

The Four Virtues of a Good Mainframe Network Monitor

SDS White Paper, October 2003

The more complex the network, the simpler the monitor has to appear.

In other words, the monitor has to watch over TCP/IP traffic, report on multiple stacks and systems, catch today's problems yesterday, and fully inform service-level agreements, without getting in the network's way.

The monitor interface has to present far more data than any 3270 screen, and still make quick sense to users. It has to be readily available from all kinds of platforms. And it has to serve a technical support staff that, more and more often, grew up with the Internet rather than green screens.

Hence the four requirements for a truly useful network monitor:

1. The monitor reports network status and performance fully, in real-time, without itself hindering performance.
2. The monitor provides a central view of *all* of your network's z/OS hosts, TCP/IP stacks, applications, and connections.
3. The monitor provides simple, intuitive navigation to full details regarding every resource and event on the network.
4. The monitor provides easy-to-use, graphic tools for network diagnosis, management, and repair.

In short, a good mainframe monitor is low-overhead, it provides a central view of nearly everything and intuitive navigation to almost anything, and it simplifies z/OS operations.

Think efficient data-collection architecture: Getting MIB data without SNMP requests. Measuring stack traffic without constant packet tracing.

Think graphic summary reports: A single screen showing you health of multiple TCP/IP stacks on a z/OS system, and of multiple z/OS systems. Intuitive links to individual TCP/IP resources: stacks, interfaces, applications, and connections. Equally intuitive links to other z/OS resources: Enterprise Extender, CICS, VTAM, and CSM buffers,

And think simple graphic tools for network diagnosis: Packet trace results in real-time. ICMP and UDP pings. Operator commands to any z/OS system from one browser screen.

This is a plan for a pretty ideal health and performance monitor for z/OS and TCP/IP. The rest of this paper describes these ideals in more detail.

Monitor Virtue #1: Fast and Light, in Real-Time

For a moment, consider the obvious: Network monitors are meant to help us improve network performance.

Then recognize the corollary: Your network monitor must not hinder network performance.

The monitor must get data frequently enough to provide real-time reports and immediate alerts, yet run efficiently enough that it doesn't interfere with network business.

For z/OS and TCP/IP, that means the monitor had better not depend on SNMP, constant packet tracing, or anything else that consumes more than minimal amounts of bandwidth and CPU.

A direct interface to the TCP/IP stack can retrieve all the same monitoring data available via SNMP, the standard TCP/IP MIBs, and the IBM enterprise MIBs—at a fraction of the overhead. It also reveals to the monitor information that it can't get by analyzing packets.

The Value of Efficient Architecture: A monitor that needs 1% or less of your CPU capacity is a helpful tool. A monitor that consumes 5% of CPU is suspect. The cost in overhead may prohibit real-time monitoring. The hidden cost of ownership may also be prohibitive.

When you monitor TCP/IP, understand that the only *real* real-time data, in the strictest sense of the term, is that created by the TCP stack itself.

A monitor might copy every packet header into its database in real time. But it still analyzes that database according to some time-based sampling routine.

The more often the monitor samples its database, the closer its reports and alerts approach real time. But the more often it samples, the more CPU it requires and the more likely it is to slow system performance.

To put it another way: The less CPU a monitor consumes, the closer it can get to providing real-time reports and alerts.

And then there's the issue of cash money. Barry Graham at Xephon recently compared the cost of ownership for high- and low-overhead software on IBM mainframes.

Graham discovered the following:

"Consider two competing software products: One that uses 1% of CPU capacity. Another that uses 6% of CPU. Over 5 years, the low-overhead alternative will save \$1 million to \$7 million in hardware purchases and software licensing fees."

—The Cost-Savings of Low-Overhead Software, *TCP/IP Manager eNewsletter*, Aug.-Sept. 2003, www.sdsusa.com/tcpipnews/augSept03Overhead.htm.

SNMP, the Usual Data Source for Monitors, is Inefficient. The SNMP (simple network management protocol) daemon that runs alongside a TCP/IP stack can report a rich lode of data about stack traffic, but the performance price is high.

Getting SNMP data requires configuring each stack's daemon, then constantly polling the daemon with multiple get requests. Over and over again, the stack writes its SNMP data, gets a request, and reads the data back out again. Bandwidth demands are considerable; real-time monitoring is out of the question.

Accessing that analysis directly from the stack is more efficient than using SNMP traffic to recreate it.

Tracing Every Packet also Seems an Expensive Way to Recreate Existing Data. Using a nonstop packet trace to analyze every packet as it passes through a stack has at least two significant drawbacks.

One, the CPU time, paging rates, and virtual storage requirements can be considerable. Furthermore, as the amount of TCP/IP traffic increases on the network, so do the monitor's demands on the system—just when you need your monitor most, it draws more resources to itself.

Two, if a user runs a trace for some specific diagnostic purpose, the monitor will likely get confused. If the trace facility gets turned off, the monitor fails.

Constantly tracing and analyzing every packet, like querying the SNMP daemon, is consuming resources to duplicate monitoring data already being written to the stacks' MIBs.

z/OS performance expert Jim Keohane tells a story about a timing problem in a TCP/IP stack on an MVS system in Benton, Arkansas. He couldn't find the cause because tracing the process made the bug disappear. In other words, tracing significantly slowed down the system. (To Troubleshoot Effectively, Use Trace Wisely, *TCP/IP Manager eNewsletter*, Aug.-Sept. 2003, www.sdsusa.com/tcpipnews/june03trace.htm.)

Packet Tracing Finds Only Part of the Data.

There are things about a TCP/IP stack that the packets don't reveal.

A packet trace can show, for example, that a stack is overflowing and has dropped connection requests. But tracing won't show that the stack is *about to* drop requests.

Suppose connection requests arrive at a server port faster than the server can accept them. That's a *connection backlog*, a relatively common problem for TCP applications. TCP deals with it by holding

the requests in a queue, but there's a limit. When that maximum backlog depth is reached, TCP simply drops any new requests.

Packet tracing can alert you when it's too late; requests have already been dropped. A direct interface with the stack can warn you when the queue is approaching its maximum.

For another example of the limits of packet tracing, consider coming enhancements to the z/OS Communications Server IP stack.

IDS (Intrusion Detection Services), TRM (Traffic Regulation Manager), and other tools coming down the line from IBM are increasingly driven by *policies*—rules applied out of sight of the packet-trace interface. For inbound packets, policies apply after the trace; for outbound packets, they apply before the trace.

A monitor with a direct interface to the stack is in a position to adapt and take advantage of these new policy tools.

Monitor Virtue #2: A Central View of All Your Network

We increasingly need to see a complex interconnection of computers as a single machine. And our network monitors need to present us with a single, coherent image of system health and configuration.

Monitors for z/OS TCP/IP stacks have tended to not see the big picture, however.

In some cases, individual monitors are closely associated with individual stacks—the one-stack:one-monitor strategy. Such a monitor is dependent on its stack; it doesn't know the others exist. To see data or get alerts from more than stack, you have to bring up more than one monitor. If a stack goes down, the monitor can't tell you it's down. When you bring the stack back up, you may have to restart the monitor too.

On the other hand, if the monitor runs independently of any one stack, and communicates with all the stacks on the system, it can provide reports that span the entire system.

Take that concept a step further: It's possible for the monitor to run independently of any one z/OS system, communicate with all the z/OS systems on the network, and provide reports that span the entire network.

In an Agent-Server-Client architecture, the Agents reside on z/OS systems, where they monitor TCP/IP stacks. All the Agents on the network push data up to a single Server (or maybe two, the second serving as a redundant backup).

The Server does much of the report processing and responds to Client requests by sending displays down to common desktop web browsers.

That Server may run on one of the z/OS systems. On the other hand, it may run on a Linux or Windows box, independent of any one z/OS image.

In that case, the monitor's allegiance is not to any one z/OS system, but to the entire network of z/OS systems. And, as Scott McNealy of Sun Microsystems put it, "*The network is the computer.*"

Monitor Virtue #3: Easy Navigation to Any Detail

A browser-based GUI is more than a pretty face. It combines breadth, depth, and easy navigation.

People have been managing mainframe networks since long before the GUI interface, the Internet, and Windows were invented. So many of today's mainframe monitors started as 3270 terminal applications. Now they scrape data off of green-screen displays and patch it onto graphic screens.

To take full advantage of its medium, however, a browser interface is best designed from the ground up.

Stacks, interfaces, TCP and SNA applications, routers, buffers—all can be monitored from a single screen. System health, critical statistics, and alerts can all be seen at a glance.

For details regarding individual resources, users can drill *straight* down to applications, interfaces, connections, stacks; rather than travelling up and down through a complex tree structure.

Users are able to diagnose a TCP/IP problem and take corrective action within minutes, because the intuitive interface guides them through the process. Monitoring tasks might pass to less technical users and help-desk staff, freeing up experienced mainframe administrators for more important work.

What else makes a GUI better than a green screen?

- A picture is worth a thousand words.
- A GUI is consistent from PC to PC.
- Help desk workers can easily learn a GUI-based product.
- Remote management: Anywhere in the world, all you need is a PC with a browser.
- Web applications are based on open standards.

Monitor Virtue #4: Browser-Based Tools for Mainframe Diagnosis, Management, & Repair

The packet trace facility that IBM built into its Communications Server stack is an extremely powerful *last resort* for problem solving. And for good reason. It's extremely difficult to use:

- Starting the service and defining the filters requires a cryptic set of console commands and replies.
- Archiving trace data requires a separate external writer.
- You can't see trace results until you stop the trace (another set of console commands) and disconnect the writer.
- To format and read the trace data, initiate an IPCS session.
- If the trace didn't capture what you need, start over.

That process is difficult; it's results are not real-time; and it can serve only one user at a time.

Mainframe monitors have tended to address the issue incompletely. They may provide a simpler interface for the operator command and reply sequences, but no real-time access to trace results. Or they may provide real-time results, but only capture packet headers and only provide them on-line, never in archives.

A really complete packet trace tool would provide the following:

- A one-step process for defining and starting traces, in an interface that allows multiple traces by multiple users, and allows users to come back later and re-use a previous trace definition.
- Filtering tools that make it easy to get headers and message data, or just headers; to select packets for specific interfaces, addresses, ports, and/or protocols.

- The power to view packets almost immediately, as they're passing through the stack. The power to view the packets in EBCDIC and/or ASCII. And the power to save traced packets to datasets, probably in standard IBM component trace format.

With a tool like that, packet tracing will become a common and relied-on tool for network administrators, help-desk technicians, and even application developers.

With a *Ping*, a TCP/IP stack asks whether another stack exists and how far away it is. (Exactly what a sonar ping does among submarines.) StackA pings StackB; StackB "echoes" the ping back to StackA. The result tells StackA that StackB exists, and how long it takes to get a message there and back.

The standard ping uses the ICMP protocol—the language that stacks use to communicate among themselves. The ICMP ping has two weaknesses:

1. Because sending a flood of pings is a simple way to attack a stack, administrators often configure firewalls to block pings. They do not return the expected echoes. The pings result in false negatives.
2. Because it uses ICMP, ping messages generally travel different routes than standard TCP or UDP

data messages. Thus the round-trip time for an ICMP ping is not an accurate measure of round-trip time for application data.

Ideally, a ping tool will provide an alternative form of ping, by the UDP protocol. Such a ping is less likely to be ignored by its target, is more likely to travel the same route as application data, and it can be made to send back a "hop count," a tally of how many routers intervene between StackA and StackB.

And to be really useful, a ping tool will provide a facility for determining how big a packet can travel between StackA to StackB. That is, it will allow users to measure the network's MTU (maximum transmission unit) by sending pings of increasing size, probing for the first size that gets fragmented.

System operator commands: Suppose your monitor makes you suspect a security breach and you need, right now, to query the intrusion detection systems at every z/OS image:

```
D TCPIP,tcpip,NETSTAT,IDS.
```

Maybe you start up a half-dozen terminal emulators. Or maybe you'd like a browser-based tool that will send and system operator commands to any or all of your systems.

Software Diversified Services has been providing mainframe administrators with first-rate tools and technical support since 1981. To learn about Vital Signs VisionNet VIP, the z/OS-TCP/IP monitor from SDS, visit www.sdsusa.com/vip.