Creating a Management Suite for a Mobile, Wireless, Serverless Testbed

Wade Gobel

The structure of the modern internet formed so quickly and organically that it is highly probable that several aspects of its current organization can be improved and optimized. Research in the areas of computerized networks must therefore scrutinize the efficiency, practicality, and effectiveness of current standards instead of merely adding to them. One area that holds great potential for improvement lies in the physical structure of the internet itself.

The backbone of the modern internet consists of immobile wired connections (ethernet, telephone lines) and stationary servers at known locations. Servers are computers that host online media and direct requests for online data to other servers. As more and more people begin to use the internet, and as more resources become available online, the load on each individual server increases exponentially. The larger the capacity of the server, the more electricity it requires for running and cooling, and the greater amount of information that is lost if a server becomes inoperable. Although the use of servers is somewhat intuitive for networked communication, it is hardly necessary for communicating across the internet. With the advent and improvement of wireless internet technologies, the reliance on servers may prove to be completely unnecessary.

At the University of California, Santa Cruz, the wireless networks lab has created a wireless network that operates without the use of servers: the Santa Cruz mObile Radio Platform for Indoor / Outdoor Networks, or SCORPION. While this network is hardly ready to replace the modern internet, it possesses some key strengths, even as a prototype. First, with the removal of servers, the information available on SCORPION must be stored on the nodes themselves – that is, the computers requesting internet service must also provide it to one another. Second, each SCORPION node is only connected to the internet wirelessly, so any protocol used for sending and receiving data must be able to tolerate long delays in connectivity. Finally, every node can be moved about the UCSC campus, thereby eliminating the assumption that any node is always at a predefined location, and also providing connectivity to a wider physical range. So while the modern internet is for the most part stationary, wired, and server-dependent, SCORPION is mobile, wireless, and server-free. With this infrastructure in place, new internet protocols can be tested in real-world conditions, with nodes playing the roles of laptops, cell phones, or any other modern mobile wireless-capable computer.

Although the SCORPION nodes are intended as placeholders for laptops and cell phones, they possess hardly as much computational power or infrastructure as their counterparts. Many SCORPION nodes are little more than a Linux operating system installed on a motherboard with a wireless radio. As such, the nodes possess neither monitors nor keyboards, and must therefore be accessed and modified in other ways. Before the summer of 2009, the best method for communicating with any node was through a secure wireless connection (ssh). Establishing such a connection could be time-consuming as it required a user to enter a password. If a wireless connection was weak, then any communication could be significantly delayed, occasionally by several minutes. In addition, once a secure connection was established, it became difficult to tell if the connection had been terminated, or if the signal strength had merely deteriorated. To complicate matters further, the hardware on some of the nodes was sensitive to the point that the software would freeze and become unresponsive if the outer casing was jostled. Carrying out experiments with SCORPION nodes was therefore tricky. First, each node used in an experiment would have to be turned on in the lab. After establishing a connection with each node individually, all the nodes' cron job lists would have to be modified so that the appropriate test scripts would be executed in the near future. The nodes would then be transported to and set up at their physical destinations for the test, hopefully before the cron jobs began. At any point in this process, a node could have been jostled, causing it to become inoperable until restarted. Such a process of experimentation was difficult, fragile, and error-prone, and once on the field, it became outright impossible to test whether the experiment was running correctly. The solution to these problems was to create a management suite.

The purpose of a management suite is to simplify communication with the nodes in the SCORPION testbed, thereby facilitating experiments. The management suite provides useful functions for sending requests to and receiving data from the nodes, communicating with several nodes simultaneously, and all without a slow secure connection. Each node runs a simple program that listens for requests in a predefined, trusted format (on a predefined port), and whenever such a request is received, the node fulfills the request and replies with the results. A user sends formatted requests from a laptop or other wireless-enabled device, thereby setting up and conducting experiments in the field (instead of setting up the nodes one by one in the lab and then transporting them to the field). For example, one experiment was conducted by transporting all nodes to the field before turning any of them on. The management suite was then used to check that all nodes were running properly. Once everything was ready, the management suite was used once more to begin the experiment on all the nodes simultaneously. And during a second experiment that used a plane node, it became apparent halfway through the run that the test program had not started properly on the plane (this was checked remotely using management suite functions). The management suite was then used to remotely start the appropriate code without landing or turning off the plane. Although the experiment ran for a shorter time than expected, the management suite was able to completely eliminate the necessity of running the experiment a second time, despite the initial error.

The management suite written for SCORPION in the summer of 2009 offers four simple functions that are able to significantly improve the ease and speed of testing. These functions allow a tester to 1) Determine which nodes are running and responsive, 2) Obtain general statistics on running nodes, 3) View differences between local files (on the tester's laptop) and remote files (on the node), 4) Run command-line commands or scripts on the nodes, and 5) Overwrite the nodes' cron files. With just this functionality, a great deal of information and control becomes available remotely to the tester.

The four functions of the management suite were named nodels, nodediff, noderun, and nodecron.

Nodels (node-ell-ess) effectively pings all nodes by broadcasting a nodels request. Any nodes that receive a nodels request respond with the following information: the node's designated number (hostname), the node's kernel version, the amount of time the node has been running since it was last turned on, the node's latitude and longitude – if it was able to receive a GPS signal, or N/A if it was unable – the subversion revision number of the main source code directory, the percentage of the hard disk currently in use, and the percent usage of the CPU.

Nodediff is a simplification of the Linux "diff" command: given the path of a local file and a remote file, each node that receives a nodediff request checks whether its copy of the given file matches the requester's version of the file. The node then replies with one of "Same", "Different", or "File not found". Instead of sending the entire local file, nodediff simply sends the md5sum of the local file, along with the path of the remote file, so the node need only check the md5sums of the files to carry out the comparison.

The noderun command requires either a –c or a –s flag, which indicates whether the receiving node will be running a command-line command (–c) or a script provided by the user (–s). A node replies to a noderun –c request with the text outputted to stdout as a result of the command. If the script provided for noderun –s terminates with value 0, a node replies with "Success", or "Fail" otherwise.

Nodecron takes a file and sets each node's cron job list to the provided file. Similarly to noderun –s, if the cron job list is installed successfully, a node replies with "Success", or "Fail" otherwise.

In addition to the base functionality, each function takes at least one optional argument. The –t flag can be provided to any function in the management suite, along with a nonnegative integer or floating-point timeout in seconds. By default, the management suite waits 1.5 seconds after sending a request for all nodes to reply. Occasionally, functioning nodes take slightly longer than this to receive, process, and respond to a request. Therefore, having the option of extending the timeout can be valuable on the field.

Since noderun –s and nodecron send entire files to certain nodes, the connection that must be established with each receiving node needs to be more reliable than a connection that only receives a short string. Such a connection takes time to establish, and might not work the first time it is attempted. Therefore, noderun –s and nodecron each take an optional –r flag, along with a positive integer indicating the number of times to (re)try to establish a connection with the receiving nodes. Increasing the number of connection retries may delay the termination of the request, but also increases the chances of establishing a working connection. The default number of connection attempts is 1.

Finally, since not all nodes require the same modifications, the –n flag allows the user to specify a node list – i.e. a subset of the available nodes to send a particular request. Since nodels always broadcasts its request, it does not take the –n flag. Nodediff and noderun –c take the –n flag optionally, and since noderun –s and nodecron send potentially long files, the –n flag is required in order to minimize the time it takes to send the requests to the desired nodes. If a node list is not provided for a particular function, the request is broadcast, any node that receives the broadcast replies to it, and any responses received by the requesting machine are displayed. If a node list is provided, only the indicated nodes are sent a request, and after the timeout, either a node's reply is outputted, or the string "No response" is printed, indicating that a reply from that node was not received.

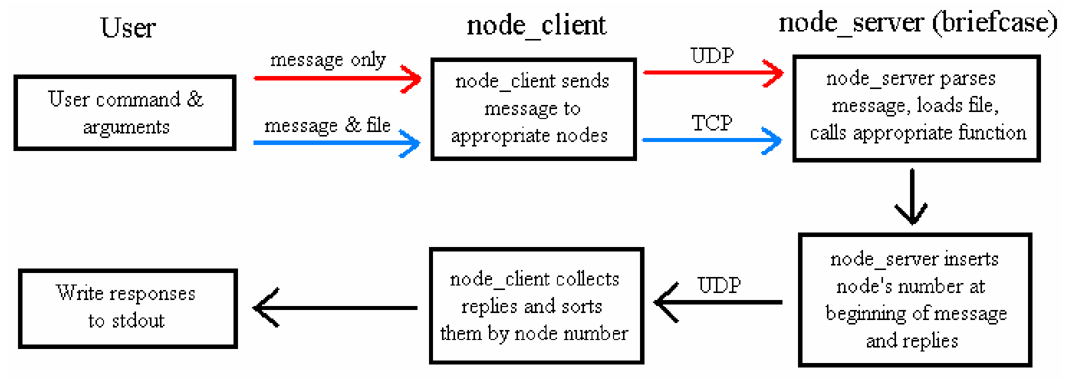
The program flow for the management suite follows a simple client-server pattern. First, the user enters the desired command and arguments at the command line. Then the indicated function parses the arguments and sends its request to node_client, which handles communication with the nodes. The program that runs in the background on the nodes is called node_server, and when it receives a request from node_client, it parses the request, calls the appropriate function, and sends back a string indicating the node's number and the result of the request. Node_client collects all such replies, sorts them by the responding node's number, and outputs them to standard output.

The current version of the SCORPION management suite only uses the protocols UDP and TCP.

UDP – the User Datagram Protocol – does not guarantee that a message will arrive at its destination, but if the message arrives, it is guaranteed to be error-free. In addition, if a UDP message is relatively short (less than 1K), it will probably be sent in a single packet. But if multiple UDP packets are sent, they are not guaranteed to arrive in order.

TCP – the Transmission Control Protocol – is more reliable than UDP but requires more time to work. If a TCP connection is established, then anything sent along this connection is guaranteed to reach its destination error-free and in the same order that it was sent (in the case of multiple packets). Since TCP is slower, it is only used when sending files to the nodes, where reliability and ordering are crucial. In fact, the –r flag indicates the number of times that node_client should attempt to establish a TCP connection with the receiving nodes.

If node_client is only sending a request, it uses UDP. If node_client needs to send a request and a file, it uses TCP. Since all replies are assumed to be relatively short, node_server only uses UDP for its responses. The program flow of the management suite therefore looks like this:


User
node_client
node_server (briefcase)
User command & arguments
message only
message & file
node_client sends message to appropriate nodes
UDP
TCP
node_server parses message, loads file, calls appropriate function
node_server inserts node's number at beginning of message and replies
UDP
node_client collects replies and sorts them by node number
Write responses to stdout

The request/response protocol used for node_client and node_server is relatively simple. For messages, node_client simply sends a whitespace-delimited string using UDP, where the first token is the command and all following tokens are command arguments. For example, the command 'nodels –t 5' would be broadcast as just 'nodels', since the nodes receiving the request do not need to know the timeout. The command 'noderun –c "uname -a"' would be sent as 'noderun c uname -a'.

For messages and files, node_client uses a slightly different format. The first four bytes of the TCP message are set aside as indicators of the lengths of the message and the file. The first byte indicates the message length (no message can be longer than $2^8 – 1 =$ 255 bytes), and the next three bytes indicate the file length (maximum file length is therefore $2^{24} – 1$ bytes). Next comes the message (i.e. the command and arguments), then the text of the file.

After calling the appropriate function with the given arguments or file, node_server responds to each request with a UDP message of its own. A reply message consists of the responding node's number (as text), followed by a single space, then the string outputted by the called function, up to the first newline character.

When node_client receives a reply, it strips the node's number from the string, and stores the command's result for later output. If the node's indicated number is not registered as a legal node number, the response is ignored.

Possible future work on the management suite includes a nodeupdate function, as well as inter-node request routing. Nodeupdate is a function that runs on each node for the first 15 seconds after boot up, listening for a laptop registered in the vicinity (the laptop broadcasts its name along with a specific identifying string). If a laptop is found, then the node attempts to rsync its main source code directory to the laptop's copy of the same directory. This effectively automates the nodes' synchronization without the need to run a subversion update command on each node individually.

And if a laptop is unable to send a management suite command to all desired nodes, one possible solution would be to use intermediary nodes to send the request farther. One way this could be accomplished would be by sending a list of the desired recipient nodes along with any management suite request. Any node that receives the request would then broadcast a copy of the request, perhaps with its own number removed from the list of intended recipients.

With this selection of simple, lightweight, yet powerful functions, the SCORPION management suite has been able to significantly simplify the testing process of a new, fundamentally different version of the internet.

Acknowledgements

National Science Foundation: Sponsor of SURF-IT 2009

Katia Obraczka: Faculty advisor

James Koshimoto: Graduate advisor