Auralization of Process and Port Status Using Program Binaries to Generate Semantically Meaningful Non-Fatiguing Noise as Canonical Sound Signals

Ronald Loui, *University of Illinois Springfield*
Dylan Dozier, *Lincoln Land College*
Evan Barber and Jeswanth Harish, *University of Illinois Springfield*
r.p.loui@gmail.com

Abstract

We describe using auralization to monitor process stack and netstat status on a server or small group of servers. Using the auditory channel is especially useful for monitoring because it can be done without taking much attention. Our contribution is to suggest generating noise-like sounds that more naturally fade into the background, and are less fatiguing over long monitoring sessions. Although our noises are not musically or tonally interesting, they are nevertheless semantically identifiable and succeed in indicating changes of state.

I. INTRODUCTION

Server monitoring by systems administrators is important for computer security, and usually requires well focused visual attention. For example, running a top or netstat repeatedly requires that the sysadmin interrupt other tasks and read the output. Various kinds of textual reports of change of state can reduce the frequency of interruption, but still require the operator to read the alert. The auditory channel, however, is usually available for simultaneous processing. Several authors have considered using active alerts and passive monitoring of state, in lieu of visual monitoring (see PRIOR WORK section). The obvious advantage is that a sysadmin can "keep an eye" on the state of the server constantly, without actually using one's eyes, i.e., without putting a load on the precious visual information channel. Auditory information is particularly well suited for simultaneous cognitive processing.

Although this is clearly a good idea, it has not found its way into standard practice. One problem is that tools that permit operators to generate sounds do not give much guidance on what sounds to generate. At best, this leaves the sysadmin with a problem of assigning "ring tones" to various events. At worst, it puts the sysadmin in the position of being a musical or soundscape composer. While some may be capable of making excellent assignments of musical scores to events, others may not, and it would be good to have a neutral, canonical way of generating sonic information for wide, default distribution of an auralizing monitoring package. It is as if SPAMASSASSIN had been distributed without any of its several hundred default regular expressions; probably few would have actually installed and used it, requiring days and weeks of development of their own filters. SPAMASSASSIN's success is due partly to the fact that it was distributed pre-programmed.

Moreover, one man's *Stairway to Heaven* musical alert is another man's *Copacabana*. For rare events like incoming phone calls, it may be acceptable to have an auralization that captures entertains, enthralls, startles, or captivates. However, the more "chatty" the dialogue, and the more often one must hear an event reported, the less welcome users find clever and delightful tonal musical signifiers. Rhythm seems less problematic in this respect than tone. Previous authors have talked about chirps and sleep-assisting soundscapes instead of playing sampled music or utterance (e.g., from a film or tv show). For constant passive monitoring, our experiments show that we need to take another step away from recognizable sounds in order not to drive the listeners to annoyance.

II. MAIN IDEA

Our idea is to solve both of these problems by using the information itself as the sound, and to do so in such a way that the resulting sounds would ordinarily be described as noise.

We began our study by using single-tone, well-spaced beeps to report the server load, and the server id. This required the operator to attend to the beeps, to count them and decide whether the count was interesting. We now know better, and can convert that beeping count into something less intrusive, like a warbling whistle. But at the time, we chose instead to report the top running process by playing the binary of the process itself. As a sonification, this binary is just noise. But it quickly becomes recognizable noise, and noise that easily fades into the background.

To implement this, a few modifications were required. First, short binaries had to be repeated in order to avoid sounding alike, and long binaries had to be truncated in order to avoid taking too much time. More interestingly, in order to have aural significance at a low volume against background noise (such as server fan noise), we found it useful to repeat the binary's prefix a few times, even if the entire binary was long enough that it did not have to be sampled. This repetition adds a rhythm to the noise that is less grating over time.

To monitor the top process on the server process stack, we first create a list of processes that are uninteresting, which we do not want to hear reported, the do-not-report list. These are daemons that run constantly, such as compiz, kthreadd, Xorg, init, dbus-demon, ibus-demon, httpd, etc. on a sample machine.

Here is a typical listing of top processes:

```
USER   VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
root   4448   2460   1456 S   0.0  0.1   0:01.53 init
root      0      0      0 S   0.0  0.0   0:00.01 kthreadd
root      0      0      0 S   0.0  0.0   0:01.40 ksoftirqd/0
root      0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
root      0      0      0 S   0.0  0.0   0:05.20 rcu_sched
root      0      0      0 S   0.0  0.0   0:00.00 rcu_bh
root      0      0      0 S   0.0  0.0   0:00.05 migration/0
root      0      0      0 S   0.0  0.0   0:00.18 watchdog/0
```

A monitoring process repeatedly looks down the list of processes using non-trivial cpu, finds the first that is not on the do-not-report list, e.g., diff, and reads from the binary, e.g., /usr/bin/diff. This binary has 447 lines and is 116k bytes, certainly long enough to be sent to the sound card as a complete sound. We play sounds with the padsp command:

```
padsp tee /dev/audio > /dev/null
```

by sending a file as input, e.g.,

```
cat /usr/bin/diff | padsp tee /dev/audio > /dev/null
```

But to create a more rhythmic and still noise-like sound, we take the first 5000 bytes, or the full length, eg. 3645 bytes of /usr/bin/diff, and we repeat it at least 3 times as needed until we reach 15kb (in this case, after the 5th iteration, 15kb has been achieved):

```
limit = 150000/topproclength

for (i=1; i<=limit; i++)
system("head -c 5000 " topproc " >> nextsound")

system("cat nextsound | padsp tee /dev/audio >
/dev/null")
```

This generates the kind of sound one might hear when interrupting an old dial-up modem connection. Depending on the binary, it may also contain the "R2D2" recognizable single tone noises that would be familiar during modem connection negotiation, but these tones tend to be sufficiently embedded in "modem noise" so as not to irritate.

Using anything shorter than 5000 bytes makes the binaries hard to differentiate aurally, even when repeated. 15000/5000 guarantees at least a 3x repetition of each binary prefix. Repetition is less jarring, permits less attention to be paid while still permitting alarm-type notification, and helps the sound to recede into the background.

Here are the sizes, by line, word, and byte count, of a few of the binaries in /usr/bin (almost all binaries are longer than 5000 bytes):

```
  2822    27690   716228 /usr/bin/cc
    14      255    17976 /usr/bin/cd-create-profile
    33      414    17768 /usr/bin/cd-fix-profile
    10       93     9576 /usr/bin/cd-iccdump
  3419    17881   395876 /usr/bin/cdrecord
    72      560    22136 /usr/bin/c++filt
    32      230     9764 /usr/bin/chacl
   200     1243    49420 /usr/bin/chage
   159      469     4459 /usr/bin/chardet
   123     1405    68272 /usr/bin/charmap
    16      156     9712 /usr/bin/chattr
   243     1722    59176 /usr/bin/chcon
```

Since sounds cannot be included in a paper, and our descriptions are bit subjective, we reproduce a part of a binary here, repeated three times:

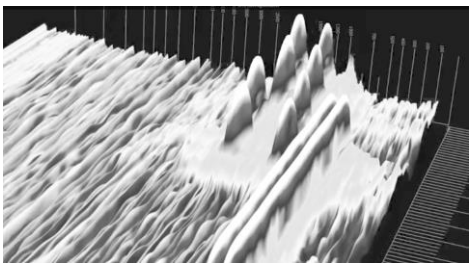
```
A#H D@#@:@@|@#A @#A#@#C #] #C A#A
K#AA#(D#$;@@2@#C #C#@#hA#(p#8;@@2@#C @#C#@#h
#L;@@@#@#A@#A#@#A#@F@
```

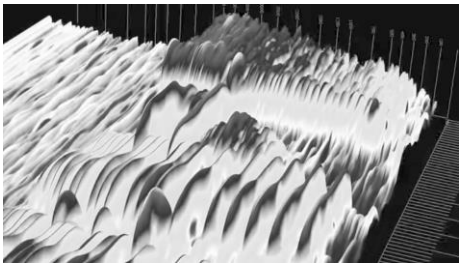
A#H D  #:|#A #A##C #] #C A#A
K#AA#(D#$;  2#C #C##hA#(p#8;2#C #C##h
#L;  ##A#A##A#F
A#H D  #:|#A #A##C #] #C A#A
K#AA#(D#$;  2#C #C##hA#(p#8;2#C #C##h
#L;  ##A#A##A#F

and compare it to a relatively pure tone:

9898989898989898989898989898989898989898989898989898989
8989898989898989898989898989898989898989898989898989898
9898989898989898989898989898989898989898989898989898989
8989898989898989898989898989898989898989898989898989898
9898989898989898989898989898989898989898989898989898989
8989898

which should give some visual indication of how much more complex it is. A youtube video of the power spectrum of modem noises shows the difference between pure tones (with power band limited to a small number of frequencies) and noise-like sounds (with power allocated to a range of frequencies):





To sum at this point, the main benefit of this idea is that it uses the binary to produce sounds, requiring no knowledge of the sound generating tool, no knowledge of music, no attachment or downloading of sounds, no commitment to a soundscape theme or musical style; and it adds a basic repetition idea to create palatable noise-like auditory signals that can fade into the background, but be recognized when not paying much attention.

III. WARBLED WHISTLING

In our rack of servers we want to monitor the status of more than one server at a time. This requires indicating which server is reporting with each auralized status. One way to do this is to assign an arbitrary noise signature to each server. Our solution is actually to use a tonally more pure signal, but to emit this sound after the status report. Again, we arrived at this solution after trying several different strategies for signaling.

Our indication of which server is reporting actually uses a count of beeps, but it embeds those beeps in a more noise-like rhythmic sound. The result is more like a brief whistle that contains a number of warbles. By adding vibrato, the sound does not require attention and does not cause much fatigue. It may be difficult to distinguish a 7-warble whistle from an 8-warble whistle at first, but after listening for a few hours, and hearing servers report in a fixed order (not necessarily enumerating in numerical order, 1, 2, 3, etc., but possibly in an order that makes juxtaposed reports more distinguishable, such as 1, 5, 2, 6, 3, 7, 4, 8), we find it easy to note which server is reporting, when one decides to listen.

The key idea here is that the counting does not require attention, but can transmit an id successfully to a person who decides to listen closely, to determine the count. At first we announced the id of the server before generating the sound associated with its status. This required too much attention. We preferred announcing the id of the server after auralizing the status. In this way, most status sounds and id counts could simply be ignored. When a status was interesting, the operator could then pay attention to which server was being described.

Here is an abridged text of the sound for id=2, so the reader may “visualize” it. Note that it is a mixing of pure tone, 99, and noise-like injection, a random real number (not just an integer).

9999999999

.4296891.790228.177651.77297.05551.744669999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999

The awk code for generating warbling whistle ids is simple:

```
 system("rm -f whoamisound")
 for (i=1; i<whoami+1; i++) {
   for (j=1; j<=beeplength; j++) print 99 > "whoami-
sound"
   for (j=1; j<=beeplength; j++) print 10*rand() >
"whoamisound"
 }
 close("whoamisound")
 system("cat whoamisound | padsp tee /dev/audio >
/dev/null")
```

This is a general strategy for embedding semantically interesting information in noisem, to decrease fatigue during monitoring.

IV. ASSIGNING BINARIES TO PORTS

To auralize port status, we use the tcp connections reported by netstat, a fragment of which might look like:

```
10.0.0.20:50029 199.16.156.201:443 ESTABLISHED
10.0.0.20:49722 199.59.148.11:443 ESTABLISHED
10.0.0.20:48640 91.189.92.10:443 CLOSE_WAIT
10.0.0.20:41137 104.73.160.113:80 ESTABLISHED
10.0.0.20:33543 91.189.92.24:443 CLOSE_WAIT
10.0.0.20:46051 91.189.92.11:443 CLOSE_WAIT
10.0.0.20:36159 184.50.238.89:80 ESTABLISHED
10.0.0.20:45428 23.79.220.138:80 ESTABLISHED
10.0.0.20:33554 91.189.92.24:443 CLOSE_WAIT
10.0.0.20:48630 91.189.92.10:443 CLOSE_WAIT
```

The interesting information is the foreign address in the middle column, the local port in the first column, and possibly the port status. However, playing this text in its current form results in a noise-like sound that is not very different from a netstat report showing an intrusion on an important port from an IP address that is unrecognized.

One of the practical problems is that we would like to auralize the DNS-resolved name of the foreign address, or at least the country to which the IP address is assigned. Unfortunately, this DNS-resolution usually takes too much time for real-time monitoring processes.

In this auralization, we make use of the observation that the listener naturally learns to recognize, and habituates, noise-like signatures, but these signatures need only be consistently associated with the semantics, not actually generated from a specific textual representation of the event. In plain words, we can assign an arbitrary noise-like sound to an event; we just have to maintain that assignment. (And the assignment need not even be one-to-one.)

To each port, we assign a randomly generated noise pattern. For our first experiments, it sufficed to take the prefixes of the binaries in /bin and /usr/bin. Port 80, for example, might be /bin/sh, and port 22 might be /bin/ls. Binaries were also arbitrarily assigned to the first two fields of the foreign IP addresses.

Because the netstat report can be long, we recommend reporting only the new entries, not the whole list each time. Also, when there is new activity there is often a lot of activity. This can make the reports too long, filled with repeat visitors. It may be good to maintain a whitelist of IP addresses that routinely access the server, and suppress reporting those entries.

The resulting sounds depend on the binaries assigned to the most common port numbers and IP prefixes. In our experience, a dozen or two netstat entries sound like distant machinery with slight variations. Novel access is perceived like anomalies in the machine; this can cause the operator to take note, usually by interrupting the task at hand, and visually attending to the server logs. This is exactly the kind of auralization we were attempting to provide.

V. PRIOR WORK

Auralization is a larger subject than its name might suggest (Kleiner 93, Vorlander 2007). LSL is an early tool for creating program auralizations (Boardman 93, 95, Khandelwal 95), for example, generating sounds that correspond to the states in a sorting algorithm. Much of this work is driven by authors with a musical background or interest (e.g., Vicker 2006).

Hermann and Hunt (2011) is a “sonification handbook” that would classify this work as “serendipitous-peripheral (push/nudge), where “attention is focused on a primary task whilst information that is useful but not required is presented on a peripheral display and is monitored indirectly.” (p. 456) They quote several authors (Tran 2000, Jenkins 1985, Brown 1996) who share our understanding of auditory channel monitoring advantages.

PEEP (Gilfix 2000) is the first paper we know to consider auralization for system security. Gopinath (2004) and Prasath (2004) explicitly considers monitoring a webserver and intrusion detection using a tool called

JLISTEN. Gopinath is more interested in events than sounds. Prasath's conclusions were that auralizations were helpful, and that musical knowledge had no effect on the ability to use auralization effectively. The library of sounds used by Prasath include: “Tinkle Bell, Sea Shore, Bird, Telephone, Music Box, Syn Bass 2, Fantasia, Whistle, Goblins, Piano, Slap Bass 2, Trumpet, Tubular Bell, Syn Bass 1, Sound Track, Woodblock, Helicopter.”

Kolano (2007) is recent NASA-supported work along similar lines, but uses PEEP for auralization: “bird chirps, cricket chirps, and water flows.” PEEP's water flows, which are used for reporting states and status, are probably chosen for many of the reasons that motivate our work.

Garcia-Ruiz et al. (2010) consider best practices, which is essentially what our paper aims at producing, but their work is aimed at teaching network intrusion and uses musical sounds.

Kennedy (1971), Childs (1976), and Galinsky et al. (1993) are examples of Human Factors research into fatigue and stress while doing auditory monitoring.

VI. FUTURE WORK

Future work includes parameterization of the auralization scripts to permit some customization. We aim to distribute a self-contained program to monitor the process stack, and another self-contained program to monitor the port activity. Our desire was to close the gap between auralization tools in theory, and ubiquitous use among sysadmins. Getting sysadmins to use the tool “right out of the box,” with no programming required, will require development and packaging.

As a broader audience is exposed to our noise-like, intentionally non-intrusive but semantically assignable sounds, data will become available for ascertaining the effectiveness of the ideas.

VII. CONCLUSION

The impetus for this work was the desire to produce practical linux server software that improves monitoring. It quickly became clear that visual monitors can be too dense, and the challenge was to get administrators to monitor *more often*, not necessarily monitor *more data*. Probing for details can be done once curiosity has been piqued, or an alarm has been sounded. The question was how to get sysadmins to monitor their machines when they are already busy doing other tasks that consume their attention.

One author happened to hear a RADIOLAB program on NPR where MIT brain researcher Matt Wilson describes listening to the “snap, crackle, and pop” of dreams; after enough listening, his lab could figure out what animals were dreaming about, recognizing the patterns. This led us to the auralization of server states using recognizable but fairly arbitrary noise-like sonifications, the potential for auralizing without fatigue, and the possibility that this would facilitate long-term monitoring.

Using the auditory channel for monitoring is too good an idea not to put into wide practical use. In conclusion, we recommend (1) assigning canonical sounds to important properties so users do not need to program their own sounds; (2) using binaries as those sounds to “play the data directly”; (3) truncating and repeating those binaries to create a more rhythmic sonification; (4) using binaries even when they are assigned arbitrarily as signals and sempahores, precisely because they are noise-like; (5) using tonal signals with numerical content on occasion, but intermixing and injecting noise, deliberately, to reduce the demand for attention and fatigue over prolonged “vigilant” listening.

REFERENCES

Boardman, David B., and Aditya P. Mathur. "Preliminary report on design rationale, syntax, and semantics of LSL: A specification language for program auralization." *Dept. of Computer Sciences, Purdue University, W. Lafayette, IN Sept* 21 (1993).

Boardman, David B., et al. "Listen: A tool to investigate the use of sound for the analysis of program behavior." *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*. IEEE, 1995.

Brown, John Seely, and Mark Weiser. "The coming age of calm technology."*http.//www. ubiq., com/hyp ertext/weiser/acnlI Lture2endnote. Htm* (1996).

Childs, Jerry M. "Signal complexity, response complexity, and signal specification in vigilance." *Human Factors: The Journal of the Human Factors and Ergonomics Society* 18.2 (1976): 149-160.

Garcia-Ruiz, Miguel A., et al. "Best Practices for Applying Sonification to Support Teaching and Learning of Network Intrusion Detection." *World Conference on Educational Multimedia, Hypermedia and Telecommunications*. Vol. 2010. No. 1. 2010.

Gilfix, Michael, and Alva L. Couch. "Peep (The Network Auralizer): Monitoring Your Network with Sound." *LISA*. 2000.

Gopinath, M. C. "Auralization of intrusion detection system using Jlisten."*Development* 22 (2004).

Hermann, Thomas, and Andy Hunt. *The sonification handbook*. Berlin: Logos Verlag, 2011.

James J. Jenkins. Acoustic information for object, places, and events. In W. H. Warren, editor, *Persistence and Change: Proceedings of the First International Conference on Event Perception*, pages 115–138. Lawrence Erlbaum, Hillsdale, NJ, 1985.

Galinsky, Traci L., et al. "Psychophysical determinants of stress in sustained attention." *Human Factors: The Journal of the Human Factors and Ergonomics Society* 35.4 (1993): 603-614.

Khandelwal, Vivek. *On program auralization*. Diss. Purdue University, 1995.

Kleiner, Mendel, Bengt-Inge Dalenbäck, and Peter Svensson. "Auralization-an overview." *Journal of the Audio Engineering Society* 41.11 (1993): 861-875.

Kolano, Paul Z. "A scalable aural-visual environment for security event monitoring, analysis, and response." *Advances in Visual Computing*. Springer Berlin Heidelberg, 2007. 564-575.

McGregor, Colin. "Controlling spam with SpamAssassin." *Linux J* 153.1 (2007).

Prasath, R. Jagadish. *Auralization of web server using jlisten*. Diss. Master's thesis, Purdue University, BITS, Pilani, India, 2004.

Tran, Quan T., and Elizabeth D. Mynatt. "Music monitor: Ambient musical data for the home." *Extended Proceedings of the HOIT* (2000): 85-92.

Vickers, Paul, and James L. Alty. *The well-tempered compiler: The aesthetics of program auralization*. MIT Press, Boston, MA, 2006.

Vorländer, Michael. *Auralization: fundamentals of acoustics, modelling, simulation, algorithms and acoustic virtual reality*. Springer Science & Business Media, 2007.