Using ICTs to Meet the Operational Needs of Community Radio Stations in India

Zahir Koradia[§]
Dept. of Comp Sc
IIT Bombay
Mumbai, India

Achal Premi
Dept. of Mathematics
IIT Delhi
New Delhi, India

Balachandran C.
Gram Vaani[§]
New Delhi, India

A. Seth[§]
Dept. of Comp Sc
IIT Delhi
New Delhi, India

ABSTRACT

Community Radio Stations are short range FM radio stations that attempt to meet the information needs of communities situated around them. While the concept of community radio is not new, the widespread proliferation of mobile phones has generated renewed interest in using phones in conjunction with radio for closer community engagement. Radio stations in developing regions are keenly looking for solutions to do this efficiently, as well as solve several other ICT challenges including content management, fault diagnosis, and reduced cost of setup. We have designed the Gramin Radio Inter Networking System (GRINS) to address these issues. In this paper, we describe the GRINS architecture, experimental setups to evaluate audio performance on low-cost commodity hardware, and a formally expressible framework to describe and diagnose radio station configurations. Our techniques and insights can serve to help other technological interventions meant for developing regions.

1. INTRODUCTION

Social media such as blogs, video and photo sharing websites, and online social networks have greatly influenced the ways in which people share knowledge, socialize, and express themselves. Often this has led to improved accountability and transparency in governance, better access to employment options, and greater awareness about current issues of global importance. How can similar tools for social media be made available in the context of developing regions, where literacy and network connectivity are significant challenges? We explore the medium of community radio in this paper, which just like its social media counterparts of blogs and content sharing websites, serves to strengthen information flow within and across communities.

CR stations are short range FM radio stations that cater to the information needs of communities living in the surrounding areas. These stations are typically setup and operated by non profit organizations or the local communities themselves. Community participation is an integral part of CR station activities with programs centered around discussions on local civic amenities, health and hygiene, advice on common economic activities such as agriculture, and even local folk songs and cultural events. The participatory philosophy of CR stations, and the use of voice as the communication medium, makes CR an attractive tool for empowering rural communities.

Although community radio has been around since the 1920s [8], a new opportunity to enhance the medium presents itself with the growing proliferation of mobile phones in developing regions and access to low-cost IT infrastructure. Combining the broadcast medium with telephony can enable CR stations to crowd-source feedback from stations, build more interactive programs, and obtain ideas for new programs. CR stations are keen to make use of these technologies to engage better with their communities and smoothen the day-to-day operations of their station, but no low-cost and comprehensive solutions exist currently. The stations therefore often end up with an ad-hoc mix of technologies that were not designed for CR stations, making them complex and hard to use. We have designed GRINS specifically in the context of rural community radio stations to provide an end-to-end solution to radio station automation. Based on preliminary surveys and visits to several CR stations in India and elsewhere, we focus on four specific goals that our ICT intervention for CR stations should address:

Seamless integration of communication technologies: Community participation in radio station activities is one of the primary guidelines for operating CR stations. To enable and foster this participation, stations are keen to use telephony and SMS to engage with their communities. However, many stations use inefficient mechanisms, for example, to record phone calls by holding the phone next to a handheld recorder, or to manually read and log SMS messages in notebooks. Integration of radio content, telephony, and SMS into a single cohesive automated system can greatly simplify the running of these CR stations.

Low cost: CR stations have limited access to advertising or other sources of revenue because their catchment area is

ACM DEV'10, December 17–18, 2010, London, United Kingdom.

small (10-15 km in the Indian context) and often economically under-developed. The stations are therefore very sensitive to the cost of any intervention. Any solution should preferably make use of low-end commodity hardware to keep costs low, but not compromise on audio quality or hamper other functions of the station.

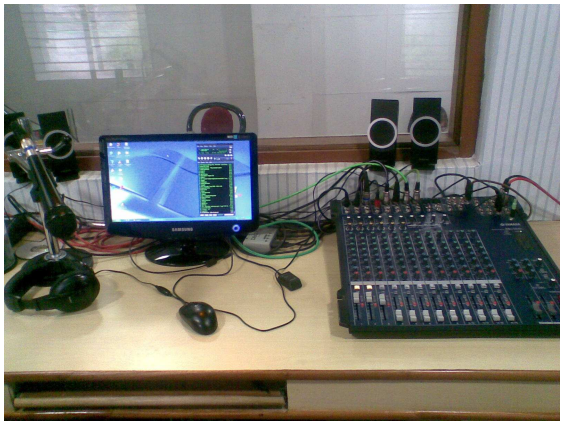


Figure 1: An image showing audio connection setup at Radio Bundelkhand

Flexibility in deployment: Different CR stations require different setups. For example, a newly established CR station with limited funds may choose to perform all its tasks on a single low-end machine, but older or bigger stations may want to handle many phone lines in parallel or manage a large content database and will require a more extensive setup. Similarly, we have seen that stations that have been running for a while and have become comfortable with existing ways of recording or editing audio programs, do not want to change their processes drastically and prefer simple plug-n-play enhancements. Thus, any solution for CR stations must allow them to choose a setup suited to them.

Local fault management: A radio station setup can get quite complicated, with audio cabling between the mixer and various other components, power supply connections, and LAN and Internet setup. This complexity is evident from Figure 1! It is hard for CR stations to find staff with sufficient skills to be able to diagnose faults and failures in such complicated setups. Sufficient trouble-shooting and diagnostic methods should therefore be included upfront so that trivial problems can be repaired locally.

We have designed, developed, and deployed a hardware and software platform in line with these goals. Our open-source system is currently running in nine community radio stations in India. In this paper, we describe our experience with building GRINS (Gramin Radio Inter Networking System), and outline various novel techniques we used to diagnose faults and monitor the performance of our system. Our contributions are as follows:

- We have identified the technology needs of CR stations, including the need to leverage the growing penetration of mobile phones, and have designed GRINS accordingly (Section 1).
- We highlight a simple and easily replicable experimental setup to study the performance of GRINS on low-end commodity hardware. This setup also helped us to identify and eliminate bugs in various open-source software components that we were using. Similar testbeds can be used to evaluate the performance of other audio based systems (Section 4).
- We outline a formal framework to describe radio station configurations, which to the best of our knowledge has not been done before (Section 5).
- We use the framework to build a generic fault diagnosis utility which generates a series of simple troubleshooting questions to debug a given configuration. We also use the framework to develop metrics to choose a good radio station configuration from among many possible options (Section 5).

The rest of the paper is organized a follows. Section 2 provides a brief introduction to GRINS and its features, and Section 3 describes its architecture. In Section 4, we highlight the primary challenge of running GRINS on low-and commodity hardware and describe a novel experimental setup to its study performance. The following section describes the formal framework and its use to diagnose station faults. Section 6 discusses the larger impact of our work and plans for future work.

2. GRINS OVERVIEW

We first describe the typical working of a CR station to set the context:

Content creation: This is mostly done offline and involves recording programs in the studio or in the field, and editing the audio to make it ready for broadcast.

Broadcast: The station operator creates playlists to schedule pre-recorded programs for broadcast, and may also introduce brief slots to speak live on air or take phone calls.

Content management: Recorded content needs to be archived and tagged to make it searchable for later.

Community interaction: Community interaction at CR stations is an on-going effort to involve the community in various activities of the CR station, such as to solicit feedback about programs, entertain requests for repeat telecasts, hold live phone conferences and discussions, etc.

GRINS is an automation system for radio stations that makes many of these tasks easier. It provides a single console to perform a large variety of operations, including:

- play out programs on air,
- preview programs while another program is playing,
- speak live on air and record the live speech,
- monitor the broadcast audio on headphones (monitor headphones are used by the station operator to listen to audio going on air),
- receive and make phone calls, record conversations
- record voice messages from callers when telephony is not in active use

- add/edit metadata for programs, and
- search for programs using faceted metadata search [4]
- automatically diagnose cable and hardware faults

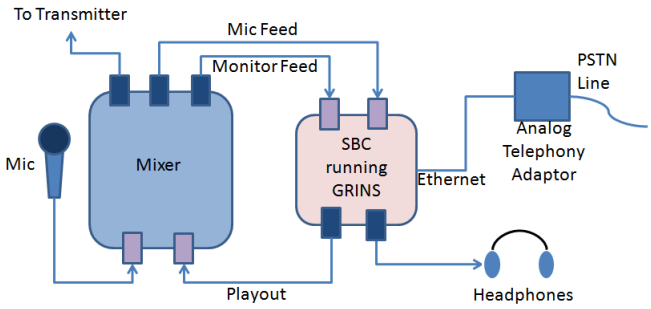


Figure 2: A radio station setup using GRINS: The single board computer (SBC) running GRINS is connected to the mixer, which is in turn connected to the transmitter. The Telephony adapter allows connecting the PSTN line to the SBC.

We are also building SMS processing and IVR capability in GRINS, to enable applications like automated feedback collation and polling to help the radio station know its audience better.

Figure 2 shows the schematic of how a *GRINS box* typically plugs into a radio station in a single machine configuration. The GRINS playout soundcard and an external microphone connect to the inputs of an audio mixer for broadcast, and the combined mixer output goes to the FM transmitter. A duplicate of the mic feed is also brought into GRINS for archival. Similarly, a duplicate of the broadcast feed ($\sim$ monitor feed) is brought in separately through a different sound card, and directed to the headphones. The same headphones are also used to preview audio, so that the operator does not have to physically change over to a different set of headphones for monitoring and previewing.

An Analog Telephony Adapter (ATA) is used to connect a PSTN line to the GRINS box. The operator can pick up the call through the GRINS console itself, and talk to the caller using the same set of headphones as those used for preview. The call is automatically archived, and can also be put live on air by redirecting both the incoming and outgoing call legs to the playout soundcard in GRINS. Conferencing is supported if the station has multiple phone lines.

All content including programs scheduled for broadcast, archived audio from live speech, and archived phone calls, are made available in the GRINS library. Metadata such as title of the program, tags, and associated caller information can be maintained through an easy-to-use interface, and a faceted metadata search is provided for retrieval of content. A listener database is provided to help stations build a profile of their listeners. This database automatically gets populated when a new caller calls for the first time or a new contact is associated with a program.

We next describe the GRINS system architecture, detailing several design decisions and their implications.

3. SYSTEM ARCHITECTURE

GRINS has a service oriented architecture [7]. Different services have been defined for specific tasks, such as an audio service for audio playout, archiver service for archiving live speech, telephony service for phone call management, library service for content management, etc. Each service is decoupled from other services and maintains its own internal state to expose information about its *readiness* to perform tasks. A lightweight inter process communication (IPC) mechanism, using TCP sockets, enables communication between services.

Decoupling of services makes the system more robust. With this design, even if a service fails due to a bug or an unexpected crash in an external open-source component, the rest of the system remains unaffected. Coupling this with an auto-restart mechanism in case of service failure potentially increases the uptime of the system, a critical requirement for rural regions.

The IPC mechanism allows services to all run off the same machine, or be distributed across different machines. For example, the phone service can be chosen to run off a different machine, or the library service can be moved elsewhere, as long as the mixer supports the necessary audio connections. This flexibility in deployments allows deployment of GRINS at stations with different requirements.

Flexibility of GRINS deployments can sometimes lead to more than a dozen possible ways to plug GRINS into a given radio station setup! Later in Section 5, we present a formal framework to describe all possible configurations in a given setting, and introduce certain metrics to choose the best configuration.

The same framework is also used to develop a fault diagnosis utility. Faults can occur because of a wide variety of reasons such as loose cable connections, or a malfunctioning LAN, or software crashes, etc, and are hard to identify specifically. Our diagnosis utility helps make logical inferences to locate the exact point of failure.

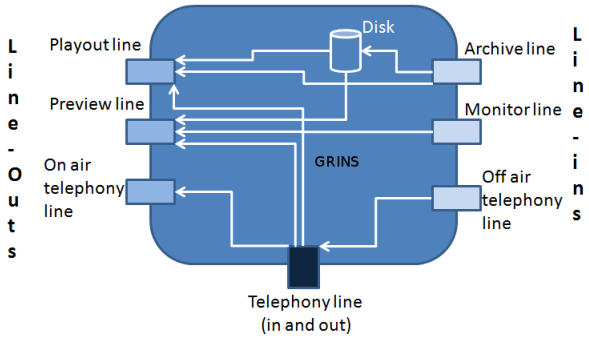


Figure 3: Different audio flows in GRINS

As may be already evident, GRINS has to handle many audio flows running in parallel. For example, a flow for playout from disk to soundcard, a flow from soundcard to disk for recording the operator's speech, two flows between soundcards and the telephony hardware corresponding to phone calls, and two flows from the telephony hardware to the disk for recording phone conversations. Figure 3 shows several possible flows in GRINS in a typical single-machine setup. Given our requirement to use low-end commodity hardware,

it becomes a challenge to ensure good audio quality with multiple flows running in parallel. In the next section, we describe novel experimental setups we used to study latency and other metrics to make sure that GRINS works well even under such constrained scenarios.

4. EVALUATING LOW-COST SETUPS

We observed a variety of issues while running GRINS on low-end commodity hardware, most of which are unlikely to appear in high-end devices. For example, encoding audio while recording live speech, or sample rate conversion during playback, consumed unacceptably high amount of CPU [6] . Similarly, scenarios where incoming audio from one soundcard was redirected to the output of another sound card, resulted in audio clicks and latency. Audio clicks are gaps when the soundcard plays out silence because audio samples were not provided to it in time. Clicks also occur in recorded audio when audio samples are not collected from the soundcard in time. These are typical situations when interrupts are missed because the CPU is not able to service them quickly enough. While clicks are definitely undesirable in broadcast audio, high latency in telephone conversations is also unacceptable. High latency during live speech can also be uncomfortable for the operator if she hears her own voice on the monitor headphones after a slight delay.

We used a novel experimental setup for measuring clicks and latency. This benchmarking exercise helped us identify and eliminate several bugs in different open-source components we were using. Eventually, on the hardware described below, we were able to completely eliminate audio clicks and reduce the latencies to quite acceptable values. We first present details about the audio architecture in Linux based systems, and then describe our setup and results.

4.1 Audio Routing in GRINS

Figure 4 shows a high level operation for line-in to line-out audio routing in GRINS. GRINS uses Gstreamer (`www.gstreamer.net`) as its media library and ALSA (`www.alsa-project.org`) as the soundcard driver. In the capture part shown to the left of the figure, the soundcard generates an interrupt when it has enough data to push to ALSA. The CPU services this interrupt and passes the control to ALSA, which collects the data and writes it in its internal buffer. Gstreamer makes blocking read calls to ALSA, which return when there is any data available in the ALSA buffer. We use input interrupt inter-arrival time (IIT) to denote the time between two successive interrupts generated by the soundcard for capture. Input IIT dictates the rate at which the CPU needs to service interrupts for audio capture.

On the playout part shown to the right of the figure, user processes push data through blocking write calls, which return as soon as the data is written to the ALSA buffer. Audio is then written from the ALSA buffer to the soundcard in response to interrupts from the soundcard. Analogous to input IIT, the output IIT dictates the rate at which the CPU needs to service interrupts for playout.

The $IITs$ directly impact audio clicks and latency. In the case of clicks, as IIT decreases, the number of interrupts to be serviced by the CPU increases. This increases the chances of the CPU missing an interrupt and causing clicks. $IIT's$ influence on latency is more subtle. During capture, the first interrupt is received by ALSA an IIT amount of time after the capture has actually begun. This causes a latency of at least one input IIT. On the other hand, during playout, ALSA waits for a Start Delay (SD) duration before beginning playout. SD is measured in multiples of output IIT and is usually set equal to the size of ALSA output buffer. This is done to avoid buffer under-runs immediately after a playout begins. Thus, delays are caused at both input and output, and are proportional to their corresponding IIT values.

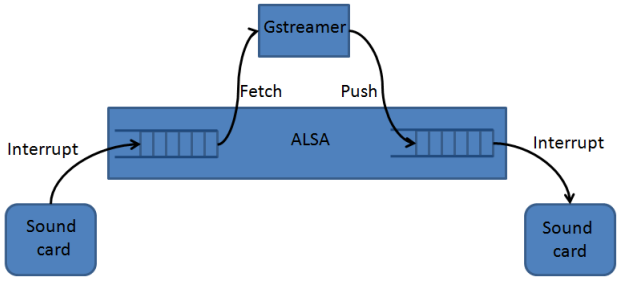


Figure 4: Audio transfer from line-in to line-out of a soundcard in GRINS

Next, we describe the experimental setup for measuring clicks and latency. The key insight from the experiments is that while latency increases monotonously with increase in $IITs$, number of clicks reduce with increase in $IITs$. Thus, a balance needs to be achieved between the two.

4.2 Experiment Setup and Methodology

We used a Via EPIA LN10000 EG mainboard with a 1GHz processor and Via VT8237 soundcard as our test machine. The machine ran Ubuntu 8.10 with the ALSA sound driver version 1.0.17. The Gstreamer multimedia framework was used for audio play-out, recording, and routing audio from line-in to line-out across soundcards. The Asterisk 1.6.1.4 (`www.asterisk.org`) PBX software was used for telephony management, with a PSTN line terminating in an external Analog Telephony Adapter (ATA) connected to the Via board over the LAN.

4.2.1 Line-in to Line-out routing latency

Figure 5 shows the experimental setup for measuring latency of line-in to line-out audio routing. The line-out of a high-end machine H_1 is connected to the line-in of the GRINS box, and the line-out of the GRINS box is connected to the line-in of the second high end machine H_2. The GRINS box routes audio from its line-in to its line-out using Gstreamer. A second audio path is created by connecting cable A_3 directly from the line-out of H_1 to line-in of H_2. The high-end machines H_1 and H_2 are assumed to have enough computational capabilities to read and write audio without causing any clicks. H_1 plays audio on its line out consisting of a sequence of *spikes* separated by an interval of 2 seconds. This audio travels to A_3 with negligible delay, but traverses the GRINS box incurring some latency. Spikes from both the paths are recorded at H_2. The difference between the start of the spike at A_3 and its corresponding spike at A_2 gives the latency of routing audio through the GRINS box.

For measuring the impact of input IIT, we fixed the value of output IIT to 10 ms and varied the input IIT from 2 ms to 50 ms. Similarly, for measuring impact of output IIT, we

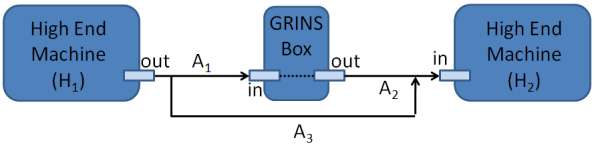


Figure 5: Experimental setup for measuring latency in audio routing

fixed input IIT at 10ms and varied output IIT. The Start Delay (SD) at line-out was set to 3 times the output IIT for all the experiments [1]. Audacity (*audacity.sourceforge.net*) was used to measure the latency manually at one millisecond accuracy. We conducted each experiment 10 times and took the average of the results.

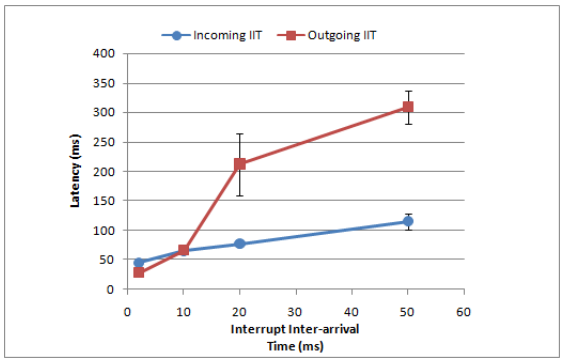


Figure 6: Impact of input and output interrupt inter-arrival times ($IITs$) on audio routing latency. Latency monotonously increases with increase in $IITs$.

Figure 6 shows the results of these experiments. As expected, both input and output $IITs$ have a linear impact on latency, but the change in output IIT impacts latency more than the input IIT because of the Start Delay on the output. IIT values of up to 20ms seem tolerable for good audio quality. However, note that these results correspond to benchmarking a single audio flow, whereas in GRINS we need to run up to four audio flows at the same time. We also conducted experiments for this *maximal setup* and eventually chose IIT values of 15ms, which gave us latencies for different flows as shown in Table 1. IIT values of 15ms also eliminated clicks, as we describe in the next section. We do not present the detailed experimental setup for brevity; details are available in [6].

These latency values will of course be different with different hardware; our contribution here has been to develop a novel experimental setup that can be used to evaluate other audio-based systems.

4.2.2 Clicks

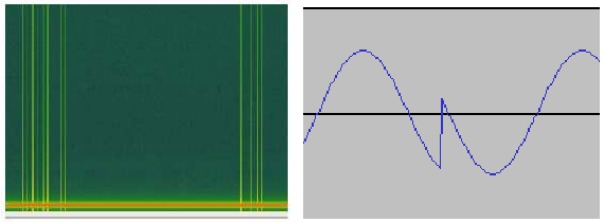
We used a similar setup as above to study the effect of IIT on clicks, with the difference that A_3 was no longer needed

[1]This value of SD was chosen after micro-benchmark experiments showed that higher values of SD increased latency while lower values caused clicks

Table 1: Latencies observed in the maximal setup for GRINS. Input and output $IITs$ were set to 15ms.

Audio Flow	Latency (ms)
Soundcard line-in to soundcard line-out	94 ± 21
Soundcard line-in to remote caller	41 ± 8
Remote caller to soundcard line-out	64 ± 11

and H_1 now played out a single-tone sine-wave audio. Gaps in the recorded audio at H_2 reveal clicks caused by line-in to line-out routing on the GRINS box. Each experiment was run for 3 minutes, repeated 5 times each, and the results were then averaged. The output IIT was kept the same as the input IIT.



(a) Audio clicks as seen in a spectrogram (b) Waveform of a recorded click

Figure 7: Audio clicks

We compute a spectrogram of the audio to identify clicks. Since the original audio is a single sine wave, the spectrogram has only one *hot* line corresponding to the frequency of the tone. However, when a click occurs then there is a sudden disruption in the waveform. This is captured by the Fourier transform as several frequency components, making all the frequencies *hot* for that part of the audio. As a result, clicks are distinctly visible in the spectrogram and can be counted. Figure 7 shows a spectrogram, and a sample audio waveform with one click.

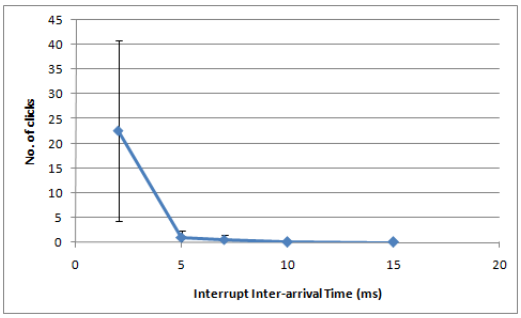


Figure 8: Impact of input and output interrupt inter-arrival times ($IITs$) on clicks in audio routing. Number of clicks reduce with increase in $IITs$.

The results of click-measurement experiments are shown in Figure 8. The number of clicks steadily reduce with increase in $IITs$. No clicks are observed at $IITs$ of more

than 10ms. We used similar setups to evaluate clicks in telephone conversations, and under high CPU loads. After some amount of debugging, we eventually succeeded in eliminating clicks altogether. Thus, the simple and novel experimental setup allowed us to guarantee the service quality that GRINS can deliver, and also helped us discover and resolve bugs in open-source software.

5. FORMAL REPRESENTATION

We described earlier that we designed GRINS in a service-oriented manner so that it could be deployed flexibly in a variety of settings. Variance in the number of machines, availability of mixer, and type of mic lead to a large number of different radio station configurations. Too many options can easily confuse a person wanting to setup a radio station. Thus, our first goal is to enumerate these options and compare them in a meaningful manner.

Recall that it is hard to find technically skilled people in rural areas. At the same time, it is important to resolve technical problems locally as much as possible, to avoid long travel times and costs of visits by technician from cities. Therefore, our second goal is to build diagnostic capabilities to pin-point the exact point of failure and guide the local staff to resolve simple issues such as faulty cables and improper connections.

To achieve these two goals, we have developed a formal framework describing radio station configurations. Based on this framework we have defined metrics for comparing configurations. We have also developed a diagnostics utility that takes a configuration as its input and, through a series of questions, guides the station operator to identify faults. We next present our framework for describing radio station configuration.

5.1 Framework

As described in the previous section, GRINS controls several audio flows. We define a flow formally and use this definition to describe a configuration.

Element: An element is any hardware or software unit that can generate or consume audio. For example, a headphone is an element that consumes analog audio, while a soundcard is an element that consumes digital audio and generates its analog form, and vice-versa. Mic, mixer, FM transmitter, file system, and the Asterisk PBX system are all examples of elements.

Channel: A channel defines the form of audio an element can consume or generate. Channels can be of different types:

- **Analog:** A mic has an analog outgoing channel.
- **Digital:** The Gstreamer element has digital incoming and outgoing channels.
- **VoIP:** A special channel to describe communication between Asterisk and an Analog Telephony Adapter (ATA).

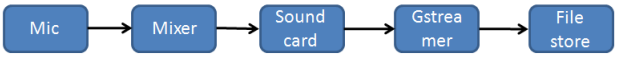
Connection: A valid connection from element A to element B is possible if there exists an outgoing channel of A, O_A and an incoming channel of B, I_B such that O_A and I_B are of the same type. For example, a Gstreamer element can be connected to a soundcard, but a Gstreamer element can not be connected to a headphone element.

Flow: A sequence of connected elements is a flow. Figure 9 shows some example flows used in GRINS.GRINS requires all of the following flows to be enabled to run correctly:

- **Playout:** File-system to the FM transmitter
- **Archive:** Mic to the file-system
- **Preview:** File-system to the headphone
- **Monitor:** Mixer output to the headphone
- **Mic:** Mic to the FM transmitter
- **Tele-Playout:** Telephony hardware to the FM transmitter
- **Tele-Preview:** Telephony hardware to the headphone
- **Tele-Archive:** Telephony hardware to the file-system
- **Tele-Mic:** Mic to the telephony hardware

The opportunity for flexibility and choice of different configurations in GRINS arises from the different ways in which these flows can be enabled. For example, a mic-flow can be enabled by either plugging the mic into the playout machine, or plugging it into the mixer. In the former case, the mic audio has to be passed in through a soundcard to a Gstreamer routing module and then to the soundcard that feeds the FM transmitter. In the latter case, the mixer may be feeding audio directly to the transmitter, and so the mic audio need not be brought into GRINS at all (unless the audio needs to be archived).

(a) Tele-Preview flow



(b) Archive flow

Figure 9: Examples of flows of a configuration

Configuration: A configuration is defined by a collection of the nine flows listed above. A configuration is considered valid if all the flows in the collection are enabled.

5.2 Configuration Enumeration

We have developed a tool using Prolog to automatically generate valid configurations. It takes as input a 4-tuple that corresponds to the answer of the following questions:

- Whether or not a mixer is available (yes/no)
- Whether the mic can plug into a mixer ($\sim$ *XLR* connector), or into the soundcard ($\sim$ *TRS* connector)
- Whether the FM transmitter is external (*ext*), or can be mounted as an internal PCI card (*int*)
- The number of machines that can be used in the setup

Using the 4-tuple input and above defined framework the tool generates all valid configurations that can be compared with each other. We skip the details for brevity and direct the interested readers to [10] for more information.

5.3 Comparing configurations

To enable meaningful comparison of configurations we classify them based on three parameters: cost, performance, and debugability. Although a combined utility score can be created, we choose to simply classify the configurations as either good or bad based on some rules of thumb.

- Using more than four soundcards is considered costly. We noticed that out of all possible configurations, 80% could be enabled with four or less soundcards, thereby rendering other solutions as relatively ineffective.
- Flows which have three or more audio conversions (digital to analog and digital to VoIP) are considered poor on latency. As observed in Table 1, these conversions increase latency in the flow.

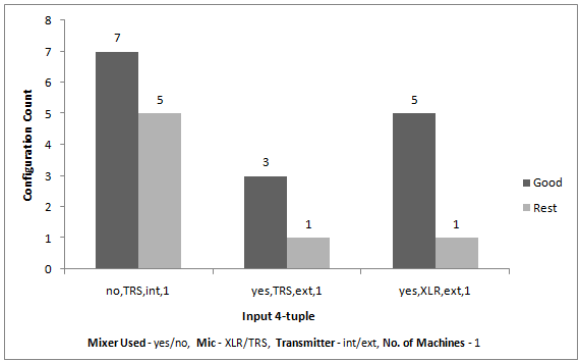


Figure 10: Classification of all single machine configurations based on cost and performance.

The debugability metric is described in the next section. Fig. 10 shows the classifications based on cost and performance for three different values of input 4-tuple. The graph shows that for a given input 4-tuple several configurations are possible. However, only a few of those may be usable when pruned using the thumb-rules above. For example, for the first input 4-tuple only seven out of 12 possible configurations are good in terms of cost and performance.

5.4 Diagnostics

We next describe how the formally expressive framework can be used for fault diagnosis, and then apply these insights to develop the *debugability* metric for classifying the quality of a radio station configuration.

We begin by noting that if a particular flow malfunctions, then a fault could lie either with some element, or with a channel connecting two elements. For example, if the audio being transmitted on air cannot be heard on the monitor headphones, then either (a) the soundcard used for monitoring may not be connected, or (b) the cable going into the monitor soundcard may be disconnected, or (c) the cable coming out of monitor soundcard and going to the monitor headphones may be disconnected. Overall, if there are n elements, then as many as $O(2^{2n})$ possible faults could arise. Clearly, if a flow functions correctly it implies that all its constituent elements and channels are functioning properly. We use this principle to break the GRINS network of flows into components that can be tested separately. Some of these tests may be automated by creating single-tone sine-wave audio at one end and performing a Fourier transform at the other end, while some may be manual requiring the user to either speak something into the mic or listen to audio on their headphones or radio sets. The goal is that after some number of tests, the inference algorithm should be able to identify likely points of failure in the system. We next describe in more detail the steps that the algorithm follows.

Phase 1 - Test cases: In the first phase, the algorithm takes configuration information as input to generate test cases. We define a test case as an evaluation of a subsequence of elements and connections in a flow such that the subsequence starts at some software element and ends at some tangibly observable element. As shown in Figure 9(a), an example of a test case is a subsequence of *Tele-Preview flow* that starts from *Asterisk* and ends in the *Head phones*, and can be observed by listening to audio on the headphones. A test case may also be a subsequence that starts at some tangible input element and ends at a software element, for example, from *Mic* to *Gstreamer* in *Archive flow* from Figure 9(b). The result of each test case is assumed to be either true or false.

Phase 2 - Diagnosis: The algorithm iteratively orders the test cases in decreasing order of their lengths, and in the case of a tie it prioritizes tests that do not require user input. The longest test cases are preferred because if the test is successful, it will indicate that all the elements and connections in that test were working correctly, and subsequences of the test can be eliminated.

The algorithm eventually comes up with a classification of elements and connections that are guaranteed to be working correctly, or not working correctly, or those that are ambiguous.

5.5 Debugability

We next describe the debugability metric that can be used to rank configurations, together with the cost and performance rules stated in Section 5.3.

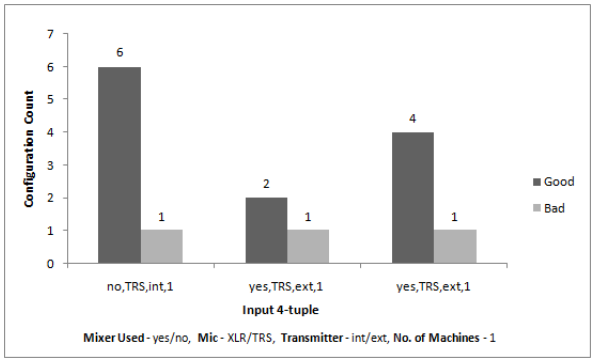


Figure 11: Classification of single machine configurations based on debugability. Only configurations found good based on cost and performance are considered.

Given a configuration, we simulate the diagnosis algorithm considering all possible results for the tests. In each case, we observe the number of elements or connections that are left ambiguous $= a$, and define the ambiguity for this case as $(2^a - 1)$, denoting the number of possible eventualities when an element or connection could be faulty. For the given configuration, the overall ambiguity is then defined as the harmonic mean of $(2^a - 1).t$, where t is the number of tests executed in each case. The debugability of a configuration is then taken as the inverse of the ambiguity. Thus, higher debugability score indicates that a configuration is more easily debugable.

We evaluated, through simulations, the debugability of all single machine configurations as in Fig. 10. The 70%ile of the scores was taken as the cut-off to classify configurations as good or bad in terms of debugability. Fig. 11 shows the classification of all configurations that were found good in Fig. 10. The debugability metric can be used to prune the number of good configurations derived from cost and performance metrics. More stringent filtering can be achieved by using higher cut-off percentile. Thus, a thumbrule based on debugability can be added to the configuration classification rules stated in Section 5.3 to find the best possible configuration. Some interesting cases are presented in [10] where configurations are cheaper but can actually be harder to debug.

6. IMPACT AND FUTURE WORK

Our experience with GRINS deployments and feedback from its users indicates that radio stations indeed value its features. While a formal usability study is underway and will be the subject of a different paper, we present here some anecdotal evidence and insights into GRINS usage.

GRINS has been deployed at nine CR stations across India, and the flexibility in deployment allowed by our architecture has been extremely useful all across. Three of our setups run in a two-machine configuration mode, with the UI and playout services running on a Windows machine and the rest of the services on the GRINS box. This was because these stations had been operational since a while and preferred using Windows, instead of switching to a pure Linux-based environment. All other setups are in new radio stations, and are hence single machine setups where the staff were trained on Linux systems upfront.

Prior to using GRINS, the stations in India were using media players such as Winamp and iTunes to schedule programs. This did not allow the station operator to schedule broadcast at a pre-specified time, or manage time slots of when to speak live on air. Several stations now schedule their broadcast to start automatically as early as 6 a.m. and have the convenience of not being in the station that early. Stations also routinely use "live slots" to explicitly define the time when they want to speak live on air.

Before using GRINS, content was stored in Windows folders, often distributed across multiple machines, making search and retrieval hard and heavily reliant on personal memory. GRINS provided powerful metadata recording and content search mechanisms to the stations. However, very few stations have taken the effort of adding metadata to programs. Additionally, stations still continue to search for programs using folders. Initial insights suggest that the reluctance to enter metadata and use faceted search is related to resistance in changing existing processes. We hope to gain a better understanding of this challenge and possible remedies through more detailed interviews with users.

Before GRINS, telephony was hard and significant manual operations had to be performed on the mixer to archive the call or put it on air. In addition, there was no way of obtaining and managing caller-id of the callers. Now, several stations have reported using integrated telephony feature of GRINS to record folk song requests, collect traffic information, and log civic amenities related complaints. One station had recorded a message over telephony, from a local celebrity encouraging listeners to vote in local elections. A second station had put phone call with traffic police commissioner live one air, during which he had explained changes in traffic regulations for the common wealth games.

We are currently also building new features in GRINS for SMS and voice messaging for stations to open more ways of engaging with their communities. Yet another direction we plan to pursue is to help CR stations develop good content. Creating quality content is hard because of the lack of knowledgeable resource personnel in rural areas, and poor access to information sources like libraries and the Internet. Further, creating quality radio content is a hard-to-master skill. It requires considerable thought in conceptualization to choose the best format of the program, prepare a script, and finally record and edit the audio content. Many staff from different CR stations in India do come together for periodic workshops, but it is cost prohibitive.

We believe that a suitably designed knowledge-sharing platform, augmented with voice forums, could greatly help CR stations learn from each other and sustainably produce good content. We intend to build such a platform to enable sharing of content, expertise and experiences among CR stations. Owing to intermittent Internet connectivity available at CR stations, the platform will be designed to operate in disconnected environments. Content browsing will be enabled through caching and intelligent prefetching, while CD or USB memory stick will be used for heavy data transfer.

7. RELATED WORK

There have been several successful attempts at using technology to assist social media in developing regions. Avaaj Otalo [9] provides an asynchronous telephony based system for farmers to ask questions and answer each other's questions by dialing into a toll-free number. Digital Green [3] pioneered a novel pedagogy for teaching better agricultural practices through mediated sessions where farmers watch videos of other fellow farmers practicing new methods. Video Volunteers [1] trains rural community members in video production on issues of relevance to local communities. Complementary to these efforts, we focus on building technologies for community radio.

Some features provided by GRINS such as playout and scheduling of audio programs is similar to those offered by other radio automation systems. Some of these are also free and open source like Campcaster (`www.sourcefabric.org`), but most like WideOrbit Automation for Radio (`www.wideorbit.com`), and Nautilus Jukebox (`www.nautilus.hu`) are licensed and prohibitively expensive for CR stations. However, GRINS goes much beyond these systems with integrated telephony management capabilities, good audio quality guarantees on computationally low-end machines, diagnostic capabilities to overcome the lack of technical expertise in rural areas, and flexibility in deployments.

Our approach to formally represent a radio station configuration is similar to work on expressing communication networks [5]. Simple primitives for elements and ports, and operations for local lookup of IP headers or remote lookup of DNS servers, were used to express the entire network stack and several routing protocols. Our approach is, however, specifically tailored to the radio station context, and has been used to develop a practically realizable system. [2] has proposed a similar abstraction to represent routers, switches, hosts, etc as devices with communication pipes between them, although the focus was more on the development of a management plane to exercise policies on the network without being concerned about the implementation details. We plan to use these ideas in future work when we will consider live transmission across a network of radio stations, where some stations may open phone connections to each other while others may stream live audio, provided they have good Internet connectivity.

8. CONCLUSION

In this paper, we described the ICT needs of community radio stations including the need to integrate mobile phones with community radio, and outlined the design of GRINS to meet these requirements. We also presented an experimental setup to evaluate audio systems, and a formal framework to express and diagnose radio station configurations. Many insights from our work will be applicable to other technological systems slated for deployment in similar developing region contexts. The main message however that we would like to convey is that it is extremely hard (but fun) to build technological systems for developing regions. Constraints presented by the reluctance of users to change their existing practices, careful thought on robustness and cost, and a good understanding of the requirements are all essential to build usable systems. Now that we are meshed closely in the community radio landscape of India, we are looking forward to building more applications to nurture and support the CR movement.

9. REFERENCES

[1] Video Volunteers. http://www.videovolunteers.org/, June 2010.

[2] H. Ballani and P. Francis. Conman: a step towards network manageability. *SIGCOMM Comput. Commun. Rev.*, 37(4):205–216, 2007.

[3] R. Gandhi, R. Veeraraghavan, K. Toyama, and V. Ramprasad. Digital Green: Participatory Video for Agricultural Extension. In *ICTD 2007*, December 2007.

[4] M. A. Hearst. UIs for Faceted Navigation: Recent Advances and Remaining Open Problmes. In *Workshop on Computer Interaction and Information Retrieval (HCIR)*, October 2008.

[5] M. Karsten, S. Keshav, S. Prasad, and M. Beg. An axiomatic basis for communication. In *SIGCOMM 2007*, pages 217–228, New York, NY, USA, 2007. ACM.

[6] Z. Koradia, C. Balachandran, and A. Seth. Design of an Automation System for Community Radio Stations in Developing Regions. http://www.cse.iitd.ernet.in/ aseth/act4d/reports/grins-design.pdf.

[7] Q. Mahmoud. Middleware for Communications. John Wiley and Sons Ltd., 2004.

[8] A. O'Connor. The Miners' Radio Stations in Bolivia: A Culture of Resistance. In *Journal of Communication*, volume 40, 1990.

[9] N. Patel, D. Chittamuru, A. Jain, P. Dave, and T. S. Parikh. Avaaj Otalo - A Field Study of an Interactive Voice Forum for Small Farmers in Rural India. In *CHI 2010*, 2010.

[10] A. Premi. Framework for Inferring and Debugging Community Radio Station Configurations. http://www.cse.iitd.ernet.in/ aseth/act4d/reports/grins-formal-expression.pdf.