How to Optimize CICS® Socket Applications and Cut Costs

HostBridge® Socket Support

A HostBridge “Letters from the Trenches” White Paper
Copyright Notice

Copyright © 2011-2013 by HostBridge Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, optical, chemical, manual, or otherwise, without prior written permission. You have limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry this copyright notice. No other rights under copyright are granted without prior written permission. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Revision date: 7/1/2013

Trademarks

HostBridge and the HostBridge logo are trademarks of HostBridge Technology. IBM, CICS, System z, and z/OS are trademarks of IBM. All other trademarks mentioned are property of their respective owners.
How to Optimize CICS® Socket Applications and Cut Costs

HostBridge® Socket Support

Shortly after we announced HostBridge® support for the zIIP specialty engine, a large CICS® customer approached us with a challenging situation. Like any organization, this customer has to respond to competitive pressure by managing costs wherever possible. They happen to do very high-volume CICS application processing and were planning to add more System z® workload to CICS since it was their most cost-effective platform. With this prospect in mind, they asked if we could help them reduce CPU burn of their general purpose processor (GP) associated with socket applications and infrastructure.

Our engagement unfolded in two stages – (1) test and understand the performance characteristics of their CICS socket application and (2) improve performance and reduce CPU burn wherever possible. We tested thoroughly, and were pleasantly surprised by the results. Repeatedly, testing pointed toward potential improvements. Ultimately, we found we could exploit z/OS®, CICS Transaction Server, CICS Open Transaction Environment (OTE) capabilities, and the zIIP engine in the creation of a new high-performance socket infrastructure. We call it HostBridge Socket Support.

The approach has general applicability to any CICS customer that relies on socket applications. The higher your socket I/O volume, the more it matters. If you have a high-volume socket application doing hundreds of thousands or millions of transactions on a daily basis or at peak intervals, the solution we are about to describe may help you reduce MIPS consumption on the GP, shift workload to the zIIP, and, in both these ways, reduce mainframe TCO.

Initial Conditions

Since we were responding to a real situation, it makes sense to start with an understanding of the customer’s actual infrastructure:
As Figure 1 illustrates, the customer has (1) a z10 mainframe with CICS TS V4 running critical business applications. On the distributed side (2), an application server interacts with CICS through various communication layers, providing customer service agents with access to the CICS applications.

A critical component of this integration architecture is the CICS Socket Support infrastructure (outlined by the orange dashed line). Provided as part of z/OS Communication Server, it includes:

- Socket APIs for C and COBOL
- A standard set of components, specifically:
  - Standard Listener
  - Enhanced Listener
  - A Definition and Management component known as an EZAO transaction.

CICS Socket Support is a well documented workhorse. All of us in the industry have been using it for years, and it has earned its keep. It predates CICS OTE, and much of its original architecture was in fact reengineered to support OTE. But as we observed during this project, some of its original performance characteristics still persist.

---

1 Discussion of CICS Socket Support is found in the Communications Server section of the z/OS documentation.
2 As seen from the TCP/IP perspective, listeners appear as "servers."
CICS Socket Support Is Not CICS Socket Domain

It is important to make clear at the outset that CICS Socket Support is not the CICS Socket Domain.

As you can see in the following diagram, both CICS Socket Support and CICS Socket Domain provide code paths through the CICS environment to the z/OS UNIX System Services Callable BPX Sockets layer. But these code paths are very different.

The CICS Socket Support pathway (shaded orange, to the left) traverses the CICS Socket Layer, the Sockets Extended Support API, and the Sockets Extended Assembler Macro API before arriving at the actual BPX Sockets layer. By contrast, the CICS Socket Domain has a much cleaner pathway (shaded green, to the right).

Unfortunately, this cleaner pathway is not available to customers and ISVs. So anyone who has to develop socket applications must do so on the CICS Socket Support side. This had been true for the customer, and it was now true for us. Given these realities, our focus was to understand the performance metrics associated with the CICS Socket Support code path – and figure out how we might be able to make them better.

I repeat for absolute clarity: our focus is on the CICS Socket Support side, not the Domain side.

---

3 Figure 3 is from an IBM presentation created when IBM began developing the CICS Socket Domain. The original can be found in z/OS Communications Server, IP Sockets Application Programming and Interface Guide and Reference.
Zooming into CICS Socket Support

The customer had been using CICS Socket Support to get socket application requests to and from CICS application transactions. We zoom in on that in the shaded area below:

![CICS Socket Support Infrastructure](image)

**Figure 3. CICS Socket Support infrastructure.**

The architecture, fairly typical for CICS-based socket apps, is characterized by:

- Persistent connection between the distributed Communication Gateway and the CICS application transactions
- Multiple simultaneous Gateway-to-CICS connections
- Extremely high socket I/O volume – millions of socket operations per day.

**Research Focus**

Initial metrics led us to focus on the functional and performance attributes of one specific set of components:

- EZASOCKET application design patterns, performance, and APIs
- CICS Socket Listener design patterns
- CICS Socket Definition and Management patterns.

**Test Methodology**

Our objective was to perform I/O operations between the Communication Gateway and the CICS applications while keeping a sharp eye on the CPU burn in the CICS Socket
Handler componentry. In our testing, we used and compared results from two test harnesses – one “internal” and one “external.”

For the internal approach, we used a CICS-based tool that we developed specifically for this purpose. But we quickly discovered that results did not reflect the reality of the customer’s actual situation, in which all the work comes from outside System z. In large part, this was because when processes are initiated inside z/OS, the functionality of the TCP/IP stack known as hypersockets comes into play automatically. Hypersockets performs very well, cutting out much of the overhead associated with socket I/O.

When the process is driven from outside System z, the issues change. For our external approach, which more accurately represented the customer’s real situation, we used a PC-based test harness that the customer already had in place. On a network PC, the customer had installed a load injector, which we used (and later modified) to create workload profiles, drive the CICS application, and get a real view of socket I/O.

**Standard Test Cycle**

Our standard test cycle followed a rigorous procedure. In their high-volume process, the customer uses a dual socket approach, that is, for every communication from the outside world to System z there are actually two sockets used – one for sending and one for receiving.

We replicated this in our testing such that each test cycle caused the gateway to:

- Open two sockets via Listener Transaction – one for sending, one for receiving
- Start send/receive transactions to handle socket I/O
- Generate 2,500 request-response iterations (with no delays or other processes intervening).

Each request caused a LINK to a customer program, and bytes in and out were modeled consistently according to a preset “average” usage.

For benchmarking, we ran 1 concurrent test cycle (2 sockets and 2,500 iterations) and then 5 concurrent test cycles (10 sockets and 12,500 iterations).

We measured region-level CPU burn for various configurations. We also differentiated between CPU burn associated with socket applications and with socket infrastructure because we were trying to change only the socket infrastructure in and out of CICS, not the socket applications.

**Tooling Developed**

From the start we knew that we needed to take snapshots of a CICS region’s total resource consumption and that our measurements had to be:

- High-resolution – accurate down to the microsecond level
- Low-overhead
• Immediate
• zIIP- and zAAP-inclusive.

However, meeting all four of these conditions turned out to be challenging. The main reason was we simply needed better tooling. There are a few commercial products for performance testing, but none could meet all four requirements, and neither our customer nor, presumably, most others had such a product.

To remedy the situation, we ended up developing four tools:
• HBZT – a CICS transaction that provides a summary of MVS ASSB timers
• CICS XMNOUT exit to log transaction metrics via WTO
• CICS-based test harness to generate socket activity
• PC-based test harness to augment that customer’s existing load injector.

With this new tooling, we were able to drive our testing very fast and assess results from all angles. Because these are targeted tools to measure performance that is otherwise hard to capture, we are happy to make them available to CICS customers for free. (For a quick overview of HBZT and CICS XMNOUT, see Appendix: Tooling.)

Data and Solutions

As anticipated, under volume testing, the CPU burn of the CICS Socket Support infrastructure proved to be both precisely measurable and linear. (Representative test results for both CICS Socket Support and HostBridge Socket Support are provided starting on page 11.)

With accurate measurements in hand, we began to think about whether and how we could improve performance. Again, the customer was looking for marginal value of every additional CICS transaction to go up relative to marginal cost. If we could lower CPU usage, either directly or by shifting workload or both, we could make that happen.

Our measurements allowed us to isolate various components and their performance impact. Based on this data, we thought through and measured the potential benefits of various alternatives. We also looked closely at how we could exploit both CICS OTE and z/OS USS. Ultimately, we found that the low-hanging fruit was the CICS Socket Support, and particularly the code path associated with the EZASOKET API.

With the fundamental objective of creating a better performing solution without changing the target application or the CICS application, we developed two solutions.

---

4 We would like to extend a special thanks to two friends of ours in the industry, Larry Lawler and Ed Jaffe, who were kind enough to answer late-night questions about technical details while we were building these tools.
5 Anyone interested in using these HostBridge tools to test socket I/O is encouraged to contact Susan Henderson, Susan.Henderson@HostBridge.com.
6 Note: we do not characterize the CPU burn as “high” or “low”; the thing that really mattered to the customer was whether it could be lower or, more to the point, not so linear.
Solution One

The first solution is illustrated here:

In Solution One, everything shaded green remained unchanged: CICS application transactions, CICS Socket Definition/Management\(^7\), and Receive, Listener, and Send transactions. Components in the orange-shaded boxes were on the table.

In the design and development of Solution One, we:

- Leveraged the EZASOKET API as an established design pattern
- Replaced the CICS Socket Handler
- Exploited CICS TS OTE, z/OS, and USS.

Solution One worked quite well, successfully lowering the GP CPU burn associated with Socket I/O. All it required was a re-link of applications using the EZASOKET API with an alternative load module. The modification was completely transparent to existing listener, sender, and receiver transactions.

As good as the results were, we found there was still some overhead in terms of EZASOKET API emulation. As an API (historically a COBOL-based API), EZASOKET has to go through certain gyrations to get certain parameter transactions into a form acceptable to the operating system. We were confident this resource consumption could be reduced. We also knew that by shifting part of the workload to HostBridge components we could optimize zIIP-enablement.

\(^7\) We left the CICS Socket Definition/Management component unchanged because it works well. The EZAO transaction lets us configure start/stop, and enable/disable TCP/IP ports and the listeners associated with them.
Looking back at the performance implications of Solution One, we realized that the whole set of CICS listeners, senders and receivers has been around for a long time. Any large organization that relies on CICS Socket Support either uses the IBM® tools or has written their own. The upside is that the design patterns are fairly straightforward.

With the aim of creating new transactions compatible with what the majority of customers actually use – and thus improve efficiency even further – we turned our attention to the listener/receiver/sender transactions. The result was Solution Two.

**Solution Two**

Here is a diagram of Solution Two:

![Diagram of Solution Two](image)

**Figure 5. HostBridge Socket Support, Solution Two**

In Solution Two, CICS Socket Definition/Management remains untouched, as do the CICS applications themselves. In Solution Two, however, we:

- Replaced receiver, listener, and sender transactions with HostBridge alternatives
- Eliminated the EZASOKET API as an underlying design pattern
- Exploited CICS TS OTE, z/OS, USS, and zIIP.

One of the keys to this solution is the HostBridge alternative receiver, listener, and sender transactions. As a licensee of the zIIP-enablement technology from IBM,

---

8 Again, EZAO performs well and as a transaction it is still used to configure, start, stop, or enable socket applications for listeners.
HostBridge is permitted to run its software code and process its workloads on the zIIP. As such, HostBridge receiver, listener, and sender processes could now be shifted from the GP to the zIIP.

Performance improved further with Solution Two. We eliminated EZASOKET API emulation, the new socket infrastructure was entirely transparent to the customer's application, and CPU burn on the GP went down even further. As mentioned above, zIIP-enablement was a major contributing factor, and Solution Two maximizes zIIP potential. Socket processes running within HostBridge components now run on the zIIP, and the new component architecture also minimizes switching between the GP and the zIIP.

The New HostBridge Socket Pathway

Recall the old CICS Socket Support pathway, diagrammed above in Figure 2, which traverses the CICS Sockets Layer, the Sockets Extended Support API, and the Sockets Extended Assembler Macro API. HostBridge Socket Support dramatically simplifies the pathway, circumventing all of these components and going straight from the application layer to the z/OS USS BPX socket layer as directly as possible.

Test Results Without Concurrency

Figure 6 below provides representative results from three tests of three different socket configurations:

- Standard CICS Socket Support
- HostBridge Socket Support (alternative socket infrastructure) without zIIP
- HostBridge Socket Support (alternative socket infrastructure) with zIIP.9

---

9 All tests were performed on a z10 mainframe with GPs and zIIPs running at the same speed. On this high-end mainframe GPs were not capped; if your GP is capped, your results may be substantially better.
<table>
<thead>
<tr>
<th>Send TX (GP)</th>
<th>Recv TX (GP)</th>
<th>Total</th>
<th>% Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>140714</td>
<td>332702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>138355</td>
<td>317988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141509</td>
<td>336017</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg 140193</td>
<td>328902</td>
<td>469095</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (GP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128676</td>
<td>285711</td>
</tr>
<tr>
<td>125736</td>
<td>271014</td>
</tr>
<tr>
<td>119938</td>
<td>240784</td>
</tr>
<tr>
<td>Avg 124783</td>
<td>255836</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (GP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>54056</td>
<td>165486</td>
</tr>
<tr>
<td>54056</td>
<td>155751</td>
</tr>
<tr>
<td>54056</td>
<td>155751</td>
</tr>
<tr>
<td>Avg 54056</td>
<td>151630</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>% reduction - old vs new w/out zIIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>% reduction - new w/out zIIP vs new w/zIIP</td>
</tr>
</tbody>
</table>

**Figure 6. Test Results: One Concurrent Instance, 2,500 Total Requests**

Each test consisted of 2,500 load test iterations. The volume of results is substantial, so for the sake of clarity, we show only the representative results of three load tests and then calculate to arrive at the average number of microseconds of GP CPU burn.

The first set of tests, using CICS Socket Support, represent our baseline test for comparison with new configurations. The second set shows the results of testing the new HostBridge Socket Support without zIIP support, i.e., with all processing on the GP. As the results indicate, HostBridge Socket Support without zIIP reduced GP CPU consumption by about 17%.

The third set of results is from HostBridge Socket Support with zIIP enabled, and the results improve further. HostBridge with zIIP was 34% faster than HostBridge without zIIP. But the key takeaway is the final percentage. HostBridge Socket Support with zIIP reduced GP CPU consumption for socket processes by about 45%.
Test Results with Concurrency

While our single-threaded tests provided valuable and actionable metrics, we also wanted to test concurrency. Figure 7 shows results of testing 5 concurrent instances and 12,500 total requests processed.

<table>
<thead>
<tr>
<th>test: concurrent instances = 5; total requests = 12,500</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Socket Infrastructure (EZA-based)</strong></td>
</tr>
<tr>
<td>Send TX (GP)</td>
</tr>
<tr>
<td>1: 609800</td>
</tr>
<tr>
<td>2: 614881</td>
</tr>
<tr>
<td>3: 617669</td>
</tr>
<tr>
<td><strong>Total</strong></td>
</tr>
<tr>
<td>1226658</td>
</tr>
<tr>
<td>1234086</td>
</tr>
<tr>
<td>1259704</td>
</tr>
<tr>
<td><strong>% Reduced</strong></td>
</tr>
<tr>
<td>1854293</td>
</tr>
</tbody>
</table>

| **Alternate Socket Infrastructure (without zIIP)**         |
| Alt Send TX (GP)                                          |
| 1: 451684                                                |
| 2: 456651                                                |
| 3: 502901                                                |
| **Total**                                                |
| 782429                                                  |
| 780384                                                  |
| 804619                                                  |
| **% reduction - old vs new w/out zIIP**                   |
| 31%                                                      |

| **Alternate Socket Infrastructure (with zIIP)**            |
| Alt Send TX (GP)                                          |
| 1: 417841                                                |
| 2: 417338                                                |
| 3: 409281                                                |
| 4: 410077                                                |
| **Total**                                                |
| 657107                                                  |
| 613641                                                  |
| 613252                                                  |
| 600015                                                  |
| **% reduction - new w/out zIIP vs new w/ zIIP**           |
| 19%                                                      |
| **-44%**                                                 |

Figure 7. Test Results: Five Concurrent Instances, 12,500 Total Requests

With concurrent as with single processes, HostBridge Socket Support again improved performance. With concurrent processing, however, performance of HostBridge Socket Support without zIIP was even better than it was with single processes – increasing from a 17% reduction to 31%. This is testament to the IBM TCP/IP stack. Not unlike other TCP/IP stacks, it gets more efficient the harder you load it. Overall, HostBridge Socket Support with zIIP continued to show the best performance improvement – in the mid-40% range.

Of course your results are likely to vary for any number of reasons. Still, we think these measurements are order-of-magnitude accurate and that most any customer with high-volume socket I/O can expect CPU consumption savings in the 40% range using HostBridge Socket Support with zIIP- enablement.
Socket Support and HostBridge

Needless to say, our customer was very pleased. And other CICS customers with high volumes of CICS socket I/O are likely to see comparable performance improvements with HostBridge Socket Support.10

At HostBridge, we fully appreciate the workhorse capabilities that CICS Socket Support has provided. At the same time, the CICS Lab continues to evolve the CICS TS Open Transaction Environment with the aim of giving customers and ISVs new opportunities to create value. We developed HostBridge Socket Support in this spirit.

Our mission is unwavering. We want customers to continue to embrace CICS for high-volume, business-critical transaction processing long into the future, and we want to help make CICS better and more valuable.

Our products provide deep integration for real-world CICS applications with an extreme degree of precision, control, and optimization. We like complex CICS environments because we like to do the hard stuff. We want to accelerate integration projects, and our customers continue to provide amazing examples of speed and productivity implementing HostBridge solutions.

Ultimately, we want you to get the best bang for your integration buck. We want to lower your costs, particularly by ensuring that all of your integration services run inside of CICS for the highest performance and on the zIIP for lowest processing cost. We also offer flexible licensing. Our objective is to be attuned to your business and your value propositions, and to make sure ours matches yours.

We hope you’ll agree that HostBridge Socket Support further fulfills our mission.

10 Remember that only a licensed ISV can zIIP-enable this type of software.
Appendix: Tooling

HBZT CPU Transaction

This first tool we developed is called HBZT Transaction. It is a simple one-screen transaction designed to yield a range of information about address space CPU consumption, with a bare minimum of CPU time consumption inside of CICS. When you run the HBZT Transaction, you see a screen like this:

![Figure 8. HBZT ACTUAL Mode.](image_url)

In ACTUAL Mode, shown here, HBZT provides a snapshot of the MVS/ASSB for a single CICS region at the moment of entry. Inside the dashed line are various ASSB counters with explanations to the right. These values represent a baseline taken at the beginning of a test (baseline values can be reset to zero at any time by pressing PF1).

At this point, we simply run a test cycle, press Enter, and HBZT returns in DELTA Mode, which computes and presents the difference between the two sets of MVS/ASSB values between the two measurement points. DELTA Mode is shown here:
The HBZT tool enabled us to see real resource consumption and allowed us to run and measure a great many test cycles in a short period of time. As you can see, it shows CPU time for both the CP and the zIIP.

**XMNOUT Exit Metrics**

We also developed a CICS exit called the XMNOUT Exit, a tool that confirms CPU consumption exactly. From the perspective of CICS, XMNOUT Exit provides a point of reference for a range of CICS activities and EXEC CICS requests issued by the program. In our testing, we executed 40,000 EXEC CICS requests in the HBSR transaction and another 60,000 in the HBSS transaction. The metrics provided by XMNOUT Exit allow us to begin correlating them with actual CPU burn.

To learn more about this free tooling, contact Susan Henderson (Susan.Henderson@HostBridge.com).