary to understand that the difference between arrivals and completions is visible only when they occur in different time intervals. If the interval is long enough, a short-lived stall can occur entirely within one interval, and remain undetected. There is a possibility that short-lived stalls can be detected with a large interval, if the stall happens to occur across the interval boundary, but this becomes less likely as the interval grows larger. If the stall is larger than the interval, on the other hand, then it is certain to be detected.

However, if the interval is too small relative to the stall, then the difference between the number of arrivals and completions is likely to be spread across many intervals, instead of a single interval where it can grow large enough to be noticeable.

The number, duration, and maximum request concurrency of stalls has a statistical distribution, just as the response times do, and these are related. Short-lived stalls are likely to occur more often than longer ones, and high-concurrency stalls will tend to be shorter as well. In fact, computer systems are networks of queues and servers at all levels, and the nature of such a system is that the queues are constantly filling and emptying. Thus, if you look carefully enough, you will surely find queueing (stalls), although it might not be severe enough to matter.

As a result, it makes sense to choose an aggregation interval that ranges between some multiple of the typical response time of the queries and the maximum acceptable stall time. In practice, if the aggregation interval is similar to the typical response time, then many

queries will arrive in one interval and complete in another, and the plot of the differences will look similar to Figure 3:

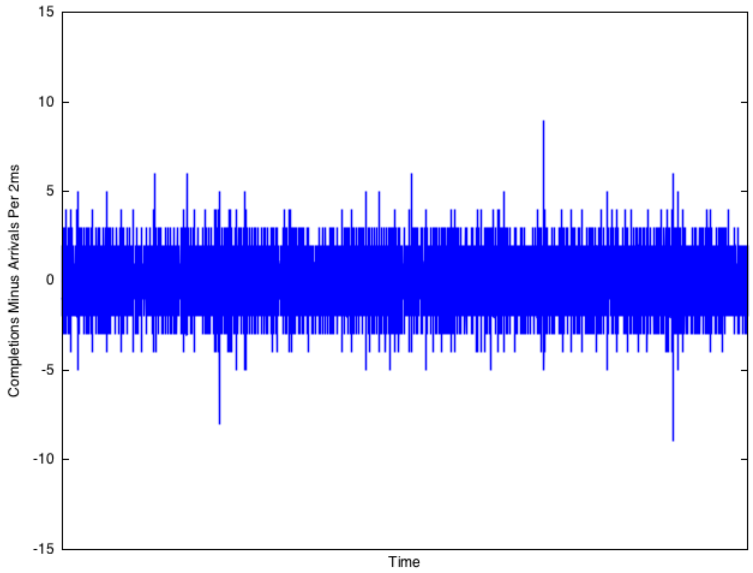

Figure 3: Completions versus arrivals in 2ms intervals

Figure 3 demonstrates that a short aggregation interval results in a chart that has many small spikes, with the effect that it simply looks like a thick line around the y-axis. It is best to choose an aggregation interval that is some multiple larger than the typical response time to avoid this problem.

This leads to the following approach to finding stalls: the aggregation interval should be more than approximately ten times the typical response time, but no more than the maximum acceptable stall time.

You can find the typical response time by aggregating the entire sample and looking at the 99th percentile response time. Do this by aggregating the sample with a large `--run-time` interval, which will result in a single line of output. The sample input has a typical response time of just over 2 milliseconds; that is, 99% of the queries are faster than 2 milliseconds. If we want to detect typical stalls as defined by this metric, then, it might be best to aggregate the queries into intervals at least 20 milliseconds long. We chose 200 milliseconds for this paper because it was an appropriate upper limit on stall length for the application in question.

Analyzing Response-Time Variability

Another very useful black-box analysis technique is to find areas where response time is inconsistent or highly variable. Good performance requires consistency: a database server that is usually speedy but occasionally becomes erratic cannot be loaded to its peak capacity, but must be limited to the lowest load at which its behavior is acceptable. Inconsistent performance also indicates an opportunity for optimization, because queries that have

highly variable response times might be possible to optimize so that they always choose a faster execution plan, instead of choosing it only sometimes.

Many statistical measures of a distribution's variability are available, but some of them are difficult to analyze because they retain too many of the characteristics of the source data. For example, the standard deviation has the same units and scale as the source data, so it is difficult to say whether a given standard deviation is good or bad without knowing more about the data. Thus, it is best to choose a metric that is normalized relative to the distribution's mean, so that it can be compared easily from server to server. Perhaps the most straightforward is the index of dispersion, also called the variance-to-mean ratio. The aggregated data we already generated contains this metric in the ninth field of each line, so it is straightforward to plot it:

```
gnuplot> plot "sliced.txt" using 9 with lines
```

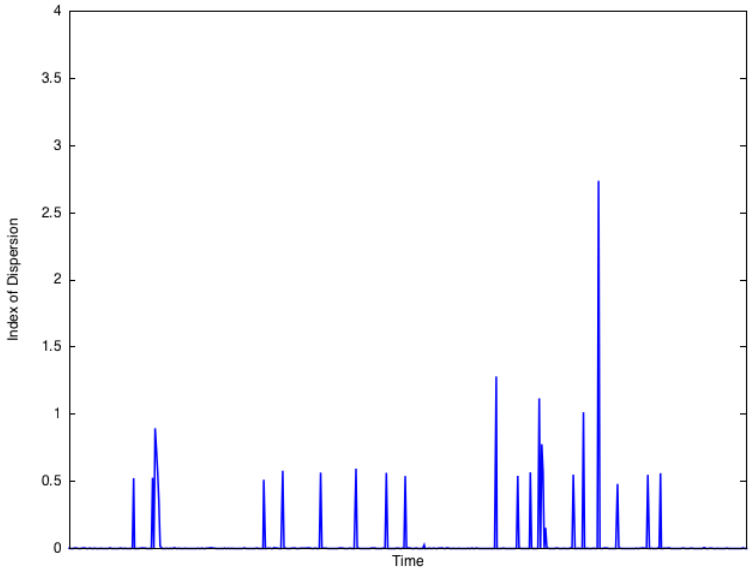

Figure 4: Index of dispersion, or variance-to-mean ratio

The spikes on Figure 4 represent areas where the server's performance is inconsistent or highly variable, and which might be fruitful to investigate more deeply. A comparison shows that most of the spikes align with the same artifacts that were visible in Figure 1, but again this is not always the case, especially when the response-time plot does not show obvious problems.

Percona Toolkit's *pt-query-digest* tool can help analyze individual queries' variability. It prints the variance-to-mean ratio by default in each query's detailed report, as well as in the overall profile.

Conclusion

Black-box performance analysis with TCP/IP packet headers requires only a few moments of work. It uses a small sample of data that is easy and nonintrusive to collect, and does not include sensitive information. It can reveal problems when other analysis techniques might obscure them or require a large investment of time or resources.

In this paper we demonstrated how to collect and transform data with open source tools such as *tcpdump* and Percona Toolkit's *pt-tcp-model*. We showed a single sample of data from three points of view: a response-time plot, the difference between arrival and completion counts, and intervals with highly variable response times. Many types of problems can appear in these plots, including daisy-chain locking amongst a group of queries, server-wide stalls, and unstable query execution plans.

About Percona

Percona provides commercial support, consulting, training, and engineering services for MySQL. You can contact us through our website, and we invite you to call us. In the United States, you can reach us toll-free at (888) 316-9775. Outside the United States, please dial +1-208-473-2904. You can reach us in Europe at +44-208-133-0309.

