;login:
THE MAGAZINE OF USENIX & SAGE
April 2001 • volume 26 • number 2

inside:

SYSADMIN

Introducing Peep: The Network Auralizer

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

introducing peep: the network auralizer

by Michael Gilfix

Michael Gilfix is completing his degrees in electrical engineering and computer science at Tufts University.

<mgilfix@eecs.tufts.edu>

Peep is a tool which provides an alternative to current network monitoring solutions. Peep creates an audio representation of network activity using natural sounds, providing large amounts of information in a compact, non-intrusive, and perhaps soothing form. Use of Peep to monitor your network is based on the concept of normalcy, where your network is functioning correctly if Peep "sounds right."

Why Peep?

Even though a good portion of our job as system administrators is to be as knowledgeable as possible about the state of our network at any given instant, it is difficult to use current approaches to live monitoring while completing other tasks. In general, one must intermittently suspend other work to check the monitor, which has profound negative consequences for accomplishing work efficiently, or perhaps wait (or pray?) for an email or page when something goes wrong.

These approaches are highly problem-centered and provide mainly negative reinforcement. Most tools are very limited in the domain they can address and report only when they discover a problem. In the meantime, the administrator is left in the dark, hoping the tool will do its job when the crucial time comes. These monitors do not regularly inform the administrator when the network is functioning well.

Peep's approach is to generate a realtime sonic representation of network state using natural sounds. Peep's audio output is meant to play in the background as a sort of white noise. With Peep, an administrator can keep tabs on how well the network is performing and what sort of activity is occurring at all times, without interrupting other work. In addition, Peep leaves all the subtleties of interpretation to the listener, thereby avoiding limitations on the kind of problem domain addressed.

Using audio to interface with the user brings several major benefits: it allows us to exploit our human instinct to notice deviations with little effort, to determine what "sounds right," and to discern singular important sounds from a collection of many sounds. These abilities are exercised continuously, with little or no conscious effort. Since computer interfaces mainly require the visual senses, using the audio senses to perform this unconscious processing does not interfere with our ability to do other work.

An audio interface also allows us to take advantage of our ability to do abstract processing. Instead of attempting the difficult and sensitive problem of determining when a network crisis has occurred or is about to occur, Peep provides contextual, continuous sound information and leaves interpretation to the listener. Decisions are then based not only on the quantitative measure of things, but also the relative amount and absence of things.

Yeah. But Won't That Get Annoying?

Nope. Most people envision audio tools as spewing some sequence of beeps or sounds that eventually annoy the listener and do more harm than good. Past approaches to audio monitoring have made use of beeps, midi, and singular sound samples to alert the user. These approaches, however, are greatly limited by what they can represent and still sound pleasing to the ear. For music to remain pleasant, one must limit one's represen-

tations to a limited number of relatively pleasing harmonic combinations. Singular and overly striking sounds are another dead-end, since we cannot tolerate these sounds in large measures (think ICQ!).

Natural sounds have an advantage over these approaches because they seem to "occur together" and "sound right" in virtually any combination. For example, birds, when singing in nature, have no coordination and yet they are still agreeable to listen to. Natural sounds also offer another advantage over conventional sounds – they have personality. We can map natural sounds to network events in the manner in which we think of them, making our representations more expressive.

Audio Representation

Since audio representation is such an important part of making an audio tool successful, Professor Alva Couch and I spent a substantial amount of time considering how to best represent network occurrences. Sound representation in Peep is divided into three basic categories: events in networks are things that occur once, naturally represented by a single peep or chirp; network states, or ongoing events, are represented by changing the type, volume, or stereo position of an ongoing background sound; while heartbeats represent the existence or frequency of occurrence of an ongoing network state by playing a sound at varying intervals, such as by changing the frequency of cricket chirps.

Peep represents discrete events by playing a single natural sound every time the event occurs, such as a bird chirp or a woodpecker's peck. These sounds are staccato in nature and easily distinguishable by the listener. We noted that certain events tend to occur together and found it convenient to assign them complementary sounds. While monitoring incoming and outgoing email on our network, we perceived that the two events were often grouped together, since both types of email were usually transferred in a single session between mail servers. To better represent this coupling between incoming and outgoing email events and make the representation sound more natural, we used the sounds of two conversing birds. Thus, a flood of incoming and outgoing email sounds like a sequence of call and response, making the sound "imagery" both more faithful to our network's behavior, as well as more pleasing to the ear.

State sounds correspond to measurements or weights describing the magnitude of something, such as the load average or the number of users on a given machine. Unlike events, which are only played when Peep is notified of them, Peep plays state information constantly and need only be signaled when state sounds should change. Peep represents a state with a continuous stream of background sounds, like a waterfall or wind. Each state is internally identified as a single number measurement, scaled to vary from extremely quiet to loud and obnoxious. The idea is that background sounds should be soothing while the network is functioning normally and annoying when an administrator should be inspired into action.

The idea is that background sounds should be soothing while the network is functioning normally and annoying when an administrator should be inspired into action.

Heartbeats are sounds that occur at constant intervals, analogous to crickets chirping at night. A common folk tale is that one can tell the temperature from the frequency of cricket chirps; likewise we can represent network load as a similar function. Intermittent chirps might mean low load, while a chorus might mean high load. Heartbeats can also report results of an intermittent check (or ping) to see if a given machine, device, or server is functioning properly.

Peep Architecture

Peep is based on a producer/consumer architecture where many client producers around the network gather information and send it to a few, centralized server consumers for playback. Configuration information is stored in a single configuration file

Perhaps the most difficult part of this process is choosing which sounds you prefer.

that is replicated for clients and servers around the network. Clients alert servers to network events via short UDP messages, keeping overhead to a minimum. This architecture allows for status reports from any number of network devices or nodes, as well as a great deal of flexibility when structuring a given network implementation.

To ease the management of Peep's distributed architecture, Peep uses a mechanism called auto-discovery and leasing. Upon startup, each client and server broadcasts its existence to the network once and the appropriate peers automatically "discover" each other. Because of the statelessness of UDP, a mechanism is required to ensure that clients do not waste network resources by sending packets to non-functional servers. This is accomplished through leasing. During their initial communication, the server and client exchange a lease time. Then, at intervals before the lease expires, the client checks in with the server and renews the lease. An expired lease indicates that a server is no longer functioning, and thus the client ceases sending its network information.

Peep's auto-discovery and leasing mechanism uses a domain-class concept to group clients and servers together. This class information is specified in the single configuration file shared by servers and clients. During the startup, each server or client broadcasts only to the subnets designated by its respective classes, and announces to those subnets which classes it belongs to. Following the initial broadcast, a list of hosts is maintained on both sides and all further communications are direct. Both clients and servers can belong to multiple classes at the same time, and clients can communicate with many servers concurrently.

Getting Peep Up and Running in Your Network

Integrating Peep with your network is a four-step process: download the source code and sound repository package from the Sourceforge site, build the server, configure your server and choose your sounds, and deploy the clients throughout your network.

The process of configuring servers and clients is relatively easy. An example configuration file comes with the Peep distribution that requires only a little modification to get Peep up and running within your network. Ample documentation is also provided with the Peep distribution in HTML format or can be obtained at <*http://peep.sourceforge.net/docs/peep-doc.html*>.

Perhaps the most difficult part of this process is choosing which sounds you prefer. KeyTest, a utility provided with Peep, is used for this exact purpose. KeyTest maps different keys to different events and states, and each keystroke plays the corresponding events or state on the server. This allows the user to experiment with changing stereo location, state volumes, and event priorities. My suggestion is to load all the sounds into the server and literally "play" the server to see how they all might sound together. KeyTest will support up to 24 different event sounds and 10 different state sounds at a given time. As this process can provide much amusement, it tends to be the lengthier part of Peep's setup time.

The last step is to deploy the clients provided with Peep. Currently, two clients are provided with the Peep distribution: Uptime and LogParser. Uptime reads state information from the UNIX utility "uptime," as its name would suggest, and reports a scaled measurement of the machine load and number of users to the server. LogParser, a real-time log parsing utility similar to Swatch, scans logs as data is appended and performs regular-expression pattern matching to extract event data. LogParser is a rather flexible tool, and the patterns it matches can be entirely customized in the Peep configuration file.

In addition, if the utilities provided with Peep do not meet your needs, all of the auto-discovery and leasing part of the Peep protocol has been encapsulated nicely within two Perl libraries provided with the Peep distribution. Example code exists in the documentation to help you write your own utilities quickly and efficiently.

Some Known Problems and Where We Are Going

One of the biggest problems with Peep is training your ear to recognize the intricacies of different network occurrences and make a complex diagnosis. This post appeared on slashdot shortly after the LISA 2000 conference:

> Since I'm on call, I'm looking forward to my first conversation with a monitoring guy after this is in place . . .
> MG: "Yeah, there's a problem with system XYZ . . ."
> Me: "How so?"
> MG: "Well, usually it goes 'ree-ree-tinktinktinktink,' you know? But right now it's going 'ree-ree-tinktink-bong-bong-tink'!"
> Me: "Is that 'bong' like a doorbell chime, or more like a big Chinese gong?"
> MG: "In between but more like a gong, I think."
> Me: "Well, shit."
> By nakaduct on slashdot – mike.muise@digital.com

But this example is part and parcel of dealing with inexperienced users of any system. Any system requires a user to become trained and "conditioned" to its proper use before attaining maximum benefit. In general, user feedback has been positive, and admins have reported that they have detected a large range of different behavior using Peep, most notably email spam. So, although training your ear may be of some concern, it has not been a problem for past users.

We are working on adding a recording feature to Peep that will save past events in a playback file for review. Currently, if you think you have heard some sort of anomaly, there is no way to go back and reevaluate the sound. A playback feature would allow admins to trade playback files amongst themselves to get second opinions.

We are also exploring visual playback methods that will provide visuals to network events similar to what a graphic equalizer does for sound. This will allow for a visual analysis similar to how Peep works, where a network is functioning correctly if the graphic just "looks right."

Summary

Whether we know it or not, we all have the ability to utilize our "peripheral" senses in doing our day-to-day work. Too many of us can instantly recognize the sound of a bad fan or a hard disk crash. We did not consciously study this or take a course in it. We learned it because these sounds form an integral part of our daily work environment. If we can add Peep to this environment, it is only a matter of time until we react to a cheerful chirp with the sure knowledge that our servers are working and that we can rest easy.

Obtaining Peep

Peep is freely available at <*http://www.sourceforge.net/projects/peep/*>. If you want to find out what this tool might sound like, there is an mp3 demo available on the home page under the introduction section.

"Well, usually it goes 'ree-ree-tinktinktinktink,' you know? But right now it's going 'ree-ree-tinktink-bong-bong-tink'!"