Optimizing Demand Response Management Systems

How to Control High Volumes of Two-Way Communicating Devices

CONTENTS

WHITE PAPER

INTRODUCTION

The emergence of two-way controllable devices has opened up a wealth of possibilities for utilities and their customers. It has enabled customers to adjust the temperature of their home from thousands of miles away while giving utilities precise forecasting of demand response capacity. It has also brought technological challenges to vendors communicating with thousands of devices. At Itron, we take great pride in the ability of IntelliSOURCE® Enterprise™ to deliver large-scale demand response programs for our clients and it was essential that our system deliver the same results for programs that used modern two-way devices as it did for legacy one-way paging devices. We recently completed a project to analyze how efficiently we communicate with devices and how we could optimize our performance. In this series of three blog posts, we examined the process we took to test our system, the results we obtained and some lessons learned.

Here's a bit more background. Traditional paging networks are able to send the same single message to ten thousand devices just as simply as sending to ten devices. Besides being costly to install and maintain, the downside of those networks is that they do not provide real-time feedback from the devices[1]. Two-way devices solve that problem and most (including all Wi-Fi devices) are IP addressable. The downside of IP addressing is that every device must be sent a unique message so controlling ten thousand devices takes 9,990 more messages than controlling ten devices.

Some demand response management systems (DRMS) work by breaking up load into large groups (treating ten thousand thermostats like a large industrial facility), determining which groups to control, then sending a proprietary or OpenADR signal to a different system that actually communicates with devices. Itron's IntelliSOURCE Enterprise coupled with our IntelliTEMP® DirectLink™ and IntelliPEAK® DirectLink™ two-way devices has the more complex job of actually sending those messages to the ten thousand individual devices. The benefit of this approach is greatly improved forecast granularity and the flexibility for surgical event dispatch.

DETERMINING WHAT TO TEST

The first question we asked was "what is our goal?" The answer, of course, depends on your exact business need. Working with our customers, we reached a goal of initiating a demand response event, selecting the devices to control, sending messages to 100,000 devices and receiving the two-way acknowledgment in less than 120 seconds. Future testing will scale to 500,000 – 1,000,000 devices. We had a very specific and measurable goal, but it doesn't take into account the larger picture.

The Itron IntelliSOURCE Enterprise DRMS is used by more than just control room operators. Device installers use the system to manage their daily work. Customer support specialists use the system to answer customer questions. Measurement and verification specialists use the system to gather and process device telemetry. We needed to determine what other areas of the system would be in use that could have an impact on the performance.

We began by analyzing the production web logs to profile what actions were being taken by users. We also looked at the logs from the application's background processes. We concentrated on those executed around the time demand response events were called and those with longer run times. This gave us a good starting point, but it was difficult to determine the performance impact of each action we were seeing in the log.

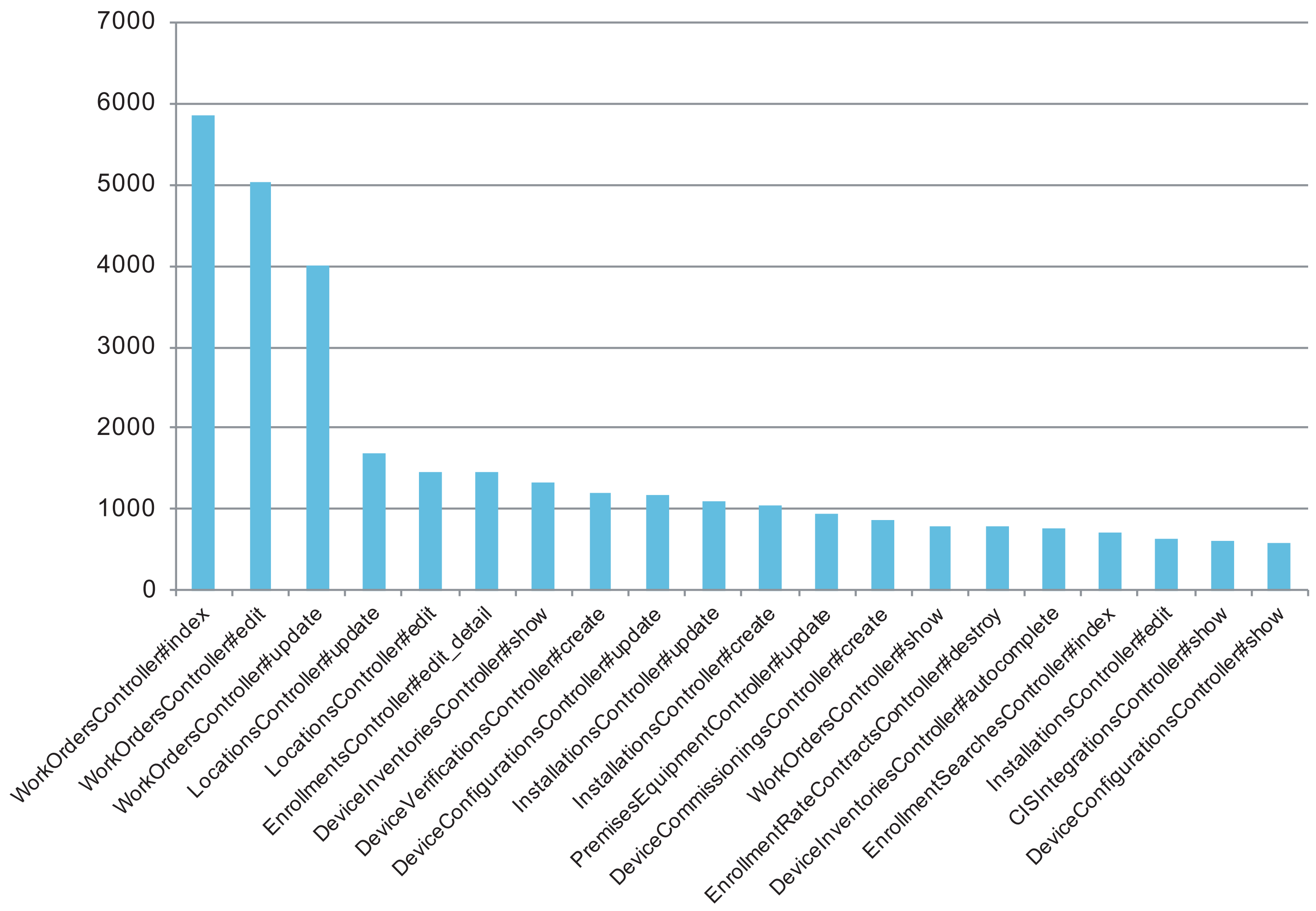
Next we used *tcpdump* to capture the raw network traffic between our application and its MySQL® database. We then used Percona's *pt-query-digest* to analyze the captured data. This approach gave us more insight into the system's behavior than MySQL's standard logging toolset with zero downtime or impact to our customer. The results of *pt-query-digest* allowed us to easily see the frequency and execution time of different database queries and to understand the impact of different system actions.

[1] These devices also do not provide customer engagement functionality and are limited to a utility direct install channel.

Putting this all together, we came up with our test script:

» Execute a demand response event sending messages to a rotating portion of the population
» Process the two-way telemetry received from the devices
» View the real-time status of devices acknowledging the demand response event
» View the real-time system status and forecasted capacity
» Process multiple device installations, registrations and commissionings

Counts of production web requests used to determine test steps



SETTING UP TESTING DATA

The second question we asked was “how do we set up our data?” This is a very important question. It is drastically different to test 100,000 devices all in a single load control group versus one hundred thousand groups each with one device. Both are valid, interesting tests and both will produce wildly different results and optimizations. We again worked with our customers and their data to profile not just how their data was set up today, but also how they planned to expand in the future. Ultimately, we created 7,000 groups with different numbers of devices based on the profiles.

Just as testing the demand response event creation alone was not enough, simply creating 100,000 devices is not enough. By using our analysis of operations, we identified the objects in our data model that have an impact on our performance. We added to our test system over 2,000,000 prospective customers, 750,000 legacy one-way devices and 5,000 weather readings. The scripts that we created will help us in the future scale to 500,000 – 1,000,000 devices.

It’s worth repeating the importance of setting up the test data correctly. There will always be an easier or simpler way to set up the data, but getting this part wrong can completely invalidate testing. For example, we saw a significant slowdown when iterating over a hash table of a certain type of device group. This type of group was part of the production system but not being used in the demand response events. It would have been easy to ignore. However, by setting up the test data correctly, we were able to move from a hash table to a set and see a dramatic improvement in our performance.

SIMULATING DEVICES

Since we don't have 100,000 extra IntelliTEMP DirectLink smart thermostats lying around for our performance testing, we built a software simulator to mimic an individual device and a testing harness that allowed us to easily launch thousands of simulators.

We again looked at the data in our production systems to identify the key functionality to simulate. Certainly, connecting to our server was important, but we also knew that devices do not stay connected 100% of the time. So we built in a variable disconnect rate and distributed simulators based on the
observed behavior of deployed devices. Receiving and acknowledging messages are also core functionality, but not all devices respond immediately. So we built in a variable lag rate and distributed simulators again based on the observed behavior of deployed devices. Lastly, we built functionality for the simulators to push our periodic telemetry information along with demand response status messages in the same manner as our deployed devices.

To implement the device simulators, we turned to the Erlang VM. Erlang's inherent parallelism and ease of scaling made it an ideal choice to run thousands of messaging based simulators. We also developed a test harness, also built in Erlang, to launch and monitor the simulators. The test harness ran simulators based on details[2] from a configuration file that was generated automatically from our test data and distribution profiles. This gave us tens of thousands of simulated thermostats and load control switches, each with individual characteristics based on observations from our real-world devices.

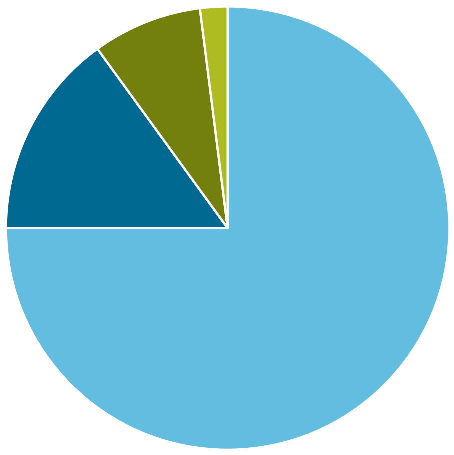
EXECUTING TESTS

With our test steps defined, test data created and device simulators implemented, we could begin our actual testing. We began by building an automated script to repeatedly and consistently execute our test steps. These scripts executed the test steps either by mimicking a series of user web requests or by accessing existing IntelliSOURCE Enterprise APIs. To help us monitor the tests, the scripts pushed intra-test logging and preliminary results into our Slack channel.

So that we could understand the performance trend as we scaled, we started our tests with 25,000 devices and progressively built our way up to our goal of 100,000. In the future, we'll scale to 500,000 – 1,000,000 devices. We built additional scripts that allowed us to automatically add simulated devices[3] to our test environments. Removing devices proved more complicated, so we saved system snapshots before adding more simulated devices.

While our test script ensured we executed our tests in a consistent and repeatable manner, we observed identical test cycles. The interactions of unrelated components (from the application layer to the testing tools to the physical hardware) and the intentionally created randomness (different devices targeted for demand response, distributed lag rates, etc.) both contribute to differences in results. While this variability can be frustrating, successful performance testing must embrace (or at least accept) it. We managed the variability by executing our tests numerous times, ultimately ending up with more than 250 test cycles and over 1,500 data points.

Distribution of simulated device attributes

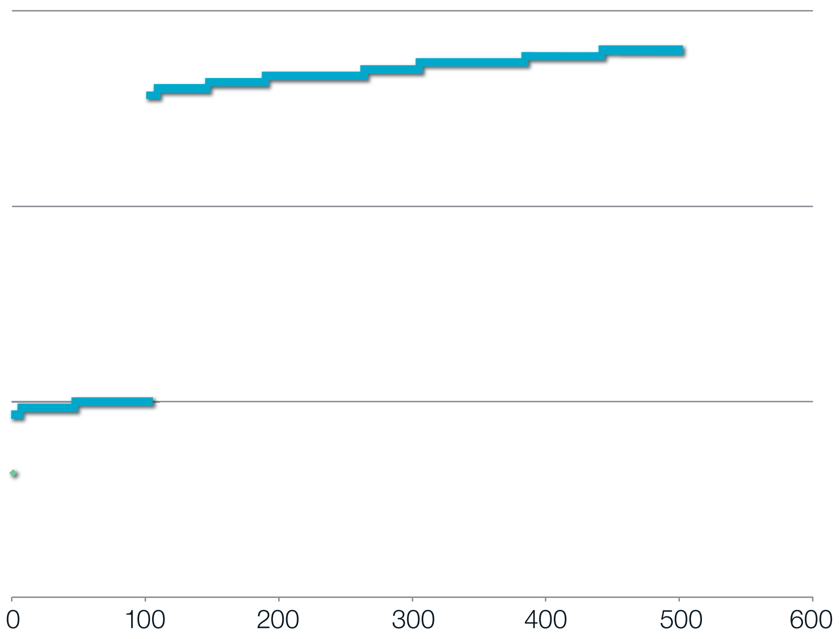


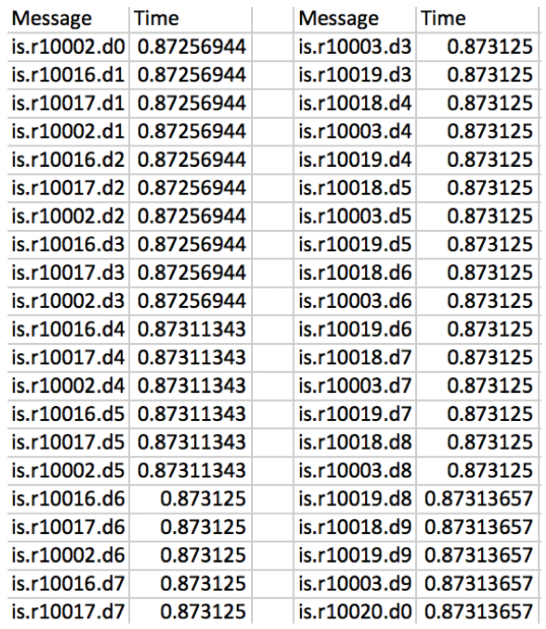
- 🟦 0.4% chance of disconnection
- 🟦 1.0% chance of disconnection
- 🟩 2.5% chance of disconnection
- 🟩 10% chance of disconnection

[2] Credentials, disconnect rate, lag rate, telemetry frequency, etc.

[3] In addition to devices, our script added the associated accounts, premises and enrollments managed by the IntelliSOURCE Enterprise DRMS.

Effective data visualizations expose patterns that are difficult to discern in raw measurements





Message	Time	Message	Time
is.r10002.d0	0.87256944	is.r10003.d3	0.873125
is.r10016.d1	0.87256944	is.r10019.d3	0.873125
is.r10017.d1	0.87256944	is.r10018.d4	0.873125
is.r10002.d1	0.87256944	is.r10003.d4	0.873125
is.r10016.d2	0.87256944	is.r10019.d4	0.873125
is.r10017.d2	0.87256944	is.r10018.d5	0.873125
is.r10002.d2	0.87256944	is.r10003.d5	0.873125
is.r10016.d3	0.87256944	is.r10019.d5	0.873125
is.r10017.d3	0.87256944	is.r10018.d6	0.873125
is.r10002.d3	0.87256944	is.r10003.d6	0.873125
is.r10016.d4	0.87311343	is.r10019.d6	0.873125
is.r10017.d4	0.87311343	is.r10018.d7	0.873125
is.r10002.d4	0.87311343	is.r10003.d7	0.873125
is.r10016.d5	0.87311343	is.r10019.d7	0.873125
is.r10017.d5	0.87311343	is.r10018.d8	0.873125
is.r10002.d5	0.87311343	is.r10003.d8	0.873125
is.r10016.d6	0.873125	is.r10019.d8	0.87313657
is.r10017.d6	0.873125	is.r10018.d9	0.87313657
is.r10002.d6	0.873125	is.r10019.d9	0.87313657
is.r10016.d7	0.873125	is.r10003.d9	0.87313657
is.r10017.d7	0.873125	is.r10020.d0	0.87313657

MEASURING RESULTS

One of the benefits of two-way devices is knowing exactly which device received a message and the time the message got to the device. This information is stored in IntelliSOURCE Enterprise and formed the basis of our measurement. For example, we recorded when outgoing messages were created and when responses from each device were stored in the database. We began with manually executed SQL queries and analysis in Excel. This was cumbersome, but gave us a great way to iterate our queries and experiment with different presentations. Once we were happy with our measurement, we built a simple Ruby on Rails application to automatically fetch data, perform analysis and produce reports for each test event. A huge benefit of this investment is that it can be applied to measure how our customer's production systems are currently performing.

We followed a similar approach to our log file analysis. We began with standard UNIX® tools to parse the logs and Excel to perform analysis. This evolved into a set of custom Splunk reports and dashboards enabled by automatic log forwarding.

One of the key items that both our custom measurement application and Splunk provides is data visualization. Meaningful data visualization is invaluable when comparing different test cycles, uncovering performance issues and communicating with others. Meaningful data visualization is also hard; it requires strong knowledge of the application to determine what to present and patience to experiment with the best way to present it. Throughout our analysis we used column charts, scatter charts, pie charts and candlestick charts to help us answer different questions.

In the data on the next page, there is a series of slowly changing numbers that don't lead us to any conclusions. However, by looking at the graph we could see a gap in outgoing messages for period of time. This helped identify a bottleneck within our message broker's disconnect process.

RESULTS

At the end of this project, we demonstrated that with 100,000 DirectLink devices the IntelliSOURCE Enterprise DRMS can calculate which devices to include in a demand response event, send messages to each device and receive two-way acknowledgments from all of the devices in under 100 seconds. This result met our goal and has surpassed all of our customer's service level agreements. It also gives us a great foundation as we continue to scale to 500,000 – 1,000,000 devices.

No performance testing effort would be complete without finding and fixing a few bottlenecks:

» **Optimizing data structures.** While newer programming languages have made it easier and more enjoyable to write code, they have also made it easier to write code that performs poorly (especially at large scale). Refactoring data structures by focusing on the most efficient type for each use case yielded significant performance impacts.

» **Optimizing SQL queries.** This is one of the first places people look when addressing performance concerns. We found some performance increases by simply changing SQL queries (particularly by removing IN clauses), but the real payoff was denormalizing specific parts of the schema to make the queries significantly simpler[4].

» **Separating and buffering operations.** It's natural to implement a single functional requirement (receive and store telemetry) as a single process. However, at large scale, a delay in one operation (storing telemetry) can impact the other operation (receiving messages). Separating those operations into multiple processes connected by a queuing[5] mechanism can alleviate this bottleneck.

Lastly, we've been able to build a framework for testing our solutions in an environment that more closely resembles our customers' production systems. This is an important addition to our existing automated unit and integration tests. We've already re-used that framework for other projects within Itron and are excited to continue enhancing it.

For more information, please visit our IntelliSOURCE Enterprise webpage.

4 The tradeoff is that we are storing more data.

5 We're using Redis.

Join us in creating a more **resourceful world**.
To learn more visit **itron.com**

CORPORATE HQ

2111 North Molter Road
Liberty Lake, WA 99019 USA

Phone: 1.800.635.5461
Fax: 1.509.891.3355