Performance White Paper

GSW RF DTIO Engine Performance Overview

The GSW Directed Terminal (Windows Console) I/O Engine is an add-on component that intercepts a specific set of terminal input/output operating system calls initiated by your Windows 32-bit application and directs terminal I/O through a specialized high performance interface within the GSW Universal Terminal Server (UTS).

The GSW DTIO Engine is specialized software, focused on a narrow set of goals. The primary objectives are to:

1. Provide significant performance improvements with respect to Terminal I/O processing
2. Increase the number of sessions on a server
3. Allows Bell to sound on the client instead of the server as one would expect
4. Allow a framework where specialized features can be incorporated on a custom basis

While the internal design/workings of the RF DTIO are complex, installation and use are not.

This paper will provide an high level overview of the mechanics performed in order to achieve items 1 and 2 listed above. It is useful to understand some of the internal software design related to the problem that is being addressed to see the significance and value of the software as well as gauging expectations.

Increases in performance by RF DTIO Engine are due to optimizations on a per session basis by reducing the number of instructions executed and the number of context switches, and DTIO Super Fine Tuning configuration.

The most significant performance improvement provided by RF DTIO is adding intelligence to the data flow between the application and the SSH2/Telnet Server. There are other areas such as fine tuning the configuration that provide additional improvements but this paper is focused on the data flow between the application and the SSH2/Telnet Server.

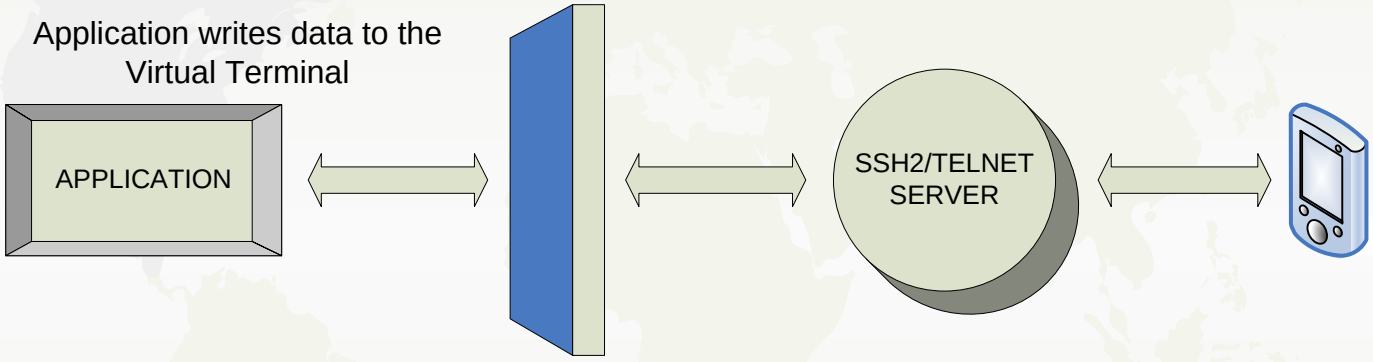
Eliminates unnecessary polling/scanning and reclaims valuable processing time.

Performance White Paper

High Level Overview
Data Flow Between the Application and SSH2/Telnet Server

To understand how RF DTIO improves performance, we will first review the basic data flow from the application to the device when using a SSH2/Telnet server.

When an application sends data (writes) to a device, it is really writing to locations on a Virtual Terminal (Windows Console) that are mapped to the device.



Virtual Terminal (VT)

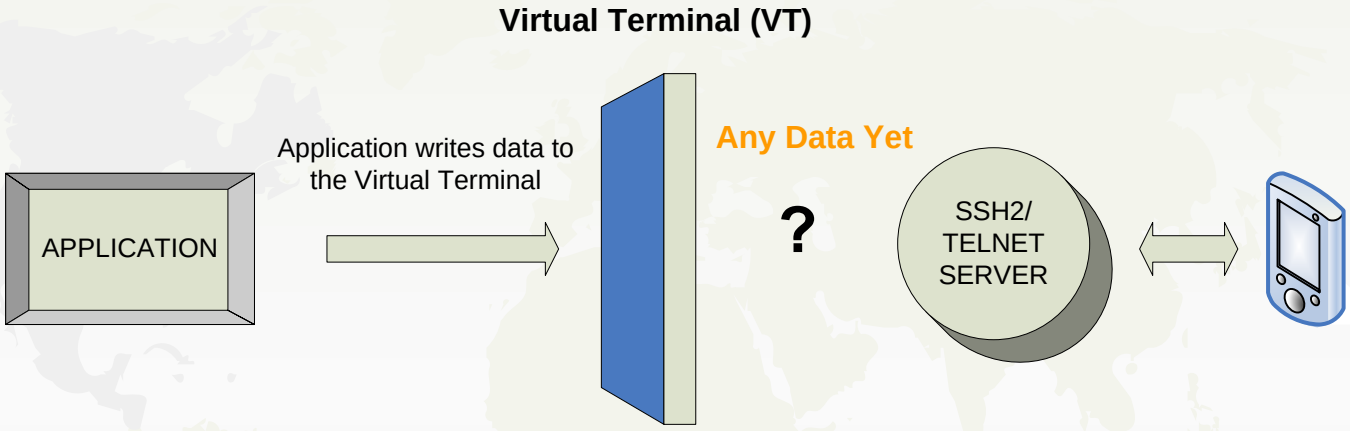
Note: The Virtual Terminal is the Console Window maintained by the SSH2/Telnet command prompt and accessed through Windows Win32 Console API's

The Telnet or SSH2 Server obtains (scans/ reads) the data from the Virtual Terminal and sends it to the Telnet/SSH2 Client for display on the device.

However, it's more involved or complicated than the diagram would suggest….

Understanding Data Flow

The mechanism used by the Application and SSH2/Telnet Server to recognize when the application writes data to the Virtual Terminal may not be intuitive.



- The Application and the SSH2/Telnet Server are unrelated programs. The application simply writes data to the Virtual Terminal.
- SSH2/Telnet does not have the intelligence to know when the application has written data to the Virtual Terminal.

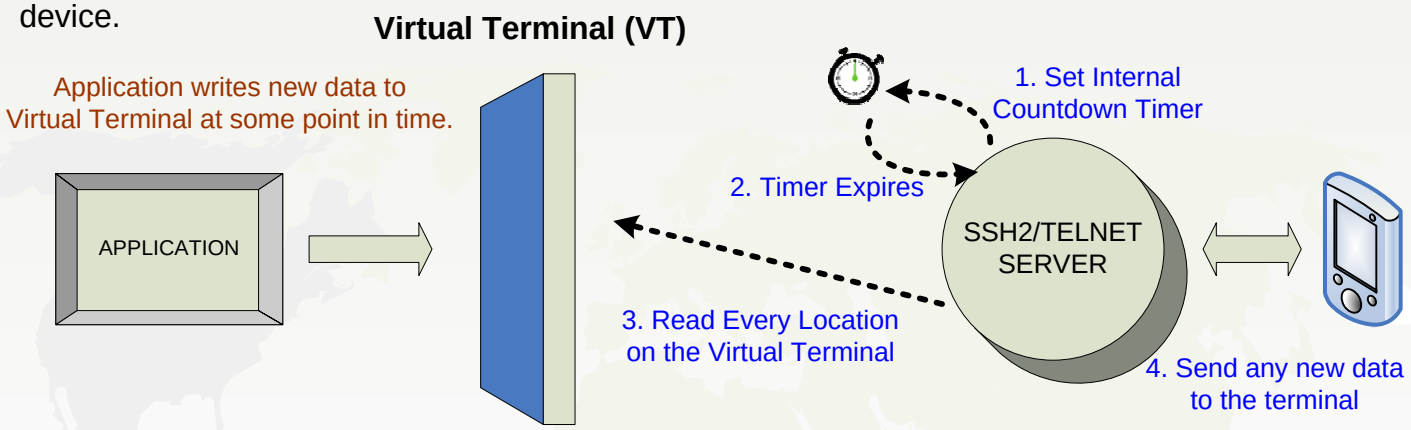
So how does the SSH2/Telnet Server know when there is new data at the Virtual Terminal?

Performance White Paper

Standard Un-Intelligent Mechanism to Determine when New Data is Present at the Virtual Terminal

The SSH2/Telnet server periodically polls each row/column location on the Virtual Terminal for new data from the the application. In brief the steps are described below.

A countdown timer is set (1). When the timer expires (2) SSH2/Telnet reads (3) every location on the VT to determine if new data is present. If new data is present it is sent (4) to the client device.



This type of operation works very well and is fast. However certain environments demand utmost efficiency so that the overall session throughput is maximized to gain an increased number of sessions and/or optimal session response times.

To achieve higher session throughput, the area with greatest opportunity for processor utilization is the mechanism that identifies when data is present at the Virtual Terminal. The current mechanism works well but extra processing occurs in order to provide the fastest response times to the users.

For example, the countdown timer may expire but there may not be any new data on the VT. If there is no data, there is no need to spend the processing power to read every location (row and column) on the VT. This may not consume much processor time with just a few devices but it's a different story when there are 40, 100, 200 or more devices.

Typically a screen is scanned for the presence of new data 10 times a second. Two context switches are required each time the screen is scanned, consuming even more time. If the number of sessions is 20, and each screen is scanned 10 times a second and that amounts 200 scans occurring each second. The math is simple; a system with 100 sessions will have 1,000 screen scans performed every second. And 300 sessions requires 3,000 scans every second. On a 1.47 GHz AMD Athlon system running Microsoft Windows XP, this amounts to 117 ms out of every second spent scanning for new data, and that doesn't include context switches! The unfortunate aspect is that in real world situations, most often there is no new data to transmit. If your scanner performs 1 scan every 5 seconds then **98% of your polling time is wasted** when no new data is present at the Virtual Terminal.

This is where the GSW RF DTIO Engine comes in!

Performance White Paper

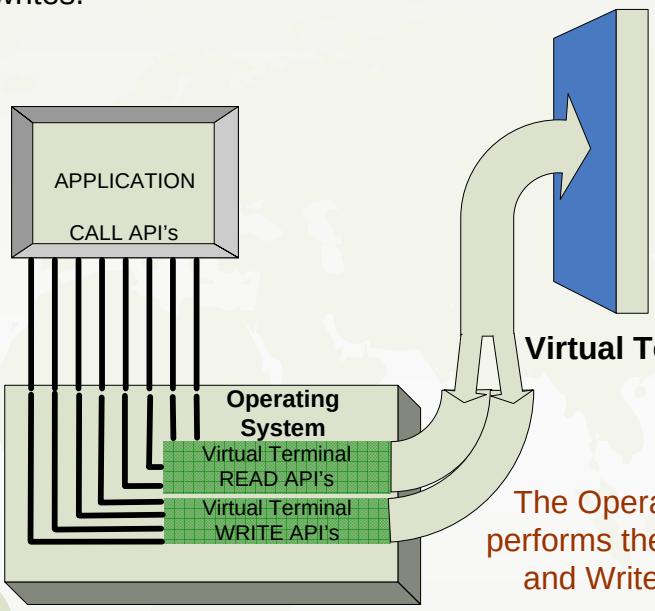
RF DTIO Engine "Hooking" adds Intelligence

The real problem that causes the unnecessary polling is a "lack of intelligence" between the application and the SSH2/Telnet server. Ideally the best situation is for the application to notify the SSH2/Telnet Server when new data is present at the Virtual Terminal. This would eliminate the time spent polling for new data.

Let's dive a little deeper into what happens when the application read/writes to the Virtual Terminal. The application uses Operating System Application Programming Interfaces (API's) to actually perform the reads and writes.

Standard Operation without GSW RF DTIO

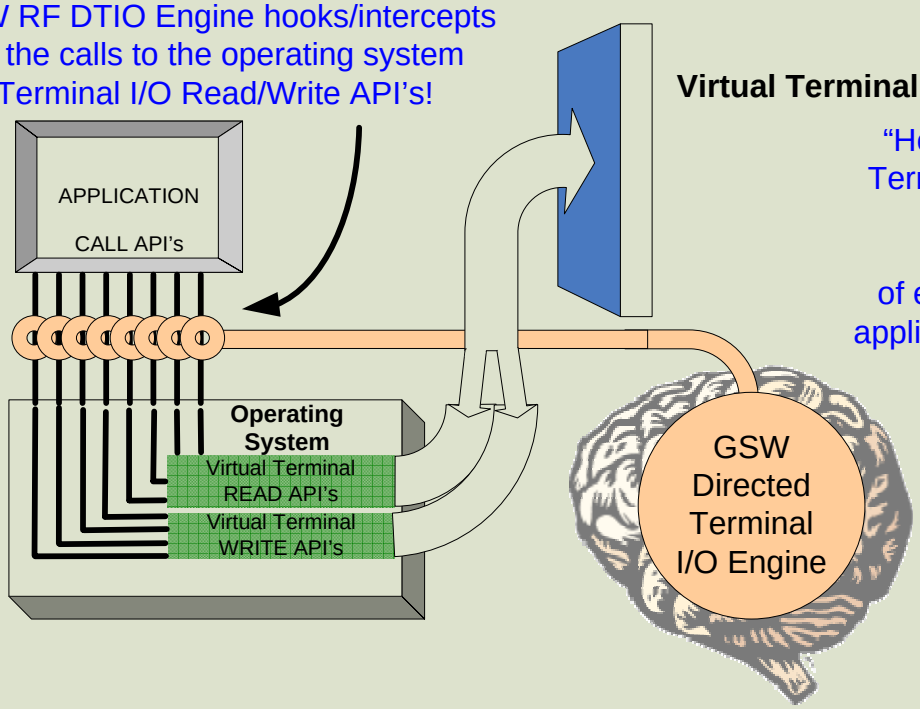
The Application calls Operating System (OS) API's to perform Reads/Writes to the VT. There are many different OS API's that may be called for Terminal I/O. Each application may have its own "favorite" API's to use.



The Operating System performs the actual Reads and Writes to the VT.

with GSW RF DTIO

GSW RF DTIO Engine hooks/intercepts all the calls to the operating system Terminal I/O Read/Write API's!



"Hooking/Intercepting" the Terminal I/O API's provides RF DTIO with the **KNOWLEDGE** of every read/write that the application performs to the VT

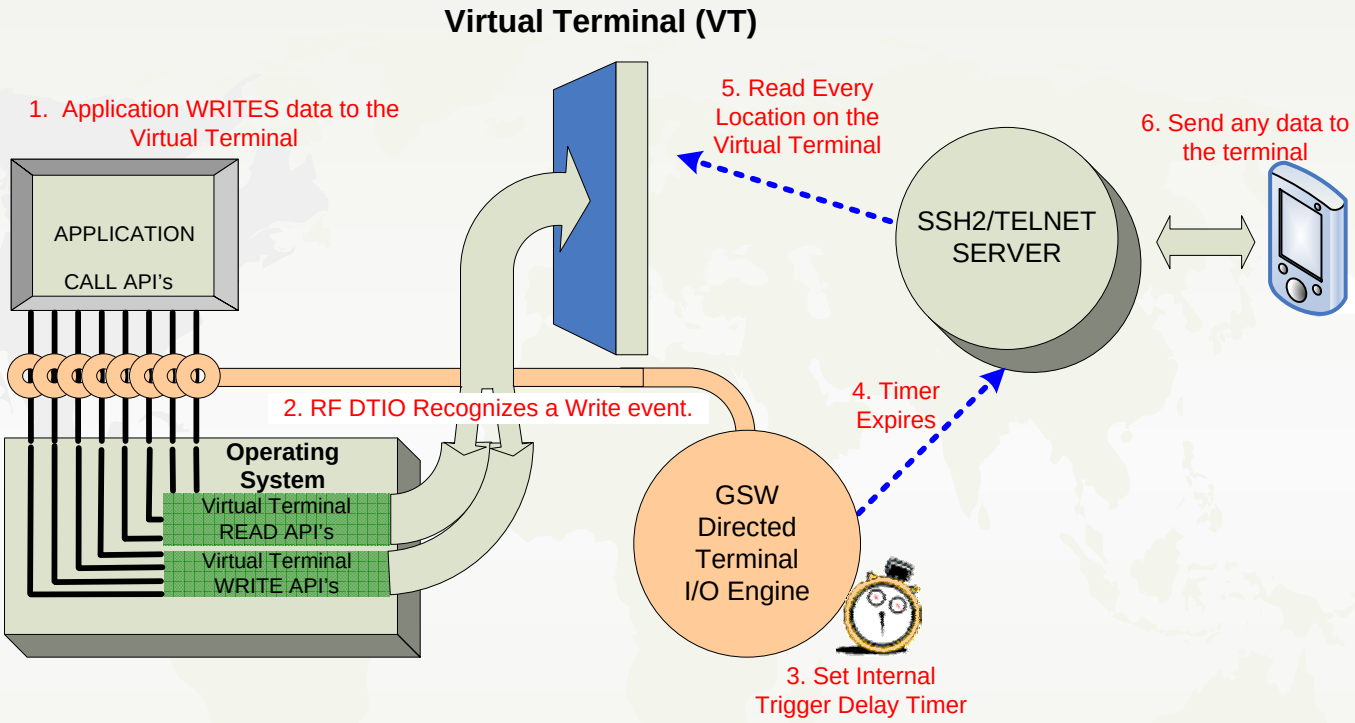
This is just what's needed!

RF DTIO incorporates an intelligent mechanism for recognizing when new data is present

Performance White Paper

RF DTIO Engine Intelligent Write
Recognizes when New Data is Present at the Virtual Terminal

With new intelligence provided by RF DTIO Engine, unnecessary polling/scanning for new data is eliminated. When the application is launched, it is launched by RF DTIO . This allows GSW RF DTIO Engine to "hook" all console I/O API's that the application needs to write/read to the Virtual Terminal.



Since the RF DTIO "hooks" all Console I/O API's it already has the knowledge every time the application reads or writes data to the Virtual Terminal. Two main cases to review are when the application Read and when the application Writes.

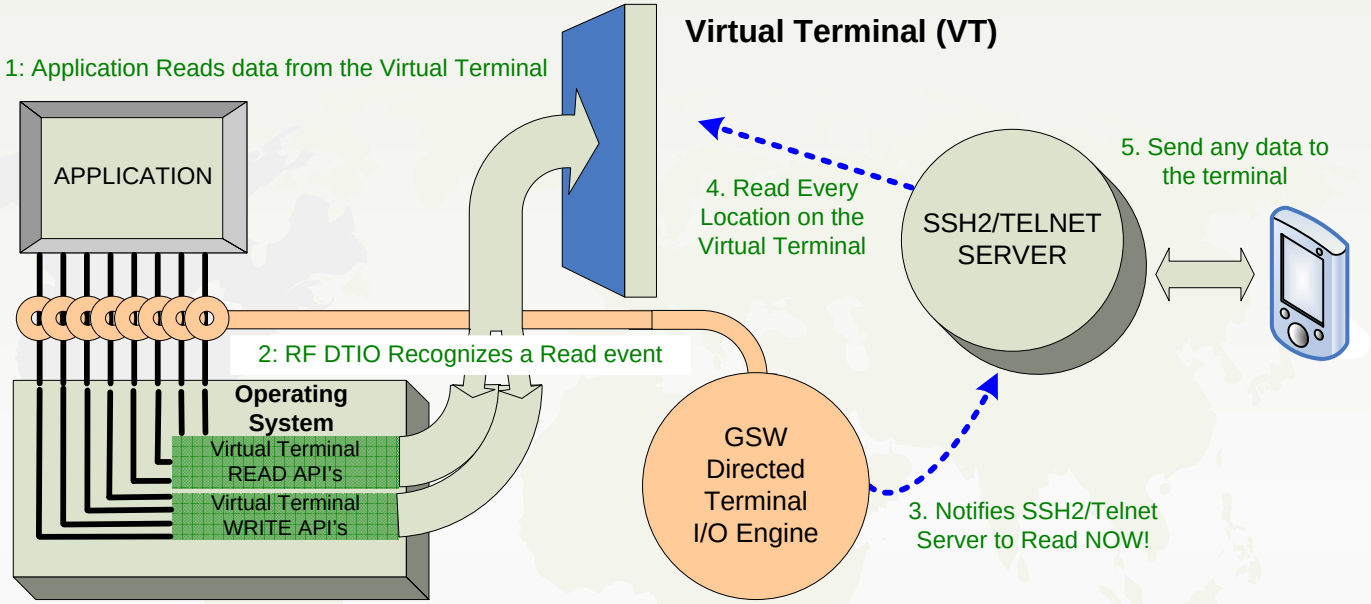
First, let's look at what happens when the application performs a write. When the application writes (1) to the Virtual Terminal, RF DTIO recognizes (2) the event and sets (3) the Trigger Delay Timer. The trigger delay timer is set because in most cases, applications will write several times before the screen is complete. The timer prevents thrashing by reading too soon and having to read again and again. When the Trigger Delay Timer expires (4) the SSH2/ Telnet Server is notified that data is present on the Virtual Terminal. Upon notification the SSH2/Telnet (5) reads (scans) the Virtual Terminal. Next, the data is sent (6) to the terminal.

RF DTIO uses this intelligence to notify the SSH2/Telnet Server that data is present.

The performance improvement in many environments is nothing short of amazing.

RF DTIO Engine Intelligent READ
Recognizes when New Data is Present at the Virtual Terminal

Another area of potential performance gain is when the application performs a Read.



Let's look at what happens when the application performs a read. Usually (but not always) when an application's writes are complete, a read is performed to see if there is a response from the operator. If the application performs a read after completing screen updates then RF DTIO can be **optionally configured** to provide further performance gains.

Knowing that an application always performs reads after screen writes allows further optimization such as described here. When the application Reads (1) from the Virtual Terminal, RF DTIO recognizes (2) this event and *immediately* notifies (3) the SSH2/Telnet Server that data is present on the Virtual Terminal. No timers are set! Upon notification the SSH2/Telnet (4) immediately reads (scans) the Virtual Terminal Next, the data is sent (5) to the terminal.

Note: Some applications have scenarios that do not perform a read after it is done writing. For example, the application may write a message such as "Processing" to the VT while handling intense database lookups (or other time consuming operations) to inform the operator that they have to wait a little longer before the next screen is displayed. Since the application is not waiting on data from the operator a read may not be performed. In this case RF DTIO would not recognize new data until an internal timer fires.

Even further performance improvements are possible in many environments with configuration options!