FIELD TRIALS OF AN ALL-SOFTWARE GSM BASESTATION

Jeffrey Steinheider (Vanu, Inc., Cambridge, MA, USA; jlstein@vanu.com);
Victor Lum (Vanu, Inc., Cambridge, MA, USA; viclum@vanu.com);
Jonathan Santos (Vanu, Inc., Cambridge, MA, USA; jrsantos@vanu.com)

ABSTRACT

Vanu, Inc. installed a GSM basestation in DeLeon, Texas, in June 2003 as part of the Mid-Tex Cellular radio access network. The basestation platform consists of an off-the-shelf HP ProLiant server with two 2.8 GHz Intel processors, running Linux, connected to an ADC Digivance radio transceiver. All of the signal processing, protocol processing, and GSM BSC functionality is implemented as application software running on the HP server. This paper describes the system, the software, and experiences with the field trial.

1. INTRODUCTION

Vanu, Inc. deployed a software radio based GSM Basestation System for field trials in DeLeon, Texas at the end of June, 2003. This paper describes the system, the software design process used for the GSM waveform implementation, and early results of the field trial.

2. DESCRIPTION OF THE TRIAL SYSTEM

The trial was initiated at the invitation of Mid-Tex Cellular, DeLeon, Texas. Mid-Tex operates an IS-136 AMPS/TDMA system covering 8,000 square miles in 6 counties, about 2 hours West of Dallas Fort Worth airport.

The trial installation consisted of two basestation transceivers (BTS) and a basestation controller (BSC), with each running on an industry standard Hewlett Packard ProLiant DL380 server with dual Intel Xeon 2.8 GHz processors. All of the signal processing, protocol processing, and BSC functionality was implemented as application level software running on top of the Linux operating system. The BTS systems used the ADC Digivance Long Range Coverage Solution as an RF interface.

Each BTS provided two TRX's of GSM capacity in one sector. Each TRX contained 8 time slots. Two of the time slots per BTS were used for the control channel, so each BTS supported 14 simultaneous voice calls. The first BTS was located in DeLeon, while the second BTS was located eleven miles away in the neighboring town of Gorman. The BSC equipment was located in the DeLeon central office. One of the goals for the trial was to evaluate hand-over between the two BTS's, so directional antennas were used to ensure overlapping coverage on the road between the two towers. The current Mid-Tex IS-136 system that operates from the same towers uses omni directional antennas and does not provide continuous coverage between the two towns.

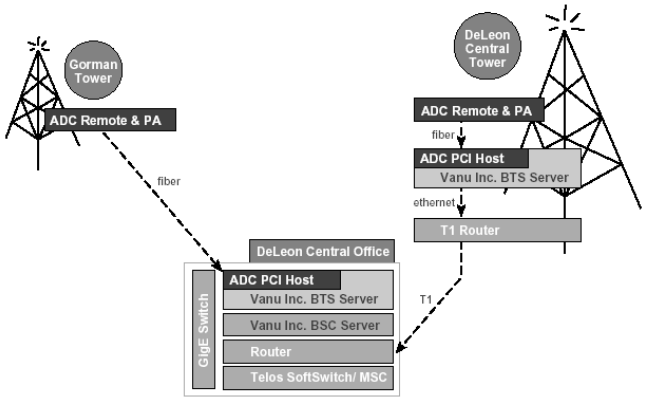

Figure 1: Trial System Architecture

The ADC Digivance system provides a fiber optic link for remote antennas. This capability was used for the Gorman BTS installation. The HP server running the BTS software was colocated in the DeLeon central office with the BSC, with its antenna 11 miles away.

Colocation of several base stations in a central location is known as base station hoteling. Hoteling offers cost savings to an operator because it reduces footprint and eliminates climate control at the cell tower, while centralizing equipment, which reduces maintenance costs.

For the DeLeon BTS, the BTS server was installed at the tower site. It was connected via a T1 link to the BSC server. This was a more traditional cellular system installation compared to the Gorman BTS.

An HP Procurve gigabit Ethernet switch at the DeLeon central office connected the BSC server, the Gorman BTS server, and the Cisco router that hosts the T1 link to the DeLeon BTS. All voice and signaling was transported over IP.

Switch (MSC) functionality for the trial was provided by a TELOS soft-switch. The first phase of the trial used a switch running at TELOS corporate headquarters in Vancouver, British Columbia, Canada. All signaling and voice traffic links between the BSC and MSC were routed over the public Internet. In late September, a TELOS switch

was installed in the DeLeon central office, connected to the same gigabit Ethernet switch as the BSC and BTS.

3. DESCRIPTION OF SOFTWARE PROCESSES

3.1 Open source tools and libraries

Open-source tools and libraries were used extensively. Several problems that were encountered with open-source components in building the BTS and BSC were solved by debugging combined with research on the public Internet. The primary tools used are compilers and debuggers (gcc/gdb), libraries for multimedia and network protocols (vovida), logging (log4cpp), and configuration management (libconfuse). The error-checking tool Valgrind helped improve the BTS and BSC code, as well as fix bugs in other open-source packages.

3.2 High level languages

The Mid-Tex system was implemented in C++. The choice of C++ represents a mid-point on the spectrum of high-level languages between C (high performance and excellent portability, but weak type-checking and little support for large-scale programming), and Java (modern conveniences, like strict type checking and garbage collection, but poor performance). A great advantage of C++ is that the whole system, from high-performance signal processing code to network protocols to high-level control, can be implemented in one language. One drawback of C++ is exposure to certain classes of errors that are difficult to debug, such as use of uninitialized memory, memory leaks, and improper memory accesses. On the other hand, C++ development knowledge is fairly widespread, and techniques for using the language and addressing its deficiencies are widely available. The 'design by contract' technique and the Valgrind error-checking tool address specific problem areas.

3.3 Test-driven development

Development of the BTS and BSC software followed a strict test-driven development strategy, often called test-first programming. In test-driven development, new functional code is added only after an automatically executable and verifiable test is written for the new functionality. In effect, the test becomes a fully precise specification for the functional code. This approach helps the developer to think through the design of the software before the code is written and it also improves the test coverage of the code. Using this approach, implementers also discover and correct errors early, keeping costs low. In addition, tests created by the developers form the core of the regression test suite used in continuous integration.

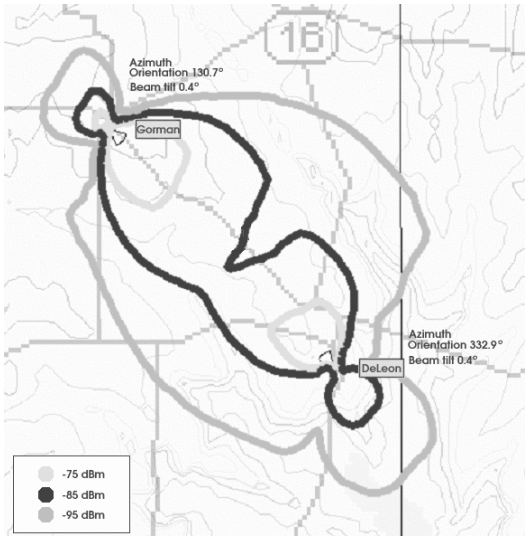

Figure 2: Trial Coverage Area

As an aid to TDD, the test coverage of the code is regularly measured using the 'gcov' tool built into gcc. This is not to enforce a particular numeric target. Instead, the coverage tool is used to indicate where in the system test coverage is weak.

When untested code is detected, one of three response options is selected. If the untested code is functionally important, tests are added. If the code is unnecessary, it is removed. If the code is error-trapping code that is difficult to test, it is validated through inspection, often in 'pair programming' sessions.

3.4 Continuous integration and testing

As the software was developed, a continuous integration system was employed. This system automatically built the software from the latest source code and ran it through a comprehensive suite of functionality tests. This process occurred several times a day.

The continuous testing provided a solid 'safety net'. An error introduced into the code, or an integration problem between two components, had a high probability of being detected within several hours while the cost of a fix was still very low. This permitted rapid progress and a good measure of system correctness at any given time.

3.5 Pair programming

Pair programming, in which two programmers collaborate side-by-side on the same task, was utilized in the software development effort. Key benefits of this practice include continuous design and code reviews as well as increased knowledge transfer.

3.6 Design for testability

In designing the system, we were careful to segment the system into components that could be tested independently. By using network interfaces and C++ abstract classes, we could replace peer system components with test apparatus to test classes and subsystems in isolation. By replacing the complex objects with simple mock objects in our tests, we could easily record the method calls made by the class under test as well as control the values returned by the calls, allowing us to setup and test particular interactions that would have been difficult to create with the actual peer object.

3.7 Design by contract

Software objects were implemented using the design by contract methodology. In design by contract, method preconditions and post-conditions as well as class invariants explicitly specify the obligations of the objects and their clients. This communication creates a better understanding among the software developers of how the various objects interact. Furthermore, the contracts are enforced through assertion checks to aid in debugging and quality assurance of the software.

The assertion checks are separated into two classes, those intended for use during the development phase only and those intended to remain in the system during operational use. Development phase assertion checks remain in the code but are disabled through a compiler option after the system has been qualified. Knowing that development phase assertion checks have no run time performance cost frees developers to insert frequent and aggressive checks of all aspects of the processing and system state.

3.8 Valgrind

Valgrind is a dynamic error-checking tool, comparable to commercial tools such as Purify or Insure++. In its basic configuration, it detects errors such as use of uninitialized memory, heap errors including leaks, boundary errors, and double-frees, and misuse of certain APIs. Another configuration is able to detect race conditions in multi-threaded code.

Valgrind's basic memory checks were integrated into our continuous integration builds to catch memory errors that escaped notice at check-in time. Perhaps most importantly, Valgrind allowed the detection and fix of bugs, such as memory leaks in third-party libraries, on which there is dependence.

3.9 Logging

Logging of system events was both a valuable development aid and an important product feature. The logging system used (log4cpp) has adjustable reporting levels and output formats, making it suitable both for development-time trace statement debugging, and for tracking system progress and anomalous events in the deployed system. The logs were invaluable during early field testing, as the team in Cambridge could remotely monitor system behavior and correlate logged information with reports from testers in the field.

4. TRIAL RESULTS

After the initial installation of the equipment and tests to ensure that it was functioning properly, Vanu, Inc. employees spent three consecutive weeks at the Mid-Tex site to test and improve the system until it was ready for Mid-Tex to trial. This was the first field trial of the Vanu, Inc. software GSM basestation in a live outdoor environment. All prior testing was in a controlled lab environment.

Figure 3: System Hardware at the DeLeon Central Office

The tests were performed onsite using commercial, off-the-shelf GSM phones that support GSM in the 850 MHz Cellular band. Most of the tests consisted of drive tests with the phones. These tests were used to exercise handover, power control, and timing advance, as well as to determine the coverage area.

The first week was spent testing mobile to mobile calls on the network. Several issues were discovered in the initial tests. Originally, the system was too aggressive when releasing a traffic channel due to poor reception, which resulted in a large number of dropped calls. This was corrected by increasing the length of time the basestation

measured mobile receive quality before deciding to release the call. In a related problem, channels that were being released by the basestation were not properly cleared, preventing future phone calls on those channels. This was detected early on and fixed in the next release.

Another problem discovered during the first week was that on some calls, there was a significant audio latency that was introduced by the VOIP libraries. This example took longer to solve, as it required the creation of new system tests to allow engineers to reproduce the problem. It occurred because of differences between the trial system network and the network used for initial testing. The T1 link added additional latency to the VOIP packets, and the delay was accumulating to cause the noticeable latency in the voice calls. An open source program, NIST Net, was used to simulate the latency added by the T1 routers so the problem could be reproduced and tested in the lab. The voice latency was removed in a release that was deployed during the second week of onsite testing.

The power control algorithm was modified throughout the three weeks in an attempt to find the balance between conserving a mobile phone's battery life and maintaining a high quality voice link in all conditions. The first algorithm was too dynamic, frequently requesting the mobile to significantly change its transmit power every SACCH period. The algorithm frequently instructed the mobile to reduce its transmit power, resulting in poor reception by the basestation. Trying to fix this resulted in a new problem: the algorithm did not increase the mobile's transmit power fast enough when the mobile was moving away from the tower at high speed. This resulted in dropped calls while driving away from the tower.

The adjustable reporting level of the logging system was instrumental in working on the power control adjustments. In normal operation, the basestation does not log all of the power measurements during a phone call, to reduce the load on the basestation as well as reduce the amount of information in the logs. For these tests however, the reporting level was increased so the basestation reported the power level measured by the basestation on each timeslot. This allowed engineers to compare the measured power with the power control messages sent to the mobile, in order to determine when the power control algorithm was failing. After testing several implementations of the power control algorithm, a balance was reached that smoothed the mobile's changes in transmit power, while still increasing the mobile phone's power at a high enough rate to match the movement of the mobile traveling in a car away from the basestation at high speeds.

The handover algorithm suffered from similar problems. The first version caused the mobiles to handover too quickly, resulting in many handovers back and forth between the two towers as a phone traveled in a straight-line path between the towers. In this case a drive tester would converse with an engineer back in Cambridge, while the engineer examined the current log messages. Instructions could easily be given to the drive tester to try different test cases, and the engineer could be notified of any problems with the voice link. The handover algorithm was tuned so that handover now occurs at roughly the half way point between the two towers.

The network architecture and physical deployment of the trial system also created a few problems for the basestation system. There was not enough capacity at the Mid-Tex sites to place all of the basestation system on their high availability power supplies, so several parts of the system lost power at different times due to storms. The sudden shutdown of a BTS introduced several unseen states in the BSC, which were caught through the design by contract method. The contracts made the problems obvious, by reporting the violations to the log. This provided a starting point in the logs enabling engineers to determine the sequence of events that caused the problem.

Using the public Internet as the voice and signaling connection between the MSC and BSC introduced the same type of problems. The link between the MSC and BSC could sometimes disconnect for several minutes at a time, a situation that was not handled well at the start of the trial. As in the previous case, design by contract found many of the error cases, and the messages from the logs were used to find the exact cause of the problem.

As a result of following the software processes described earlier, Vanu Inc. iterated through 12 releases of the basestation software during this three week period. New releases were downloaded to the system over an Internet connection, and could be installed in a few seconds. The ease of deploying new releases allowed the team to make many small, incremental changes with each release. Improvements were made to both the BSC and BTS, across all aspects of the system. The radio interface, the voice over IP transport protocol, and the link detection between the BSC and MSC are a few examples of areas where improvements were made.

5. CONCLUSION

Design by contract and the use of a logging system were very important for catching errors and debugging the system. The other software processes described earlier allowed Vanu, Inc. engineers to increase the speed of development and to make many changes with the confidence that these changes would not result in new bugs elsewhere in the system.

The use of off-the-shelf servers and the Linux operating system also streamlined the development and deployment process. Capabilities such as log retrieval and remote software installation were built in to these commodity

components, meaning these did not have to be custom-developed.

The deployment phase described in this paper was completed in three weeks, at which point the system was ready for Mid-Tex begin its evaluation. After extensive tests, Mid-Tex declared itself highly satisfied with the performance of the Vanu, Inc. devices.

This successful field trial represents an important milestone in commercial acceptance of Vanu Software Radio, and in the acceptance of flexible SDR systems in the general telecoms marketplace.

Vanu, Inc would like to acknowledge the National Science Foundation for partially funding the development and testing of the Vanu Software Radio basestation system.