ARL-TR-7540 ● Nov 2015

US Army Research Laboratory

Analyzing GAIAN Database (GaianDB) on a Tactical Network

by Ryan Sheatsley and Andrew Toth

ARL-TR-7540 ● NOV 2015

US Army Research Laboratory

Analyzing GAIAN Database (GaianDB) on a Tactical Network

by Ryan Sheatsley and Andrew Toth
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE						*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* November 2015	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** 07/2015–09/2105
4. TITLE AND SUBTITLE Analyzing GAIAN Database (GaianDB) on a Tactical Network		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Ryan Sheatsley and Andrew Toth		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CIN-T 2800 Powder Mill Road Adelphi, MD 20783-1138		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-7540
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

This report covers the integration of the GAIAN Database (GaianDB) with the second-generation common sensor radio (CSR). Performing measurements with the second-generation CSRs, we discovered that Internet Protocol (IP) encapsulation of GaianDB traffic in radio frequency (RF) transmissions added minimal overhead to RF networks, and we observed high percentages of query success across our tests and a reasonable performance impact on query round trip time. In this report, we first provide background concerning GaianDB and the CSRs, and how these 2 pieces fit with the current scope of the Army. Next, we state our goals and how we accomplished them. Afterwards, we describe our experimentation, including setup, tests, and results. Finally, we conclude with an interpretation of our results and describe future goals.

15. SUBJECT TERMS
Gaian Database, GaianDB, Tactical

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Andrew Toth
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	18	**19b. TELEPHONE NUMBER (Include area code)** 301-394-2746

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

The primary author would like to thank Andrew Toth for his mentorship, guidance, and encouragement throughout this project; Ron Tobin for providing test radios and giving us a technical background on the common sensor radios; Peter Lawrence for the serial platform code we augmented without which this project would have not been possible; Theron Trout who helped identify evasive bugs and provided recommendations for alternative standard C functions from legacy functions; and Mark Reinsfelder who set up the environment we used to perform our research and experiments, as well as provided insights on network configurations.

INTENTIONALLY LEFT BLANK.

1. Introduction

The GAIAN database (GaianDB) is a dynamic distributed federated database written in Java. This biologically inspired database automatically discovers other GaianDB nodes, creating an ad-hoc network based on metrics that contribute to a value called the fitness factor of a node creating a topology that is independent of the underlying network topology. GaianDB nodes federate 1 or more data sources, which may include plain text, relational database management system (RDBMS) databases, and other files, and exposes them as 1 unified structured query language (SQL)-compliant data source. This “store locally query anywhere” (SLQA) approach reduces overhead associated with replication of databases by leaving the data at their source and only retrieving them when needed. GaianDB also features query caching, an explain-network-route query option, and dynamic configuration. A small 4-MB footprint and support for ad-hoc network formation make GaianDB an appropriate database for sensors connected to tactical networks.[1]

The common sensor radio (CSR) is a military mesh network radio for use with unattended sensors. The radios are highly energy efficient; they are designed to function for lengthy periods of time before requiring battery replacement. Much like GaianDB, the CSRs have built-in discovery commands, which initialize the ad-hoc network.

A challenge that the Army faces today is how to maintain battery efficiency and performance in locations with active adversaries. Presently, sensors will transmit data as they receive them to a central base for processing. Imagine if multiple radios were collecting and sending sensor data to central base continuously. Not only could there be a significant impact on radio battery life, but the radio frequency (RF) networks themselves could become slow or unresponsive with their finite bandwidth. To solve this, the CSRs could manage their own sensor data and send information back to a central base *when requested.* At the cost of a sustained increased energy consumption, we perform a tradeoff of lower network bandwidth consumption and burst energy consumption. GaianDB allows us to perform this tradeoff when coupled with the CSRs.

2. Implementation

Figure 1 shows an image describing how GaianDB communicates over the CSRs with our implementation.

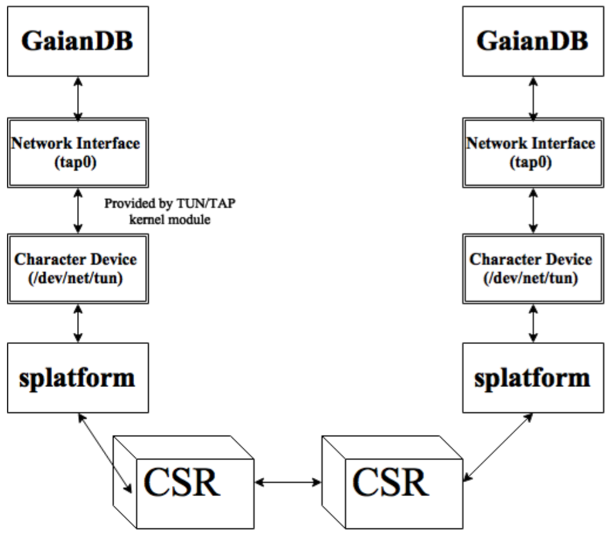
GaianDB
Network Interface (tap0)
Provided by TUN/TAP kernel module
Character Device (/dev/net/tun)
splatform
CSR
CSR
GaianDB
Network Interface (tap0)
Character Device (/dev/net/tun)
splatform

Fig. 1 GaianDB communication over the CSRs

An obstacle we had to overcome was how we would integrate the GaianDB with the CSR. Our challenge was to answer the following questions:

- What protocol does GaianDB use?
- How does GaianDB establish and maintain connections?
- How could we interface GaianDB with the CSRs?
- How do we determine if integrating GaianDB with the CSRs is feasible?

To start, we used low-level networking tools to see the traffic coming in and out of a host running GaianDB. Upon application startup, we observed Internet Group Management Protocol (IGMP) traffic on our network coming from the GaianDB node. GaianDB uses IGMP to establish multicast group memberships. Interestingly, we also observed multicast group requests on all network interfaces. GaianDB does not pick one interface to transmit traffic through exclusively, which turned out to be a behavior we'd exploit later on.

Once another GaianDB node appears on the network and also joins the multicast group, the GaianDB nodes will (depending upon their configuration files) stop sending out multicast group requests and begin to directly communicate one another via either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) network protocols. Afterwards, if a GaianDB node drops out of the network (again, depending upon a node's configuration file), the remaining node will revert back to broadcasting multicast group membership messages to obtain another GaianDB connection.

After we understood the basic operation of GaianDB from initialization to process termination, we revisited our initial problem with new approaches on how to solve it. Our first approach involved modifying the GaianDB source code to communicate with a TCP/UDP server that could communicate directly with the CSRs via the CSR's serial port. However, GAIAN has over 800,000 lines of source code. It initiates and terminates TCP/IP connections in multiple locations throughout its runtime, and has complex logic for timeouts, caching, and multicast group management, by which all would have to be modified to communicate with our server and maintain utility. Not only did we quickly realize that this approach would be infeasible with the short 8 weeks we had, but this would also affect the core functionality of GaianDB and would require modifications to any subsequent updates to GAIAN. Instead, we had to create a cleaner and more modular solution to the problem.

Our next approach was to implement a packet sniffer that would capture outgoing GaianDB traffic, pipe packets through the radios by leveraging code we used last summer to prove the CSRs could be IP-enabled, and finally send the packets to the appropriate network interface card (NIC) connected to a host running GaianDB. Unfortunately, this approach did not work. We were able to successfully capture outgoing packets and send them over the radio, but we were not able to send a packet to another NIC. When we "sent" a packet to a remote host, we observed that the remote NIC reported that *it* sent that packet, instead of receiving it. We believe some fields at the link layer were changed when we sent the packet to the remote host that caused this behavior. Thus, we were unable to instigate a response from GaianDB running on the remote host.

After we realized this shortcoming in our implementation, we sought for new solutions to our hurdle. In a couple of days, we came across a standard library in C that could be used to create virtual network kernel devices, which uses the network tunnel (TUN)/network tap (TAP)[2] kernel module in most UNIX-based systems. The TUN portion of the module emulates a network layer device and operates with layer 3 (e.g., IP) packets. Conversely, the TAP emulates a link layer device and operates with layer 2 (e.g., Ethernet frames) packets. A TUN is used to establish host-to-host communication with Point-to-Point Protocol over Ethernet (PPPoE), while a TAP is used to create a network bridge or, in our case, create an ad-hoc network by encapsulating raw packets (which contain preambles and Ethernet frames) inside CSR packets.

Thus, we leveraged some code we used last summer, which was originally written by the CSR development team as a "just-add-water" solution to communicating with the radios through their serial ports called `splatform` (serial platform), and augmented it with a TAP. On application startup, our process would use the TUN/TAP kernel module to create a dummy network interface that would appear as “tap0” in the list of available NICs. Our first indication of success, was observing GaianDB send IGMP packets to all network interfaces, including our tap0 interface. By exploiting this behavior, we were able to direct GaianDB to always communicate with our dummy interface without touching any of the underlying GaianDB functionality.

The TAP interface is simple in nature: all incoming traffic sent to the dummy interface is piped to a character device in `/dev/net/` called either `tap` or `tun`, depending upon how the kernel module is configured. Afterwards, in our application, we can read data from this character device through familiar file input/output (I/O) operations, like `read` and `write`. After a packet is sent to our local tap0 interface, we can perform a `read` on the file descriptor associated with our `tap` character device, pipe the packet through the code Peter Lawrence wrote, and finally execute a `write` on the remote node’s character device once it receives our packet. From there, we again used low-level networking tools to verify incoming and outgoing packets through our TAP interface. Remote GaianDB nodes would see the incoming multicast group membership packets, promptly join the multicast group, and begin to communicate directly with another GaianDB node.

As far as GaianDB is concerned, it is communicating to another node via IPv4, its native protocol for communication. Any GaianDB node is completely agnostic to the fact that, in reality, it is communicating over RF. Thus, we have achieved our minimally invasive and modular solution to our GaianDB and CSR integration question.

3. Experimentation

Our experiment setup (Fig. 2) is a network of 3 CSRs and Raspberry Pi’s running our augmented `splatform` and GaianDB.

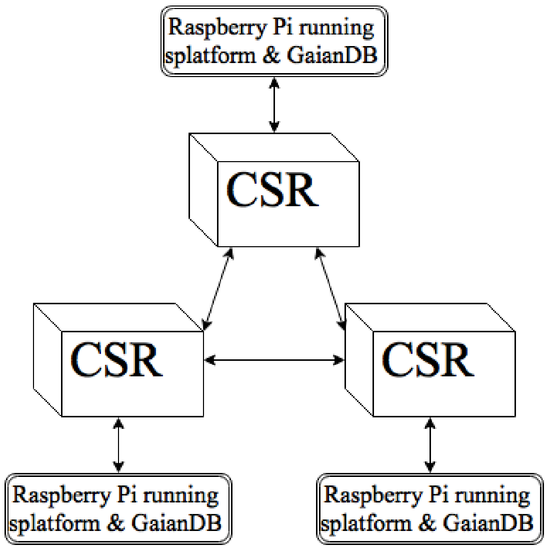
Raspberry Pi running splatform & GaianDB
CSR
CSR
CSR
Raspberry Pi running splatform & GaianDB
Raspberry Pi running splatform & GaianDB

Fig. 2 Experimental setup using 3 CSRs and Raspberry Pi's running `splatform` and GaianDB

Once we finalized our implementation, we could now formally analyze how GaianDB would perform on a tactical network. To perform our experiments, we connected 3 Raspberry Pi's running GaianDB and our augmented version of `splatform` to a network of 3 CSRs. The Raspberry Pi is a low power, low cost, small form factor computer measuring 85.60 mm x 56 mm x 21 mm and weighing 45 g. The device runs a distribution of the Linux operating system based on Debian from a connected secure digital high capacity (SDHC) card or a universal serial bus (USB) device. The Raspberry Pi comes equipped with an Ethernet port, high-definition multimedia interface (HDMI) and general purpose input/output (GPIO) connections.[3] The combination of size, power requirements, capabilities, and cost make the Raspberry Pi a useful device for sensor experimentation. From there, we performed 3 types of benchmarks (burst, sustained, and extended) and measured total execution time (TET) as well as query success rate. We define TET as the time between when a query is sent and a response is received on a local node and query success rate (QSR) as the ratio of receiving all data from both remote nodes versus the total number of queries made on a local node.

In the burst benchmark, we sent queries simultaneously in a short period of time in a remote node. In the sustained benchmark, we sent queries in a serial fashion: once a remote node received a query and began to send back data, it would receive another one. In the extended benchmark, we would only send another request once data had been received plus a period of time of total inactivity. Across all benchmarks, we sent a total of 30 queries from 1 local node connected to 2 remote nodes.

Finally, we performed the same set of experiments in an emulated network in `CORE,` the common open research emulator developed by the US Navy Research Laboratory[3] with 3 virtual nodes running GaianDB with emulated wireless NICs. We compared the differences in TET between the emulated network and our CSR network to see if there would be a tangible impact on GaianDB performance when using IP encapsulation through RF. In our results (Table 1), we observed the following average TETs.

Table 1 Average TETs comparing the CSR and CORE networks

Test Type	CSR Network		CORE Network	
	TET	QSR	TET	QSR
Burst	7,000 ms	73%	900 ms	100%
Sustained	10,000 ms	85%	1,000 ms	100%
Extended	12,000 ms	90%	1,100 ms	100%

4. Conclusion

Initially, we had concerns that flooding the CSR network would have a significant impact on QSR, but we observed a negligible decrease in our results. GaianDB's ability to cache recent queries had a substantial impact on the QSRs across all tests in the CSR network; each test exhibited an average of <35% QSR in the first 6 queries. By the 15th query, the average QSR for each test rose to between 50% and 75%. It is worth mentioning that, in all tests, the last 12 queries made were 100% successful. On the subject of TET, while our results show an increase by a factor of 8 to 10, we can conclude that such latencies would not have a significant impact on a Soldier as delays are common in tactical networks due to their ad-hoc nature.

Thus, we can conclude from our results that there is a negligible impact on QSR and a reasonable delay in TET of the GaianDB when paired with the CSRs. While our emulated results were faster than our CSR network, the emulated nodes had hardware that far surpassed the low-end hardware found in the Raspberry Pi's; the emulated nodes had more than double the random access memory (RAM), double the number of central processing unit (CPU) cores, and double the CPU frequency. Even with the increased latency, we observed minimal query failure (when a query is made, yet no data are returned), which may suggest that formal integration between the GaianDB and CSRs is feasible and practical. However, this is only a preliminary indicator. More research is necessary to conclude if formal integration should be pursued. Currently, there are performance enhancements that can be done to improve our results:

- `splatform` does not support multicast protocols. We had to use a broadcast protocol, adding redundant traffic to the network, which may have had a negative impact on our results.
- Raspberry Pi does not support multi-queue flag options. When creating the TUN/TAP interface, a multi-queue flag can be specified to enable simultaneous reads and writes to the character device to increase throughput. Currently, the Raspberry Pi does not support this flag, so we had to serialize our reads and writes, which may have also had a negative impact on our results.

5. Current and Future Activities

In the future, we would like to improve our implantation by completing the performance enhancements listed above. Additionally, we would like to perform a finer granularity of experimentation concerning integration practicality and feasibility. We suggest performing experiments on throughput using `tcpstat`, increasing our network node count from 3 to 10+, measure query failure rates based on CSR topology, among other metrics. While our initial results seem promising, performing these experiments will give us a better indication if formal integration between GaianDB and CSRs may be of interest to the Army.

6. References

1. GAIAN Database – An overview [accessed 2015]. https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=f6ce657b-f385-43b2-8350-458e6e4a344f.
2. Universal TUN/TAP device driver [accessed 2015]. https://www.kernel.org/doc/Documentation/networking/tuntap.txt.
3. Raspberry Pi Computer [accessed 2015]. https://www.raspberrypi.org.
4. Common Open Research Emulator (CORE) [accessed 2015]. http://www.nrl.navy.mil/itd/ncs/products/core.

List of Symbols, Abbreviations, and Acronyms

CPU	central processing unit
CSR	common sensor radio
GaianDB	GAIAN database
GPIO	general purpose input/output
HDMI	high-definition multimedia interface
I/O	input/output
IGMP	Internet Group Management Protocol
NIC	network interface card
PPPoE	Point-to-Point Protocol over Ethernet
QSR	query success rate
RAM	random access memory
RDBMS	relational database management system
RF	radio frequency
SDHC	secure digital high capacity
SLQA	store locally query anywhere
SQL	structured query language
TAP	network tap
TCP	Transmission Control Protocol
TET	total execution time
TUN	network tunnel
UDP	User Datagram Protocol
USB	universal serial bus

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
2	DIRECTOR
(PDF)	US ARMY RESEARCH LAB
	RDRL CIO LL
	IMAL HRA MAIL & RECORDS
	MGMT
1	GOVT PRINTG OFC
(PDF)	A MALHOTRA
4	US ARMY RESEARCH LAB
(PDF)	RDRL CIN
	A KOTT
	RDRL CIN T
	R SHEATSLEY
	A TOTH
	B RIVERA