System administration as communication over a noisy channel

Mark Burgess

Oslo University College

1 Introduction

Claude Shannon, the father of modern information theory, died on February 24th 2001, one of the most influential scientists of the last century. In one obituary[1], it states that Shannon — in addition to working on codes and transmissions in electrical circuits — wrote his doctoral thesis '*An algebra for theoretical genetics*' on the mathematics of the genetic code[2]. He was, doubtless, not surprised by the subsequent metamorphosis which occurred in biology, changing it from a "hunt for protoplasm" to a study of gene propagation. Today, Shannon is generally associated with his most famous work '*The mathematical theory of communication*'[3], a legacy which is now known as modern information theory[4], but the influence of that theory of communication on biology distilled an important message, which has since pervaded many branches of science. For biology, it was this: life is information, which flows through a noisy ecosystem. The reason information theory is so important is that virtually everything which happens in the world can be viewed as a transfer of symbolic information from one place to another. The same is true of system administration, so it seems fit to honour Shannon's work by viewing it in this light.

2 Communication and preservation

Shannon saw the communication process as being essentially stochastic in nature. His theory of communication was not about meaning, but about the integrity of pattern, just as system administration is, to some extent, about the integrity of machine configuration. Since the conveyance of meaning requires symbols or tokens to pass from sender to receiver, he viewed the study of patterns as being more fundamental than the meaning they imbued. This abstract view has allowed Shannon's theory to inspire many similar discussions, from fundamental physics[5], to studies of computer behaviour[6]. The semantic interpretation of information plays no role; one simply views information simply as a sequence of symbols. In the Shannon paradigm, information originates from a 'source' (originally thought of as modelling some kind of transmitter) and is transmitted over a 'channel' (a cable, a wireless wave, a vibration etc.) to a 'receiver'. The theory is extremely general, however, because, in spite of using these radiophonic terms, the physical nature of these elements is entirely unspecified. Thus any person or object that emits a symbolic signal is a transmitter (human being, computer network card, computer screen, klystron); any medium which carries such a signal is a communications channel (wire, radio-wave, diskette, memory chip, human brain, piece of paper, carrier pigeon); and anything which receives the signal is a receiver (human ear, human eye, network interface, CPU register). The theory of communication is thus, not so much a single theory, as a framework for discussion.

In earlier work, I related this viewpoint to system administration, as a paradigm for explaining the changes to a computer system. By thinking of a system, whose purpose is specified within certain limits, as a system interacting with an environment of users and external inputs[6], the matter of computation becomes one of communication. This viewpoint has been central to the idea behind the system administration tool cfengine[7], for instance, which attempts to communicate a sense of configuration policy over time.

Another analogy which has been used to discuss system administration is that of *immunology*, or the study of how organisms fight disease. This is a controversial subject, even in biology, because no one really agrees on what disease is, or what the immune system does: does it fight any foreign bodies (immigration control), or dangerous objects (pest control)? For the purpose of discussing computer immunology the following interpretation is useful: an immune system regulates system function.

Sickness is not unnatural: nothing unnatural or even abnormal is happening during an illness — all of the processes are normal and even expected; all one say is that during sickness, there is *an unwanted deviation from the intended functioning of a system*. Intended function has to be described by a *policy*, which is partly based on human wishes, and is partly a result of nature's tendency to settle for short term stability, in the face of long term change. For example, there is nothing biologically wrong with a person being eaten by bacteria: we are just raw material and the bacteria are just machines following a simple program. This is the normal functioning of bacteria, and whether or not it is normal in humans, it is certainly not a *desirable* function of the human organism to exist as food for parasites. Nature provides us with immune systems only because the genetic program for organisms is to reproduce, and that requires the organism survive long enough to do so.

We refer bacterial infection as a sickness, because we have our own ideas about what we are 'for', and we would like to preserve that usage. Other bacteria (in our guts, for example) live and cooperate with us as part of 'the organism', assisting us in preserving that policy. Old age, broken bones, viruses and genetic disorders are all considered sickness for this reason, even though they are all normal developments in biological machines. The same thing applies to computer systems: is arbitrary use of a system right or wrong? The system is just doing its job, whether it is executing a computer virus, or whether it is executing a word-processor. The difference between acceptable and unacceptable behaviour is thus a matter of *policy*.

This sounds like a contradiction: if Shannon viewed the theory of communication as being about pattern, not about semantic meaning, then why are we talking about policy? The answer is simple: neither computer systems nor biological organisms care what the policy is that they propagate. All they see is a sequence of coded instructions which they communicate and re-code. What emerges is the tree of consequences resulting from that policy, as it branches due to environmental influences. Organisms have developed a strategy for regulation, against undue alteration of their programming, so that they maintain sufficient stability as to be able to propagate their genes further still. Evolution is testimony to the fact that this propagation is not completely error free.

Let us summarize the contention of this essay: there is a sense in which everything which happens in the world is an expression of communication between different parts of a system. Anything which influences anything else must do so by transmitting some signal or information: this is simply the law of cause and effect.

Information theory is about the *representation* and *transmission* of patterns of symbols or *digits*. Organisms attach meaning or *function* to these patterns, and data are transmitted from a *source* to a *receiver*, over a communications *channel*. The theory of information seeks to answer questions such as how rapidly or reliably the information from the source can be transmitted over a channel. Shannon's work is significant, because it addresses fundamental limits on the ability to resolve and transmit information. He addressed the problem of coding of symbolic information, in such as way as to maximize the rate and fidelity of transmission, given that the representation might change. He also achieved considerable success with his technique of random coding, in which he showed that a random, rather than a sequential, encoding of data can, with high probability, give essentially optimal performance.

3 How is system administration communication?

The process of communication is essential in any information system. System administration is no different; we see essential bi-directional communications taking place in a variety of forms:

- Between computer programs and their data,
- Between computers and devices,
- Between collaborating humans (in teams),
- Between clients and servers,
- Between computer users and computer systems,
- Between policy decision makers and policy enforcers,
- Between computers and the environment (spilled coffee).

The intent of these communications is constantly being intruded upon by a environmental noise. Errors in this communication process can occur in two ways:

- Transmission: Information is distorted, symbols are changed, inserted or omitted, by faulty communication, or by external interference,
- Reception: Information in interpreted incorrectly; symbols are incorrectly identified, due to imprecision or external interference.

For example, suppose one begins with the simplest case of a standalone computer, with no users, executing a program in isolation. The computer is not communicating with any external agents, but *internally* there is a fetch-execute cycle, causing data to be read from and written to memory, with a CPU performing manipulations along the way. The transmission of data, to and from the memory, is subject to errors, which are caused by electrical spikes, cosmic rays, thermal noise and all kinds of other effects. These errors are normally corrected by error-correction mechanisms, originating from Shannon's work. The computer program itself, manipulates the data in the memory and rewrites it to the memory with a new coding. From this point of view, one may think of the memory of the computer itself as being both transmitter and a receiver, and passing through the 'CPU plus computer program' communication channel. This communication channel does not transmit the data unaltered; rather, it transforms the data according to specific rules, laid down in the computer program (see fig. 1). In other words, the very operation of a computer fits the paradigm of communication over a coded channel. The fact that the channel is also noisy, is a result of the physical environment. Computer operation, at this level, is largely *immune* to environmental noise, because it employs error correction methods. At higher levels, there are no standardized error correction mechanisms in common usage.

Suppose now that an administrator sends a configuration message to a host, or even to a single computer program. Such a message takes place by some agreed form of coding: a protocol of some kind, e.g. a user interface, or a message format. Such a configuration message might be distorted by errors in communication, by software errors, by random typing errors. The system itself might change, during the implementation of the instructions, due to the actions of unknown parties, working covertly. These are all issues which contribute uncertainty into to the configuration process and, unless corrected, lead to a 'sickness' of the system, i.e. a deviation from its intended function.

Consider a straightforward example: the application of a patch to some programming code. Programs which patch bugs in computer code only work reliably if they are not confused by external (environmental) alterations performed outside the scope of their jurisdiction. If a line break is edited in the code, in advance, this can be enough to cause a patch to fail, because the semantic content of the file was distorted by the coding change (noise). One reason why computer systems have been vulnerable to this kind of environmental noise, traditionally, is that error correcting protocols, of sufficient flexibility, have not been available for making system changes. Protocols, such as SNMP or proprietary change mechanisms, do not yet incorporate feedback checking, of the higher level protocols, over extended periods of time.

The idea of *convergence* was introduced by the configuration engine cfengine[7], precisely in order to implement such extended error-correction. Cfengine's idea of convergence, is rather like that a biological system, in that its view of error correction is a dynamical one. Cfengine exerts a process of continual regulation, in order to correct deviations from system policy, bother in terms of configuration state and in terms of competition between different subsystems. In other words, it does not simply ensure that a configuration message is transmitted correctly once, it views the entire time-development of the system as the on-going transmission of the message, and seeks to correct it at every stage.

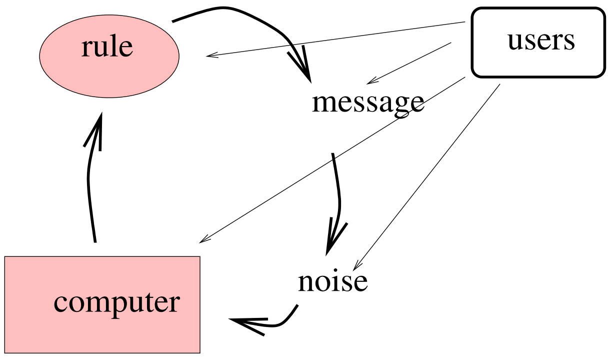

Figure 1: An instruction loop, showing the development of a computer system in time, according to a set of rules. The efficacy of the rules may be distorted by users from local and remote domains, who change the conditions under which the message was applicable. This change may be viewed as an intentional change, or as a stochastic error.

At each level of computer operation, one finds messages being communicated between different entities. System administration is a meta-program, executed by a mixture of humans and machines, which concerns the evolution and maintenance of distributed computer systems. It involves:

- Establishing guidelines,
- Configuring systems within policy guidelines,
- Keeping machines running within policy guidelines,
- Keeping user activity within policy guidelines.

Clearly, the scope of system administration is rather ambitious. Indeed, it is reminiscent of the task faced by biological organisms in maintaining their health in an environment of noise and unpredictability[8]. There are also many links between system administration and work carried out in the Human Computer Interaction community[9, 10].

4 Semantic and stochastic

There are two approaches to studying this dynamical information system: an immunological approach, where one attempts to discern some semantic meaning from the communication in order to filter out unwanted elements[11], or a stochastic approach, in which one uses generic strategies (à la Shannon) to analyse and regulate information flow[6, 7]. These are complementary viewpoints, because they discuss the same essential communication problem at different levels.

System administration requires computer-computer interaction, human-computer interaction, and human-human interaction (fig. 2). Each of these communication channels is subject to error, or misinterpretation.

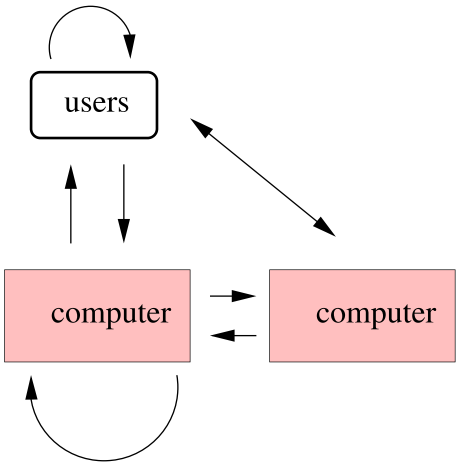

Figure 2: The human-computer interaction is also a form of communication. Collaboration between system administrators, or users is the least reliable form of communication, since one can not be sure that the two parties even use the same symbolic alphabets. Thus, there is the danger not only of noise, but also of misunderstanding, i.e. perversion of meaning.

- **Instruction**: system administration is about developing a policy (instruction manual for machine and humans) for use of the system, including the choice of programs which are executed by the system. A complete policy can therefore be identified with the sum of all programs and ad hoc rules which lead to any planned change of the system. This includes normal usage of the system.

- **Propagation**: as the system evolves in time, according to policy (or not), it propagates information from the past into the future, modifying the system. If the system is stable, this iterative mapping will not lead to any divergent behaviour; if it is unstable, then even a small error might cause a runaway breakdown of the system.

- **Collaboration**: programs and humans exchange information in the course of this loop of instruction and propagation. If collaboration is interrupted, or errors occur, then the enactment of policy-correct behaviour is spoiled, with possibly dangerous consequences for the system.

- **Automation**: automatic processes which monitor systems, even perform routine maintenance, are not immune to errors, because they depend, for input, on data which are influenced by the external environment.

- **Reparation**: if some influence causes an error, then the error must be corrected in order to uphold policy, else the correct propagation of policy over time is corrupted. The same thing applies to human policy and purely automated policy. This process of maintenance, reparation, or regulation is central to the stability of information systems.

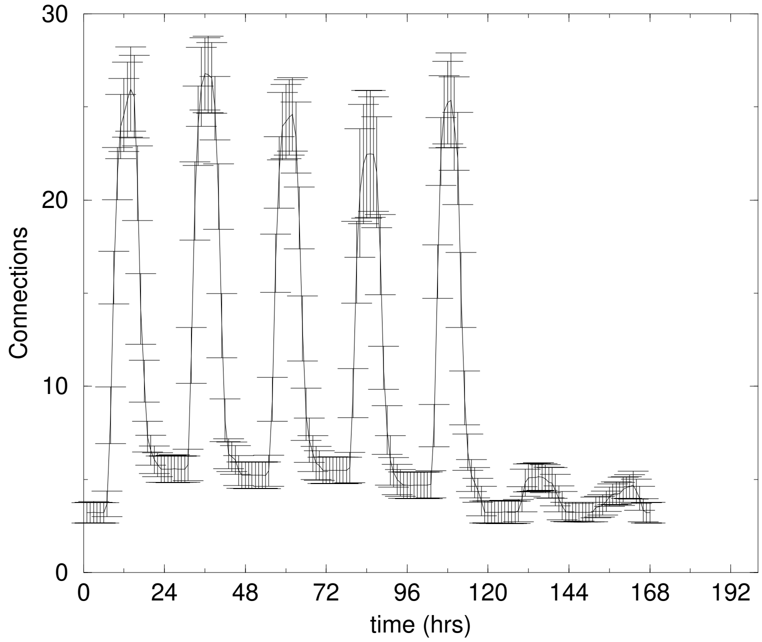

Figure 3: An example of computer behaviour is the number of connections to a Samba server, serving user home directories, over time. This graph shows that there is a definite average pattern to the behaviour, but the error bars indicate that there is random noise in the signal. The graph shows a period of several weeks, plotted om a periodic graph, from Monday to Sunday. The average (mean) behaviour (solid line) has a clear pattern, but the actual measurements are randomly distributed around the mean. The error bars denote the standard deviation of the data, at each given time of week.

Apparently we are interested in having computer systems which converge to stable averages of communication, or at least ones which do not diverge[12]. This idea has been promoted especially in connection with the system administration agent cfengine[7]. To summarize, one may view system administration as communication over a communications channel at several levels.

- **Input**: policy, instruction.
- **Noise**: stochastic user activity, illegal behaviour, random error, systematic error.
- **Output**: the system as we see it.

Having made this identification, the question becomes: does this help us to build computer systems which behave in a stable and predictable fashion? Biological organisms have significant maintenance and regulatory systems, which uphold a genetically programmed policy (subject to a small drift), by attempting to eliminate errors by repair, immune cleansing of biological noise (parasites, bacteria, viruses).

5 Some communication theory

Is it true that computers are stochastic systems? Are they not, after all, governed by users' decisions, which are far from random? This argument is a naive one. While it is clear that individual users behave in an apparently non-random fashion, a random mixture of users,

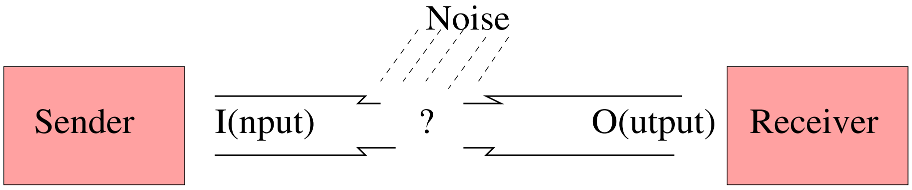

Figure 4: The transmission of information over a channel takes place from a sender (channel input) to a receiver (channel output). Along the way, the signal might become distorted by noise.

working at random times leads to random behaviour. Note also that random does not imply that there is no pattern to the behaviour (see fig. 3), that is not true at all. At Oslo University College, we have studied the interaction between local and remote users and machines , and have found them not only to be random in nature, but maximally random[6].

Imagine a signal sent into the input of a communications channel and transmitted to the output, with some level of reliability. In order to talk quantitatively about this process, we need to introduce some quantities. We associate meaning primarily to discrete events, or symbols. A signal $f(t)$, digitized into symbols, or digits, by a digitization procedure (see fig. 5).

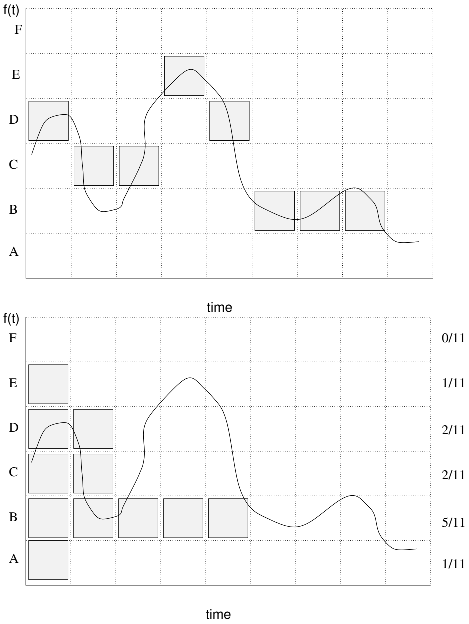

Figure 5: Coarse-graining or digitization is a labelling of the signal. First we divide up the signal variability into discrete symbols or finite sized blocks (first picture), and then we approximate the true signal to the coarse representation. Collapsing the digit-blocks into a histogram, by counting the number of occurrances (second picture) gives a graphical representation of the average numbers of digits in the signal. By scaling this histogram, one obtains a probability distribution for the signal. This is used to calculate the entropy, or average information content.

Information is coded into the channel by the variation of the signal. This is digitized into a string of digital symbols (called a message) which occur in a certain order, with a particular frequency. When a digit occurs n_i times out of a total of N digits, where

$$\sum_i n_i = N, \tag{1}$$

one may say that the probability of the symbol's occurrence was

$$p_i = \frac{n_i}{N}. \tag{2}$$

The probability distribution of symbols in the signal is found by collapsing the image of a signal's symbol series into a histogram (see fig. 5). Note that, for simplicity, it is assumed that only a single symbol can occur at any one time, i.e. that the channel is a serial channel.

This probability distribution measures the average amount of information in the message being transmitted. This information is measured by the *entropy*.

$$H[p] = -\sum_{i=1}^{A} p_i \log_2 p_i, \tag{3}$$

where A is the number of symbols in the alphabet.

The entropy is a single scalar characterization of a probability distribution. Since it collapses all of the changes of the signal into a single number, it cannot distinguish equivalent signals, i.e. signals with the same frequency distributions of symbols in them. However, its interpretation as a transmission rate is very useful for considering the efficiency of communication.

Suppose we consider the transmission of information over a communications channel, then there is information at the input of the channel and at the output. Under ideal conditions, the average information entering the input would be the same as the average information leaving the output; however, this is not necessarily the case if the channel is affected by noise. (Note that it is the average information, or sum of symbols over time, which is important, since one might choose to code information such that information does not necessarily arrive in the same sequence which gives it meaning – this is true at the lower levels of the TCP stack, for instance.) Noise is simply unpredictable information, which enters the system from the environment and changes some of the symbols.

Fig. 6 shows how the distribution of symbols in a message can be altered by its transmission over a channel. The entropy may be used to characterize the average information content at the input I and at the output O.

$$\begin{aligned} H[I] &= -\sum_{i=1}^{A} p_i(I) \log_2 p_i(I), \\ H[O] &= -\sum_{i=1}^{A} p_i(O) \log_2 p_i(O), \end{aligned} \tag{4}$$

where $p_i(I)$ are the probabilities at the input, and $p_i(O)$ are the probabilities at the output. Note that I and O label *sets* of symbols, whose elements are the alphabet of digits.

In order for the message to be transmitted sensibly, the transmission would normally be a *congruent mapping* of symbols. Usually the input and output alphabets are assumed to be the same; however, if the channel is encrypted, compressed or otherwise encoded, this need not be the case. For now, we shall assume that the number of symbols is A in both cases, and that the symbols map congruently, in the same sequence. If tranmission of information is perfect, then,

$$H[I] = H[O]. \tag{5}$$

This can only occur, if the information extracted from the output is predicated by the information at the input, i.e. if they care correlated. If $p(I)$ and $p(O)$ are independent events,

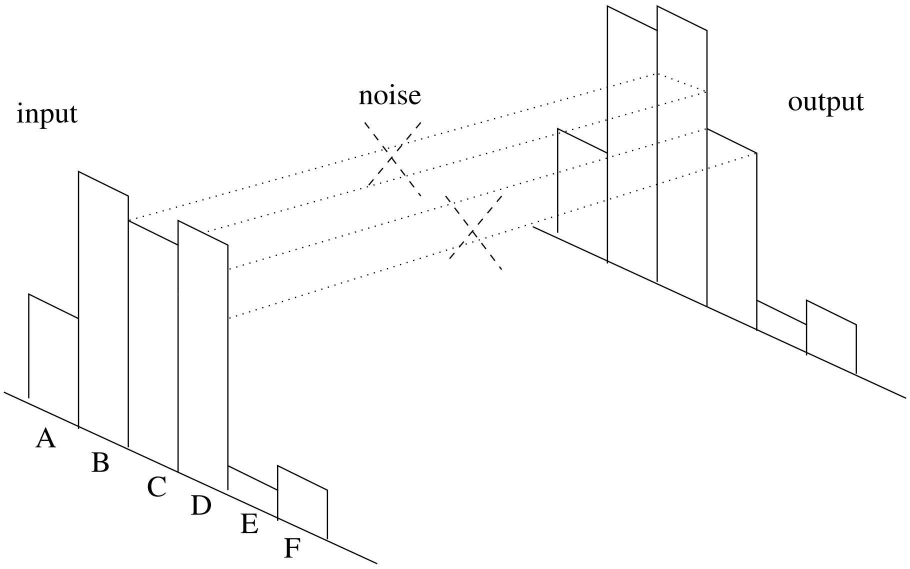

Figure 6: Information passing through a noisy channel. The frequency distributions of symbols can be altered by noise introduced by the environment, as the data pass through the channel. Ideally, these two distributions would be equal. In the worst case, they would have nothing in common.

this cannot be the case, since output and input would be unrelated. This is clearly undesirable. It is natural to apply joint probabilities to the input and output of a communications channel. Since we are interested in communication, the worst case scenario is when the probability of an event at the output occurs completely independently of an event being sent at the input. In that case, the output is purely 'noise', and the only chance of seeing the same data at input and output, is when there is a *coincidental* overlap

$$p_{ij}(I,O)_{\mathrm{worst-case}} = p_i(I)p_j(O). \tag{6}$$

The ideal case is when a transmission is reproduced with perfect fidelity; i.e. $p_{ij}(I,O)$ is a diagonal matrix

$$p_{ii}(I,O)_{\mathrm{best-case}} = p_i(I) = p_i(O), \tag{7}$$

i.e. the probability distributions at input and output are identical.

Using the joint probability of events at the input and output, of a communications channel, we can construct a *joint entropy*,

$$H(I,O) = -\sum_{i,j=1}^{A} p_{ij}(I,O)\log_2 p_{ij}(I,O), \tag{8}$$

which measures the information in the whole channel.

The final measure of information is called the relative entropy or *mutual information*:

$$H(I/O) = H(I;O) = +\sum_{ij} p_{ij}(I,O)\log\frac{p_{ij}(I,O)}{p_i(I)p_j(I)}. \tag{9}$$

The semi-colon is used to indicate that this quantity is a symmetrical function of the input and output, i.e. the information is reversibly usable. It measures the difference between a source and a receiver which are completely independent (Q) and a general source and receiver, connected by a partially reliable communications channel. Using the definitions, it can be expressed in a number of different ways.

$$H(I;O) = H(I) - H(I|O) \tag{10}$$

This is the information (uncertainty plus signal) at the input minus the uncertainty in the input, given a definite output. in other words, it is the part of the input which was transmitted to the output, or the likelihood of transmission (the fidelity). Another representation is:

$$H(I;O) = H(O) - H(O|I). \tag{11}$$

This is the information arriving at the output (signal plus noise), minus the uncertainty information at the output, given a fixed input (i.e. the noise picked up along the way). Again, this is the likelihood of correct transmission. A third form is

$$H(O;I) = H(I) + H(O) - H(I,O). \tag{12}$$

This is the sum of information entering the input and leaving the output, minus the total amount of independent information in the system. What is left, must be the information in the overlap region (see fig 7), i.e. the information which is common to both input and output. For system administration, one is looking for a scheme for propagating policy

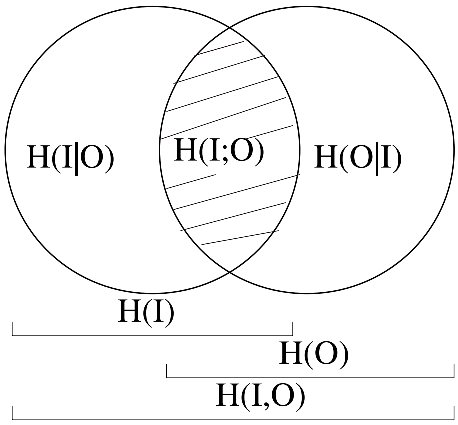

Figure 7: The entropies visualized as measures on the space of all possible input and output signals. The Venn diagram shows where the mutual information received support from the input-output probabilities; it does not strictly represent the magnitude of the entropy.

throughout system evolution. In this case, we identify $H(I)$ as the information in policy description and initial state, $H(O)$ as the outcome after running and using the system for a given time interval, and $H(I;O)$ is thus the *compliance* with policy.

6 Coding

Although, at their lowest level, computer systems are essentially bit-configurations, operated on by rules over time, this viewpoint ignores an important issue, which is the level of granularity with which the user interacts with it. Even the simplest ASCII symbol requires a minimal representation of 7-bits; most data of significance require complex representations which require one to deal with high level objects such as files and directories. The coarser the level of a system's coding, the lower the probability that lower level noise will affect it and cause errors, because the classification into robust symbols is relatively insensitive to small variations. Coarser levels also mean that retransmission of erroneous symbols is easy.

The transition from low level to high level is accomplished by successive levels of abstraction, such as the hierarchy of objects expressed in the table in table 1. From a system administrators point of view, this hierarchy determines the scope and extent of policy, and also the type of response required when dealing with errors.

Level	Example objects
6	Groups, departments, LANs
5	Users, virtual machines, agents
4	Compound objects, files, processes and attributes
3	int, float, char etc
2	bytes, words
1	bits (RAM,ROM,disk)

Table 1: Coding levels in a computer system

The objects in the table represent new effective entities of the system; they form new and higher alphabets of non-overlapping objects. The state of higher level system objects demands a new set of symbols, for each level, in order to convey and perhaps correct information about the state. Information about these levels is coded separately for each layer in the hierarchy (as, for instance, in the Open Standards Interconnect (OSI) model of network layers). For example, the symbol alphabet for correcting byte-transmission requires only two symbols: ONE and ZERO; the symbol alphabet which describes information in a plain text file consists of all of the printable ASCII symbols; the set of symbols required to implement a policy decision about file permissions is the set of all rules (one for each item), which for Unix might look like this:

```
files:

 # symbol 1

 file1 mode=0644 owner=mark group=users action=fix

 # symbol 2

 directory1 mode=a+rX owner=root recurse=true action=fix
```

In this case, each entire rule is a symbol of the higher level policy alphabet, and when it is coded in this fashion, it is more robust than trying to deal with each of its sub-components (all of the possible combinations of explicit file permissions, different owners etc.). It can be viewed as an alphabet rather than a language, because it has a limited, denumerable set of values in a given policy. This alphabet can be combined into higher grammatical forms, in order to code a complete policy, but the basic objects are denumerable, because of the finiteness of the system. As a compressed code, the grammar of the language has little meaning. For this reason, it is possible to make these atomic operations of the cfengine policy alphabet very robust, by building in ‘convergence’ or error-correction features to each symbol.

This kind of coding of policy, is robust to accidental or random error, and is easy to reapply (retransmit), should external factors alter its result. Thus policy integrity is served by expressing policy in terms of a robust, coarse alphabet. This, of course, is the reason why iconic user interfaces are popular: they are robust to error, because they offer very limited choice. They are alphabetic rather than grammatical.

The layers carve up the system into slices, which, in turn, affect the way that policy, at that level, is encoded. In short, the resources of the computer may be viewed at many different levels, from bits to abstract data-types, files, programs, volumes etc. This procedure of defining entities which express meaning at different levels is called *coding*. Coding of high level objects, such as 'jobs' or tasks, is usually called *scheduling*, in computer science. A coding scheme is a definition of entities and a schedule for their transmission. In our work at Oslo, we have examined how effective random strategies for enforcing policy can be. It is not only 'spot checks', rather than systematic parsing of a structure, which lead to efficiency, but also randomization of policy details, schedules and content about some mean. One of Shannon's conclusions was that, if one splits up a signal into a coding scheme which is randomized in ordering, then many this is a good defence against random noise, because any systematic error will be reduced to random error.

7 Error correction

One of Shannon's accomplishments was to prove that any communications channel has a limited capacity, and that it is possible to find a coding scheme which achieves reliable transmission of symbols with an efficiency which is arbitrarily close to that channel capacity. This is known as Shannon's theorem. Shannon imagined a generic model for transmission over a noisy channel, shown in fig. 8. This model can also be applied to the transmission of *policy*, or system *health*.

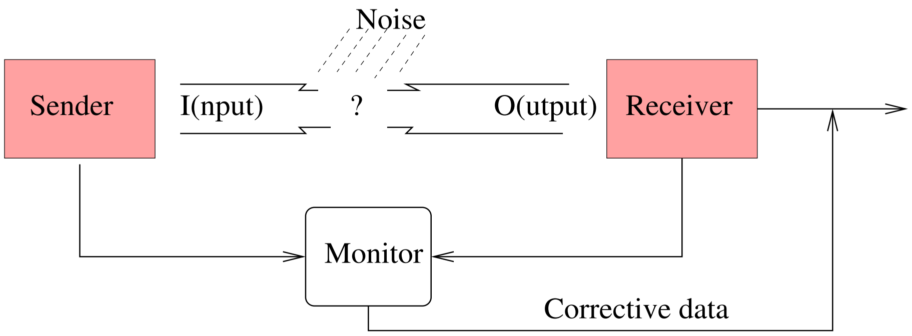

Figure 8: Shannon's view of the correction of data on a coded channel, can also be applied to the correction of policy propagation, or system 'health'. It makes clear the need for a correctional system.

Errors creep into the transmission of rules and actions, even with coarse digital channels. One of the reasons for introducing graphical user interfaces was to reduce the possibility of error, by condensing difficult operations into a few simple symbols. In spite of the effort, users still hit the wrong icon, or menu item, and the simplifications which are assumed do not always apply, because external conditions invalidate the assumptions that were made in selecting the symbols.

Shannon's theorem, however, tells us that a suitable coding scheme (protocol) one can assure the transmission of a system policy to within arbitrary accuracy. Windowing systems use error correction protocols, such as 'Are you sure?' dialogue boxes, in order to catch random errors — not usually because they believe that users will change their minds very often. Cfengine uses its notion of 'convergence' to check and correct errors in policy actions, with respect to the symbols coded by the user.

The signal we wish to propagate is this:

$$\text{Ideal signal} = H(\text{user}|\text{policy}) \tag{13}$$

Policy can be communicated by declaration[7, 13], or by simulated dialogue[14, 15]. The latter is becoming more common, with technologies like SNMP [16] and XML based services [17]. In either case, its transmission or implementation through fallible humans and technologies is subject to the incursion of 'noise'.

The issue now, at this high level, is more subtle than for simple bits however. Do we really want to achieve such a level of precision as to lead to no errors at all? As long as humans are part of the equation, there is the question of user-comfort and human welfare. Machine configuration issues can be dealt with, using explicit error correction, but where humans are involved, the situation is less reliable. Clearly 'sweat shop' styled policy enforcement could force users to behave rigidly in accordance with policy, but we do not normally mean for every aspect of policy to be interpreted this rigidly. As humans, we like to distance ourselves from machine-like conformity; we are creative and need a certain freedom in order to prosper. In biology, at least, it is fortunate that perfect transmission has never been achieved, else there would be no adaptive development or *evolution* of the system; humans would not even exist. Thus, in order to apply error correction to larger social and ecological systems, we must choose the symbols rather carefully. Unlike Shannon, we must pay attention to the semantic content of symbols when formulating policies which one wishes to have enforced[9]. By observing this semantic content, one finds a behavioural model of regular trends in the environment, which can be exploited.

Proactive error correction is one way of dealing with this issue: by requiring confirmation of a policy breach, one can avoid spurious over-reactions to acceptable transgressions. For example, the double keys used in nuclear missile launches, or the double signals used in Co-stimulation of the immune response, are simple security features which prevent potentially damaging responses to innocent errors. Hamming codes and checksum confirmation are other examples of this type of protocol coding.

8 End remarks

Computer science often assumes that computers can only be understood within rigid frameworks of logical propositions, and that what has once been decided must remain until future management decisions are implemented. Artificial intelligence researchers have long abandoned such a view; however, the belief persists amongst system designers and system administrators. In this paper, I have tried to draw attention to the *unpredictable* element of computer systems. Shannon's view of communication provides an interesting viewpoint on this problem, with an important message. We should be looking for protocols and systems which have a built-in sense of correctness, and can rectify random and systematic errors.

Cfengine was the first system administration agent to perform continual error correction; others are on the way. Intelligent SNMP agents are being proposed, for instance, which will not only set configurations, but also maintain their integrity. Even so, these are only the beginning, inhabiting only the most primitive level of error description.

If policy is to be preserved, or communicated into the future without error, one needs error correcting mechanisms which are effective at counteracting all kinds of noise. Graphical user interfaces, and smart interpreters can protect against simple human error, but the environment which generates noise is vast, and the sources of error are many.

If sickness is the deviation of a system from its intended purpose, then it is all a question of timescales. The challenge with living organisms, and extended computer systems is that what is normal and healthy depends very much on the scale at which one views the system. Biology enjoys local stability, but long term fragility; its protection mechanisms have only evolved to keep organisms stable for long enough to fulfil the execution of their program: to reproduce. In doing so, they contribute to an ecology which also enjoys local stability, but long term fragility, through communication between organisms. The message here is rather like that of Dawkins' *Extended phenotype*[18], namely that the success of an organism in carrying out its program is not only dependent on its own properties, but on the properties and policies of all the organisms it is in contact with. We do not need to possess the genes which make bacteria break down waste, in order to be dependent on them for our efficient functioning or even survival. In an environment, one cannot think of systems in isolation.

What makes the protection of "life" and the administration of computer systems challenging is the ambiguity of purpose. In any feedback system, the intended function of the system is always changing in response to the environment. It is wrong to think of static policies and standalone entities, oppressed by their environments. It is right to think of them as dynamical entities, participating in an adaptive ecology. Computers, as organisms, exist to fulfil some functional niche in a higher level system, so change in response to the broader scale conditions is not only desirable, but inevitable. How then is it possible to speak of health and sickness, when the very conditions of stability are changing?

The definition of health seeks to promote the concept of short-term stability, so that a system can complete some kind of orderly function, in spite of the environment it is coupled to. The world is a noisy channel, and error correction is a necessary part of stability in a complex system. Without it, chaos will so easily reign. Error correction, or immune response, is thus about providing information systems with stable patterns of behaviour, to which we can attach consistent meaning. Integrity of purpose is the challenge for system administration.

Acknowledgement

I am grateful to Trond Reitan for suggesting the extended phenotype analogy, and to Alva Couch and Dan Klein for reading drafts of the manuscript.

References

[1] W. Atmar. A profoundly repeated pattern. *Bulletin of the Ecological Society of America*, pages 208–211.

[2] C.E. Shannon. *An algebra for theoretical genetics*. Massachusetts Institute of Technology, Dept. of Mathematics, 1940.

[3] C.E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.

[4] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. (J.Wiley & Sons., New York), 1991.

[5] J. Schwinger. *Particles, Sources and Fields (Volume 1)*. (Addison Wesley, California), 1970.

[6] M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems (to appear)*, 2001.

[7] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[8] M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998.

[9] L.A. Zadeh. Outline of a new approach to the analysis of complex systems and decision process. *IEEE Transactions on Systems, Man and Cybernetics*, 3:28–44, 1973.

[10] T.B. Sheridan. Allocating functions among humans and machines. *In D. Beevis, P. Essens and H. Schuffel (Eds.), Improving Function Allocation for Integrated Systems*

Design, Wright-Patterson Airforce Base, CSERIAC State-of-the-Art Report:179–198, 1996.

[11] A. Somayaji, S. Hofmeyr, and S. Forrest. Principles of a computer immune system. *New Security Paradigms Workshop, ACM*, September 1997:75–82.

[12] M. Burgess. On the theory of system administration. *Submitted to J. ACM.*, 2000.

[13] N. Damianou, N. Dulay, E.C. Lupu, and M. Sloman. Ponder: a language for specifying security and management policies for distributed systems. *Imperial College Research Report DoC 2000/1*, 2000.

[14] D. Libes. Using *expect* to automate system administration tasks. *Proceedings of the Fourth Large Installation System Administrator's Conference (LISA IV) (USENIX Association: Berkeley, CA, 1990)*, page 107, 1990.

[15] A. Couch. An expectant chat about script maturity. *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV) (USENIX Association: Berkeley, CA)*, page 15, 2000.

[16] S. Omari, R. Boutaba, and O. Cherakaoui. Policies in snmpv3-based management. *Proceedings of the VI IFIP/IEEE IM conference on network management*, page 797, 1999.

[17] XML-RPC. Internet remote procedure call. *http://www.xmlrpc.com/spec*.

[18] R. Dawkins. *The Extended Phenotype*. (Oxford University Press,, Oxford), 1990.