Sonification of a Network's Self-Organized Criticality for Real-time Situational Awareness

Paul Vickers[a,*], Chris Laing[b,1], Tom Fairfax[c]

[a]*Northumbria University, Newcastle upon Tyne NE1 8ST, United Kingdom*
[b]*Sciendum Ltd, 20-22 Wenlock Road, London, N1 7GU, UK*
[c]*SRM Solutions, The Grainger Suite, Dobson House, Regent Centre, Gosforth, Newcastle upon Tyne, NE3 3PF, UK*

Abstract

Communication networks involve the transmission and reception of large volumes of data. Research indicates that network traffic volumes will continue to increase. These traffic volumes will be unprecedented and the behaviour of global information infrastructures when dealing with these data volumes is unknown. It has been shown that complex systems (including computer networks) exhibit self-organized criticality under certain conditions. Given the possibility in such systems of a sudden and spontaneous system reset the development of techniques to inform system administrators of this behavior could be beneficial. This article focuses on the combination of two dissimilar research concepts, namely sonification (a form of auditory display) and self-organized criticality (SOC). A system is described that sonifies in real time an information infrastructure's self-organized criticality to alert the network administrators of both normal and abnormal network traffic and operation. It is shown how the system makes changes in a system's SOC readily perceptible. Implications for how such a system may support real-time situational awareness and post-hoc incident analysis are discussed.

Keywords: Auditory Display, Sonification, Information Visualization, Self-Organized Criticality, Network Monitoring

1. Introduction

With the large volumes of traffic passing across networks it is important to know about the state of the various components involved (servers, routers, switches, firewalls, computers, network-attached storage devices, etc.) and the types and volume of the data traffic passing through the network. In the case of the hardware, network administrators need to know if a component has failed or is approaching some capacity threshold (e.g., a server has crashed, a hard drive has become full, etc.) so that appropriate action can be taken. Likewise, the administrators need to be aware of traffic type and flow. For example, a large increase in traffic volume (perhaps as would occur if the network were to broadcast a live stream of a major sporting event) might require extra servers to be brought online to handle and balance the load. A sudden increase in certain types of traffic (such as small UDP packets) might indicate that a distributed denial-of-service attack is in progress, for example, and corrective action would need to be taken to protect the network.[2]

Given the large volume of traffic passing through a network every second in the form of data packets and the fact that each packet will be associated with particular sender and receiver IP addresses and port numbers, understanding what is happening to a network requires information about the traffic data to be aggregated and presented to the network administrator in an easy-to-understand way. This problem of information presentation and interpretation, or 'situational awareness', was addressed by the military leading to Boyd's OODA (observe, orient, decide, act) model (see [1]),

*Corresponding author
Email addresses: paul.vickers@northumbria.ac.uk (Paul Vickers), christopher.laing@sciendum.org.uk (Chris Laing), tom.fairfax@srm-solutions.com (Tom Fairfax)

[1]This work was done while Chris Laing was at Northumbria but he is now at sciendum.org.uk.

[2]UDP, or user datagram protocol, is a way of sending internet packets without handshaking. It means that packets can be lost, but in some real-time systems (e.g., online gaming) it is preferable to lose a packet than to wait for a delayed one.

and others have followed (notably Endsley's three-level model [2]). Situational awareness, as Cook put it, "requires that various pieces of information be connected in space and time" (Nancy Cooke in McNeese [3]).

Computer networks possess high tempo and granularity but with low visibilty and tangibility. Administrators rely on complex data feeds which typically need translatation into language that can be understood by decision makers. Each layer of analytical tools that is added can increase the margin for error as well as adding Clausewitzian friction (see von Clausewitz's '*On War*', 1873). Furthermore, it is practically impossible for most administrators to watch complex visual data feeds concurrently with other activity without quickly losing effectiveness [4].

In military circles there is debate about whether cyberspace has become the fifth warfighting domain (the others being sea, land, air, and space) [4]. Computer networks are increasingly coming under strain both from adversarial attacks (warfighting in military parlance) and from load and traffic pressures (e.g., increased demand on web services).

Another term that has made its way from the military lexicon into the wider world of network administration is situational awareness. Endsley[2, p. 36] defined situational awareness (SA) as the "perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future". So, SA facilitates an administrator in becoming aware of a network's current state. The perception phase of SA comprises the recognition of situational events and their subsequent identification. Sonification is a process of computational perceptualisation which Vickers [5] suggested is well suited to the monitoring of time-dependent processes and phenomena such as computer networks.

Fairfax *et al*. [4] noted that the cyber environment is increasingly being viewed as the fifth warfighting domain (alongside land, sea, air, and space). They stated the challenge for maintaining situational awareness in the cyber environment as:

> ... whilst land, sea, air and space are physically distinct and are defined by similar criteria, cyberspace is defined in a different way, existing on an electronic plane rather than a physical and chemical one. Some would argue that cyber space is a vein which runs through the other four warfighting domains and exists as a common component rather than as a discrete domain. One can easily see how cyber operations can easily play a significant role in land, sea, air or space warfare, due to the technology employed in each of these domains [4, p. 335].

Thus, in this environment where human perception is constrained, adversaries and protagonists alike are dependent on tools for their perception and understanding of what is going on. Many tools on which we rely for situational awareness are focused on specific detail. The peripheral vision (based on a range of senses) on which our instinctive threat models are based is very narrow when canalised by the tools we use to monitor the network environment. The majority of these tools use primarily visual cues (with the exception of alarms) to communicate situational awareness to operators. Put simply, situational awareness is the means by which protagonists in a particular environment perceive what is going on around them (including hostile, friendly, and environmental events), and understand the implications of these events in sufficient time to take appropriate action.

When network incidents occur experience shows that the speed and accuracy of the initial response are critical to a successful resolution of the situation. Operators observe the indicators, orient themselves and their sensors to understand the problem, decide on the action to be taken, and act in a timely and decisive way. Traditional approaches to monitoring can hinder this by not making the initial indication and its context clear thus requiring an extensive orientation stage. An ineffective initial response is consistently seen to be one of the hardest things for people to get right in practice [4]. D'Amico (see McNeese [3]) put the challenge of designing visualizations for situational awareness this way:

> ... visualization designers must focus on the specific role of the target user, and the stage of situational awareness the visualizations are intended to support: perception, comprehension, or projection.

While work has been carried out to use information visualization techniques on network data we note that the *perceive* and *comprehend* stages in Endsley's three-level situational awareness model (the third being *project*) [2] align themselves with Pierre Schaeffer's two fundamental modes of musical listening, *écouter* (hearing, the auditory equivalent of perception) and *entendre* (literally 'understanding', the equivalent of comprehension). Vickers [6] demonstrated how Schaeffer's musical context can be applied sonification. This paper proposes a sonification tool as one of the means by

which real-time situational awareness in network environments may be facilitated. A more detailed discussion of situational awareness and its relationship to network monitoring (specifically within a cybersecurity and warfighting context) can be found in Fairfax *et al*. [4].

1.1. Sonification for Network Monitoring

Sonification has been applied to many different types of data analysis (for a recent and broad coverage see *The Sonification Handbook* [7]). One task for which it seems particularly well suited is live monitoring, as would be required in situational awareness applications [5]. The approach described in this article provides one way of addressing the challenges outlined above by enabling operators to monitor networks concurrently with other tasks using additional senses. This has the potential to increase operators' available bandwidth without overloading individual cognitive functions, and could provide an immediate and elegant route to practical situational awareness.

It has been suggested that understanding the patterns of network traffic is essential to the analysis of a network's survivability [8]. Typically, analysis takes place post-hoc through an inspection of log files to determine what caused a crash or other network event. Lessons would be learned and counter measures put in place to prevent a re-occurrence.

For the purpose of keeping a network running smoothly load balancing can sometimes be achieved automatically by the network itself, or alerts can be posted to trigger a manual response by the network administrators. Guo *et al*. [8] observed that "from the perspective of traffic engineering, understanding the network traffic pattern is essential" for the analysis of network survivability.

Often, the first the administrators know about a problem on a network is after an attack, or other destabilizing event, has taken place or the network has crashed. Here, the traffic logs would be examined to identify the causes and steps would be taken to try to protect against the same events in future. Live monitoring of network traffic assists with situational awareness and could provide administrators either with advanced warning of an impending threat or with real-time intelligence on network threatening events in action.[3]

Real-time network monitoring offers a challenge in that, except for alarms for discrete events, the administrator must be looking at a console screen to observe what is happening. To identify changes in traffic flow would this require attention to be devoted to the console [4]. Vickers [5, p. 455] categorised monitoring tasks as direct, peripheral, or serendipitous-peripheral:

> In a direct monitoring task we are directly engaged with the system being monitored and our attention is focused on the system as we take note of its state. In a peripheral monitoring task, our primary focus is elsewhere, our attention being diverted to the monitored system either on our own volition at intervals by scanning the system ... or through being interrupted by an exceptional event signalled by the system itself.

Serendipitous-peripheral is similar to peripheral monitoring except that it uses what Mynatt *et al.* [9] term "serendipitous information", that is, the information gained "is useful and appreciated but not strictly required or vital either to the task in hand or the overall goal" [5, p. 456].

Thus, a system to sonify network traffic may allow us to monitor the network in a peripheral mode, the monitoring becoming a secondary task for the operator who can carry on with some other primary activity. Network traffic is a prime candidate for sonification as it comprises series of temporally-related data which may be mapped naturally to sound, a temporal medium [5].

Gilfix and Crouch's Peep system [10] is an early network sonification example. They used natural sounds to represent network states and events and hoped that repeated listening would enable users to build up an understanding of what normal operation of their network sounds like. The system was offered very much as a proof-of-concept and no specific guidance was given on particular ways in which Peep could be used.

Kimoto and Ohno [11] developed a network sonification system called Stetho which uses HTTP traffic data to generate MIDI events which are in turn rendered into sound by MIDI-compatible sound synthesis software.[4] An experiment showed that four participants who used the system for five minutes to identify peaks in HTTP traffic. Kimoto and Ohno concluded that the system was suitable to grasp "traffic vaguely", so like Peep there was

[3] By threat, we do not only mean a hacking/DDOS attack, but also include 'natural' disasters such as component failures, legitimate traffic surges, etc.

[4] MIDI (musical instrument digital interface) is a set of software and hardware protocols developed by leading synthesizer manufacturers in the 1980s to allow interoperability between previously incompatible devices.

a lack of a sense of real use cases that STETHO might support.

Ballora *et al.* [12, 13, 14] built on these ideas to address the particular case of situational awareness. Rather than use environmental sounds, Ballora *et al.* used synthesized musical instruments to represent network data as pitched tones. Using an auditory model of the network packet space they produced a "nuanced soundscape in which unexpected patterns can emerge for experienced listeners". Their approach used the five-level JDL fusion model which is concerned with integrating multiple data streams such that situational awareness is enhanced (see Blasch and Plano [15]). Rather than focus on simple bytes and packets coming in and leaving the network, their system allowed differentiation between the geographic origin of packets (via IP addresses), and the nature of the traffic (via port numbers). However, Ballora *et al*. [12] noted that the high data speeds and volumes associated with computer networks can lead to unmanageable cognitive loads. Endsley and Connor (in McNeese [3]) came to the same conclusion, stating that the "extreme volume of data and the speed at which that data flows rapidly exceeds human cognitive limits and capabilities." They concluded:

> The combination of the text-based format commonly used in cyber security systems coupled with the high false alert rates can lead to analysts being overwhelmed and unable to ferret out real intrusions and attacks from the deluge of information. The Level 5 fusion process indicates that the HCI interface should provide access to and human control at each level of the fusion process, but the question is how to do so without overwhelming the analyst with the details.

Like STETHO, Giot and Courbe's INTENTION (Interactive Network Sonification) system mapped network activity to a musical aesthetic via MIDI [16]. Four sound channels were implemented. The first three processed HTTP, FTP, and DNS traffic respectively, while the fourth channel dealt with traffic from all other protocols together. The system mapped several details of traffic properties to the parameters of the output sounds. For instance, packet size controlled the frequency of a tone while the TTL (time to live) of a datagram controlled the duration of the tone. Geographic distance (estimated from IP addresses) controlled the amount of reverberation applied to the tone. Unfortunately, no target use case was stated and no description or demonstration of the system was provided. It remains to be determined how effective this deliberate approach to consider musical aesthetics was.

Wolf and Fiebrink [17] designed the SONNET system to help users (artists or people have an interest in network traffic information) to easily access network traffic through a simple coding interface without requiring knowledge of Internet protocols. The system used three levels of abstraction dealing with raw packet data, temporal aspects and directionality of traffic (via source and destination IP addresses, port numbers, and time since the last packet, and aggregated information over multiple packets (via packet state and flags) respectively.

The system's default operation is to process TCP packets on port 80 (i.e., HTTP traffic), though users can select to monitor UDP traffic and traffic on all network ports if they wish. The sonification itself was left to the user to specify by writing a script to control a ChucK module.[5]

The system was evaluated with four composers and students of music composition. The objective was to discover whether SONNET would support composers in creating a musical piece. Therefore, the target use case is quite different from the systems mentioned above which were more concerned with assisting with the monitoring of a network.

Worrall's NETSON project [18] is a network sonification tool that aims to "sonically reveal aspects of the temporal structure of computer network data flows in a relatively large-scale organization". The system began as an exploratory tool for an art and technology event and includes visualizations alongside the auditory output and aims to assist people with the peripheral monitoring of a network. The sonification design is not explained in detail, but it is based on using the features of raw traffic data to control various aspects of the output sound. The overall design is explained thus: "in contradistinction to much parameter mapping sonification, 'melodic' pitch structures are used very sparingly in favour of a diverse *klangfarben* (timbral) palette."

One particular configuration of the system is described as revealing "a combination of interesting features (such as printer server activity) and load-balancing" Worrall [18]. However, in its present version NETSON is presented as a sonification for public spaces so further work is necessary to see how well it supports specific network monitoring tasks and goals.

As seen in the work mentioned above, network sonification typically approaches the task by representing

[5] ChucK is a concurrent music programming language that can be used to generate audio (see `http://chuck.cs.princeton.edu`).

the raw traffic data (packets) or aggregated information about those packets. To address managing the complexity we propose that the study of self-organized criticality has the potential to provide a way of aggregating network behaviour and presenting the 'health' of the network as a simple variable, or set of related variables.

2. Self Organized Criticality in Network Traffic

The 20th century witnessed a number of advances in our understanding of complexity in dynamical systems. In 1987 Bak, Tang, and Wiesenfeld [19] brought together the concept of emergent complexity in simple systems, the mathematics describing the complexity of fractals in natural systems, and the scale-invariant power laws, fractal geometries, and the pink $(1/f)$ noise observed at the critical points between phase transitions in physical systems in a single explanatory model they termed *self-organized criticality*, or SOC.

They showed that these factors could be observed in a cellular automaton and that they were linked to critical point phenomena. While critical point phenomena are typically associated with the phase transitions of thermodynamical systems (e.g., when a liquid transitions to the vapor phase), SOC could be observed in a range of natural systems. SOC accounts for the emergence of complexity in a way that does not depend on the way a system is configured internally. That is, parameters of a system could be manipulated without affecting the emergence of SOC (the scale-invariant power laws, fractal geometries, $1/f$ noise are all still observed), hence the criticality was self-organized, not being dependent on external influences.

The classic example Bak, Tang, and Wiesenfeld offered was the sandpile model. Avalanches in the sandpile (critical points) happen as a result of grains of sand being sprinkled onto the pile. It is the sandpile organizing itself that leads to an avalanche (a system reset). SOC is a function of an external driving force and and internal relaxation process with a separation of timescales between them [4]. In the case of the sandpile the external driving force is the addition of sand grains and the internal relaxation process is the avalanche. The avalanche can take seconds to happen yet the external driving force can operate over a longer timescale (minutes or hours). Since then, SOC has been demonstrated in other natural systems such as earthquakes (in which the relaxation process can take seconds compared to the years or decades involved in the external driving force) and forest fires and has subsequently been observed in artificial systems such as stock markets and, latterly, computer networks.

The separation of timescales also comprises two other essential elements: thresholds and metastability [20]. Since the time taken before an internal relaxation process occurs is non-deterministic, so is the threshold at which the internal relaxation process occurs. Thus, a system can exhibit many many differing states, each of which is 'barely stable', a condition called metastability [19].

Modern computer networks demonstrate periods of very high activity alternating with periods of relative calm, a characteristic known as 'burstiness' [21]. It was commonly thought that ethernet traffic conformed to Poisson or Markovian distributions. Traffic would thus possess a characteristic burst length which would be smooth when averaged over a timescale [22]. However, network traffic has been shown to have significant variance or burstiness over a range of timescales. Such traffic can be described using the statistical concept of self-similarity and it has been established that ethernet traffic exhibits this property [23].

In a wavelet analysis of the burstiness of self-similar computer network traffic Yang *et al*. [24] demonstrated that the avalanche volume, duration time, and the inter-event time of traffic flow fluctuations obey power law distributions. According to Bak *et al*. [19] such power law distributions in complex systems are evidence of SOC. Fukuda *et al*. [25] demonstrated the existence of phase transition phenomena in network traffic and Valverde and Solé [23] showed how network traffic exhibits the critical states associated with SOC.

Yang *et al*. [24] suggested that SOC might be a better explanation of network traffic than traditional Poisson models. They argue that the power laws evident in inter-event timings indicates that network traffic "exhibits long-term memory (its behavior across widely separated times is correlated)" offering new ways to model and understand network traffic and behavior. This supports Fukuda *et al*. [25]'s suggestion that that self-organized criticality could be the origin of the fluctuation of burstiness in network traffic. In experiments with an ethernet traffic simulator to investigate competition among nodes and the exponential back-off that occurs when packets fail to be transmitted due to congestion, they discovered that when traffic flow rate was low there were few collisions and the traffic statistics were "dominated by the random input" [25, p. 299]. However, when the input reached the critical rate the number of collisions became significant and the output traffic became "correlated in long time scales". They observed that at this critical point the traffic hovered or fluctuated randomly about the two phases and that these fluctuations exhibited self-similarity.

2.1. Identifying and Measuring the SOC

SOC is not a discrete variable that can be identified and monitored directly. Instead, its presence is inferred through the analysis of a system's behaviour or properties, specifically by looking at some time-dependent characteristics. For networks such analysis would typically focus on the traffic, that is, the packets passing through the system. We may observe the SOC by measuring these time-dependent characteristics and comparing changes in successive samples. This is typically done by calculating a log return. The log return, r, of two data values on a stream S at intervals t and t' is given by equation (1).

$$r = \ln[S(t')] - \ln[S(t)] \tag{1}$$

That is, two successive data samples are converted to logarithms, which are then subtracted to give the log return value. During normal behaviour the log return differences will be small. However, a repeated series of large changes may well indicate a network instability, and the possibility of some form of network 'reset'. Here, a reset does not necessarily mean a catastrophic failure of the network, but could rather mean the existence of a rapidly increasing level of service traffic restrictions [4].

Some simple examples may illustrate what we mean by service traffic restrictions. The log returns (r) of normal network traffic and a network undergoing a distributed denial of service (DDoS) attack were compared using a Daubechies wavelet (part of the wavelet transformation package within Matlab). Since we are concerned with the notion of self-similar properties, then it made sense to use this particular approach [26].

As can be seen in Figure 1(a), the residuals have the characteristic burstiness of normal network traffic. This can be seen more clearly in Figure 1(b), were the residuals have been denoised. In addition, the FFT spectrum for the normal traffic displays almost consistent energy levels across the entire frequency range.

Figure 2(a) shows DDoS attack traffic. Again the characteristic burstiness can be seen in the residuals, this time slightly more intense and regular. However, note the energy levels and distribution in the FFT spectrum. The energy levels have increased by a factor of 10, while the distribution is confined towards the upper end of the frequency spectrum, and note the rising trend, possibly an indication of increasing SOC activity, and an unstable situation.

In Figure 2(b), the residuals of the beginning of a malicious network attack have been denoised. On a cursory inspection it appears to be very similar to Figure 1(b). Both figures plot the same data sets, but Figure 2(b) is a representation of the denoised residuals of normal traffic data that is also carrying malicious traffic data. Consequently, one would expect to see some differences, and on a closer inspection, the differences become clear. Firstly, at approximately 500 (x-axis) in Figure 2(b), a small amount of SOC activity can be observed (this is not present in Figure 1(b)). Secondly, between 1000 and 1500 (x-axis) on both figures, it can be seen that the level and intensity of SOC activity has increased in Figure 2(b). Whereas between 1500 and 2000 (x-axis), the SOC spike has moved, while between 2000 and 2500 (x-axis), the SOC activity has intensified. In the next section we describe a system for sonifying the SOC characteristics of network traffic.

3. The SOC Sonification System

A prototype SOC sonification system, socs, was designed and constructed to facilitate the real-time auditory perception of the SOC properties of network traffic. The tool was implemented using the Pure Data audio programming environment (freely available from `http://puredata.info`) and a custom Python script that used the Python `socket` library for dealing with the capture of network packets and the transmission to the tool of the log return values of the variables being monitored.[6]

Network traffic is fed into the Python script either via a live capture device (e.g., the Wireshark program) or from a file of previously captured data that is played back via a script which can maintain the original timing of the events, or resample the data to allow the playback of different timescales (see below).

For purposes of illustration, the example chosen here sonifies the log returns of the following time-dependent network traffic data items: number of bytes sent, number of packets sent, number of bytes received, number of packets received by the network which we call bs, ps, br, pr respectively. These variables represent the total number of packets and bytes sent and received in a given time interval, t, t'. As SOC has been shown to exist across multiple timescales, network traffic could be sampled at any regular interval. The size of the interval is not specified and is at the discretion of the user. SOC properties can be observed by comparing the log return values of successive samples of time series data. Thus,

[6] The system and example audio output can be accessed from the project's repository at `https://github.com/paulvickers/nuson-SOCS` [28].

in this example we calculate four log return values for the variables *bs*, *ps*, *br*, *pr*:

$$rbs = \ln[bs(t')] - \ln[bs(t)] \quad (2)$$
$$rps = \ln[ps(t')] - \ln[ps(t)] \quad (3)$$
$$rbr = \ln[br(t')] - \ln[br(t)] \quad (4)$$
$$rpr = \ln[pr(t')] - \ln[pr(t)] \quad (5)$$

This may result in negative values for the log return which can be used to indicate the direction of a SOC event's change in level (i.e., an increase in value means a step up to the next level of steady state, whilst a decrease means a step down). Therefore, the system can also use absolute (unsigned) log return values to keep all values positive (which might be done if one were interested only in large changes of level regardless of direction). In addition, all values can be squared with the sign retained or discarded. Thus, if using absolute values, the squares will all be positive, but if signed values are used, then the squares retain the sign of the original value (e.g., a log return of -2 becomes -4 when using signed values, but 4 when using absolute values). The reason for squaring the values is discussed in Section 4.

The Python script calculates the log return values and feeds them as input to the sonification engine. Each log return value is used to control the parameters of an individual sound generator (or *voice*), a technique known as parameter-mapping sonification, or PMSon, (see Grond and Berger [27] for a detailed discussion of PMSon).

3.1. Sonification Parameters

There are many possible mappings between the input data values and the various parameters that affect the audio. For example, this may be done by increasing/decreasing the amplitude, altering a sound's position in a sound field (e.g., left-right pan in a stereo field, front/back/left/right in a surround-sound field, or front/back/left/right and azimuth in a full three-dimensional sound field), altering the sound's phase, or altering its spectral characteristics (e.g., by changing the parameters of a filter). The following sections describe the processes that were used in the system.

3.1.1. Scaling

SOC evidences itself through orders-of-magnitude changes in the log return values but audio processing units tend to require restricted ranges of digital input values (say, $0 \ldots 127$, $0 \ldots 15$, $-256 \ldots 255$, etc.). Therefore, it was necessary to scale all incoming data so a scaler module was built that takes four arguments: the minimum and maximum values of the input range and the minimum and maximum of the desired output range. Any any value received on the scaler's input is converted to a corresponding value in the specified output range.

3.1.2. Amplitude Control

An amplitude control module was constructed that adjusts the amplitude, or level, of the output sound according to the value of the module's input variable, in this case, the log return values. The lower the log return value the quieter the sound, the higher it is the louder the sound that is played back. Thus, the real-time monitoring of the network leads to constant fluctuations in the amplitude of the output, but only large changes in level are readily perceived.

3.1.3. Filtering

There are several ways that the spectral characteristics of an audio signal may be processed, each of which will cause a change in the timbre of the audio. For this example the input log return values were used to determine the coefficients of a biquad filter which has been configured with initial values of $\{1.41409, -0.9, 1, -1.41421, 1\}$ which defines a notch filter. The advantage of a biquad filter is that it offers a richer set of filtering options than a single type of dedicated (e.g., band pass) filter and its coefficients can be altered in run time to changed the sound processing in real time. If desired, a basic band pass (BP) filter may be used by swapping the `loopChannel3` sub patch for the `loopChannel` sub patch which is available in the `abstractions` folder of the project's repository [28]. A notch or band-reject filter prevents frequencies within a certain range from passing. A BP filter, by contrast, allows frequencies within a certain range of the central frequency to pass unhindered and attenuates frequencies falling outside this range. In the Pure Data language the range's width is specified by the BP filter's *resonance* input. The higher the resonance the wider the band of frequencies that are allowed to pass through. The choice of filter depends on the kinds of sounds being loaded into the system. The disadvantage of the BP filter was that it tended to muffle the soundscape during normal operation, whilst the notch filter retained the soundscape's original brightness and altered it only when large changes were detected in the log returns.

A filtering unit was used for each audio channel. The filter's initial parameters are hard-coded into the `loopChannel3` sub patch to best fit the sonic material being used but ultimately it is intended to expose this functionality to the end user. The value of the input variable being monitored by each channel was then used to

alter the filter's response in real time. This means that the timbre of the audio changed as the input variable changed.

3.1.4. Sampled and Synthesized Voices

Because the prototype system monitored four variables four voices or channels were used, one per variable. The system can be extended to include as many voices as there are data dimensions to be monitored. A voice can be a synthesized tone generated in real time or it can be a segment of sampled audio that is played back as a repeated loop. The loop playback method incorporates a sub patch designed by Farnell [29] which loads a wave audio file and plays it continuously, restarting it when it reaches the end. The amplitude of the loop is controlled by the amplitude control module and its timbre is controlled by the filter above. Thus, the log return value of each data stream is used to modulate the corresponding voice.

The loops and synthesized voices could, in principle, be any sound, but it is recommended to use sounds that complement each other (e.g., the different sounds of a natural ecology) to minimize perceptual distraction. In the version described here, the channels contained different sounds that combined to make a countryside soundscape. Any wave files can be loaded into the system. For the examples described here voice 1 was mapped to a woodland sound with a variety of bird calls. voice 2 used a recording of a running stream, and voice 3 used a recording of wind. All the audio files were downloaded in mp3 format from the freeSFX web site [30] (country_sounds.mp3, stream02.mp3, and vientos.mp3), edited for length, and exported as wave audio files. Voice 4 used a synthesized rain sound; the sub patch for generating the rain sound was taken from Farnell [29]. This enables the various audio channels to be attended to as a single coherent whole, but alterations in any single channel will stand out.

3.2. System Architecture

Figure 3 shows the architecture of the sonification design used for sonifying four data streams. Each of the numbered items in Figure 3 is explained below.

1. Capture network traffic: Network traffic is captured from a log file or a live packet sniffer program by the custom Python script.
2. Select variables: The variables chosen here are *bs*, *ps*, *br*, *pr* (see above).
3. Calculate the log returns: Calculation of the log returns is done in the Python script and the values fed to the sonification engine built in Pure Data.
4. Scaler: This module scales the input values to the ranges required by the various audio processing units.
5. Amplitude control: Modulates the amplitude of the voice by the log return value.
6. filter: Alters the parameters of the band pass or biquad filter according to the log return value.
7. Voice *n* loop: A looped sample playback. The amplitude of the loop is controlled by the Amplitude control (above) and its timbre is controlled by a filter above.
8. Voice *n* (synthesiszed tone): A synthesized rather than sampled sound channel.
9. Mixer: The four audio channels are combined into a single stereo output which is then sent to the audio system of the host computer.

Figure 4 shows a screen shot of the application as it looks to the user. The application has four principal sections: network input (A), channel processing (B), the mixer (C), and the graph view (D). The network input section contains a module that receives the log returns generated by the `trafficSender.py` Python script. The channel processing section contains four similar units: three for dealing with audio loop playback and one for dealing with synthesized tone playback. Each of the four units contains a scaler module and a filter module. The three loop-based units also contain modules for loading and playing back the pre-recorded audio files. The synthesizer unit contains modules for generating and filtering white noise. Each of these four channel processors contain a real-time graphic plot which shows the values of the log returns. The mixer section (C) allows the relative amplitudes of the four channels to be set. These four channels are then mixed down to a single stereo output which is sent to the host computer's audio hardware. The audio output can also be captured in real-time and saved to a wave audio file via the 'start_rec' 'stop_rec' buttons. The graph view (D) plots the aggregate network traffic in real-time which allows visual reference to be made when something of interest is heard.

The network input section (A) contains taps to turn the four data streams that are being sonified on and off. This allows the operator to generate an overall soundscape of all the network variables being monitored or to focus on an desired subset. Additionally, the mixer section (C) allows the overall balance between the soundscape channels to be adjusted as desired. For example, if the operator wished to focus more attention on the bytes sent per time interval then they could raise the level of voice 1.

4. Discussion

The system was driven by a number of traffic data sets captured from live networks. Traffic data were aggregated over 1 s intervals and the number of bytes and packets sent and received per interval were fed to the socs application via the Python script. Each time a set of log return values is received the system uses the values to modulate the four respective audio channels.

When the traffic is exhibiting normal patterns small fluctuations in log return values do not lead to very noticeable changes in the soundscape, either in amplitude or timbre. Using the mappings described above one hears a soundscape comprising the combination of sounds described above.

When one or more very large log returns occur (such as would be expected during a dynamic system relaxation event) the corresponding soundscape experiences a very noticeable change: the amplitude varies greatly and the timbre alters as filters are adjusted (see Figure 8). In practice this was experienced as a sudden attenuation, loud rumble, or click (depending on the filter being used and whether signed, unsigned, or squared log return values were being used).

4.1. Audio examples

Audio files demonstrating the system output can be found in the project repository [28]. Table 1 lists the available example audio files.

Table 1: Example sound files

Audio file name	**Description**
`normal.wav`	No large log returns
`spike2_1s_a.wav`	Two audible spikes using unsigned (absolute) log returns, see Fig. 5.
`spike2_1s.wav`	Same spikes but using signed log returns.
`spike2_20ms_a.wav`	Same spikes, but at 20 ms speed, absolute log returns.
`multiSpike_1s_a.wav`	Multiple spikes, 1 s playback, absolute log returns, see Fig. 7.
`multiSpike_1s_as.wav`	Same spikes, 1 s playback, but squared log returns.
`multiSpike_1s_s.wav`	Same spikes, 1 s playback, squared log returns retaining original sign.
`multiSpike_1s.wav`	Same spikes, signed log returns, see Fig. 6.
`multiSpike_20ms_a.wav`	Same spikes, 20 ms playback, absolute log returns.
`multiSpike_20ms.wav`	Same spikes, 20 ms playback, signed log returns.
`multiSpike_20ms_as.wav`	Same spikes, 20 ms playback, squared log returns.

All files available in the `examples` directory at `https://github.com/paulvickers/nuson-SOCS`

Running the Python script at different playback rates and with signed, unsigned, and squared log return values leads to different auditory outputs. For example, running the system using signed log returns (the system default) reveals that the traffic spikes shown in Fig. 5 are both negative. The sound file `spike2_1s_.wav` demonstrates that at a playback speed of 1 s per record (the timescale at which the traffic data were captured) the negative spikes reveal themselves as gaps in the soundscape. In this case, the woodland sounds and the wind sound are attenuated as these are mapped to the sent and received bytes variables. However, when playing back the traffic data at a higher rate, as one might do when spooling through a log file to get a feel for where any problems might lie, these attenutations are not so perceptible. In this case it is preferable to run the system using absolute (unsigned) log return values. This renders all large changes as positive spikes which results in all spikes being heard as large increases in amplitude and filtering effects. The sound file `spike2_20ms_a.wav` is a recording of the system processing the traffic data at 20 ms per record. The two spikes are now clearly audible.

Fig. 6 shows a series of spikes in the traffic data. The sent and received bytes variables each have three spikes the first two being negative and the third positive, while the sent and received packets variables have a single positive spike each. In the audio file `multiSpike_1s.wav` the negative spikes are clearly heard as gaps in the soundscape whilst the positive spike is very audible, particularly in the wind sound which represents the received bytes variable.

Using absolute log returns the spikes appear as in Fig. 7. This time, at a playback rate of 1 s per record (`multiSpike_1s_abs.wav`) the first two spikes in the sent and received bytes are heard as a very loud noise signal, which is made even stronger by the sent and received packets spikes coinciding. The third spike is dominated by the wind sound of the received bytes channel. The same spikes when played back at 20 ms per record are much shorter and more percussive.

4.2. The Situational Awareness Loop

On hearing an events such as described above (situational awareness level 1— Perception) the network aministrator would be drawn to inspecting the state of the network (situational awareness level 2 — Comprehension) to decide whether any action needs to be taken (situational awareness level 3 — Projection). After deciding what action to take (level 3) then comes the stage of managing the action, which itself requires situational

awareness as actions are taken to address the situation. The final step in all UK military decision support methodology is to ask the question "has the situation changed?", thus restarting the OODA loop.

In a healthy network one would expect a number of significant changes in the soundscape over time as relaxation events occur (much as a sandpile would undergo shifts in its topology as sand is added to it over time). Some of these events may go unnoticed by the administrator (if, for example, they left the monitoring station for a short period of time) but individual events are not a matter of great concern. What will be of particular interest is when there is an extended series of repeated high log return values which might indicate growing instability in the network. An extended period of increased soundscape amplitude signals as a clear alert to the administrator.

4.3. Timescales

The system was run with log return intervals of 1 s and 20 ms. The traffic data set used in the examples was collected at 1 s intervals, so the 20 ms playback was done in a post-hoc examination mode.

The running of the socs system at a higher rate than the traffic data's initial sample rate allowed historical feeds to be listened to post-hoc in a manner analogous to spooling quickly through an audio tape (the main difference being that there is no consequent alteration of pitch). This means that logs can be auditioned quickly and interesting areas of activity spotted. This is useful for post-incident investigations and means that the system can be used for more than live monitoring. The Python script can be supplied with run-time arguments to focus on certain sections of the traffic data, and/or to slow down playback once a particular point is reached. For example, the command:

```
python trafficSender.py -t 0.02 -f 6000 -w
6960 -f 7100
```

tells the Python script to send unsquared signed log returns (the default) to the socs system at a playback rate of 20 ms (0.02 s), starting at record number 6000 and slowing down to a default rate of 1 s per record once record number 6960 is reached, and then stopping at record number 7100.

5. Concluding Remarks

The combination of using a system's self-organized criticality as the underlying data set for situational awareness and a tool for sonifying this SOC offers a number of potential advantages. First, because SOC is an emergent property of the network as a whole, and can be seen at different timescales, it means that one can get an impression of the overall state of a network by monitoring a relatively small number of data streams, thereby ameliorating the problems of extreme volumes and speeds of data identified by previous researchers. Second, the sonification approach allows for the real-time presentation of simple, but relevant data via a medium that lets network administrators work at situational awareness levels 1 and 2 using without having to keep a visual focus on a complex graphical display. Third, because SOC manifests itself fractally and across timescales, whatever data sampling interval is chosen, the network SOC ought still to be perceptible regardless of the interval over which traffic data are sampled and aggregated.

While the work described here focused on the traditional traffic metrics of bytes and packets sent and received, it is important to explore what other variables and characteristics are implicated in a network's SOC and this is the subject of ongoing work. For example, rather than using log returns, Valverde and Solé [23] explored network criticality through packet density, congestion, and the "critical load rate" measured by λ_c". It will be instructive to investigate how different views of network behavior exhibit SOC and how each of these can be used in the situational awareness loop.

The present system allowed the creation of a soundscape of up to four independent audio streams (mixed down to a pair of stereo channels). The underlying system architecture promotes interactivity by letting the user select the combination of incoming data streams to be sonified and the sonic balance of the auditory streams.

Another aspect of the ongoing work is to explore combining sonification with a multitouch display to create a richer interaction experience. The following example use case describes how this might be realized. A possible intrusion is detected through an anomalous change in the SOC variables. The administrator now wishes to investigate the network's behaviour. To do this a diagram showing a network setup is projected onto a multi-touch display. The data indicate a problem between the router and the internet, and between the switch and the laptop. A tangible user interface object (e.g., a cube) with a fiducial marker on its bottom surface is placed above the router. Another object is placed on the channel between the switch and the laptop. A camera beneath the display recognizes the fiducials which are coded to specific traffic data variables.

Rotating the objects controls the auditory and/or visual parameters of the data streams. Visual feedback can be projected onto the surface (e.g., printing data above the interface object) with auditory feedback being via loudspeakers or headphones. The interface objects become probes to monitor chosen network locations for particular events or data types. In this way the administrator can gain intelligence about the state of the network in a hands-on way.

Acknowledgements

This work was funded by the United Kingdom's Technology Strategy Board (Innovate UK) (grant no. BK008B). The authors gratefully acknowledge the input of Jonathan Christison, a final-year student on Northumbria University's BSc Ethical Hacking for Computer Security who provided assistance with constructing the Python packet sniffer. This system described in this article was the subject of UK Patent Application no. GB1205564.6.

[1] W. S. Angerman, Coming Full Circle with Boyd's OODA Loop Ideas: An Analysis of Innovation Diffusion and Evolution, Master's thesis, Airforce Institute of Technology, Wright-Patterson AFB, Ohio, USA, 2004.

[2] M. Endsley, Toward a theory of situation awareness in dynamic systems, Human Factors 37 (1995) 32–64.

[3] M. McNeese, Perspectives on the role of cognition in cyber security, in: Proceedings of the Human Factors and Ergonomics Society 56th Annual Meeting – 2012, 2012, pp. 268–271. URL: ftp://ftp.sagepub.com/PRO%202012/2012%20PRO%20pdf%20files/057_694.pdf.

[4] T. Fairfax, C. Laing, P. Vickers, Network situational awareness: Sonification & visualization in the cyber battlespace, in: M. M. Cruz-Cunha, I. M. Portela (Eds.), Handbook of Research on Digital Crime, Cyberspace Security, and Information Assurance, Advances in Digital Crime, Forensics, and Cyber Terrorism, IGI Global, 2014, pp. 334–349. URL: http://www.igi-global.com/book/handbook-research-digital-crime-cyberspace/104750. DOI: 10.4018/978-1-4666-6324-4.

[5] P. Vickers, Sonification for process monitoring, in: T. Hermann, A. D. Hunt, J. Neuhoff (Eds.), The Sonification Handbook, Logos Verlag, Berlin, 2011, pp. 455–492.

[6] P. Vickers, Ways of listening and modes of being: Electroacoustic auditory display, Journal of Sonic Studies 2 (2012). URL: http://arxiv.org/pdf/1311.5880. DOI: http://journal.sonicstudies.org/vol02/nr01/a04.

[7] T. Hermann, A. D. Hunt, J. Neuhoff (Eds.), The Sonification Handbook, Logos Verlag, Berlin, 2011.

[8] C. Guo, L. Wang, L. nv Huang, L. Zhao, Study on the internet behavior's activity oriented to network survivability, in: International Conference on Computational Intelligence and Security, 2008. CIS '08, volume 1, IEEE, 2008, pp. 432–435. DOI: 10.1109/CIS.2008.17.

[9] E. D. Mynatt, M. Back, R. Want, M. Baer, J. B. Ellis, Designing Audio Aura, in: CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998, pp. 566–573. DOI: http://doi.acm.org/10.1145/274644.274720.

[10] M. Gilfix, A. L. Couch, Peep (the network auralizer): Monitoring your network with sound, in: 14th System Administration Conference (LISA 2000), The USENIX Association, New Orleans, Louisiana, USA, 2000, pp. 109–117. URL: https://www.usenix.org/publications/library/proceedings/lisa2000/gilfix.html.

[11] M. Kimoto, H. Ohno, Design and implementation of stetho — network sonification system, in: Proceedings of the 2002 International Computer Music Conference, San Francisco: ICMA, Göteborg, 2002, pp. 273–279.

[12] M. Ballora, B. Panulla, M. Gourley, D. L. Hall, Preliminary steps in sonifying web log data, in: E. Brazil (Ed.), 16th International Conference on Auditory Display, ICAD, Washington, DC, 2010, pp. 83–87.

[13] M. Ballora, N. A. Giacobe, D. L. Hall, Songs of cyberspace: An update on sonifications of network traffic to support situational awareness, Proc. SPIE 8064 (2011) 80640P1–80640P6. URL: http://dx.doi.org/10.1117/12.883443. DOI: 10.1117/12.883443.

[14] M. Ballora, N. A. Giacobe, M. McNeese, D. L. Hall, Information data fusion and computer network defense, in: C. Onwubiko, T. Owens (Eds.), Situational Awareness in Computer Network Defense: Principles, Methods and Applications, IGI Global, 2012. DOI: 10.4018/978-1-4666-0104-8.ch009.

[15] E. P. Blasch, S. Plano, JDL level 5 fusion model: User refinement issues and applications in group tracking, in: Proc. SPIE, volume 4729, 2002, pp. 270–279.

[16] R. Giot, Y. Courbe, InteNtion–interactive network sonification, in: M. A. Nees, B. N. Walker, J. Freeman (Eds.), Proceedings of the 18th International Conference on Auditory Display (ICAD 2012), Georgia Institute of Technology, 2012, pp. 235–236.

[17] K. E. Wolf, R. Fiebrink, SonNet: A code interface for sonifying computer network data, in: NIME'13 —13th International Conference on New Interfaces for Musical Expression, Daejeon + Seoul, Korea, 2013, pp. 503–506.

[18] D. Worrall, Realtime sonification and visualisation of network metadata (the *NetSon* project), in: K. Vogt, A. Andreopoulou, V. Gourdazi (Eds.), ICAD 15: Proceedings of the 21st International Conference on Auditory Display, Institute of Electronic Music and Acoustics (IEM), University of Music and Performing Arts Graz (KUG), Graz, Austria, 2015, p. no page numbers.

[19] P. Bak, C. Tang, K. Wiesenfeld, Self-organized criticality: An explanation of the $1/f$ noise, Phys. Rev. Lett. 59 (1987) 381–384. DOI: 10.1103/PhysRevLett.59.381.

[20] H. J. Jensen, Self-Organized Criticality, Cambridge University Press, Cambridge, 1998.

[21] W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, On the self-similar nature of Ethernet traffic, SIGCOMM Comput. Commun. Rev. 23 (1993) 183–193. DOI: http://doi.acm.org/10.1145/167954.166255.

[22] M. E. Crovella, A. Bestavros, Self-similarity in world wide web traffic: Evidence and possible causes, IEEE/ACM Trans. Netw. 5 (1997) 835–846. DOI: http://dx.doi.org/10.1109/90.650143.

[23] S. Valverde, R. V. Solé, Self-organized critical traffic in parallel computer networks, Physica A 312 (2002) 636–648.

[24] C.-X. Yang, S.-M. Jiang, T. Zhou, B.-H. Wang, P.-L. Zhou, Self-organized criticality of computer network traffic, in: Communications, Circuits and Systems Proceedings, 2006 International Conference on, volume 3, IEEE, 2006, pp. 1740–1743. DOI: 10.1109/ICCCAS.2006.285010.

[25] K. Fukuda, H. Takayasu, M. Takayasu, Origin of critical behavior in ethernet traffic, Physica A: Statistical Mechanics and its

Applications 287 (2000) 289–301.

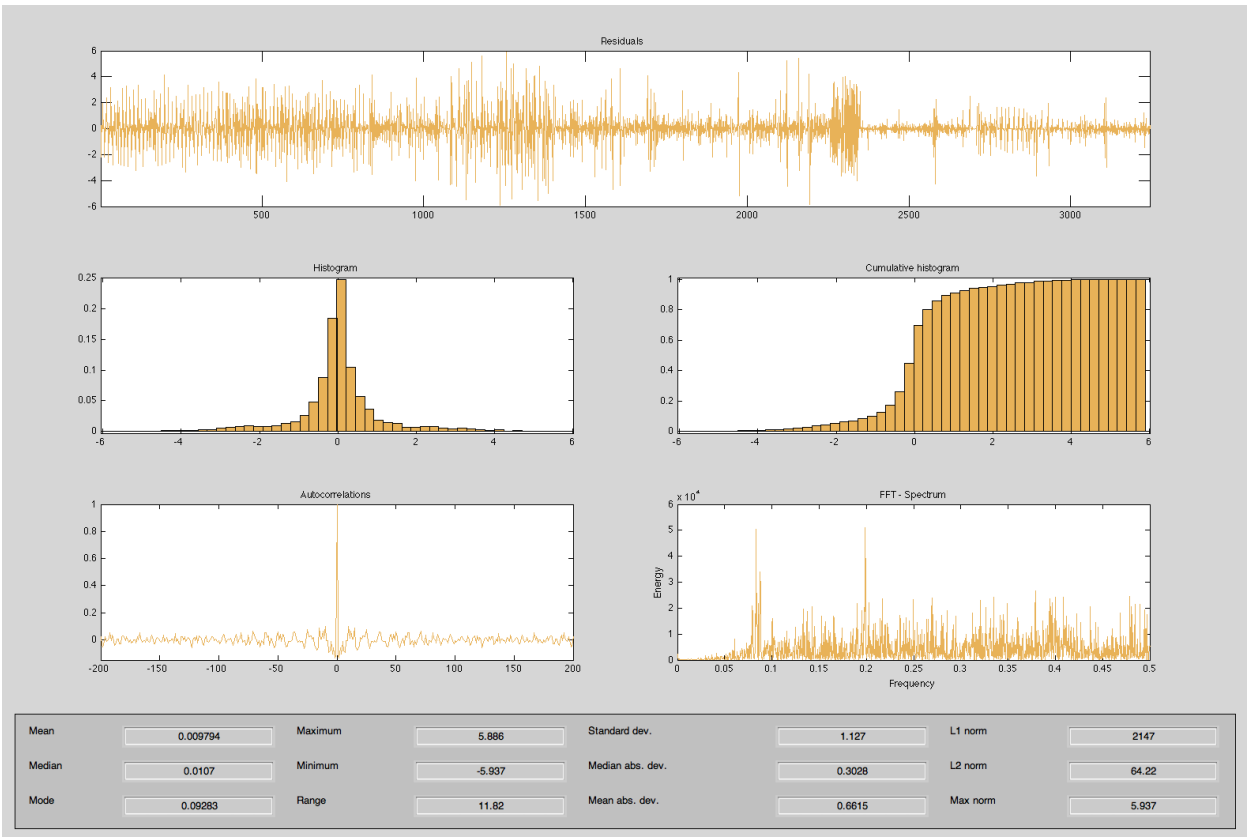
[26] N. S. D. Brito, B. A. Souza, F. A. C. Pires, Daubechies wavelets in quality of electrical power, in: Harmonics and Quality of Power Proceedings, 1998. Proceedings. 8th International Conference On, volume 1, Athens, 1998, pp. 511–515. DOI: 10.1109/ICHQP.1998.759961.

[27] F. Grond, J. Berger, Parameter mapping sonification, in: T. Hermann, A. D. Hunt, J. Neuhoff (Eds.), The Sonification Handbook, Logos Verlag, Berlin, 2011, pp. 363–398.

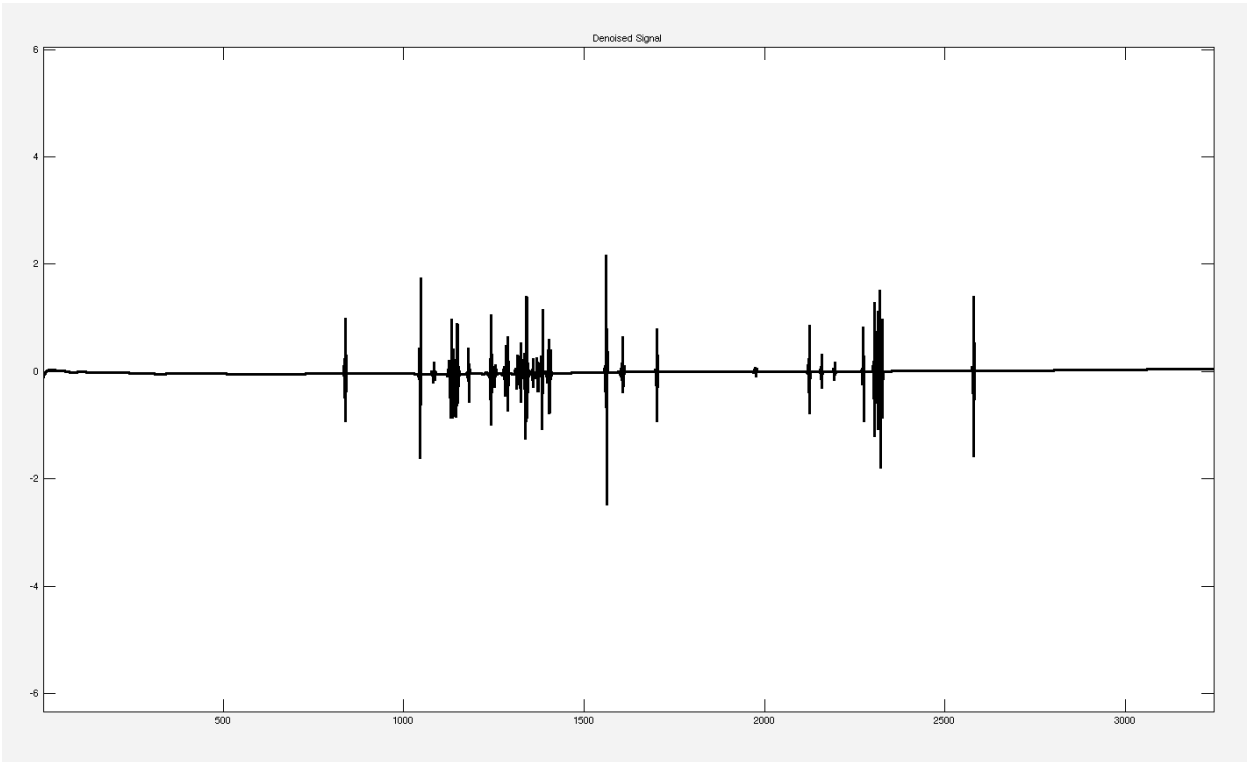
[28] P. Vickers, nuson-SOCS: Self-organized criticality sonification, 2016. URL: http://dx.doi.org/10.5281/zenodo.49844. DOI: 10.5281/zenodo.49844.

[29] A. Farnell, Designing Sound, MIT Press, 2010. URL: https://mitpress.mit.edu/books/designing-sound.

[30] freeSFX, 2016. URL: http://www.freesfx.co.uk/.

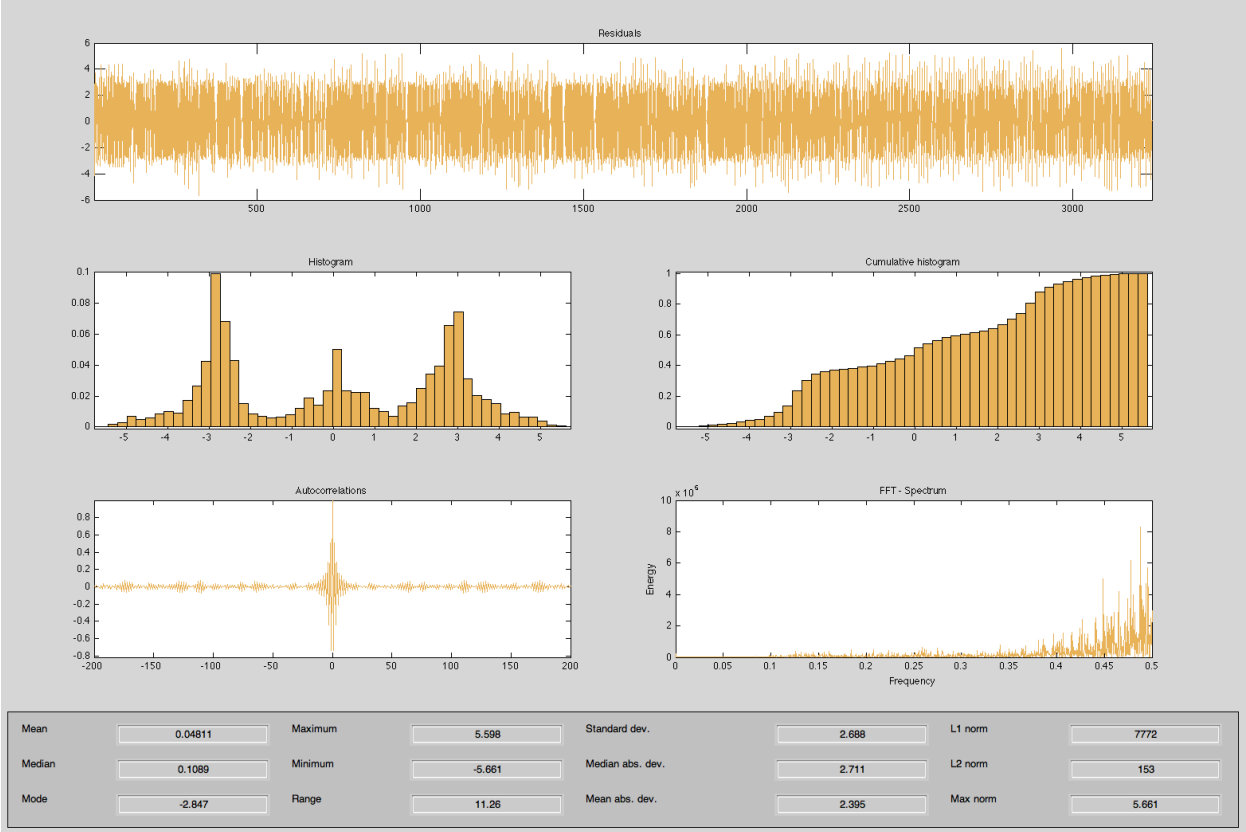


(a)

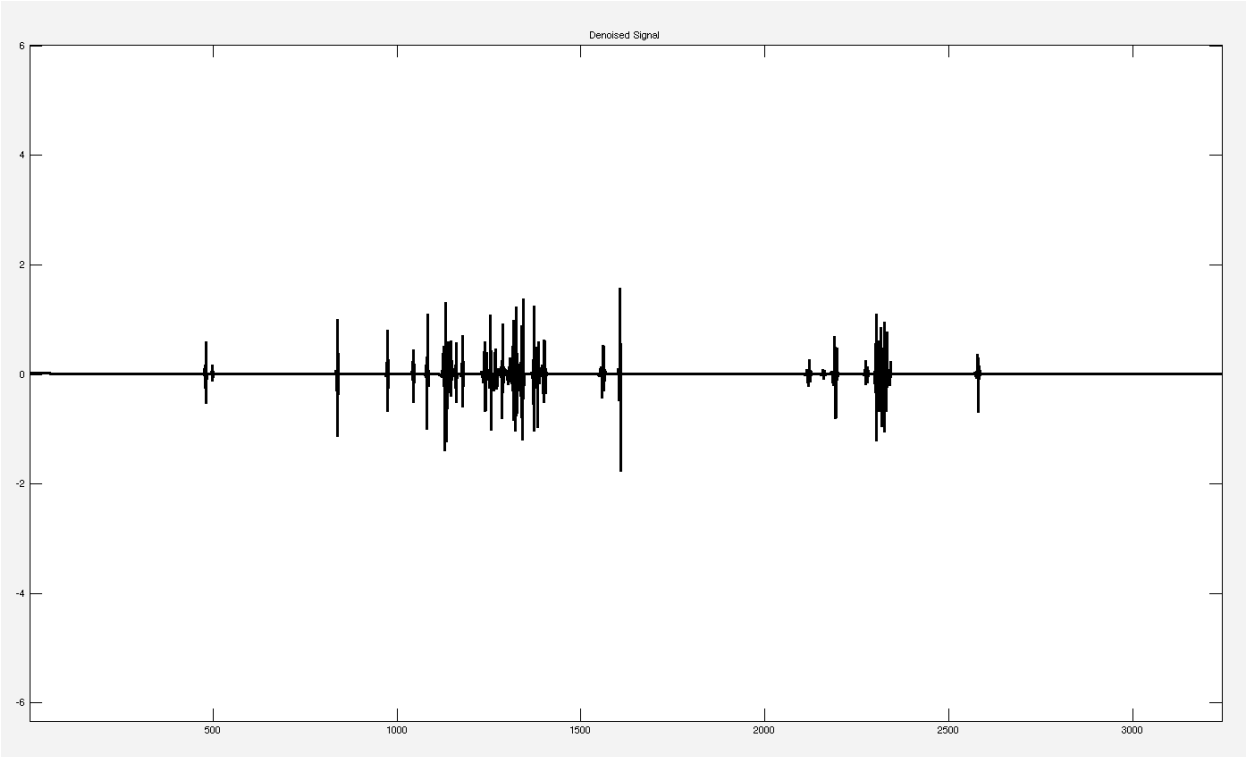


(b)

Figure 1: (a) shows a Daubechies wavelet analysis of normal traffic data while (b) shows the denoised traffic residuals.



(a)



(b)

Figure 2: (a) Shows a Daubechies wavelet analysis of DDoS traffic data while (b) presents the denoised traffic residuals.

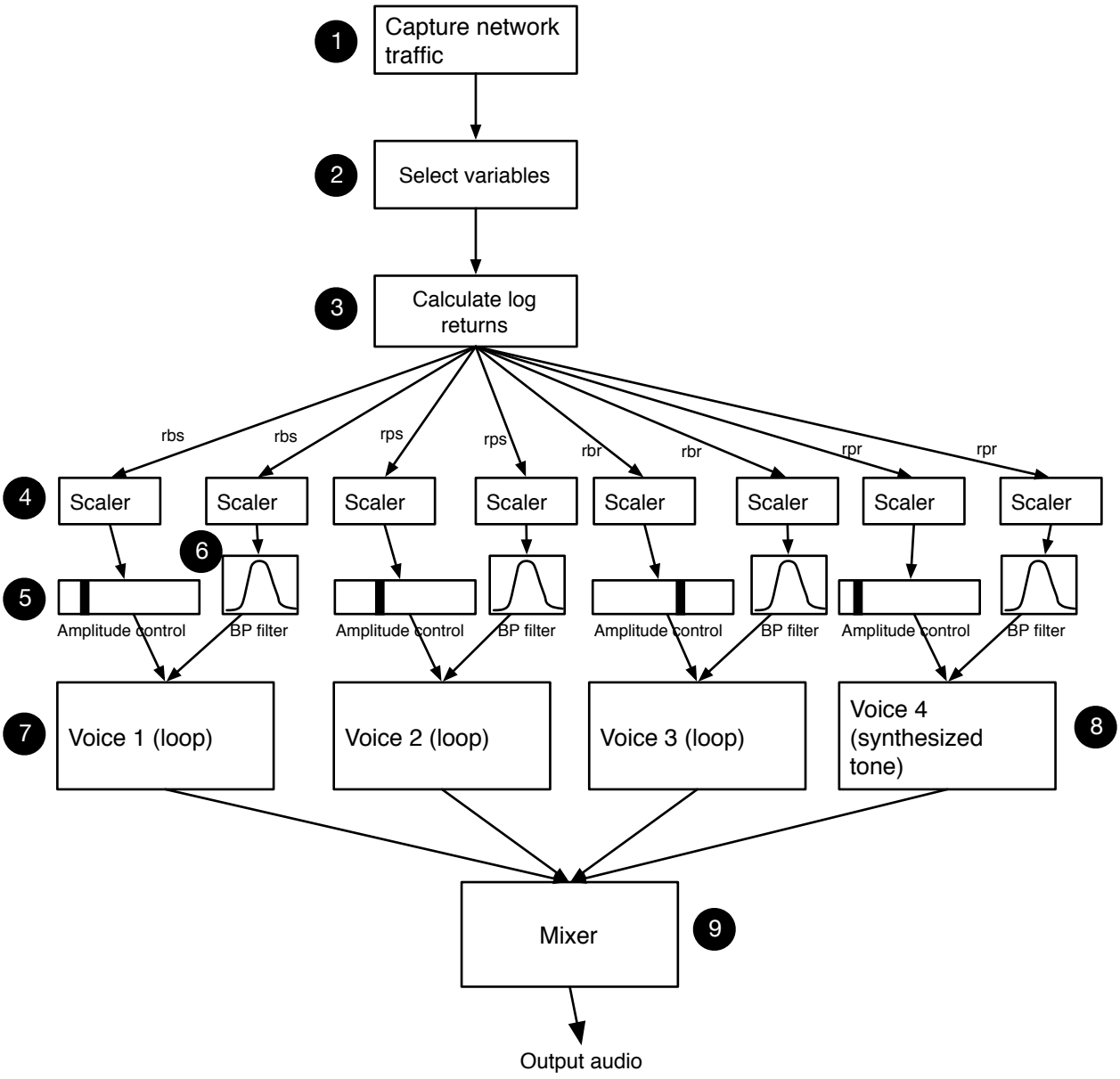


Figure 3: Schematic view of the SOC sonification system for four network traffic variables: number of bytes sent (*bs*), number of packets sent (*ps*), number of bytes received (*br*), number of packets received (*pr*).

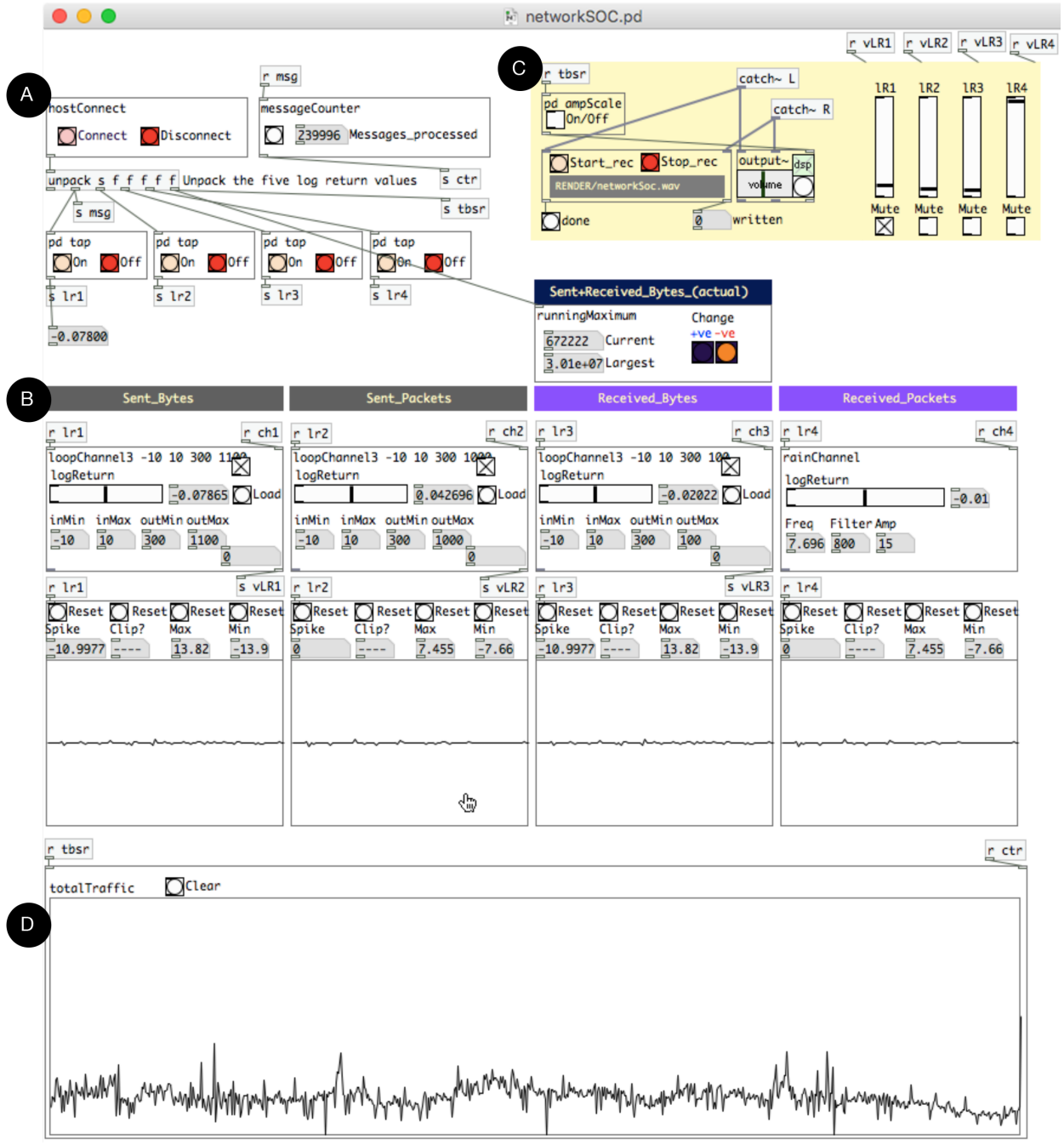


Figure 4: The socs application. Section A deals with reading the network traffic from the capture device. Section B contains the voice definitions to which each traffic variable is mapped. Section C is a mixer to convert the four separate audio streams into a single stereo feed. Section D is a graphical display of the combined variables being monitored. The channel graph plots are updated more frequently than the aggregate graph plot.

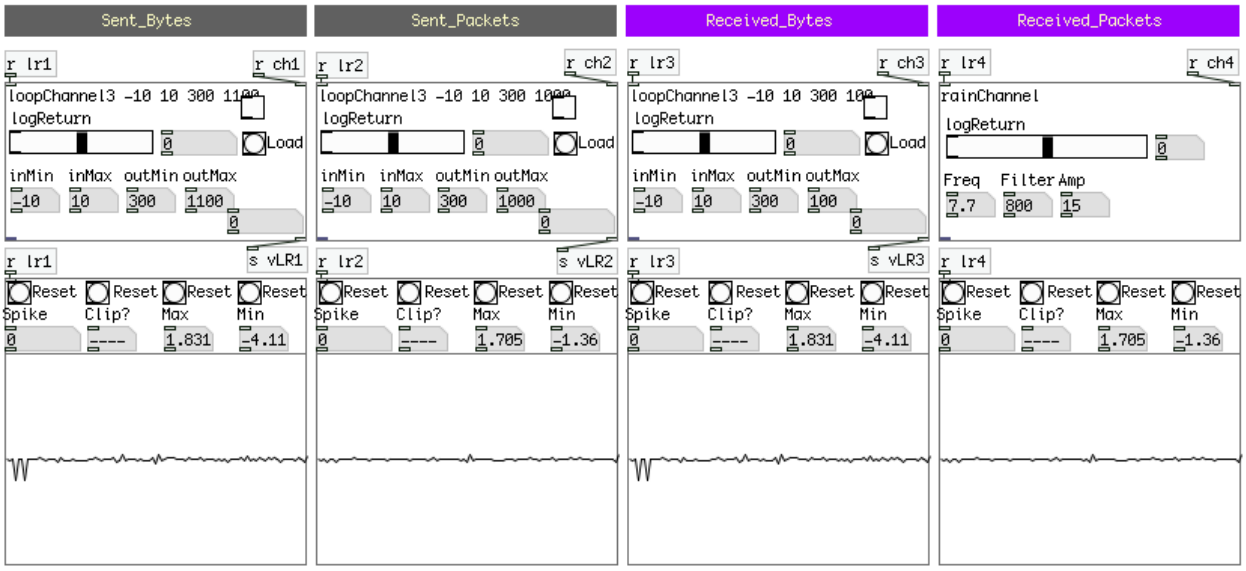


Figure 5: This screen grab shows the double spike in the sent bytes and received bytes variables. The log returns are negative jumps as shown by the descending peaks. The sound files `spike2_1s_a.wav` and `spike2_1s.wav` demonstrate how this sounds at playback speeds of 1 s per record, using the absolute and signed log returns respectively. `spike2_20ms_a.wav` is the same traffic segment but played back at 20 ms per record and using absolute (unsigned) values.

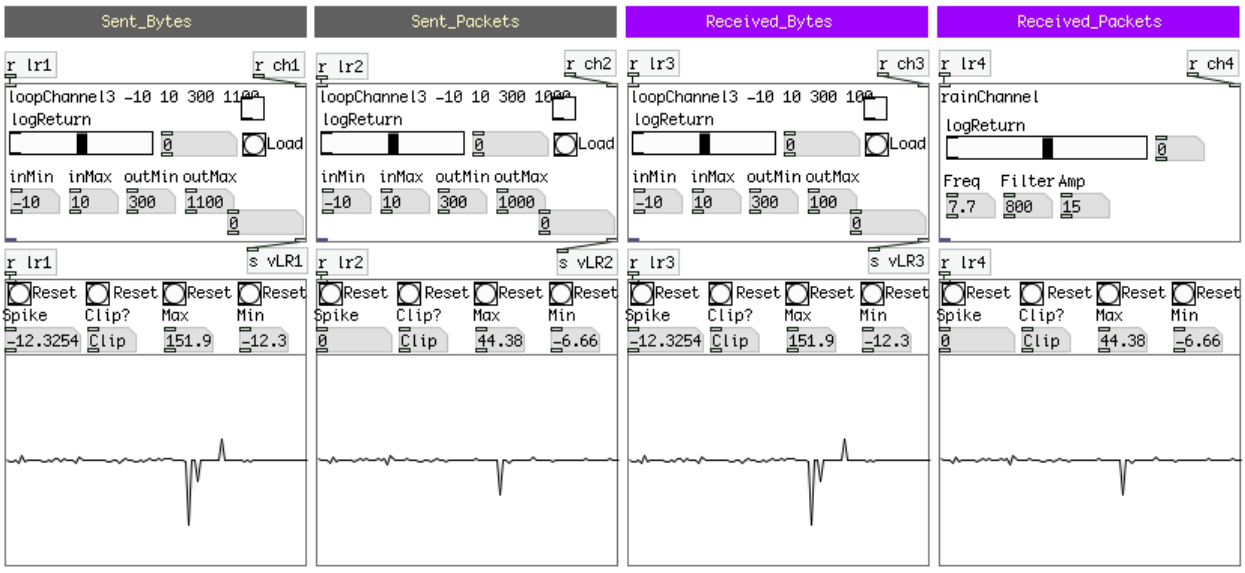


Figure 6: This screen grab shows shows a series of spikes in the sent bytes and received bytes variables.The audio file `multiSpike_1s.wav` demonstrates how this sounds at a playback speed of 1 s per record using signed log return values. There is an audible difference between the positive and negative traffic changes. However, notice how the peaks are not salient when playing back signed values at the faster rate of 20 ms as in `multiSpike_20ms.wav` — the peaks occur at around the 1.5 s mark in the audio file.

Figure 7: This screen grab shows the same series of spikes as Fig 6 but this time using the unsigned log returns. The sound files `multiSpike_1s_a.wav` and `multiSpike_20ms_a.wav` are recordings of this traffic segment played back at 1 s and 20 ms per record respectively. In both cases the traffic peaks are quite noticeable. The file `multiSpike_20ms_as.wav` demonstrates how squaring the log return values leads to an even more marked effect.

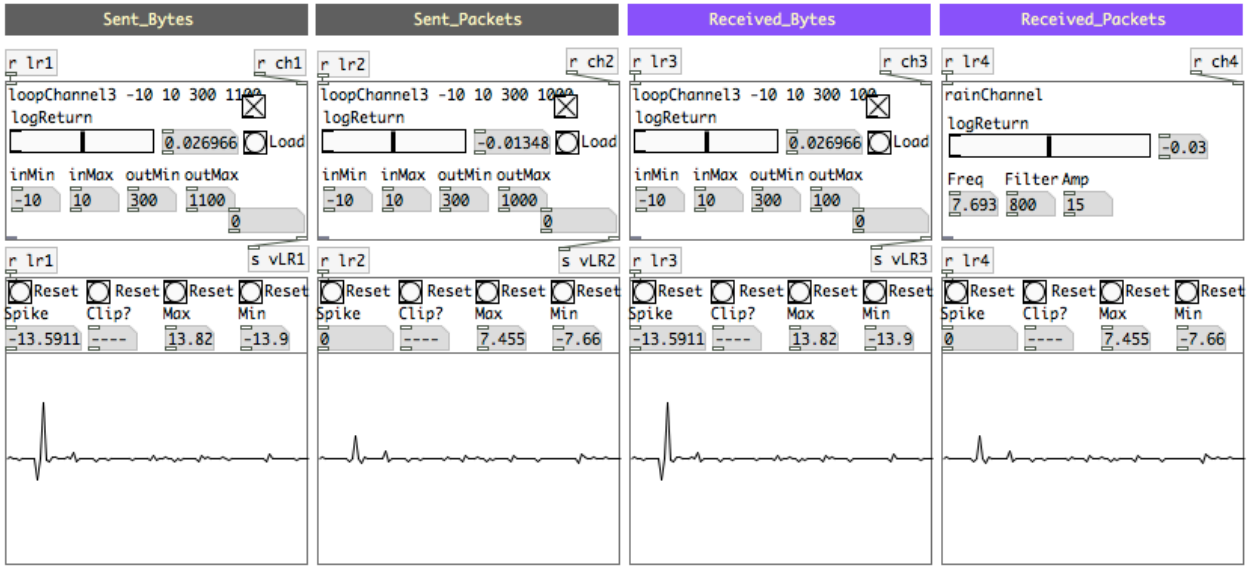


Figure 8: This screen shot of the voice channel section shows log return spikes occurring on all four channels with the largest values occuring in the sent bytes and sent packets streams. These spikes generate a noticeable increase in the amplitude and brightening of the timbre of the soundscape.