Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

TRANSFERRING INFORMATION BY RINGING A CELL PHONE

Master's thesis

2014

Bc. Martin Šrámek

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Transferring information by ringing a cell phone

Master's thesis

Study program:	Informatics
Field of study:	2508 Informatics
Department:	Department of Computer Science
Supervisor:	RNDr. Michal Forišek, PhD.

Bratislava, 2014

Bc. Martin Šrámek

10523974

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Martin Šrámek
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický

Názov: Transferring information by ringing a cell phone
Prenos informácie prezváňaním mobilného telefónu

Cieľ: Možnosť zadarmo prezvoniť mobilný telefón poskytuje užívateľom mobilnej siete postranný kanál použiteľný na prenos informácie. Hlavným cieľom tejto práce je čo najpresnejšie zistiť prenosovú kapacitu tohto kanálu. Práca by mala obsahovať praktickú analýzu tohto kanálu a následne na jej základe spravený jeho teoretický model, ktorý bude ďalej skúmaný. Súčasťou práce by mal byť samotný návrh komunikačného protokolu, vrátane preskúmania rôznych možností kódovania informácie a analýzy použitia vhodného samoopravného kódu. Vítaným prínosom by bola následná praktická implementácia dosiahnutých výsledkov. Ako možný ďalší cieľ nad rámec práce navrhujeme preskúmanie scenárov zahŕňajúcich viac ako dvoch účastníkov.

Vedúci:	RNDr. Michal Forišek, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.
Dátum zadania:	14.11.2012

Dátum schválenia: 19.11.2012

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

....
študent

....
vedúci práce

Acknowledgement

I want to express thanks to my supervisor RNDr. Michal Forišek, PhD. for presenting me with an interesting topic and for his advice and guidance during my work on this thesis.

I also wish to thank my parents for their amazing support, my girlfriend Katka for the same and for lending me a phone to test on, and to my friends for their plentiful comments on what they would like to see in the application.

Abstract

Author:	Bc. Martin Šrámek
Title:	Transferring information by ringing a cell phone
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Supervisor:	RNDr. Michal Forišek, PhD.
Number of pages:	74

Ringing a phone without answering the call can be viewed as a side communication channel in the cell phone network. We defined a theoretical model describing this channel and proposed several protocols that take advantage of it to transmit arbitrary data. We estimated the capacity of the channel and showed that our protocols reach near-optimal efficiency. As a proof of concept, we also created an Android application that uses ringing to send text messages.

Keywords: ringing, phone, side channel, Android

Abstrakt

Autor:	Bc. Martin Šrámek
Názov práce:	Transferring information by ringing a cell phone (Prenos informácie prezváňaním mobilného telefónu)
Univerzita:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci práce:	RNDr. Michal Forišek, PhD.
Rozsah:	74 strán

Prezváňanie telefónu bez prijatia hovoru sa dá považovať za vedľajší komunikačný kanál v telefónnej sieti. Vytvorili sme teoretický model popisujúci tento kanál a navrhli sme niekoľko protokolov, ktoré ním umožňujú prenášať ľubovoľné dáta. Odhadli sme kapacitu tohto kanálu a ukázali sme, že naše protokoly dosahujú efektivitu blízku optimálnej. Ako dôkaz praktickej použiteľnosti našich výsledkov sme vytvorili aplikáciu pre Android, ktorá pomocou prezváňania dokáže posielať textové správy.

Kľúčové slová: prezváňanie, telefón, vedľajší kanál, Android

Contents

Introduction

Mobile phone usage has been spreading rapidly during the last two decades. Communication protocols are being replaced by new ones, bringing ever more services for the end user. While the primary purpose of a telephone is, as the name suggests, audio (voice) transfer, mobile telephones now have long been able to send text messages. Modern mobile telephones, so called smarthpones, are full-fledged computers, which, thanks to the internet access, now allow basically any kind of data transfer one may think of.

There is, however, an information channel that has always been in use with telephones, yet it is obscure and unexplored. It is hidden in the process that telephones use to notify the callee of an incoming call - ringing. This was apparently never intended for information transfer, but it can be used for such a purpose. The simplest example of this may be ringing someone to notify them of one's arrival. Another one could be sending a bit of information by a scheme such as "one ring means yes, two rings means no". However, as we will see, there are more complicated ways to utilize this channel, which result in information being sent more effectively. The primary task of this thesis is to explore just about how effectively can this be done. We will do so by proposing various protocols that can be used to convey information and measure their bitrate. We will then try to estimate the maximum bitrate that the channel allows to see how effective our protocols are.

Regarding the OSI model of network architecture, the protocols we discuss will be at the application layer (i.e. we will only be interested in using ringing as a black box, and not in the workings of underlying protocols such as GSM). We will analyze the proposed protocols in a rather abstract way, using variables to represent the costs of operations. We will then use empirical data to understand how the protocols really fare in practice. While we are unable to do a large scale experiment that would bring exact data for a statistically significant number of different devices and networks worldwide, we at least aim to estimate the order of magnitude and approximate intervals of values which the considered variables may take.

To put the results from the analyses to use, we will implement our protocols in the form of a smartphone text messenger application. This should be a simple application allowing the user to first choose a protocol, then type in the called number, then the text to be sent and ultimately commence the transmission. As the smartphone has no way of knowing that the incoming call is not a regular call, the callee will have to have the application already running at that time. However, note that this is not a problem, because the caller may inform the callee of the incoming transmission by ringing them manually (non-programmaticaly) first, an agreed amount of time in advance. As a platform for the application, we chose the open source operating system Android. We are going to use the Java programming language.

Our motivation for this work is to investigate a communication channel that has not yet been studied. One may argue that this channel is unimportant, as the standard communication channels provided by mobile phones - calling, text messaging and nowadays even mobile internet access - offer a considerably higher throughput. However, we find theoretical interest in the idea that even such a nonstandard channel, basically existing by accident, may be taken advantage of in a much more sophisticated way than the two examples mentioned earlier. And it may be exactly this nonstandardness and obscurity that offers a use case for such a channel - for example, one may avoid eavesdroppers by not only enciphering the message, but more importantly by hiding the fact that the communication ever took place. If a phone line is being monitored and calls recorded, then never picking up a ringing phone might not trigger the recording itself.

Apart from this, there is a much more apparent usage for the ringing information channel - an economically motivated one. That is, mobile network operators usually charge the phone user only if the call is actually picked up, not solely for ringing. One may then use their mobile phone to send short messages free of charge. In this thesis, we will try to show that this can be done at a non-negligible bitrate, although it will probably still take a lot more time than a standard, nonfree service provided by the operator. By also implementing a messaging application, we make it possible to test this idea in practice.

Finally, we believe that the network operators themselves may find interest in the topic researched in this thesis. It illustrates how well exploitable an accidentaly created channel in the mobile network mechanism can be. While it is improbable that these exploits would persuade common phone users to turn from fast (high bandwidth) but paid-for phone services in favour of a free but considerably less capacious ringing communication, it is useful to know that the mobile phone network is susceptible to such exploits.

Chapter 1

The model

Before we can devise any protocols, we must first formalize our notion of a cell phone and its capabilities. Keeping in mind our intent to apply our findings by implementing a communication application for the Android system, we will limit ourselves to the possibilities of Android API.

The communication is done between two participants - the *sender* and the *receiver*. We will disregard the possibility of the communication being interrupted by unrelated calls from other cell phone network users. The sender is sending information to the receiver. We will focus on evaluating the achieved bitrate, computed as

$$r = \frac{n}{t_n}$$

where n is the number of bits sent and t_n is the time spent doing so. Note that while t is obviously a function of n, it is not necessarily a linear function - in other words, the time to send one bit may vary. It is therefore more useful to define bitrate as

$$\lim_{n\to\infty} \frac{n}{t_n}$$

which may be understood as the time spent per bit if we are sending an unlimited amount of data. In our computations, we will always consider this to be the case and therefore we will never include any explicit termination in our protocols - after the communication started, it will continue indefinitely. As bitrate will be typically less than $1s$ in our case, it will often be more comfortable to speak about the bit transmission time, computed as

$$\frac{1}{r} = \lim_{n\to\infty} \frac{t_n}{n}$$

Although we have now dealt with the problem of varying time per bit, there is still some ambiguity left in our definition of bitrate. Time needed to send a bit may vary according to the bit's value - in some protocols, sending 1 may take longer than sending a 0, or vice versa. This makes it impossible to determine the time needed to send a certain message while only knowing its length, but not contents.

Therefore, in our analyses, we will usually measure the average bitrate, i.e the expected time needed to send a bit. For this, we will assume that each message (a sequence of 1s and 0s) is equally likely. This could be equivalently formulated as that every bit is a random variable from Bernoulli's distribution with probability $p = 0.5$. This is a reasonable assumption, because even if this is not true about our actual distribution of messages, it can be achieved by compression. Prior to transmitting the data, we may apply any good compression algorithm on the message. The probability distribution of compressed messages tends to be approximately uniform. We may therefore assume that our application will always compress the messages and that this is done in a higher layer of the OSI model, so we will never have to deal with it explicitly. Of course, achieving uniform distribution aside, this is useful in its own sake, as shrinking the transferred data simply makes communication more effective.

In practice, the participants would need to include some kind of a preamble to every communication to denote that this is a ringing communication and not a regular phone call, to select a protocol and its parameters etc. Furthermore, some protocols may start with silence (i.e. waiting a certain time until the first ring) - in this case, the preamble would be necessary to even inform the receiver that the communication has already started. While such a preamble would be useful in real world applications, we will not include it in our theoretical bitrate computations. The reasons are as follows.

- It is implementation-dependent. For example, if it carries the information about protocol selection, its size depends on the number of protocols available.
- As stated above, we will focus on analysis for $\lim_{n\to\infty}$. Since the time $t_{preamble}$ required by the preamble would be constant (independent of n), its contribution would become negligible anyway.

$$r_{preamble} = \lim_{n\to\infty} \frac{n}{t + t_{preamble}} = \lim_{n\to\infty} \frac{1}{\frac{t+O(1)}{n}} =$$

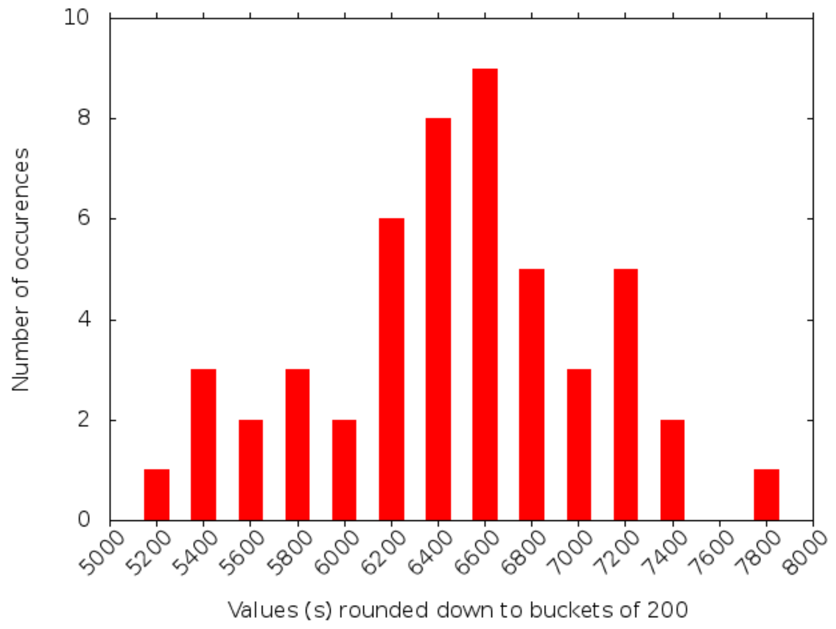
$$= \frac{1}{\lim_{n\to\infty} \frac{t+O(1)}{n}} = \frac{1}{\lim_{n\to\infty} \frac{t}{n} + \lim_{n\to\infty} \frac{O(1)}{n}} =$$

$$\frac{1}{\lim_{n\to\infty} \frac{t}{n} + 0} = \frac{1}{\lim_{n\to\infty} \frac{t}{n}} = \lim_{n\to\infty} \frac{n}{t} = r$$

1.1 Parameters

If we are to calculate the bitrate, we need to know how much time do some elementary operations (e.g. dialing time) take. However, such values will inevitably vary for different cell phone devices, their cellular network connections, and even between individual calls. These variables will constitute the parameters of our model.

We will usually express the quantitative properties of studied protocols in terms of these variables. However, to put things into perspective, we also measured their values for a number of calls using a particular pair of mobile phones we had at our disposal. The phones were in proximity of each other and were



$\mathbf{d_{min}}$	$5s$
$\mathbf{d_{avg}}$	$6.5s$
$\mathbf{d_{max}}$	$8s$
$\mathbf{\Delta d}$	$3s$

Figure 1.1: Distribution of the 50 measured values of the parameter d.

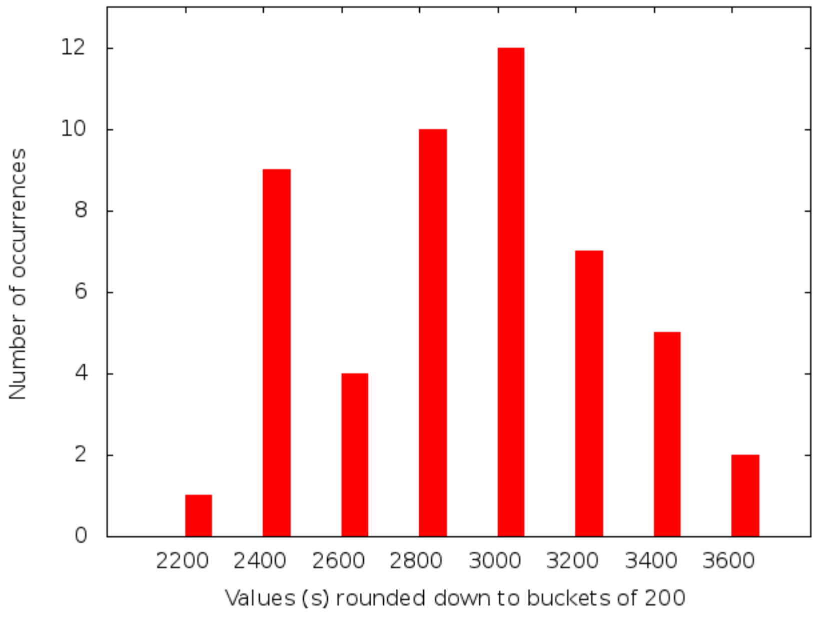
using different mobile phone operators. Although these sample measurements done in a very particular configuration are in no way representative of all the possible values the parameters can attain, we find them valuable to at least hint us on those values' magnitudes. We will also use our measurements to compare the protocols' performance in our two devices' case. This will therefore make us able to say that "In our case (for our values of parameters) protocol A achieves better bitrate than protocol B", though one must keep in mind that the results can be different for different values of parameters.

- d - the length of time since the sender starts dialing until the receiver's phone starts ringing. Our measurements of this random variable are shown in Figure 1.1. We will denote the *exclusive* lower and upper bound of this range d_{min} and d_{max}. This means that $d \in (d_{min}, d_{max})$. The mean of d will be denoted d_{avg}. We will also use the notation $\Delta d = d_{max} - d_{min}$.

- h - the amount of time since the caller hangs up until the receiver's phone stops ringing. The measurements can be seen in Figure 1.2. Similarly as in the case of d, we will consider h to be limited to the range bounded by *exclusive* bounds h_{min} and h_{max}, i.e. $h \in (h_{min}, h_{max})$. The mean is denoted as h_{avg} and the variable's range size as $\Delta h = h_{max} - h_{min}$.

- l - the maximum ringing time. This is a constant that can be set on the receiver's phone, typically as a multiple of 5 seconds. It is usually set defaultly to $30s$ or $60s$. In order to achieve better results, we will assume the latter of these possibilities, i.e. that $l = 60s$. Of course, if the receiver's phone allows setting an ever higher value of l, it is useful to do so.

- g - the "granularity of time". Our protocols will require measuring time between events, but nat-



$\mathbf{h_{min}}$	$2.3s$
$\mathbf{h_{avg}}$	$3s$
$\mathbf{h_{max}}$	$3.7s$
$\mathbf{\Delta h}$	$1.4s$

Figure 1.2: Distribution of the 50 measured values of the parameter h.

urally, this cannot be done with infinite precision. We will have to choose a discrete time unit g for the measurements that is large enough so that we can be sure that whenever the sender and receiver measure the time between certain events, rounded to a multiple of g, they arrive at the same number. And naturally, we want g to be as short as possible, as more precise time measurements will lead to more effective protocols.

There is clearly a non-negligible technical lower bound for g, stemming from hardware and software limitations (e.g. the application cannot measure smaller time intervals than how often it is scheduled). However, there can also be a semantic lower bound - the nature of events between which the time is measured may put more constraints on g. Therefore, the semantics of this parameter can only be understood in the context of a certain protocol.

We will always assume that both participants are aware of the paramaters' values. If they are not, they can find them in cooperation using the methods described in Chapter 4 - Calibration.

1.2 States, actions and events

Sender and receiver can both be in two possible states.

- Sender
  - Offhook
  - Idle
- Receiver
  - Ringing
  - Idle[1].

The Idle and Ringing states are self-explanatory. By saying that the sender's phone is offhook, we mean that it is either currently dialing the receiver or the connection has already been established and the receiver's phone is ringing. Note that we cannot distinguish which of these two situations are we in - when the sender dials, they will not know when the receiver's phone starts ringing. In our model, it means that the sender will not know what value did the random variable d take.

Note that while it is technically possible for the sender to discover approximately when the receiver's phone starts ringing (as one can hear ringing in the sender's earpiece as well), no such information is provided by the Android API[pho14]. Since one of our goals is to implement an Android application, we have to disregard this as a possibility. Similarly, we cannot know the actual value of h either, and in this case it is not just a problem of API but rather a property of the underlying telephony protocol.

Sender and receiver will participate in the protocols by taking actions and listening to events as follows.

- Sender
  - *Start dialing* the receiver. This forces the state transition from Idle to Offhook. After d time, this will cause the *incoming call* event for the receiver.
  - *Hang up.* This forces the state transition from Offhook to Idle. After h time, this will cause the *hangup* event for the receiver.
- Receiver
  - Handle the event of an *incoming call*, i.e. when the phone *starts ringing.* This event indicates the state transition from Idle to Ringing.
  - Handle the event of a *hangup.* This event indicates the state transition from Ringing to Idle.

As we can see, the rules are very simple. The sender has two possible actions to switch between the two states. The receiver observes the sender's actions as events (with some delay, d or h).

We can also add one more action-event pair to give the receiver the ability to hang up as well.

- Receiver can hang up an incoming call. This forces the state transition from Ringing to Idle. After h time, this will cause the *hangup* event for sender.

[1]We will often call this state "silence" as well. This is sometimes better as the term "Idle" suggests that no activity is in progress, while in fact the non-ringing state may have its own meaning.

- Sender can handle a *hangup* event. This indicates the state transition from Offhook to Idle.

An important restriction in our model is that the sender is not allowed to dial during hanging up. If sender wants to hang up the call and re-dial the receiver, they must wait until they are sure that the hangup signal has already reached the receiver. Not complying with this restriction means that the hangup and dial signals would travel the cellular network together. The desired outcome in this case is that the hangup signal reaches the receiver first and is followed by the dialing signal. While this could happen, it could also happen that the two signals are processed together or in a reverse order in one of the nodes of the network, resulting in the cancellation of the dialing. We therefore consider this to be an undefined behavior and that is the reason we forbid it in our model. Symetrically, neither is it allowed to hang up while dialing. Once the phone is offhook, the sender can only hang up if they are sure that the receiver's phone is already ringing. Otherwise, the order in which the signals are processed in the network is again undefined. In practical terms, this means that the sender must guarantee that at least d_{max} time has passed since the dialing started when they hang up, and similarly that at least h_{max} time has elapsed since hanging up whenever they start dialing.

It is allowed for both participants to hang up at the same time. We get two hangup signals travelling through the network concurrently, but that does not lead to an undefined behavior, as these signals have the same intent.

Chapter 2

Protocols

We are going to suggest several communication protocols.

- One-or-Two
- Ringing Length
- Ring-or-Silence
- Alternating Ringing

2.1 One-or-two protocol

The One-or-two protocol is a very simple protocol that will serve as our proof of concept, i.e. it will be our demonstration of feasibility of ringing as a data transfer method. Its main advantages are easy implementation and the fact that it does not rely on timing as heavily as other protocols do.

In this protocol, we will send 0 by ringing once and 1 by ringing twice. If we send more than one bit, we must specify a timeout to be used after each 0, so two consecutive 0s are not mistaken as 1. Let us denote this timeout t. The algorithm of both participants in pseudocode would then be as follows.

```
Sender                Receiver
input $b$ // bit 0 or 1   wait until first ringing
ring                  wait $t$
if b = 1              if another ringing event occured during the wait
  then ring             then output 1
  else wait $t$           else output 0
```

In other words, the transfer of each bit starts by the sender ringing at least once. If the sender wishes to send 1, they have to ring again in quick succession. If they wish to send 0, they must wait for the timeout t. From the receiver's point of view, every transfer starts by ringing once. If another ring comes soon enough (sooner than the timeout t elapses), 1 was transferred. Otherwise, 0 was transferred.

In this pseudocode, by the `ring` command we understand that the sender dials the receiver and waits until the receiver's phone starts ringing. The call is then hung up. However, there are two ways to do this.

- The receiver may hang up immediately when their phone starts ringing.
- The sender may hang up when they are sure the receiver's phone is already ringing, i.e. after d_{max} time has passed.

It turns out that this is not just an implementation detail - in fact, it influences the protocol significantly. We are now going to explore the consequences of both ways of hanging up.

2.1.1 Receiver hangup method

In this method, it is the receiver who hangs up the phone as soon as it starts ringing. Whether the sender wants to transmit 0 or 1, they must always ring at least once. The time it takes for the sender to dial the receiver is a realization of the random variable d and we will denote it as d_1. Thus, after d_1 time has passed, the receiver's phone starts ringing and the receiver immediately hangs up. We will denote the time it takes for the signal to reach back to the sender as h_1. Now there are two possibilities.

If the sender wants to send 1, they only needed to ring again. Thus, the process is repeated, where the dialing takes d_2 time and the hangup takes h_2 time. Obviously,

$$E[d_1] = E[d_2] = E[d] = d_{avg}$$

$$E[h_1] = E[h_2] = E[h] = h_{avg}$$

and therefore the average time required to send the 1 bit is

$$2(d_{avg} + h_{avg})$$

If, on the other hand, the sender wants to send 0, they have to wait a certain amount of time. How much time is required? When the receiver hangs up the first call, they know that it can take at most h_{max} time for the hangup signal to reach sender, and then it may take at most d_{max} time for the sender to call back again. Therefore, it is sufficient for the receiver to wait for

$$t = d_{max} + h_{max}$$

Thus, the sender knows that they must wait at least this long before ringing again to send the next bit. However, we must realize that the receiver starts waiting as soon as they hang up. When the sender is informed of the first hangup, at least h_{min} time has already passed, because it takes for the hangup

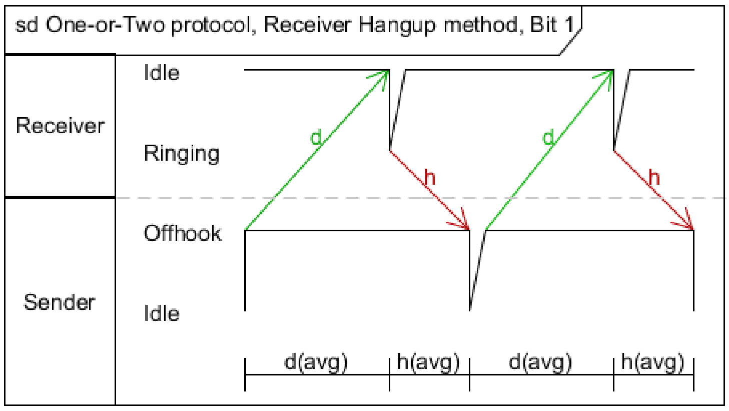


Figure 2.1: Sending 1 using the Receiver hangup method of the One-or-Two protocol.

signal at least this much to reach the sender. The sender therefore knows that at least h_{min} of the timeout has already passed, so they only have to wait for

$$t' = d_{max} + h_{max} - h_{min} = d_{max} + \Delta h$$

After waiting this much time, the sender can be sure that the receiver has understood that a 0 bit was sent. The sender may now start transferring another bit. However, there is still place for improvement. If the sender starts dialing the receiver to send the next bit after the timeout has passed, it will take at least d_{min} time for the receiver's phone to start ringing. Thus, there is an unnecessary d_{min}-long time gap when both participants are already certain that the timeout has passed and the next bit is now to be transferred, but the receiver cannot receive a call from the sender yet. This is ineffective and can be easily improved by making the sender dial d_{min} time before the timeout elapses, so that the receiver's phone may start ringing as soon as possible after the timeout.

$$t'' = d_{max} - d_{min} + h_{max} - h_{min} = \Delta d + \Delta h$$

It follows that transferring a 0 bit takes $d_1 + h_1$ time for the first ring and then the timeout from sender's point of view which is $t' = d_{max} + h_{max} - h_{min}$. Note that the timeout from receiver's point of view may be disregarded, as it never elapses later than from the sender's perspective. This is because timeout from the sender's perspective is essentialy an upper estimate of the receiver's timeout. It follows that the average 0 bit transferring time is

$$d_{avg} + h_{avg} + t' = d_{avg} + h_{avg} + d_{max} + \Delta h$$

However, as the sender already starts dialing to send the next bit during the timeout, they spare d_{min} of what would be the dialing time d_1 for the next bit. The processes of sending these two bits overlap. We can therefore say that the actual cost of sending a 0 bit is the previous value minus d_{min}, i.e.

$$d_{avg} + h_{avg} + t'' = d_{avg} + h_{avg} + \Delta d + \Delta h$$

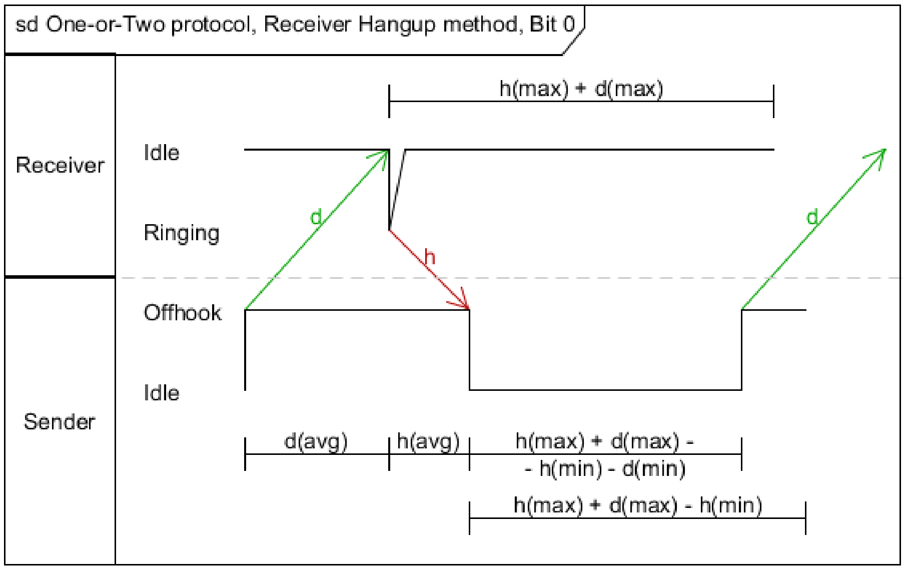


Figure 2.2: Sending 0 using the Receiver hangup method of the One-or-Two protocol. The figure also shows the early dialing for the next bit.

Of course, there is the special case if the considered 0 bit is the last one in the message. In this case, as there is no following bit to be transferred, we do not actually save the d_{min} time. In our model of random messages, the probability of the last bit being 0 is $\frac{1}{2}$, and so we should add $\frac{1}{2} \cdot d_{min}$ to the overall time of sending a message. However, as was explained before, we consider the message to consist of n bits where $n \to \infty$. Thus this one-time addition would be amortized between the n bits and contribute nothing for $n \to \infty$.

Now that we know the time required to send a 1 and a 0 bit, the average time needed to send a bit can be calculated as

$$\frac{1}{2} \cdot 2(d_{avg} + h_{avg}) + \frac{1}{2} \cdot (d_{avg} + h_{avg} + \Delta d + \Delta h) =$$
$$= \frac{3}{2} \cdot (d_{avg} + h_{avg}) + \frac{1}{2} \cdot (\Delta d + \Delta h)$$

If we substitute values for the variables in this expression, we arrive at the average bit transfer time of $16.45s$, or equivalently a bitrate of $r \approx 0.06079b/s$.

2.1.2 Sender hangup method

Just as in the previous method, the bit transfer starts by having the sender dial the receiver. Since the sender does not know precisely how long will it take until the receiver's phone ring, they must dial for d_{max} time to be sure. After d_{max} time, it is guaranteed that the receiver's phone rings (either it has just started or it may have been so for as much as $d_{max} - d_{min}$ time). Thus, the sender may now hang up, what will take h_{avg} time on average.

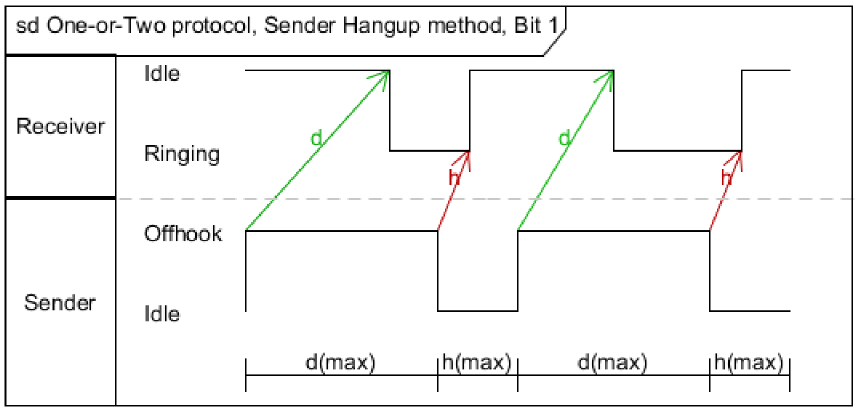


Figure 2.3: Sending 1 using the Sender hangup method of the One-or-Two protocol.

Let us again consider sending a 0 and a 1 bit separately.

If the sender is to send a 1 bit, they must ring again. Therefore, after hanging up, the sender must wait h_{max} time to be sure that the hangup signal has reached the receiver and then dial again. The sender must again wait d_{max} time to be sure that the receiver's phone is ringing and then they can hang up for the second time. There were two ringings, both taking

$$d_{max} + h_{max}$$

time, making up the total cost

$$2d_{max} + 2h_{max}$$

If the bit being sent is 0, we will witness a similar situation as in the Receiver hangup method, but now with roles reversed. The receiver will have to estimate the timeout from the sender's perspective now. From the point where the sender hangs up, it would take at most $h_{max} + d_{max}$ time to wait for the hangup signal and then ring again to reach receiver again to send 1. Therefore, the timeout is set to

$$t = d_{max} + h_{max}$$

When the receiver learns that sender has hung up, they will know that at least h_{min} time has passed while the signal got to them. The receiver therefore has to wait for

$$t' = d_{max} + h_{max} - h_{min} = d_{max} + \Delta h$$

After t' time elapses, the receiver knows that a 0 bit was sent and that the next bit is now to be transferred. How does the sender know when to start with the next transfer? From the moment when they hung up, it may have taken as much as h_{max} time for the signal to reach the receiver, and then the receiver waits for t' time. The sender therefore cannot be certain that the receiver's timeout has elapsed until

$$t'' = h_{max} + t' = d_{max} + h_{max} + \Delta h$$

time has passed since they hung up. After t'' time, both sender and receiver are certain that the other party knows that a 0 bit has been transferred. The sender may start ringing again to send the next bit. At this point, we may reuse the optimization from the previous method - the sender may actually start dialing d_{min} time before the timeout elapses, i.e. after

$$t''' = t'' - d_{min} = h_{max} + \Delta d + \Delta h$$

By similar argumentation as in the previous method, it is evident that the timeout t'' measured by the sender elapses later than t' measured by the receiver. The total time needed to send the 0 bit is

$$d_{max} + t'' = 2d_{max} + 2h_{max} - h_{min}$$

As the sender starts dialing before the timeout elapsed, they spare d_{min} from the dialing time for the first ringing of the next bit. Therefore, we get the actual average cost of sending a 0 bit as

$$d_{max} + t''' = d_{max} + h_{max} + \Delta d + \Delta h$$

As was already argumented before, the idea of subtracting d_{min} cost is invalid if we are considering the last bit, as there is no further dialing from which this time should be spared. In this case, we would have to add a term of d_{min} back to the sum. However, we again do not have to, as this constant term can be amortized among n bits for $n \to \infty$.

We can now express the average time required to send a bit as the average of the respective times for 1 and 0 bits.

$$\frac{1}{2} \cdot (2d_{max} + 2h_{max}) + \frac{1}{2} \cdot (d_{max} + h_{max} + \Delta d + \Delta h) =$$
$$= 2d_{max} + 2h_{max} - \frac{1}{2}(d_{min} + h_{min})$$

After substituting values for variables, we get that in our case the average time per bit is $19.75s$. This yields a bitrate of $r \approx 0.0506b/s$.

2.1.3 Comparison

Upon observing the bitrates of the two methods closer, we can say that in the former method the average time required to send a bit

$$\frac{3}{2} \cdot (d_{avg} + h_{avg}) + \frac{1}{2} \cdot (\Delta d + \Delta h)$$

depends linearly on

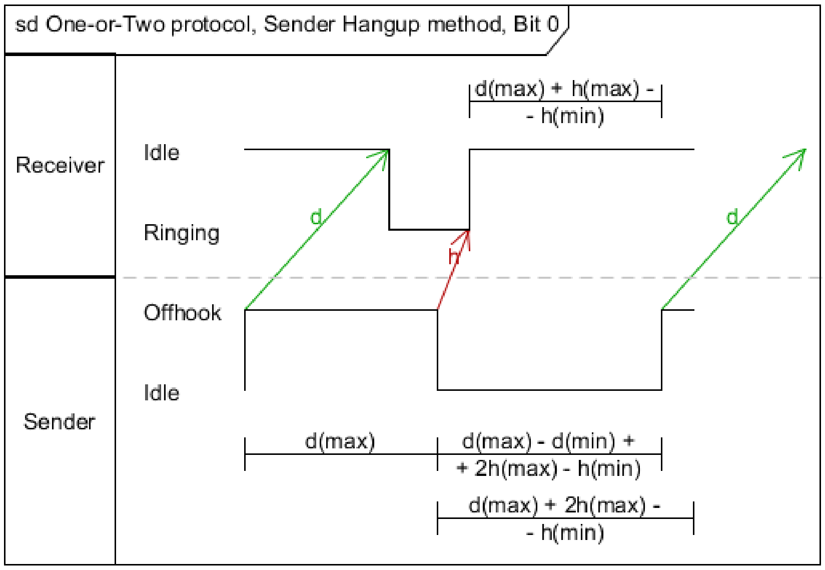

sd One-or-Two protocol, Sender Hangup method, Bit 0
Receiver
Sender
Idle
Ringing
Offhook
Idle
d
h
d
d(max) + h(max) - - h(min)
d(max)
d(max) - d(min) + + 2h(max) - h(min)
d(max) + 2h(max) - - h(min)

Figure 2.4: Sending 0 using the Sender hangup method of the One-or-Two protocol. The figure also shows the early dialing for the next bit.

- The average time of one ringing (dialing + hanging up).
- The size of intervals spanned by values of the variables d and h.

while in the latter one

$$2d_{max} + 2h_{max} - \frac{1}{2}(d_{min} + h_{min})$$

it depends linearly on

- The maximum time of one ringing (dialing + hanging up).
- The minimum time of one ringing (dialing + hanging up).

In our case, the first method turned out to be more efficient. Let us find the general condition when this holds.

$$\frac{3}{2} \cdot (d_{avg} + h_{avg}) + \frac{1}{2} \cdot (\Delta d + \Delta h) < 2d_{max} + 2h_{max} - \frac{1}{2} \cdot (d_{min} + h_{min})$$

$$\frac{3}{2} \cdot (d_{avg} + h_{avg}) + \frac{1}{2} \cdot (d_{max} + h_{max}) < 2d_{max} + 2h_{max}$$

$$\frac{3}{2} \cdot (d_{avg} + h_{avg}) < \frac{3}{2}d_{max} + \frac{3}{2}h_{max}$$

$$d_{avg} + h_{avg} < d_{max} + h_{max}$$

By definition, $d_{avg} \leq d_{max}$ and $h_{avg} \leq h_{max}$. Thus, the Receiver hangup method is always more efficient except in the special case where d and h would have no variance, so

$$d_{min} = d_{avg} = d_{max} \wedge h_{min} = h_{avg} = h_{max}$$

$$d_{avg} + h_{avg} = d_{max} + h_{max}$$

and then two methods would be equally efficient.

Note also that in this comparison we used the h parameter in both methods, but it has two different meanings. In one case, it is the time it takes for the hangup signal to reach from the sender to the receiver, while in the the other case the signal travels in the opposite way. It is not guaranteed that h, as a random variable, will have the same distribution in both cases. If it does not, this comparison is meaningless. It is, however, not an unreasonable assumption. In both cases, we expect h to be small, considerably smaller than d and if the distributions differ only slightly, this difference will not have a significant effect on our comparison.

2.2 Ringing Length protocol

Observing the previous protocol closely, one can see an obvious source of inefficiency. We dial once or twice for each bit, what takes a long time, as the d parameter takes quite large values. Naturally, the improvement that comes to mind is to send mutliple bits per one dialing instead. This is the idea of the Ringing Length protocol. While in the One-or-Two protocol we always hang up as soon as possible, in this protocol we will let the receiver's phone ring and several bits will be encoded to the ringing time length.

2.2.1 Description

A sequence of bits sent per one dialing will be called a *block*. We will denote the block size as b. The values of the b bits will be interpreted as a single number $x \in \{0, \ldots, 2^b - 1\}$. This number will be caled the *value* of a block and encoded by ringing for the time $x \cdot g$. We remind the reader that g is the "granularity of time" parameter, the smallest reasonably distinguishable time interval. As we remarked before, its semantics will be dependent on the protocol. It is the case in this protocol as well. While there is a technical lower bound on g - the actual time measurement precision of the phones, this protocol also puts a semantic lower bound on it.

The protocol will proceed as follows. The sender will dial the receiver for d_{max} time, so they can be sure that the receiver's phone will ring. Then, to send a number $x \in \{0, \ldots, 2^b - 1\}$, the sender will wait for $x \cdot g$. The sender will then hang up.

The sender will always allow d_{max} time for dialing and h_{max} time for hanging up. Since in our model all 2^b configurations of the b bits are equally likely, the average ringing time length can be computed as

$$\frac{1}{2^b}\sum_{i=0}^{b} i \cdot g = \frac{g}{2^b} \cdot \frac{(2^b)(2^b-1)}{2} = \frac{g(2^b-1)}{2}$$

Thus, the average time required to send a block is

$$d_{max} + h_{max} + \frac{g(2^b-1)}{2} \tag{2.1}$$

and since one block contains b bits, the average time needed to send one bit is

$$\frac{d_{max} + h_{max}}{b} + \frac{g(2^b-1)}{2b}$$

Observe that this computation was made from the sender's point of view. The sender assumes the worstcase dialing time d_{max}, then rings for $x \cdot g$, and only then hangs up, what will add at least h_{min} more ringing time from the receiver's perspective. However, dialing may have taken less time and hanging up may have taken more time. If they did, the receiver will perceive the ringing to be somewhat longer. In general, if the sender intends to transmit the value x, ringing will take

- at least $x \cdot g + h_{min}$ time. This is achieved if dialing takes the longest possible time d_{max} and hanging up takes the shortest possible time h_{min}.
- at most $x \cdot g + \Delta d + h_{max}$ time. This is achieved if dialing takes the shortest possible time d_{min} (thus adding the difference Δd to the beginning) and if hanging up takes the long possible time h_{max}.

Thus, the time that the receiver measures may vary on the interval

$$(x \cdot g + h_{min}, x \cdot g + \Delta d + h_{max})$$

Let us call the case when dialing takes the maximum possible time d_{max} and hanging up takes the minimum possible time h_{min} the *base case.* Therefore, $x \cdot g + h_{min}$ is the base case ringing time for the receiver. Note that since we defined the ranges of d and h as open intervals, the ringing can in fact never take exactly this time, but can take ε longer for any $\varepsilon > 0$. The cases when ringing takes longer or hangup takes shorter than in base case will be considered a tolerated *error.* Such an error may be of size up to

$$0 < \mathit{error} < |(x \cdot g + h_{min}, x \cdot g + \Delta d + h_{max})| = \Delta d + \Delta h \tag{2.2}$$

Note that the error is always positive, because we chose the base case ($\mathit{error} = 0$) as the case with the shortest possible ringing time. The bounds we defined on error are exclusive, i.e. its range is an open interval, because the range of possible ringing times is also an open interval.

In order to correctly decode x, the receiver must be able to recognize and correct the error. The error prolongs the ringing and we must make sure that it will not be prolonged so much as to be considered $x + 1$. In other words, transmission time of x even with the maximum possible error must be less than transmission time $x + 1$ with any error. Since the error is non-negative, we can simplify this statement

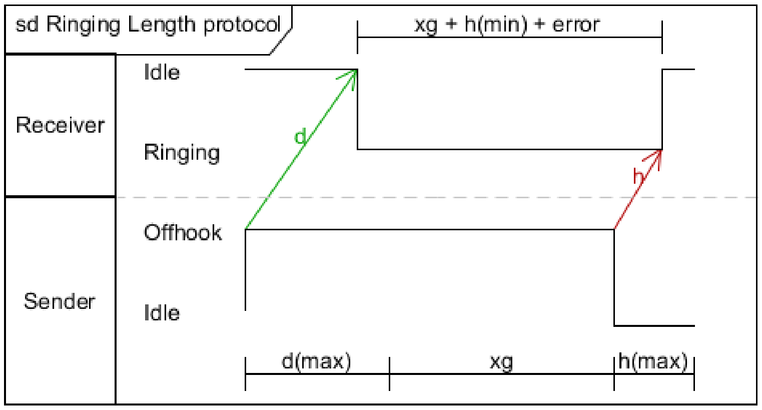


Figure 2.5: Ringing Length protocol - sending block with value x.

and say that the transmission time of x with the maximum possible error is less than transmission time of $x + 1$ without an error.

$$x \cdot g + \Delta d + h_{max} \leq (x + 1) \cdot g + h_{min}$$

$$g \geq \Delta d + \Delta h$$

Note that we could use $\leq$ instead of $<$, because the error's range is an open interval. And since we want g to be as small as possible, we set

$$g = \Delta d + \Delta h$$

Therefore in our case we get the value $g = 4.4s$, which is surely enough to not hit the technical limitations of time measurement precision.

The ringing time (denoted t_{ring}) can be decoded into block value x as

$$x = \left\lfloor \frac{t_{ring} - h_{min}}{g} \right\rfloor \tag{2.3}$$

and by translating x to its binary notation, we get the bits that were sent.

What remains is to find the optimal block size b. The idea of this protocol is that it's more effective to send several bits at once, so it could seem that we want to maximize b. However, we must also notice that increasing b by one means that the range of the encoded number x doubles. It follows that the average ringing time also approximately doubles, and so it grows exponentially with b. Therefore, b cannot be too large either. It seems that there is an optimal value of b that yields the minimum possible average transfer time per bit, while both increasing or decreasing b will lead to ever worse results. This means that the average transfer time per bit, as a function of b (denoted $t(b)$)

$$t(b) = \frac{d_{max} + h_{max}}{b} + \frac{g(2^b - 1)}{2b}, b > 0 \tag{2.4}$$

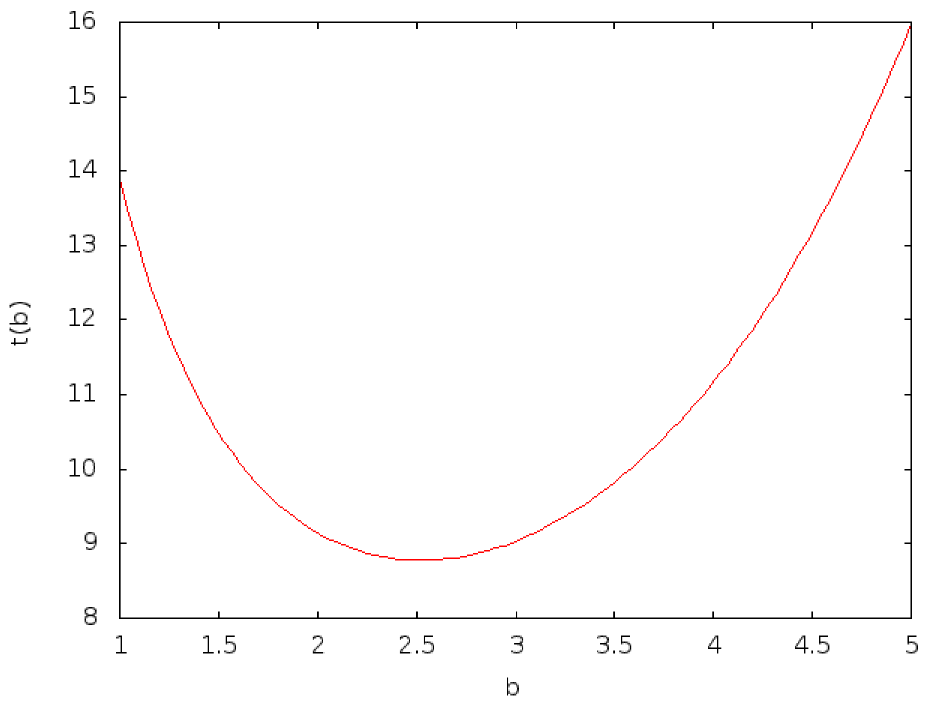


Figure 2.6: Transmission time as a function of the block size.

is a convex function with a single local and global minimum, which is the optimal value of b. Plotting a graph of this function (for our values of parameters) suggests that this is indeed true - see Figure 2.6.

To prove this, let us take the derivative of this function

$$t'(b) = \frac{(d_{max}+h_{max})'b - (d_{max}+h_{max})b'}{b^2} + \frac{(g(2^b-1))'(2b) - g(2^b-1)(2b)'}{4b^2} =$$

$$= -\frac{d_{max}+h_{max}}{b^2} + \frac{g(2^b \ln 2)(2b) - 2g(2^b-1)}{4b^2} =$$

$$= \frac{-4d_{max} - 4h_{max} + 2g(\ln 2)b2^b - 2g2^b + 2g}{4b^2} =$$

$$= \frac{-2d_{max} - 2h_{max} + g2^b((\ln 2)b - 1) + g}{2b^2}$$

To find the local extreme (denoted b_{opt}), we put

$$t'(b_{opt}) = 0$$

$$\frac{-2d_{max} - 2h_{max} + g2^b_{opt}((\ln 2)b_{opt} - 1) + g}{2b^2_{opt}} = 0$$

$$-2d_{max} - 2h_{max} + g2^b_{opt}((\ln 2)b_{opt} - 1) + g = 0$$

$$g2^b_{opt}((\ln 2)b_{opt} - 1) = 2d_{max} + 2h_{max} - g$$

$$g2^b_{opt}((\ln 2)b_{opt} - 1) = \frac{2d_{max} + 2h_{max} - g}{g}$$
$$e^{(\ln 2)b_{opt}}((\ln 2)b_{opt} - 1) = \frac{2d_{max} + 2h_{max} - g}{g}$$
$$e^{(\ln 2b_{opt} - 1)}b_{opt}((\ln 2)b_{opt} - 1) = \frac{2d_{max} + 2h_{max} - g}{eg}$$
$$\ln 2b_{opt} - 1 = W\left(\frac{2d_{max} + 2h_{max} - g}{eg}\right)$$
$$b_{opt} = \frac{W\left(\frac{2d_{max}+2h_{max}-g}{eg}\right) + 1}{\ln 2}$$

Where W is the Lambert function defined as the inverse of $f(x) = xe^x$[CGH+96], where we substited $x = (ln2)b_{opt} - 1$. To prove that this point is indeed a local minimum, we have to also show that $t''(b_{opt}) > 0$. We will, however, omit this part as it is tedious and mechanical.

We now know how to find the block size b_{opt} such that $t(b_{opt})$ is minimal. However, this number is likely non-integer. Since t is bitonic, we know that the minimum integer value is one of its integer neighbours

$$b_{int} = \arg\min_b \{t(b) \mid b \in \{\lfloor b_{opt} \rfloor, \lceil b_{opt} \rceil\}\}$$

But this is still not the solution. Larger blocks imply longer ringing times, which is limited by the parameter l. So far, we have focused on the average ringing time in our computations, now we must ensure that even the longest ringing time, transferring the block $1 \dots 1$ (value $2^b - 1$) will fit into the time window defined by the parameter l. We must also allow for possible prolongation of Δd if the dialing takes the shortest possible time. Finally, to keep things consistent, we do not want the receiver to hang up on their own. The reason is that (for some values of parameters) l time could elapse just after the sender has sent the hangup signal but before h_{min} time passed. The receiver would then hang up on their own too soon (less than h_{min} after sender decided to hang up) and thus seemingly break the assumption that $h \in (h_{min}, h_{max})$. Thus, we should also allocate extra time for the hangup signal to reach from sender to the receiver.

$$(2^b - 1)g + \Delta d + h_{max} \leq l$$
$$b \leq \log_2\left(\frac{l - \Delta d - h_{max}}{g} + 1\right) = b_{max} \tag{2.5}$$

Consider that the optimal value b_{int} does not satisfy this condition, i.e. $b_{int} > b_{max}$. It follows that as $t(b)$ is decreasing on the interval $(0, b_{opt})$, it must also be decreasing on the $(0, \lfloor b_{max} \rfloor]$, which is its subinterval. This means that the optimal value of b in this case is the largest integer in this interval, i.e. $\lfloor b_{max} \rfloor$.

To summarize all these rules, we can formulate the algorithm used by the participants to find the optimal feasible value of b (denoted b_{res}) that takes into account both b_{opt} and b_{max}. When speaking about the transmission speed of this protocol, we will always assume that we are using the optimal block size: $b = b_{res}$.

1. Calculate $b_{opt} = \frac{W\left(\frac{2d_{max}+2h_{max}-g}{eg}\right)+1}{\ln 2}$
2. Calculate $b_{max} = \log_2\left(\frac{l-d_{max}+d_{min}-h_{max}}{g}+1\right)$
3. If $\lfloor b_{opt} \rfloor \leq b_{max}$, return $b_{res} = \lfloor b_{max} \rfloor$
4. Otherwise return $b_{res} = \arg\min_b \{t(b) \mid b \in \{\lfloor b_{opt} \rfloor, \lceil b_{opt} \rceil\}\}$

For our values of parameters, we get $b_{opt} \approx 2.525$ and $b_{max} \approx 3.713$. We should therefore compare the block sizes 2 and 3. We get $t(2) = 9.15$ and $t(3) = 9.0\overline{3}$, so we should use the block size $b = b_{res} = 3$.

2.2.2 Speedup: Granularity versus error-correction tradeoff

We have established the lower bound for g to be

$$g \geq \Delta d + \Delta h$$

and since we want to minimize g, we set it to exactly match this bound. Now the question is if there is anything to gain by lowering g below this bound. Remember that this protocol requires g to be at least this large in order to assure a correct interpretation of ringing length despite the fluctuations of d and h. Therefore, lowering g must necessarily introduce transmission errors. On the other hand, low value of g makes the communication faster, so we may use the saved time to correct these errors.

Consider using a reduced granularity $g' = \frac{g}{3}$. Originally, the intervals of possible ringing times for individual values of x were disjunct. Now they overlap, as their spacing is only g' while their sizes are still $g = 3g'$. Receiver used the formula (2.3)

$$x = \left\lfloor \frac{t_{ring} - h_{min}}{g} \right\rfloor$$

to determine the sent value. However, this time using the formula

$$x = \left\lfloor \frac{t_{ring} - h_{min}}{g'} \right\rfloor = \left\lfloor 3\frac{t_{ring} - h_{min}}{g} \right\rfloor \tag{2.6}$$

we cannot distinguish between

- Sending x with an error $[2g', 3g')$
- Sending $x + 1$ with an error $[g', 2g']$
- Sending $x + 2$ with an error $(0, g']$

because
$[x+2g', xg'+3g'] = [(x+1)g'+g', (x+1)g'+2g'] = [(x+2)g', (x+2)g'+g']$
and thus this formula will map them to the same value (specifically $x+2$). From the receiver's perspective, if the formula (2.6) returns a value x, what could have happened is

- x was sent and the error was $(0, g')$
- $x-1$ was sent and the error was $[g', 2g')$
- $x-2$ was sent and the error was $[2g', 3g')$

Note that the ringing time $xg'+2g'$ could either mean value x with error $2g'$ or value $x+1$ with error g'. However, as our formula uses the floor function, it is always interpreted as the former. We will later see that this does not matter. Even if it did, we can always argue that d, h are expected to be continuous random variables and so the probability of error being exactly this (or any other) one particular value is zero anyway, so the ambiguous case never actually occurs.

We can say that for each block, receiver will read the value of x with error 0, -1 or -2 (of course, except the cases where $x=2$ or $x=1$, where only one or two of these cases are possible, respectively).

Gray code

Let us change the protocol description slightly so that whenever sender wants to send the value x, they ring not for xg' time, but for $xg'+g' = (x+1)g'$, as if they wanted to send $x+1$. The possible errors in receiver's interpretation also shift by one, so if x is received, it could mean that

- $x+1$ was sent and the error was $(0, g')$
- x was sent and the error was $[g', 2g')$
- $x-1$ was sent and the error was $[2g', 3g')$

The receiver thus interprets the value of each block with a possible ± 1 error (with exception of $x=0$, where -1 error cannot occur and $x = 2^b - 1$, where $+1$ cannot occur). It turns out that this property is very useful if we encode the b bits by Gray code[Gra53] (a.k.a. reflected binary code) instead of naïvely interpreting the bits as a binary number.

Gray code G_b is a permutation of b-bit sequences. In our case, if we want to transmit a b-bit word w, we will first translate it to $G_b(w)$ and interpret the resulting b-bit sequence as the number $x \in \{0, \ldots, 2^b - 1\}$ which becomes the block value. The receiver will then invert the process by rewriting x back to binary and applying G_b^{-1}.

This enables us to use an important feature of Gray code - codes for consecutive numbers only differ in one bit. This means that the ± 1 possible error in ringing length transforms into one flipped bit difference between $G(x)$ and $G(x \pm +1)$. This allows us to identify and correct these transmission errors using standard error-correcting codes. In this case, error in one bit can be repaired by Hamming's $(7, 4)$ code[MVM09]. Hamming's code assumes there is at most one flipped bit in every chunk of 7 bits. We can guarantee that there is at most one flipped bit in one block. Thus, we must use block size $b = 7$.

The average time needed to send one bit can now be expressed by taking the original formula (2.4) and substituting $b = 7$ in the numerator, but $b = 4$ in the denominator (as we have to send 7 bits, but only 4 bits of the message are actually transmitted, the remaining 3 serve for the error correction). We also substitute $\frac{g}{3}$ for g. Finally, we must take into account that every ringing now takes $g' = \frac{g}{3}$ longer, due to the ± 1 shift. We obtain

$$\frac{d_{max} + h_{max}}{4} + \frac{g}{3} \cdot \frac{2^7 + 1}{2 \cdot 4}$$

which, for our values of parameters, will yield ~ 26.575, almost tripling the original time. In our case, the exponential growth of ringing time with the block size has far surpassed the advantage of smaller g. It seems, however, that for different configuration of parameters, if b_{opt} was higher, there could be an improvement. For example, let us consider that we were lucky so the block size $b = 7$ suggested by Hamming code was already optimal in the naïve version of the algorithm. Let us see if this Gray code modification is then faster. Intuitively, this should hold, since we transmit at triple the speed, but send only $\frac{7}{4}$-times more bits (which only means sending $\frac{7}{4}$ more blocks, not $\frac{7}{4}$-times larger blocks; thus the time increases linearly, not exponentially).

$$\frac{d_{max} + h_{max}}{4} + \frac{g}{3} \cdot \frac{2^7 + 1}{2 \cdot 4} < \frac{d_{max} + h_{max}}{7} + g \cdot \frac{2^7 - 1}{2 \cdot 7}$$

$$\frac{7d_{max} + 7h_{max} + \frac{903}{6}g}{28} < \frac{4d_{max} + 4h_{max} + 254g}{28}$$

$$d_{max} + h_{max} < \frac{69}{2}g$$

$$d_{max} + h_{max} < \frac{69}{2}\Delta d + \frac{69}{2}\Delta h$$

Thus, the only situation when Gray code improvement would fare worse than the naïve approach is if the maxima of d and h were disproportionately larger than their ranges, e.g. if dialing and ringing times took reliably constant time instead of being random variables.

We may extend the principal idea of this section by using $g' = \frac{g}{5}$ and subsequently use an error-correcting code that corrects 2 bits, or in general $g' = \frac{g}{2k+1}$ with k faulty bits. However, such error-correcting codes necessitate large block sizes, and thus will apparently be very inefficient for our practice where $b_{opt} = 3$.

		$\mathbf{x \to y}$ **shift**		
		$\mathbf{y = x}$	$\mathbf{y = x+1}$	$\mathbf{y = x+2}$
$\mathbf{x'}$ **considered**	$\mathbf{x' = y}$	x	$x+1$	$x+2$
	$\mathbf{x' = y-1}$	$x-1$	x	$x+1$
	$\mathbf{x' = y-2}$	$x-2$	$x-1$	x

Table 2.1: Values of x' considered by the receiver.

Modulo method

Not having success with standard error-correcting codes working with the bit representation of the message, let us devise an error-correcting method working directly with the ringing times.

We will return to the idea of lowering g to $g' = \frac{g}{3}$ without the +1 shift. Recall that in this setting, if the sender sends a value x, the receiver may interpret it as x, $x+1$ or $x+2$. Let us denote this interpreted value y. The receiver wants to guess the original value of x, we will denote this guess by x'. Now the receiver knows that the value y was shifted and so it may hold that either $x' = y$, $x' = y-1$ or $x' = y-2$.

The sender does not know which of the three shifts happened to x (i.e. the value of y) and thus does not know which three values are considered as x'. Therefore, from the sender's point of view, there are five total values that could be potentially considered. This is illustrated in Figure 2.1.

The sender must therefore find a way to inform the receiver which one of the five values of x' possibly considered by the receiver is the true x. Although not knowing which three of them will be actually considered, the sender knows that the five values are $\{x-2, x-1, x, x+1, x+2\}$. The obvious way to pinpoint x is to send the value $x \ mod \ 3$, since x itself is the only number in the set with this property. Furthermore, this information is as small as possible - the receiver always has to choose between three options and $x \ mod \ 3 \in \{0, 1, 2\}$ is an information of size $\log_2 3$.

The protocol will proceed as follows. The message will be first sent using $g' = \frac{g}{3}$. Then, an error-correcting message will be sent, containing $x \ mod \ 3$ for every block value x of the original message. This will be sent using the original value of g, so as not to introduce any more errors.

The first problem we have solve is how does the receiver know which blocks carry the message and which ones carry the error-correcting information. There are two possible approaches. In the following text, let the block size of the regular blocks be denoted as b' and that of the error-correcting blocks as b. This distinction is important, because different values of g' and g may result in different optimal block sizes.

- *The offline approach.* Wait until all blocks are transmitted. Let us denote their number k. We need to compute the number of error correcting blocks e - then we will know that the message is contained in the first $k-e$ blocks and the error-correcting data in the last e ones, making us able to decode the message. The e error-correcting blocks contain eb bits total. As the correction information for each block of the message takes $\log_2 3$ bits, there are a total $\frac{eb}{\log_2 3}$ of them. However, this must correspond to the number of regular blocks. We get that

$$k - e = \frac{eb}{\log_2 3}$$
$$k = e\left(1 + \frac{b}{\log_2 3}\right)$$
$$e = \frac{k}{1 + \frac{b}{\log_2 3}}$$

Note, however, that this is always an approximation. The number of error-correcting bits we need to send is $(k-e)\log_2 3$, which is an irrational number. The error-correcting blocks can contain eb bits, which is an integer. The last block must therefore always contain some padding, unused bits in the end. Thus, there may be up to b less error-correcting bits than it seems, and therefore up to $\frac{b}{\log_2 3}$ less blocks. The easiest way to help the receiver identify the correct number of blocks $k-e$ is to also send the number

$$(k-e)\ mod\ \left\lceil \frac{b}{\log_2 3} \right\rceil$$

.

This number takes up $\log_2 \left\lceil \frac{b}{\log_2 3} \right\rceil$ bits, which is a fixed value known by both participants, so the receiver will always know at how many last blocks they have to look to extract it. The reader may wonder whether it would not be easier to simply encode the message length n at the beginning of the transmission. The problem is that this would take $O(\log_2 n)$ more bits, while our approach only adds $O(1)$ bits. But both values are $o(n)$ and so they contribute to the transmission time negligibly if $n \to \infty$.

- *The online approach.* As the name suggests, this approach suggests not to wait until the end of the message, but rather process it gradually. This is practical if the message is very long or even infinite. We will have to partition the message into *meta-blocks*, sequences of k blocks and their corresponding error-correcting blocks. Thus, after every k regular blocks we will send k modulo numbers. This way, every meta-block is essentially a separate message that can be decoded on its own, making it possible to output the data transmitted so far after every kb' bits. The sender and receiver will agree on the number k in advance. The question is, how to determine a good value of k?

  The k regular blocks will require k modulo numbers, which will take up $k\log_2 3$ bits. For any k, the number $k\log_2 3$ is irrational and thus the last error-correcting block will always be partially empty. In order not to waste the space (and transmission time with it), we want to pick k such that the last block is as full as possible. The simplest way to achieve this is to approximate the irrational value of $\log_2 3$ by a slightly larger rational number $a \in \mathbb{Q}$. Let us pick a to be the rational number obtained by evaluating $\log_2 3$ to p decimal digits, increased by 10^{-p}, so that it will hold that $a > \log_2 3$. Clearly $a - \log_2 3 < 10^{-p}$. Since a only has p decimal digits, it follows that $10^p a$ is integer, what makes $k = 10^p$ a feasible choice. As a slightly overestimates $\log_2 3$, there are in fact $\log_2 3 - a$ unused bits in the last block(s) for every error-correcting number, which means the total number of unused bits is

$$k(\log_2 3 - a) = 10^p(\log_2 3 - a) < 10^p(10^{-p}) = 1$$

The extra cost of this bit is amortized among the kb' bits in the meta-block. This means that we can reduce the extra time caused by the irationality of $\log_2 3$ to arbitrarily small value by choosing a precise enough approximation a and a corresponding meta-block size k. Therefore, we can consider this extra time to be negligible.

In both approaches, we have solved the problem of the last error-correcting block not being filled to full extent. We can therefore consider the time needed to send an error-correcting bit (as a function of the block size b, denoted $t_g(b)$) to be precisely $\frac{1}{b}$ of the time needed to send the whole error-correcting block. Thus, we can reuse the formula (2.4)

$$t_g(b) = \frac{d_{max} + h_{max}}{b} + \frac{g(2^b - 1)}{2b} \tag{2.7}$$

And similarly, we obtain the time needed to send a plain bit (without error correction) by simply substituting $g' = \frac{g}{3}$ for g and b' for b into formula (2.4). We will denote this $t_{g'}(b')$.

$$t_{g'}(b') = \frac{d_{max} + h_{max}}{b'} + \frac{g'(2^{b'} - 1)}{2b'} = \frac{d_{max} + h_{max}}{b'} + \frac{g'(2^{b'} - 1)}{6b'} \tag{2.8}$$

We send $\log_2 3$ error-correcting bits per every b'-bit block of the message. Therefore, the total time required per bit using the modulo method is derived from formulae (2.7) and (2.8) as

$$t_{g'}(b') + \frac{\log_2 3}{b'} t_g(b) = \frac{d_{max} + h_{max}}{b'} + \frac{g'(2^{b'} - 1)}{6b'} + \frac{\log_2 3}{b'} \frac{d_{max} + h_{max}}{b} + \frac{g(2^b - 1)}{2b} \tag{2.9}$$

We explained the concept of this section using $g' = \frac{g}{3}$, continuing from the previous section about Gray code. However, it is apparent that the concept may be generalized to work with $g' = \frac{g}{k}$ for any k. We will then have $y - x \in \{0, \ldots, k-1\}$ and send $\log_2 k$ bits per block to determine the value $x \ mod \ k$. We then obtain the per bit transmission time by substituting k for relevant occurences of the constant 3 into formula (2.9)

$$t_{g'}(b') + (\log_2 k) t_g(b) =$$
$$= \frac{d_{max} + h_{max}}{b'} + \frac{g'(2^{b'} - 1)}{2kb'} + \frac{\log_2 k}{b'} \frac{d_{max} + h_{max}}{b} + \frac{g(2^b - 1)}{2b}$$

We have already shown how to choose the optimal block size b corresponding to a given g. Now, we have to choose the optimal block size b' so that the total transmission time is minimal. This can be once again done by finding the root of the derivative of transmission time as a function of b'.

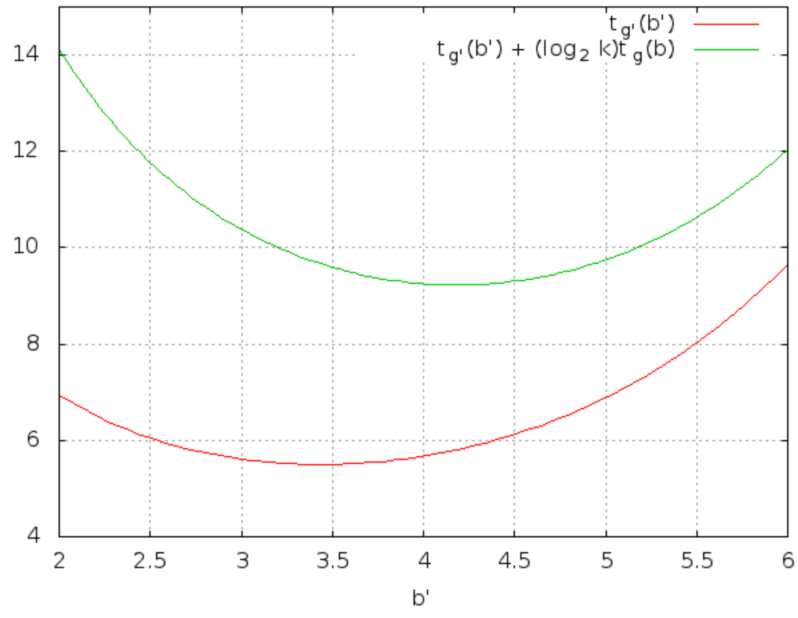


Figure 2.7: The functions $t_{g'}(b')$ and $t_{g'}(b') + (\log_2 k)t_g(b)$ for our values of parameters and $k = 3$. The integer minimum of the first is 3, while for the latter it is 4.

$$\frac{\partial}{\partial b'}\left(t_{g'}(b') + \frac{\log_2 k}{b'}t_g(b)\right) = 0$$

Note that since b' appears in the second term as well, this is not the same as simply choosing the optimal block size b' corresponding to the reduced granularity g'.

$$\arg\min_{b'}\{t_{g'}(b')\} \neq \arg\min_{b'}\{t_{g'}(b') + (\log_2 k)t_g(b)\}$$

This is illustrated in Figure 2.7. We will not provide the exact calculation of b', as it is lengthy and purely mechanical.

Results

Let us experiment with different values of k and illustrate the effect on the transmission time for our values of parameters. The results are shown in Table 2.2. For every value of k, we show

- b' - the optimal block size corresponding to $g' = \frac{g}{k}$.
- Bit transmission time for a plain message bit $t_{g'}(b')$
- Total bit transmission time for a plain bit and $\frac{1}{b'}$-th of an error-correcting bit $t_{g'}(b') + \frac{\log_2 k}{b'}t_g(b)$.

To compare the data in Figure 2.2 with the original, not sped-up version of the protocol, remember that the optimal block size corresponding to $g = 4.4s$ is $b = 3$ and $t(3) = 9.0\overline{3}$.

		Transmission time per bit	
k	$\mathbf{b'_{res}}$	**Plain**	**Total**
1	3	9.033	9.033
2	4	7.050	9.308
3	4	5.675	9.254
4	5	5.750	9.363
5	5	5.068	9.263
6	5	4.613	9.283
7	5	4.289	9.361
8	6	4.838	9.354
9	6	4.517	9.289
10	6	4.260	9.261

Table 2.2: Bit transmission time achieved by the Modulo method for $k \in \{1, \ldots, 10\}$.

We may observe that for increasing k, the tranmsission time also increases. When we reach a value of k high enough so that b'_{int} grows, the transmission time drops, then increases again while b'_{int} is fixed. However, we must conclude that neither choice of k has improved the transmission time under $9.0\overline{3}$. Although the table only shows this for $k \leq 10$, the reader may convince themselves that there is no improvement even for larger values of k.

The usefulness of the Modulo method is best visible if $b_{opt} > b_{max}$. In the original version of the protocol, this would mean having to settle for a non-optimal block size, otherwise the ringing could, in the worst case, take longer than l. Using the modulo method, we try different values of k and reduce g to $g' = \frac{g}{k}$. Decreasing the value of g' increases the corresponding optimal block size b'_{opt}. For some values of k, it may hold that $b'_{opt} \leq b'_{max}$, i.e. the optimal block size does not lead to exceeding the ringing length limit l. Or, it may at least hold that b'_{max} is closer to b'_{opt} than b_{max} was to b_{opt}. Being able to use a block size that is closer to the optimum then provides a huge speedup that dwarfs the extra time required by sending the error-correcting bits.

Let us illustrate this idea on an example. Consider that the receiver has a low ringing time limit $l = 19s$[1]. The maximum ringing time for block size $b = 3$ is $37.5s$ and for $b = 2$ it is $19.9s$, both exceeding l. We must therefore use $b = 1$, which leads to the transfer time $t(1) = 13.9s$. Let us now look at the values of $k \in \{21, \ldots, 30\}$, listed in Figure 2.3.

For every value in this range, it holds that $\lfloor b'_{max} \rfloor \leq b'_{int}$ and therefore $b'_{res} = \lfloor b'_{max} \rfloor$. For $k \in \{22, \ldots, 27\}$ we managed to achieve the desired effect - the transmission time is lower than $t(1) = 13.9s$.

[1]Mobile phones typically only allow setting the limit in $5s$ steps, but this is not important for our computations. If we wish to be pedantic, we may use $l = 20s$ and pretend that d, h parameters are $\frac{20}{19}$ larger instead.

				Transmission time per bit	
k	**b'_{int}**	**$\lfloor b'_{max} \rfloor$**	**b'_{res}**	**Plain**	**Total**
21	7	5	5	3.572	15.200
22	7	6	6	3.486	13.331
23	8	6	6	4.511	13.434
24	8	6	6	4.384	13.534
25	8	6	6	4.268	13.632
26	8	6	6	4.160	13.728
27	8	6	6	4.060	13.821
28	8	6	6	3.967	13.912
29	8	6	6	3.881	14.001
30	8	6	6	3.800	14.088

Table 2.3: Transmission time achieved by the Modulo method for $k \in \{21, \ldots, 30\}$ for $l = 19s$.

2.3 Ring-or-Silence protocol

The Ring-or-Silence protocol follows the way how digital data are usually sent through wires. Signal means 1 and no signal means 0. In our case, ringing means that a 1 bit is being transffered and silence means that a 0 bit is.

2.3.1 Description

The receiver will regularly check if the phone is ringing or idle. To be more precise, the receiver will handle its *incoming call* and *hangup* events and set a flag, then check this flag in regular time intervals, called *frames*. As expected, we will use g to denote the size of a frame. The receiver will then output 1 if the flag is set during the check at the end of the frame and 0 otherwise.

To makes things simpler at first, let us begin with the assumption that the sender and receiver have synchronized clocks. This means that the sender knows precisely at what moments the receiver is going to chceck the state. Furthermore, we will assume that the moment when the transmission starts is implicit and known. This is important because when the transmission begins by sending 0, there is no explicit event for the receiver to know. In practice, this beginning time may be mutually agreed between the participants by other means of communication. Another possibility is to start the communication by a preamble.

Sending two consecutive 1 bits means to keep ringing between frames, just as sending two 0 means to keep idling. However, sending a 1 after 0 means that the sender has to dial. Similarly, sending a 0 after 1 means they have to hang up. In both cases, the state must be changed inside a time window of g, since that is how often the receiver checks it. We will therefore require that

$$g \geq d_{max} \wedge g \geq h_{max}$$

and since it typically holds that $d_{max} > h_{max}$[2], we may simply put

$$g = d_{max}$$

Thus, the bit transmission time seems to be d_{max}. We will soon see that this is only approximately so. Still, notice that so far in every protocol it was the d_{max} parameter that had the greatest influence on the protocol's effectivity.

We must now deal with the inherent asymmetry of our model. While the phone can be in the idle state indefinitely, it can only ring for the time l. In our case, this limits the number of 1s (denoted c) that can be sent consecutively to

$$c = 1 + \left\lfloor \frac{l - \Delta d}{g} \right\rfloor = 1 + \left\lfloor \frac{l - \Delta d}{d_{max}} \right\rfloor \tag{2.10}$$

To send the first 1 bit of a long sequence, the sender will start dialing immediately in the beginning of a frame. The length of the frame is $g = d_{max}$ and since dialing may only take d_{min}, the receiver's phone may have already been ringing for $d_{max} - d_{min}$ when the receiver checks the state and outputs the first 1. The ringing will then continue for at least $l + d_{max} - d_{min}$ (possibly more, if dialing took more time, but we cannot know that, so we cannot reckon with it). During this time, receiver will find the state to be ringing at the end of every frame and then the sender will have to hang up. Hence we arrived to the formula (2.10). Note that although the receiver can hang up themself after the time limit has elapsed, the sender should always do so as this would otherwise introduce a meaningless special case. The sender should always hang up after c consecutive 1 bits were sent, despite the fact that the receiver would hang up soon or even may already have hung up and the signal just did not reach the sender yet. If we just waited for the receiver to hang up automatically after the time limit l elapses, we could sometimes accidentally send more 1 bits. This could happen if the dialing took more than the minimum d_{min} time. Suppose it took full d_{max} time. In this case, we would have as much as

$$\left\lfloor \frac{l}{g} \right\rfloor$$

receiver state checks resulting in 1 bit being output. Thus, if

$$\left\lfloor \frac{l}{g} \right\rfloor > \left\lfloor \frac{l - \Delta d}{g} \right\rfloor$$
$$l \ mod \ g < \Delta d$$

we would accidentally send at least one more 1 bit because the receiver failed to hang up in time. Of course, this could sometimes be taken advantage of to send a longer sequence of 1s, but cannot be relied upon, as the dialing time is nondeterministic and more importantly unknown to the sender.

It is not difficult to cope with this limitation. We know that after c consecutive 1 bits, the call must

[2]This is warranted by our measurements as well as by understanding that dialing is a more complicated process.

be hung up and thus a 0 bit will follow. Therefore, the receiver will understand every 0 coming after c 1s as an involuntary 0 bit only transmitted because of the ringing time limit. Of course, the sender might have actually wanted to transmit a 0, but the receiver cannot distinguish that. The sender must realize this and always include the extra 0 after c consecutive 1s, even if they hung up with the intention to send a 0 and not only because of the time limit. This slightly impacts the bitrate of our protocol, as in our random model we expect that a sequence of at least c consecutive 1 bits will occur every so often, forcing us to transmit the extra bit. Let us compute precisely how often does this happen.

We will start by computing the expected number of consecutive sequences of 1s of a certain length. Let us denote this length i. The random variable counting the number of sequences of length i will then be denoted t_i and so the value we look for is $E[t]$. In order to avoid duplicate counting, we will only consider sequences that are exactly of length i. For example, the bit string 0111010 contains one sequence of length 1 and one sequence of length 3. It also contains two overlapping sequences of length 2, but we will not count those, as they are both part of the sequence of length 3. Sequences of length i can be divided into three categories.

- The sequence starts right in the beginning of the message. This means that the first $i + 1$ bits of the message are $11, \dots, 110$, i.e. i 1s followed by a 0. Since every bit can be equally probably a 1 or 0, the probability of this happenning is $2^{-(i+1)}$. It follows that the expected number of consecutive sequences of 1s in the first $i + 1$ bits of the message, satisfying the conditions
  - length i
  - starting at the beginning of the message

  is also $2^{-(i+1)}$.
- A symmetrical situation at the end of the message. Thus, the expected number of consecutive sequences of 1s in the last $i+1$ bits of the message (that are also of length i and end at the last bit) is $2^{-(i+1)}$.
- For sequences that neither start at the first bit nor end at the last one, we must reckon with sentinel 0s on both sides. Therefore, we will count the number of sequences $011\dots110$ containing i 1s. If the length of message is n, there are $n - i - 1$ positions where such a sequence of length $i + 2$ might be. Of course, this only holds if $n \leq i + 2$, but remember that we always assume $n \to \infty$. Now in each of these "slots" of $i+2$ bits, the probability of it containing the sequence $011\dots110$ is $2^{-(i+2)}$. Therefore, in each of these slots the expected number of sequences $01\dots10$ is also $2^{-(i+2)}$.

These are all the positions where a consecutive sequence of 1s of the length i may be. As each of those positions overlap a few others, the probability of the sequence appearing in one position is not independent of the probability that it appears in others. However, we may take advantage of the linearity of mean value. Thus, the expected number of the sequences of 1s of the length i is

$$E[t_i] = 2 \cdot 2^{-(i+1)} + (n - i - 1) \cdot 2^{-(i+2)} = \frac{2}{2^{i+1}} + \frac{n-i+1}{2^{i+2}} = \frac{n+3-i}{2^{i+2}}$$

Every sequence of length at least c and at most $2c-1$ causes us to transmit an extra bit, thus on average we get $\sum_{i=c}^{\infty} E[t_i]$ extra bits. Sequences of lengths $2c \dots 3c-1$ cause us to transmit two extra bits, one after the first c consecutive 1s, another after the next c. In general, we transmit k extra bits if the sequence is of length $kc \dots (k+1)c-1$. This can be reformulated as follows. Every sequence of length at least c adds an extra bit, $\sum_{i=c}^{\infty} E[t_i]$ in total. Every sequence of length at least $2c$ adds one more extra bit, so we have to add $\sum_{i=2c}^{\infty} E[t_i]$. Then sequences of length at least $3c$ add yet one more etc. It follows that the final number of extra bits generated is

$$\sum_{k=1}^{\infty} \sum_{i=kc}^{\infty} \frac{n+3-i}{2^{i+2}}$$

Let us compute the outer sum first.

$$\sum_{i=kc}^{\infty} \frac{n+3-i}{2^{i+2}} = \sum_{i=0}^{\infty} \frac{n+3-(i+kc)}{2^{i+kc+2}} = \frac{1}{2^{kc+2}} \sum_{i=0}^{\infty} \frac{n+3-(i+kc)}{2^i} =$$

$$= \frac{1}{2^{kc+2}} \left(\sum_{i=0}^{\infty} \frac{n+3-kc}{2^i} - \sum_{i=0}^{\infty} \frac{i}{2^i} \right) = \frac{1}{2^{kc+2}} (2(n+3-kc)-2) =$$

$$= \frac{2n+4-2kc}{2^{kc+2}} = \frac{n+2-kc}{2^{kc+1}}$$

Now for the inner sum, we get

$$\sum_{k=1}^{\infty} \frac{n+2-kc}{2^{kc+1}} = \sum_{k=0}^{\infty} \frac{n+2-(k+1)c}{2^{(k+1)c+1}} = \sum_{k=0}^{\infty} \frac{n+2-kc-c}{2^{kc+c+1}} =$$

$$= \frac{1}{2^{c+1}} \sum_{k=0}^{\infty} \frac{n+2-kc-c}{2^k c} = \frac{1}{2^{c+1}} \left(\sum_{k=0}^{\infty} \frac{n+2-c}{(2^c)^k} - \sum_{k=0}^{\infty} \frac{kc}{(2^c)^k} \right) =$$

$$= \frac{1}{2^{c+1}} \left(\frac{(n+2-c)2^c}{2^c-1} - \frac{c2^c}{(2^c-1)^2} \right) = \frac{(n+2-c)(2^c-1)-c}{2(2^c-1)^2}$$

This is the contribution of the extra bits over n bits of the message. It follows that the contribution per one bit is

$$\lim_{n\to\infty} \frac{(n+2-c)(2^c-1)-c}{2(2^c-1)^2 n} = \lim_{n\to\infty} \frac{n(2^c-1)}{2(2^c-1)^2 n} = \frac{1}{2(2^c-1)}$$

With the parameter values we measured, we get $c = 8$. Thus we get an average of $\frac{1}{510}$ extra bits per a bit of message, which is negligible. The total bit transmission time can be expressed as

$$g \left(1 + \frac{1}{2(2^c-1)} \right) = d_{max} \left(1 + \frac{1}{2(2^c-1)} \right) \approx g = d_{max}$$

For our values of parameters, this is approximately $8.016s$, which yields a bitrate of approximately

$0.125b/s$.

2.3.2 Speedup by lowering granularity

In the following text and computations, for the sake of simplicity, we will usually not take the negligible contribution of consecutive sequences of 1s into account. One may also notice that by reducing the granularity g, the number c increases, which makes the contribution to the transmission time even more negligible.

We will demonstrate the idea of lowering the value of g using the particular parameter values from our measurements. Let us halve the value of g, thus getting $g = \frac{1}{2}d_{max} = 4s$. It then holds that

$$(d_{min}, d_{max}) = (5s, 8s) \subseteq (4s, 8s) = (g, 2g)$$

and also

$$(h_{min}, h_{max}) = (2.3s, 3.7s) \subseteq (0s, 4s) = (0, g)$$

which means that hanging up can still be done in one frame, while dialing always takes exactly two frames. Let us see how this change in dialing affects transmission. We dial when the message contains the subsequence 01. If $g = d_{max}$, after transmitting 0 there is enough time to dial and send 1. However, with $g = \frac{1}{2}d_{max}$, the receiver will never see ringing after g time, but only after $2g$ time. Thus the receiver will read 001 instead of 01.

This is not a problem. Whenever the sender sees 01 in the message, they can deliberately insert one more frame between 0 and 1 to allow for longer dialing time. The receiver is then sure to never encounter less than two 0s before each 1 and can always interpret 001 as 01. The message is then correctly transmitted. This forces us to send an extra 0 bit for every occurence of 01 in the message. The probability of each bit being 0 is $\frac{1}{2}$ and the probability of it being followed by a 1 is again $\frac{1}{2}$. To be more precise, there is a small probability that the considered bit is the last one of the message, thus it is not followed by anything, meaning the probability of being followed by 1 is slightly less than $\frac{1}{2}$. But for message length $n \to \infty$ the probability of a bit being last is $\to 0$. In total, we get that $\frac{1}{4}$ of bits generate this extra 0 bit. The size of message has grown to

$$n' = n + \frac{1}{4}n = \frac{5}{4}n$$

but since $g = \frac{1}{2}$, the total transmission time is

$$n'g = \frac{5}{4}n \cdot \frac{1}{2}d_{max} = \frac{5}{8}nd_{max}$$

so the message is transmitted 37.5% faster than original (disregarding the negligible contribution of extra bits from consecutive sequences of 1s). In our case, this means sending a bit every $5s$.

This idea can be generalized. Apparently nothing can be gained by $g > d_{max}$; in fact, the l limit prevents us from having g too large. However, it seems reasonable to take $g \to 0$ in order to minimize the time of one frame. However, we will require that g is not too small either, but that it holds

$$\Delta d, \Delta h \leq g$$

i.e. the whole ranges of d and h fit into one frame. This means that even if dialing or ringing take several frames, we will always know exactly in which one will the event happen. The number of frames required to dial, denoted f_d, and the number of frames required to hang up, denoted f_h, can evidently be computed as

$$f_d = \left\lceil \frac{d_{max}}{g} \right\rceil$$

$$f_h = \left\lceil \frac{h_{max}}{g} \right\rceil$$

Note that it must not necessarily hold that

$$(d_{min}, d_{max}) \in ((f_d - 1)g, f_d g)$$

$$(h_{min}, h_{max}) \in ((f_h - 1)g, f_h g)$$

By definition of f_d and f_h, d_{max} and h_{max} must always fit into the f_d-th and f_h-th frame, respectively, but their minimum counterparts might not. For example, for our values of parameters and $g = 3s$, ringing takes three frames. If the ringing time is minimal ($d = d_{min}$), the receiver's phone will start ringing in the second frame already. This can be alleviated if the sender starts dialing one second into the first frame, not right in its beginning. As $1s + d_{min} = 6s$ and $1s + d_{max} = 9s$, the ringing now always fits into the third frame. In general, we usually have to start dialing with a certain delay ε. The delay might be

$$\varepsilon = (f_d - 1)g - d_{min}$$

so that d_{min} fits right into the beginning of the f_d-th frame, or

$$\varepsilon = f_d g - d_{max}$$

so that d_{max} fits right into its end, or anything in between. Our lower bound on g implies that in either option the phone is guaranteed to happen in the f_d-th frame.

$$(\varepsilon + d_{min}, \varepsilon + d_{max}) \in ((f_d - 1)g + f_d g)$$

An analogous idea is applicable for h.

We can now revisit the problematic part of this speedup. When transmitting 01, dialing now takes f_d

intervals instead of 1, thus $f_d - 1$ extra 0s are "accidentally" transmitted as well. We end up transmitting $0 \dots 01$ with f_d 0s. For hanging up, we instead of transmitting 10 we always transmit $1 \dots 10$, consisting of f_h bits of 1 followed by 0. This knowledge can be used by the receiver to decode the message properly by ignoring the correct number of 0s or 1s in every $0 \to 1$ or $1 \to 0$ transition.

We have established that for each 0 followed by 1, we have to add $f_d - 1$ extra bits. As discussed before, this happens for $\frac{1}{4}$ of bits in the message. Similarly, in $\frac{1}{4}$ of cases we have 1 followed by 0. In this case, $f_h - 1$ extra bits are added. It follows that the total time necessary to transmit n bits (again disregarding the extra bits from consecutive sequences of 1s) is

$$n\left(g + \frac{1}{4}(f_d - 1)g + \frac{1}{4}(f_h - 1)g\right) = \frac{ng}{4}(2 + f_d + f_h)$$

The transmission time per bit is then obtained by dividing by n, i.e.

$$\frac{g}{4}(2 + f_d + f_h) = \frac{g}{4}\left(2 + \left\lfloor \frac{d_{max}}{g} \right\rfloor + \left\lfloor \frac{h_{max}}{g} \right\rfloor\right)$$

Let us explore how much improvement could be done if it was possible to decrease $g \to 0$. In reality, this can only happen if d and h have no variance, i.e. they are non-random, otherwise their ranges will not fit into arbitrarily small intervals. And of course, for small g we would also hit the technical limitation of measuring time with high precision. Taking $g \to 0$, we get

$$\lim_{g\to 0} \frac{g}{4}\left(2 + \left\lfloor \frac{d_{max}}{g} \right\rfloor + \left\lfloor \frac{h_{max}}{g} \right\rfloor\right) = \lim_{g\to 0} \frac{g}{4}\left(\left\lfloor \frac{d_{max}}{g} \right\rfloor + \left\lfloor \frac{h_{max}}{g} \right\rfloor\right) =$$

$$\lim_{g\to 0} \frac{g}{4}\left(\frac{d_{max}}{g} + \frac{h_{max}}{g}\right) = \frac{1}{4}(d_{max} + h_{max})$$

Compared with the original transmission time d_{max}, the speedup we achieved is

$$\frac{\frac{1}{4}(d_{max} + h_{max})}{d_{max}} = \frac{1}{4}\left(1 + \frac{h_{max}}{d_{max}}\right)$$

so depending on the ratio between h_{max} and d_{max}, the transmission time could be lowered as much as to $\frac{1}{2}$ or even $\frac{1}{4}$ of the original speed.

Unable to actually use $g \to 0$, we want to use the value of g that yields minimum transmission time. We may observe that the transmission time increases linearly with g in every interval whose endpoints share the same values of the ceiling functions appearing in f_d and f_h. However, increasing g beyond a value of which either d_{max} or h_{max} is a multiple, the respective ceiling function decreases by one. This means that using the minimum possible value of g

$$g = \max\{\Delta d, \Delta h\}$$

is not necessarily optimal. Instead, we must search through the relevant points

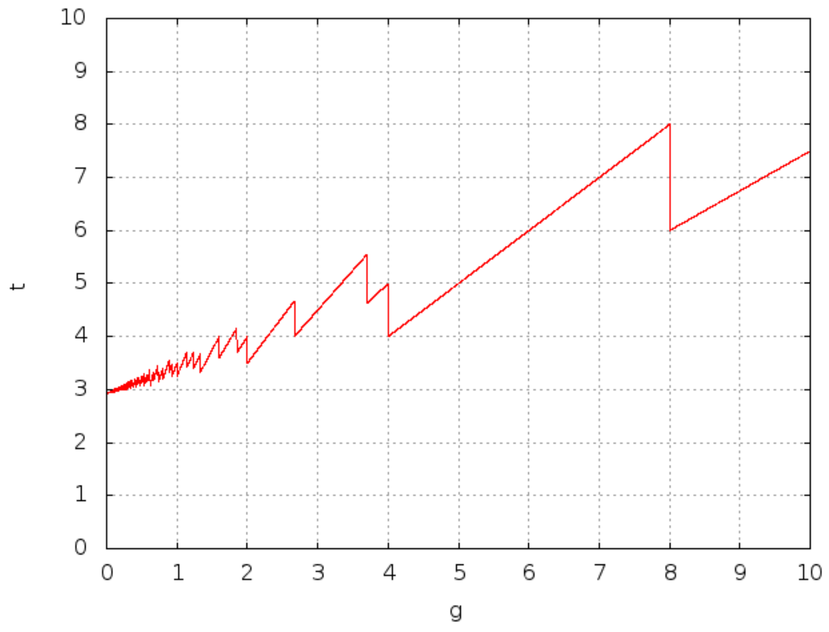


Figure 2.8: Transmission time as a function of g for our values of parameters.

$$\left\{ \frac{d_{max}}{k}, \frac{h_{max}}{k} \,\middle|\, k-1 \in \mathbb{N} \right\} \cap [\max\{\Delta d, \Delta h\}, g]$$

to find the minimum. In Figure 2.8, we can see the graph of transmission time on g for our values of parameters. In Figure 2.9, we can observe that the optimal value of g is not the minimal feasible one, $3s$, but in fact $4s$. This is actually the value we used in the beginning of this section, where we achieved the transmission speed $5s$.

2.3.3 Synchronization

So far, our formulation of the protocol assumed that the participants have their clocks synchronized, so that every frame elapses at the same time from the perspective of them both. However, this assumption is not always reasonable. First of all, the communication may take place between two participants that have never met and never had the chance to synchronize their clocks in advance. Even if they both use some kind of external synchronization, we usually require high precision (the parameters d, h and g are all in order of seconds; even as small missynchronization as one tenth of a second would make trouble).

We will therefore explore possible extensions to this protocol which make it less dependent on clock synchronization - presumably with loss of bitrate. This is also interesting for the purpose of comparing the protocols. The Ring-or-Silence protocol turned out to be much faster than the previous two protocols, but those two did not require any synchronization whatsoever. Thus, removing this requirement from this protocol as well will allow us to make a fairer comparison.

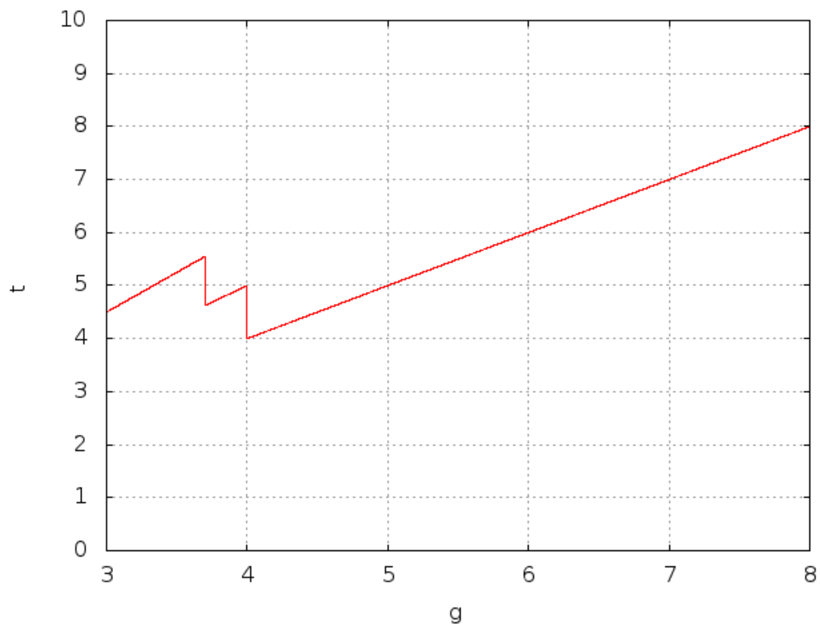


Figure 2.9: Transmission time as a function of g for our values of parameters. Detail of the feasible interval.

Our target is to align the frames of sender and receiver as precisely as possible. The simplest way to do this is to utilize their knowledge of parameters' values. The sender may commence the transmission with a preamble[3] which conists of one extra ringing. The sender dials at the time t_0^s, the beginning of their first frame. The receiver's phone starts ringing at $t_0^r \in \{t_0^s + d_{min}, t_0^s + d_{max}\}$. The receiver will then mark $t_0^r - d_{min}$ as the start of their frame. This means that the misalignment between the corresponding endpoints of receiver's and sender's frames is $t_0^r - t_0^s \leq \Delta d$. In other words, when the frame elapses from the sender's viewpoint, the sender knows that on the receiver's side this happens sometime between now and Δd time later. The sender does not know when in this time window is the receiver going to check their phone state, so the sender must assure that the state is kept constant throughout it. For example, when the sender is sending 1, the sender must make sure the receiver's phone is already ringing at the beginning of this window and keep it ringing until the end. Between these time windows, we must allow d_{max} time to change states to change states, just like in the original version of this protocol. This increases our requirements for the frame length to

$$g = d_{max} + \Delta d = 2d_{max} - d_{min}$$

However, we still want the receiver's frame to end between $t_0^s + d_{max}$ and $t_0^s + d_{max} + \Delta d$ and since we just prolonged g by Δd, we must shift the beginning of the receiver's frame Δd back. The beginning is now between $t_0^s - \Delta d$ and t_0^s.

We aligned the participants' timing using a single dialing. The uncertainty of dialing time is what

[3]As usual, the extra time contribution may be ignored, as it is amortized for message length $n \to \infty$.

determined the misalignment error size. Apparently, the same could be done with hangup, whereas the error will be equal to the range of h. The sender rings, then hangs up at what they will consider to be the time t_0^s. When the hangup signal reaches the receiver at time t_0^r, they will mark $t_0^r - h_{min}$ as the beginning of their frame. Similarly as in the previous case, we will require a time window of $h_{max} - h_{min}$ to check the state. This leads us to the frame length

$$g = d_{max} + \Delta h$$

With our values of parameters, this results in $g = 9.4s$, which also means per-bit transmission speed of $9.4s$ for this protocol. Note that we normally assume that $d_{max} > h_{max}$, but that doesn't necessarily mean that $d_{max} - d_{min} > h_{max} - h_{min}$. In our case, it was indeed the h parameter that had smaller range and thus was more effective to use in synchronization. But in general, either of the two may be the better one. For further reference, let us denote the size of this time window w.

$$w = \min\{\Delta d, \Delta h\}$$

$$g = d_{max} + w$$

Note that we can understand this version of the protocol as follows. In the original (synchronized) version with synchronized clocks, the sender always had a time interval of length d_{max} to change the state if needed and they have to make sure that at the end of every such interval, the receiver's phone is in the required state. It is very similar in this version of the protocol as well. The sender again has d_{max} time to change the state, only now there is not a single moment, but a whole time window of length w where the receiver's phone must be kept in the required state. The receiver checks the state regularly (in period of g) and although the actual check only takes a single moment, neither sender nor receiver knows where exactly in that time window this moment is.

Let us now examine how can we reuse the idea of speedup by lowering the granularity. In the synchronized version, we required that the entire ranges of d and h still fit into a single frame. The same is true now as well, but they always have to fit into the transmitting part of the frame, as the state must never change in the checking part. Therefore, the minimum frame length is now increased by w

$$g \geq \max\{\Delta d, \Delta h\} + w$$

The fact that no events may occur in the checking window of a frame now must be taken into account in the computation of f_d and f_h. Ringing can occur no later than $f_d g - w$ and similarly, hangup can occur no later that $f_h g - w$. It follows that

$$f_d = \lceil d_{max} + wg \rceil$$

$$f_h = \lceil h_{max} + wg \rceil$$

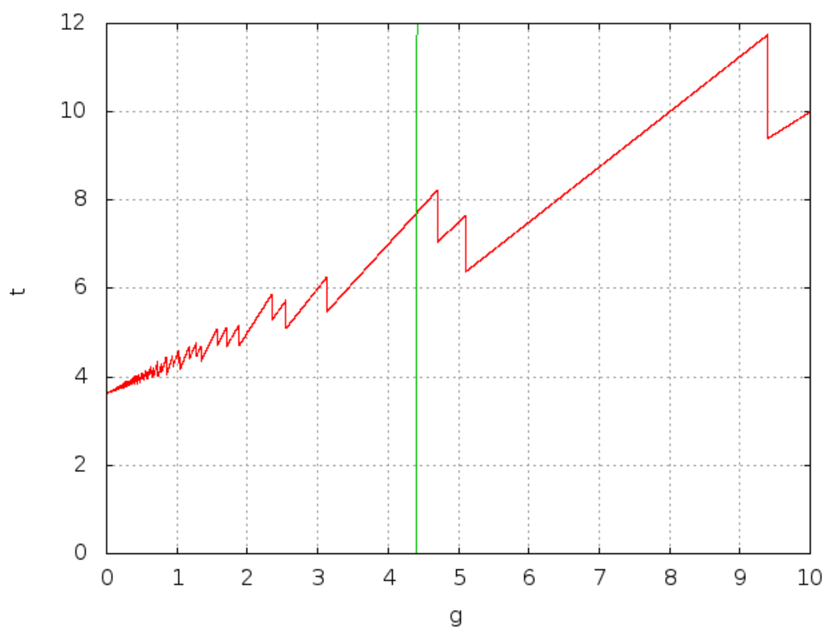


Figure 2.10: Transmission time as a function of g for our values of parameters in the unsynchronized case. The green line is the minimal feasible value boundary $4.4s$.

The transmission time can now be rewritten as

$$\frac{g}{4}(2+f_d+f_h)=\frac{g}{4}(2+f_d+f_h)=\frac{g}{4}\left(2+\left\lfloor\frac{d_{max}+w}{g}\right\rfloor+\left\lfloor\frac{h_{max}+w}{g}\right\rfloor\right)$$

The optimum value of g can be again found by trying the points in which the ceiling functions contained in f_d, f_h change value. For our values of parameters, the checking window size is $w = \Delta h = 1.4s$, which means that we require $g \geq 3s + 1.4s = 4.4s$. In Figure 2.10, we can see that the optimal value is $g = 5.1s$. From that, we obtain $f_d = 2, f_h = 1$ and finally the transmission time $6.375s$.

Although the lack of synchronization has increased the transmission time, we still achieved a better result than in the previous two protocols.

2.4 Alternating Ringing protocol

The last protocol we are going to explore is the Alternating Ringing protocol, which could be regarded as a generalization of the Ringing Length protocol. There, we had to dial and hang up once per each block and we have seen that especially dialing is an expensive operation. We can share every dialing and hangup between two blocks if we encode blocks not only into ringing, but also into the silence between two ringings. We can alternate between *ringing blocks* and *silent blocks*, hence the protocol name.

2.4.1 Description

The participants need to compute the optimal block sizes b_r, b_s for ringing and silent blocks, respectively. The protocol thus proceeds as follows. The sender splits the message into alternating blocks of b_r and b_s bits. The sender will then commence the transmission by dialing the receiver and waiting for d_{max} time to make sure the receiver's phone is already ringing. The sender will then keep ringing for $x \cdot g_r$ time, where $x \in \{0, \dots, 2^{b_r} - 1\}$ is the value of the first block. The sender will then hang up. After h_{max} time, the sender knows that the receiver's phone is now idle. The sender will now wait for $x \cdot g_s$ ($x \in \{0, \dots, 2^{b_s} - 1\}$) time to transmit b_s bits of value x from the second block. The process then repeats for every other chunk of $b_r + b_s$ bits.

Note that it is practical to add a rule that the transmission always ends with a ringing block, i.e. an odd number of blocks is always sent. In case that this does not hold, padding can be added to the end of the message so that it would. The reason is that if a silent block was the last one in the message, it would not be possible to determine what was its length, unless we want to finish the transmission by dialing.

We used g_r, g_s to denote the granularities in ringing and silent blocks. The way in which the ringing blocks are sent is exactly the same as in the Ringing Length protocol. From there, we know that that granularity has to take into account the variance of d and h parameters.
$g_r = \Delta d + \Delta h$
This result came from the observation of possible errors in the ringing length. It is not difficult to see that we deal with the same problem in the case of silent blocks, just in the opposite order. In our description of the Ringing Length protocol, we designated the base case hangup time to be $h = h_{min}$ and if $h > h_{min}$, it meant the ringing was longer and there was an error. Now, we have to look at the silent block which follows this hangup. If h_{min} is the base state, then $h > h_{min}$ will lead to a shorter silence time. Similarly in the case of dialing, $d = d_{max}$ was the base state and $d < d_{max}$ meant that the ringing took longer time. For the silent block that comes before the ringing, $d < d_{max}$ means that the receiver's phone starts ringing sooner, shortening the silence time.

To summarize, dialing may be up to Δd time shorter than expected, prolonging the ringing, but shortening the silence. Hanging up may take up to Δh time more, prolonging the ringing, but shortening the silence. When sending a block with value x, let us denote $error_r$ the difference between the actual ringing time and ringing time in the base case. Let us use $error_s$ to denote the same for a silent block. Evidently, it holds that

$$0 \leq error_r < \Delta d + \Delta h = g_r$$

for the ringing error, what we have alredy established for the Ringing Length protocol (inequality (2.2)). Symmetrically, it holds that

$$0 \leq -error_s < \Delta d + \Delta h = g_s$$

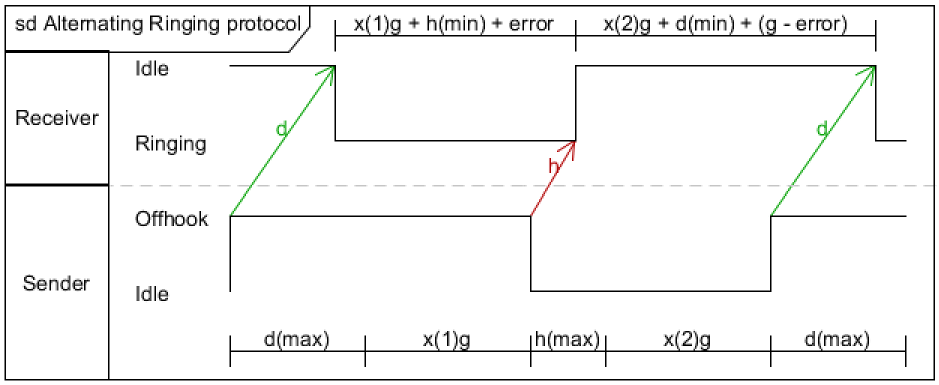


Figure 2.11: The Alternating Ringing protocol - sending a ringing block with value x_1 and a silent block with value x_2.

for the silence error. As ringing and silent blocks have the same maximum absolute size of error, they should also use the same granularity. Therefore, from now on we will write g instead of g_b and g_s.

$$g_r = g_s = g$$

Note that the error is always negative in the case of silent blocks. Let us see that there is always room for this negative error. When sending the number x, the silence length perceived by the receiver in the base case is

$$d_{max} + \Delta h + xg$$

This is because in the base case dialing takes full d_{max} time and hanging up takes only h_{min} time, while the sender waits for h_{max}. It follows that.

$$d_{max} + \Delta h + xg \geq d_{max} + \Delta h > \Delta d + \Delta h \geq |error_s|$$

for any x, which means that it is indeed always possible to encounter this error. The shortest possible ringing time is then the above expression reduced by the maximum error with the minimum value of $x = 0$.

$$d_{max} + \Delta h - g = d_{max} + \Delta h - (\Delta d + \Delta h) = d_{min}$$

Indeed, this is reached when hanging up takes full h_{max} and thus does not contribute to the silence time, while dialing takes the minimum possible time d_{min}.

Decoding

In the Ringing Length protocol (and in the ringing blocks in this protocol) the minimum possible ringing time was h_{min}. This was reflected in the decoding formula (2.3), where it had to be subtracted from the total ringing time.

$$x = \left\lfloor \frac{t_{ring} - h_{min}}{g} \right\rfloor$$

As d_{min} is the minimum possible silence time in silent blocks, we may see that a similar formula will hold for decoding the silent blocks, only containing d_{min} instead of h_{min}. Indeed, as we want to map

$$(d_{min} + xg, d_{max} + \Delta h + xg) = (d_{min} + xg, d_{min} + (x+1)g) \to x$$

it is evident that the formula for x is

$$x = \left\lceil \frac{t_{silence} - d_{min}}{g} \right\rceil$$

where $t_{silence}$ is the length of the silence measured by the receiver.

Transmission time

The communication in this protocol consists of a repeating pattern of dialing, ringing, hanging up and silence, which serves to send a pair of ringing and silent blocks. The exception is the final block, which we required to be a ringing one. However, this one-time contribution need not to be taken into account, as the time spent on this block is amortized to zero for message length $n \to \infty$.

This pattern of sending a pair of a ringing and silent block takes

$$d_{max} + h_{max} + \frac{g(2^{b_r} - 1)}{2} + \frac{g(2^{b_s} - 1)}{2}$$

The first two terms account for the dialing and hanging up, the last two stand for the average time spent on respective blocks (for derivation thereof, see the formula (2.1) Ringing Length protocol). The transmission time per one bit as a function t of two block sizes b_r, b_s can then be expressed by averaging the above formula for $b_r + b_s$ bits

$$t(b_r, b_s) = \frac{d_{max} + h_{max}}{b_r + b_s} + g\frac{2^{b_r} + 2^{b_s} - 2}{2(b_r + b_s)} \tag{2.11}$$

To find the optimal block size, we must find the pair of arguments b_r, b_s that yields the minimum possible value of t. We must also satisfy the condition on the maximum ringing length which states that b_r has an upper bound, denoted b_{max}

$$b_r \leq \log_2\left(\frac{l - d_{max} + d_{min} - h_{max}}{g} + 1\right) = b_{max} \tag{2.12}$$

The derivation of this constraint is again seen in the Ringing Length protocol as inequality (2.5). Note that there is no pair constraint for b_s, as there is no limit to the maximum length of silence. This is an important advantage of the Alternating Ringing protocol over the Ringing Length one - if the optimal

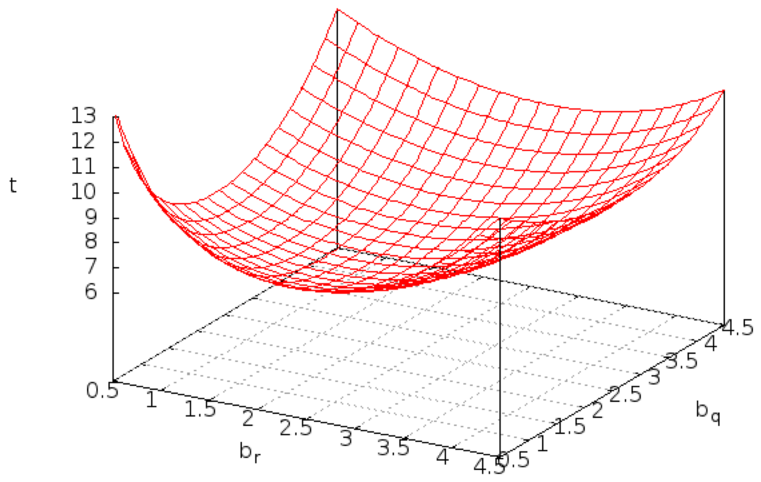


Figure 2.12: Transmission time per bit as a function of two block sizes is a convex function with a single local and global minimum.

block size is bigger than the l limit permits, we can still use it in half of the blocks (the silent ones).

It can be shown that t limited on $b_r, b_s > 0$ is a convex function with a single stationary point, which is its local and global minimum (see Figure 2.12). If it satisfies the condition on l, it constitutes our solution. We can find this point by setting the partial derivatives of t to zero.

$$\frac{\partial}{\partial b_r} t(b_r, b_s) = 0$$

$$\frac{\partial}{\partial b_s} t(b_r, b_s) = 0$$

$$\frac{d_{max} + h_{max} + \frac{g}{2}(2^{b_r}((b_r + b_s)\ln 2 - 1) + 2^{b_s} - 2)}{-(b_r + b_s)^2} = 0$$

$$\frac{d_{max} + h_{max} + \frac{g}{2}(2^{b_s}((b_s + b_r)\ln 2 - 1) + 2^{b_r} - 2)}{-(b_r + b_s)^2} = 0$$

$$d_{max} + h_{max} + \frac{g}{2}(2^{b_r}((b_r + b_s)\ln 2 - 1) + 2^{b_s} - 2) = 0$$

$$d_{max} + h_{max} + \frac{g}{2}(2^{b_s}((b_s + b_r)\ln 2 - 1) + 2^{b_r} - 2) = 0$$

Without having to solve these equations, we can see that one is produced from the other by cyclic permutation of the variables b_r and b_s. It must therefore hold for the minimum that $b_r = b_s$. We now know that it is optimal to use the same block size for ringing and silent blocks (unless the l limit prevents this). We will denote this block size b and can now rewrite t as a function of one variable, denoted $t_1(b)$

$$t_1(b) = t(b,b) = \frac{d_{max} + h_{max}}{b+b} + g\frac{2^b + 2^b - 2}{2(b+b)} = \frac{d_{max} + h_{max} + g(2^b - 1)}{2b}$$

This simplifies our task of finding the stationary point, as instead of two partial derivatives we now only have to find the derivative of a single variable function. Let the sought minimum point of t be denoted $[b_{opt}, b_{opt}]$, then the minimum of t_1 is b_{opt}.

$$t_1'(b_{opt}) = 0$$
$$-\frac{d_{max} + h_{max} + g(2^b_{opt}(1 - b_{opt}\ln 2) - 1)}{4b^2_{opt}} = 0$$
$$d_{max} + h_{max} + g(2^b_{opt}(1 - b_{opt}\ln 2) - 1) = 0$$
$$g(2^b_{opt}(1 - b_{opt}\ln 2) - 1) = -(d_{max} + h_{max})$$
$$2^b_{opt}(1 - b_{opt}\ln 2) = -\frac{d_{max} + h_{max} - g}{g}$$
$$e^{b_{opt}\ln 2}(b_{opt}\ln 2 - 1) = \frac{d_{max} + h_{max} - g}{g}$$
$$e^{b_{opt}\ln 2 - 1}(b_{opt}\ln 2 - 1) = \frac{d_{max} + h_{max} - g}{eg}$$
$$b_{opt}\ln 2 - 1 = W\left(\frac{d_{max} + h_{max} - g}{eg}\right)$$
$$b_{opt} = \frac{W\left(\frac{d_{max}+h_{max}-g}{eg}\right) + 1}{\ln 2}$$

As b_{opt} may be non-integer, the actual solution, denoted $[b_r^{int}, b_s^{int}]$, may be found as one of the four neighbouring integer points

$$[b_r^{int}, b_s^{int}] = \underset{b_r, b_s}{\arg\max}\big\{t(b_r, b_s) \mid b_r, b_s \in \{\lfloor b_{opt}\rfloor, \lceil b_{opt}\rceil\}\big\}$$

Any farther integer points will yield a higher value due to t being convex. Furthermore, the symmetrical nature of t implies that

$$t\left(\lfloor b_{opt}\rfloor, \lceil b_{opt}\rceil\right) = t\left(\lceil b_{opt}\rceil, \lfloor b_{opt}\rfloor\right)$$

so in fact we only have three points to choose from. Out of the two points in the above equation, the former one is preferred. This is because of the constraint we have on b_r. If $b_r = \lceil b_{opt}\rceil$ satisfies the constraint, then $b_r = \lfloor b_{opt}\rfloor$ also does, but not necessarily the other way around.

If the solution $[b_r^{int}, b_s^{int}]$ does not satisfy the constraint on b_r, i.e. $b_r^{int} > b_{max}$, we have to find a solution which does. This means finding the minimum of $t(b_r, b_s)$ in the half-space determined by $b_r \leq b_{max}$. From the assumption that $b_{max} < b_r^{int}$, it follows that $b_{max} < \lceil b_{opt} \rceil$ and so any integer smaller than b_{max} is also smaller than b_{opt}. From this and from the convexity of t, it follows that for fixed b_s and decreasing b_r the value of $t(b_r, b_s)$ increases. This compels us to set b_r to the highest integer value smaller than b_{max}, which is $\lfloor b_{max} \rfloor$.

$$b_r^{int} = \lfloor b_{max} \rfloor$$

We now have the solution for b_r. By fixing this value in t, we get a function of one variable. The solution for b_s then can be found by again setting its derivative to zero.

$$\frac{\partial}{\partial b_s} t(b_r^{int}, b_s) = 0$$

$$\frac{d_{max} + h_{max} + \frac{g}{2}(2^{b_s}((b_s + b_r^{int}) \ln 2 - 1) + 2^{b_r^{int}} - 2)}{-(b_r^{int} + b_s)^2} = 0$$

$$d_{max} + h_{max} + \frac{g}{2}\left(2^{b_s}((b_s + b_r^{int}) \ln 2 - 1) + 2^{b_r^{int}} - 2\right) = 0$$

$$\frac{g}{2} 2^{b_s}((b_s + b_r^{int}) \ln 2 - 1) = -\left(d_{max} + h_{max} + \frac{g}{2}(2^{b_r^{int}} - 2)\right)$$

$$2^{b_s}((b_s + b_r^{int}) \ln 2 - 1) = -\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)$$

$$2^{b_s + b_r^{int}}((b_s + b_r^{int}) \ln 2 - 1) = -2^{b_r^{int}}\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)$$

$$e^{(b_s + b_r^{int}) \ln 2}((b_s + b_r^{int}) \ln 2 - 1) = -2^{b_r^{int}}\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)$$

$$e^{(b_s + b_r^{int}) \ln 2 - 1}((b_s + b_r^{int}) \ln 2 - 1) = -\frac{1}{e} 2^{b_r^{int}}\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)$$

$$(b_s + b_r^{int}) \ln 2 - 1 = W\left(-\frac{1}{e} 2^{b_r^{int}}\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)\right)$$

$$b_s = \frac{W\left(-\frac{1}{e} 2^{b_r^{int}}\left(\frac{2}{g}(d_{max} + h_{max}) + (2^{b_r^{int}} - 2)\right)\right) + 1}{\ln 2} - b_r^{int}$$

Let us now calculate the optimal block sizes for our values of parameters. We get $b_{opt} \approx 2.0291$. Checking neighbouring integer points, we obtain $t(2,2) = 6.225$, $t(2,3) = 6.74$ and $t(3,3) = 7.08\overline{3}$. It is therefore optimal to choose $b_r = b_s = 2$. As we get $b_{max} \approx 3.713$, this solution satisfies the constraint on l and is therefore feasible. The bit transmission time of this protocol is therefore $6.225s$, what equals a bitrate of $\sim 0.161 b/s$.

2.4.2 Speedup

As it was shown to not be very useful, we will not return to the Gray code method (subsection 2.2.2) of speedup from the Ringing Length protocol. The Modulo method (subsection 2.2.2), on the other hand, can be reapplied here. To reiterate, the Modulo method works by

- Sending the message fast with a lowered granularity $\frac{g}{k}$.
- As the value x of every block can now be misinterpredted by up to $\pm(k-1)$, sending the value $x \ mod \ k$ to correct the result.

If the time saved by lowering granularity is greater than that spent sending the correction, we obtain a speedup.

Non-integer block sizes

As we will only use the Modulo method of speedup and not the Gray code one, it is no longer necessary (or useful) to regard a block as a sequence of bits, but rather as a single value. Normally, we would limit the block value (divided by g) to the range

$$\left\{0, \dots, 2^b - 1\right\}$$

for some number b, which we call block size. However, we may equally well use the range

$$\{0, \dots, r-1\}$$

for an arbitrary $r \in \mathbb{N}$, thus sending $\log_2 r$ bits. We will call the number r block range. We will use the notation r_r, r_s for the block ranges of ringing and silent blocks, respectively. The transmission time per bit is then rewritten by substituting $b_r = \log_2 r_r$ and $b_s = \log_2 r_s$ into $t(b_r, b_s)$

$$t(b_r, b_s) = \frac{d_{mar} + h_{mar}}{b_r + b_s} + g\frac{2^{b_r} + 2^{b_s} - 2}{2(b_r + b_s)}$$

$$t(\log_2 r_r, \log_2 r_s) = \frac{d_{mar} + h_{mar}}{\log_2 r_r + \log_2 r_s} + g\frac{r_r + r_s - 2}{2(\log_2 r_r + \log_2 r_s)}$$

$$t(\log_2 r_r, \log_2 r_s) = \frac{d_{mar} + h_{mar}}{\log_2 r_r r_s} + g\frac{r_r + r_s - 2}{2\log_2 r_r r_s)}$$

and the upper bound on r_r (see again inequality (2.5)) changes to

$$r_r \leq \frac{l - d_{max} + d_{min} - h_{max}}{g} + 1 = r_{max}$$

We can now get closer to the minimum of $t(b_r, b_s)$ than it was possible before. Instead of requiring b_r, b_s to be integer, we now only need them to be a binary logarithm of some integer, what is a strictly weaker condition. For our values of parameters, we have already computed that $b_{opt} \approx 2.0291$. Before, this would limit to use a block size of 2 or 3. Now, we can compute $r_{opt} = 2^b_{opt} \approx 4.082$ and use a block range of 4 or 5, i.e a block size of $\log_2 4 = 2$ or $\log_2 5$. We get $t(2,2) = 6.225$, $t(2, \log_2 5) = t(\log_2 5, 2) \approx 6.2703$ and $t(\log_2 5, \log_2 5) \approx 6.3094$. The minimum transmission time is achieved for $b_r = b_s = 2$, so in this case the integer block sizes were coincidentally already optimal.

For simplicity, let us now use assume that $r_{int} < r_{max}$, so we can use a common block range for both types of blocks, $r = r_r^{int} = r_s^{int}$. The only thing that remains to discuss is how to encode and decode the data. If a block carries $\log_2 r$ bits, to send an n-bit message we require $\left\lceil \frac{n}{\log_2 r} \right\rceil = \lceil n \log_r 2 \rceil$ blocks. The total number of configurations of values of these blocks is

$$r^{\lceil n \log_r 2 \rceil} \geq r^{n \log_r 2} = 2^n$$

Thus, by numbering these configurations in any order (e.g. lexicographical), we can transmit any n-bit message by interpreting it as a number from $\{0, \ldots, 2^n - 1\}$ and sending blocks with the corresponding configuration number. Similarly, the receiver decodes the transmission by computing the configuration number and outputting its n-bit binary notation.

The problem with this naïve approach is that we require the whole message to be transmitted before it can be decoded. Instead, we would like the receiver to be able to decode and output the message gradually, few bits after every block arrives. To achieve this, we can use Arithmetic coding[WNC87] (or rather a special case thereof, as we do not have to deal with varying probabilities of characters). The message always consists of an integer number of bits n. We will interpret the message as the binary notation of a fraction part of a certain number $v \in [0, 1)$, i.e.

$$v = \sum_{i=1}^{n} b_i 2^{-i}$$

where b_i is the $i - th$ bit of the message (counting from one). As each block represents one value from $\{0, \ldots, r - 1\}$, we may consider them to be r-ary digits. We want to express v in r-ary system. Although v has a finite binary notation, this may not be true for its r-ary notation. However, this is not a problem. Representations of all the possible n-bit messages are multiples of 2^{-n}. Each r-ary digit basically narrows down the interval of possible values of v to one r-th, so using m digits we can pinpoint v to an interval of size r^{-m}. Since we want to only match one of the multiples of 2^{-n}, the one that represents our message, we require that

$$r^{-m} \leq 2^{-n}$$

$$-m \leq \log_r 2^{-n}$$

$$m \geq n \log_r 2$$

which means that $n \log_r 2$ blocks are sufficient to encode n bits of message. Since by definition every block carries $\log_2 r$ bits, $n \log_r 2$ blocks carry $n \log_r 2 \log_2 r = n$ bits, so this number of blocks is also the minimum necessary. This means that Arithmetic coding is lossless and thus optimal in our case. There is no disadvantage compared to the naïve approach. More importantly, it has the desired property of gradual decoding. Whenever we narrow v down to an interval where all the possible solutions share first few bits, we can already output those bits.

Correlations between consecutive blocks

In the Ringing Length protocol, the Modulo method using $g' = \frac{g}{k}$ requires us to send $\log_2 k$ error-correcting bits for every block, as we always require a number $x \ mod \ k \in \{0, \dots, k-1\}$. The situation is different in the Alternating Ringing protocol. As every two consecutive blocks share either dialing or hangup, the errors that may appear in them are not independent. This is because each dialing or hangup contributes the same amount to the errors of the two respective blocks that it is a part of. This means that we will only need to send $\log_2 k$ error correction bits for every other block; for the ones inbetween we will need less information to compute the error. In our case, it means that we will send full error correction information for the ringing blocks, and only additional information for the silent blocks.

Suppose that $\Delta d \geq \Delta h$. Note that if this is not the case, the whole following process may be done the same way with the roles of h and d reversed. Let us use the granularity $g' = \frac{g}{k}$, thus causing up to $\pm(k-1)$ error in every block. We will explore the errors that could occur in a sequence of three consecutive blocks, starting with a ringing one. Let d_1, h_1, d_2, h_2 denote the dialing and hangup times of the first and the second ringing blocks, respectively. Naturally, h_1 and d_2 are also the hangup and dialing times of the silent block between them.

Consider that we received $\log_2 k$ error-correcting bits for each of the ringing blocks, making us able to decode their values - denoted x_1 and x_2, respectively. Let y_1, y_2 denote the numbers formerly interpeted by the receiver to be the values of the blocks (using formula (2.6) from section 2.2.2) before considering the error correction information. The errors of the blocks e_1, e_2 can be expressed as

$$e_1 \in [s_1 g', (s_1 + 1)g']$$

$$e_2 \in [s_2 g', (s_2 + 1)g']$$

Where $s_1 = y_1 - x_1, s_2 = y_2 - x_2$ denote the difference between receiver's perception of the block value and its real value. By definition, these errors are distances from the base state, where dialing takes d_{max} and hangup takes d_{min}. Thus,

$$e_1 = (d_{max} - d_1) + (h_1 - h_{min})$$

$$e_2 = (d_{max} - d_2) + (h_2 - h_{min})$$

Since the receiver knows x_1, x_2, y_1, y_2, they can compute e_1, e_2. With the help of above equations, this allows them to gain some information about d_1, d_2, h_1, h_2. First, let us look at the information that h_1 could give us. Consider that the error discovered by the receiver when comparing y_1 and x_1 was

$$s_1 = \left\lfloor \frac{\Delta h}{g'} \right\rfloor$$

$$e_1 \in \left[\left\lfloor \frac{\Delta h}{g'} \right\rfloor g', \left(\left\lfloor \frac{\Delta h}{g'} \right\rfloor + 1 \right) g' \right]$$

Let us inspect the possible values of h_1. It could hold that

$$h_1 = h_{min}$$

$$d_1 = d_{max} - g's_1$$

We can check that this is a correct assignment, i.e. that the error is in the correct range and that d_1 has a valid value

$$e_1 = (d_{max} - (d_{max} - g's_1) + (h_{min} - h_{min}) = g's_1 \in [s_1 g', (s_1 + 1)g']$$

$$d_1 = d_{max} - g's_1 = d_{max} - g' \left\lfloor \frac{\Delta h}{g'} \right\rfloor \geq d_{max} - \Delta h \geq d_{max} - \Delta d \geq d_{min}$$

Where the latter stems from our assumption that $\Delta d \geq \Delta h$. Similarly, the values of h_1 and d_1 could be such that

$$h_1 = h_{max}$$

$$d_1 = d_{max}$$

Indeed, we can check that in this case

$$e_1 = (d_{max} - d_{max}) + (h_{max} - h_{min}) = h_{max} - h_{min} = \Delta h = g' \frac{\Delta h}{g'} =$$

$$= g' \left(\left\lfloor \frac{\Delta h}{g'} \right\rfloor + \left\{ \frac{\Delta h}{g'} \right\} \right) = g' \left(s_1 + \left\{ \frac{\Delta h}{g'} \right\} \right) \in [s_1 g', (s_1 + 1)g']$$

We have found two different situations, one with $h_1 = h_{min}$ and the other with $h_2 = h_{max}$, and both of them yield error that leads to shift in value interpretation by s_1. If we take any linear combination of these situations, i.e.

$$h_1 = \alpha h_{min} + (1 - \alpha) h_{max}$$

$$d_1 = \alpha(d_{max} - g's_1) + (1 - \alpha) d_{max}$$

for $\alpha \in [0, 1]$, the resulting error will also be a linear combination of their respective errors

$$e_1 = \alpha g' s_1 + (1 - \alpha)\Delta h$$

and any such error must also fall in the interval $[s_1 g', (s_1 + 1)g']$, because its components did. We discovered that for a particular value of s_1, any value of h_1 is feasible. That means that some values of s_1 give us no knowledge about h_1. Since the sender cannot predict s_1, they must reckon that it may be unfavorable. Thus, they must reckon that h_1 can have any value. Note that if this was not the case, having some knowledge about h_1 would also give us some knowledge about the length of the silent block, possibly allowing us to spare some error correction bits. Unfortunately, it turns out that h_1 did not help

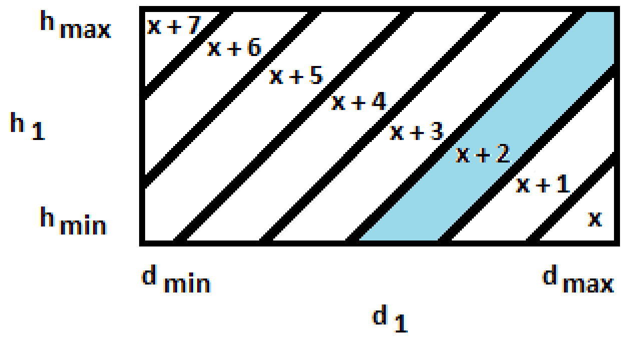


Figure 2.13: The diagram shows the value interpreted by the receiver ($y_1 = x_1 + s_1$) as a function of d_1 and h_1 for $k = 8$. There is always at least one value of s_1 (blue strip) that gives us no information on h_2.

us in this regard. See Figure 2.13 for a diagram of the above computation.

We will soon see that the situation is different with s_2. Consider any value of s_2. We are going to look at the possible values that d_2 might have. We know that error e_2 is bounded by

$$e_2 \in [s_2 g', (s_2 + 1)g']$$

$$s_2 g' \leq e_2 \leq (s_2 + 1)g'$$

$$s_2 g' \leq (d_{max} - d_2) + (h_2 - h_{min}) \leq (s_2 + 1)g'$$

If we express d_2 and use the the trivial bounds $h_{min} \leq h_2 \leq h_{max}$, we obtain

$$d_2 \leq d_{max} - s_2 g' + h_2 - h_{min} \leq d_{max} - s_2 g' + h_{max} - h_{min} = d_{max} - s_2 g' + \Delta h$$

$$d_2 \geq d_{max} - (s_2 + 1)g' + h_2 - h_{min} \geq d_2 \geq d_{max} - (s_2 + 1)g' + h_{min} - h_{min} = d_{max} - (s_2 + 1)g'$$

Thus, d_2 can only take values from the interval

$$[d_{max} - (s_2 + 1)g', d_{max} - s_2 g' + \Delta h] \cap [d_{min}, d_{max}]$$

whose size is

$$\min \{(d_{max} - s_2 g' + \Delta h) - (d_{max} - (s_2 + 1)g'), d_{max} - d_{min}\} = \min \{g' + \Delta h, \Delta d\}$$

Recall our assumption that $\Delta d \geq \Delta h$. Now if $\Delta d = \Delta h$, then

$$\min \{g' + \Delta h, \Delta d\} = \Delta d$$

We get that d_2 takes values from an interval of size Δd, which must be the interval (d_{min}, d_{max}), so we have no actual knowledge about d_2. However, if $\Delta d > \Delta h$, then for sufficiently large k (namely

$k > \frac{g}{\Delta d - \Delta h}$) we get that

$$g' + \Delta h = \frac{g}{k} + \Delta h < \frac{g}{\frac{g}{\Delta d - \Delta h}} + \Delta h = (\Delta d - \Delta h) + \Delta h = \Delta d$$

and so

$$min(g' + \Delta h, \Delta d) = g' + \Delta h < \Delta d$$

which means that we pinpointed the value of d_2 to an interval smaller than Δd. This knowledge will also locate the error of the silent block into a smaller interval, thus sparing us some error correction bits.

Let x_s denote the intended value of the silent block, y_s the value interpreted by the receiver, $s_s = y_s - x_s$ the shift between those two values and e_s the error of this block. We assume that the receiver has already received full error correcting information for the two ringing blocks and thus knows the values x_1, s_1, x_2, s_2 and thus has been able to pinpoint d_2 to an interval of size $g' + \Delta h$ by the above described process. We are now going to demonstrate that the receiver does not necessarily need $\log_2 k$ error correction bits for the silent block. We must start from the assumption that we have not yet received any error-correcting information and compute the minimum amount that we require. This also means that we do not yet know the values x_s and s_s.

Let us estimate the error of the silent block. Using the constraints we have on h_1, d_2

$$h_1 \leq h_{max}$$

$$d_2 \geq d_{max} - (s_2 + 1)g'$$

we obtain the upper estimate

$$e_s = (d_{max} - d_2) + (h_1 - h_{min}) \leq (d_{max} - (d_{max} - (s_2 + 1)g')) + (h_{max} - h_{min}) = (s_2 + 1)g' + \Delta h$$

Similarly, using the opposite constraints

$$h_1 \geq h_{min}$$

$$d_2 \leq d_{max} - s_2 g' + \Delta h$$

we obtain the lower estimate

$$e_s = (d_{max} - d_2) + (h_1 - h_{min}) \geq (d_{max} - (d_{max} - s_2 g' + \Delta h)) + (h_{min} - h_{min}) = s_2 g' - \Delta h$$

which means that we have pinpointed e_1 to the interval

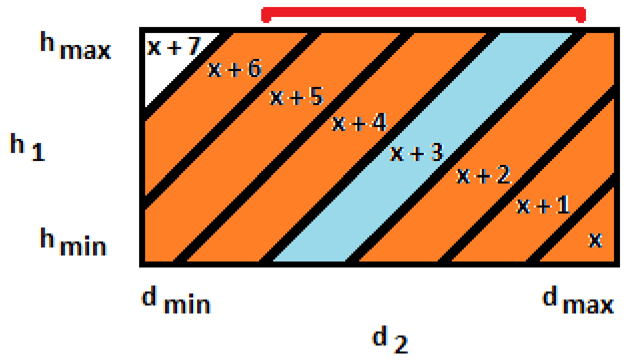


Figure 2.14: The diagram shows the value interpreted by the receiver ($y_s = x_s + s_s$) as a function of d_2 and h_1 for $k = 8$. For every value of s_2 (blue strip), there is a limited range for d_2 (red bar), and only some of the values of s_s intersect it (orange strips).

$$[s_2 g' - \Delta h, (s_2 + 1)g' + \Delta h]$$

of size

$$((s_2 + 1)g' + \Delta h) - (s_2 g' - \Delta h) = g' + 2\Delta h$$

Let us see if we can deduce something about s_s from this fact. Since e_s is at least $s_2 g' - \Delta h$, then the corresponding value of s_s would be

$$s_s \geq \left\lceil \frac{s_2 g' - \Delta h}{g'} \right\rceil = s_2 - \left\lceil \frac{\Delta_h}{g'} \right\rceil$$

On the other hand, as $e_s \leq (s_2 + 1)g' + \Delta h$, then

$$s_s \leq \left\lceil \frac{(s_2 + 1)g' + \Delta h}{g'} \right\rceil = s_2 + 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil$$

which means that

$$s_s \in \left\{ s_2 - \left\lceil \frac{\Delta_h}{g'} \right\rceil, \ldots, s_2 + 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil \right\} \cap \{0, \ldots, k - 1\}$$

as is shown in Figure 2.14. This means that there are at most

$$s_2 + 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil - \left(s_2 - \left\lceil \frac{\Delta_h}{g'} \right\rceil \right) = 1 + 2 \left\lceil \frac{\Delta_h}{g'} \right\rceil \tag{2.13}$$

possible values of s_s if we already know s_2. Let us denote this number a.

$$a = \left| \left\{ s_2 - \left\lceil \frac{\Delta_h}{g'} \right\rceil, \ldots, s_2 + 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil \right\} \right| = 1 + 2 \left\lceil \frac{\Delta_h}{g'} \right\rceil$$

Thus, the sender only needs to send the receiver $\log_2 a$ error-correcting bits instead of $\log_2 k$, which is

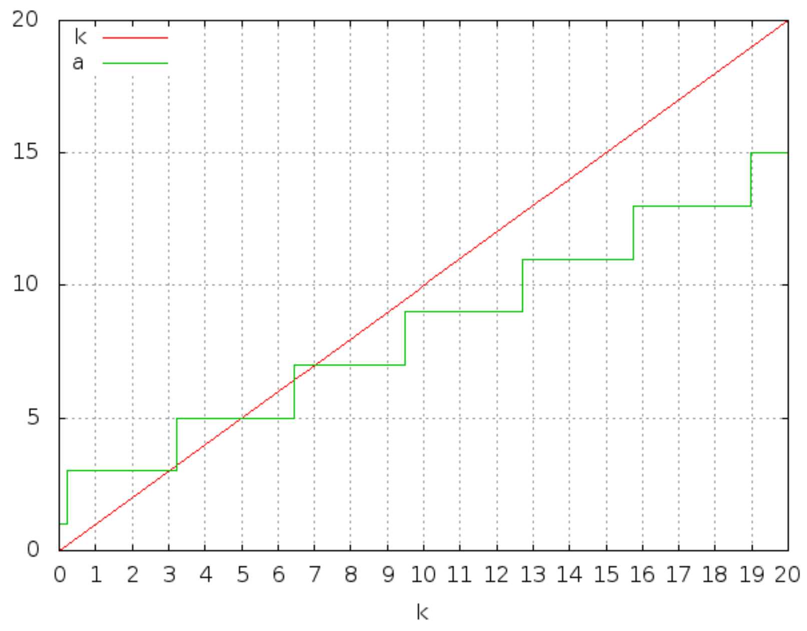


Figure 2.15: Comparison of a and k for our values of parameters.

more efficient if $a < k$. Let us see when this holds. By removing the ceiling function from a, we get its approximation a^* which is exact whenever the ceiling argument is integer.

$$a \approx a^* = 1 + 2\frac{\Delta_h}{g'}$$

$$a \approx a^* 1 + 2\frac{k\Delta_h}{\Delta d + \Delta h}$$

$$\frac{\partial}{\partial k} a^* = 2\frac{\Delta_h}{\Delta d + \Delta h}$$

$$\frac{\partial}{\partial k} a^* \leq 2\frac{\Delta_h}{2\Delta h}$$

$$\frac{\partial}{\partial k} a^* \leq 1$$

which means that a, as a function of k, increases at most as fast as k. Evidently, if $\Delta d > \Delta h$ we get a strong inequality, so a increases slower. It follows that there is a value k_0 such that for any $k > k_0$ we have $a < k$. As can be seen in Figure 2.15, for our values of parameters, $a = k$ holds if $k = 3, 5, 7$ and $a < k$ holds for $k = 6 \vee k \geq 8$.

The question that stands is how exactly does the sender encode the error-correcting information into $\log_2 a$ bits. We will show that $x_s \; mod \; a \in \{0, \ldots, a-1\}$ suffices. The receiver knows the values of x_2, y_2, y_s and is supposed to determine x_s.

$$x_s = y_s + s_s$$

$$x_s = y_s + s_2 + z, z \in \left\{ -\left\lceil \frac{\Delta_h}{g'} \right\rceil, \dots, 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil \right\}$$

$$x_s = y_s + (y_2 - q_2) + z, z \in \left\{ -\left\lceil \frac{\Delta_h}{g'} \right\rceil, \dots, 1 + \left\lceil \frac{\Delta_h}{g'} \right\rceil \right\}$$

All variables on the right hand side are known, except for z. There is an interval of a consecutive candidate values for z, out of which each has a different value $mod\ a$, thus exactly one of them will result in $x_s\ mod\ a$ having the correct value.

Results

Finally, let us apply the Modulo speedup method with non-integer block sizes and error-correction bit correlations together. Let us use granularity $g' = \frac{g}{k}$ and corresponding block ranges x'_r, x'_s (for ringing and silent blocks, respectively) to send the bits of the message. The required time (denoted $t_{g'}$), expressed in terms of block ranges, is then obtained by substituting into formula (2.11)

$$t_{g'}(\log_2 x'_r, \log_2 x'_s) = \frac{d_{max} + h_{max}}{\log_2 x'_r + \log_2 x'_s} + \frac{g}{k} \frac{x'_r + x'_s - 2}{2(\log_2 x'_r + \log_2 x'_s)}$$

To send the error-correcting bits, we will use the original granularity g (to emphasize this, we will write t_g instead of just t) and block ranges x_r, x_s.

$$t_g(\log_2 x_r, \log_2 x_s) = \frac{d_{max} + h_{max}}{\log_2 x_r + \log_2 x_s} + g \frac{x_r + x_s - 2}{2(\log_2 x_r + \log_2 x_s)}$$

We have established in inequality (2.13) that while we need $\log_2 k$ error-correction bits for the ringing blocks, for the silent ones we only need

$$\begin{cases} \log_2 \left(1 + 2 \left\lceil \frac{\Delta_h}{g'} \right\rceil \right) & \Delta d \geq \Delta h \\ \log_2 \left(1 + 2 \left\lceil \frac{\Delta_d}{g'} \right\rceil \right) & \Delta d < \Delta h \end{cases}$$

For simplicity, we will always use the first option, as this is the case for our values of parameters. Equations for the other option can be obtained analogously. These error-correcting bits are sent for every pair of ringing and silent blocks, i.e. for every $\log_2 x'_r + \log_2 x'_s$ bits. We get that the total bit transmission time (plain bit plus corresponding error correction) is

$$t_{g'}(\log_2 x'_r, \log_2 x'_s) + \frac{\log_2 k + \log_2 \left(1 + 2 \left\lceil \frac{\Delta_h}{g'} \right\rceil \right)}{\log_2 x'_r + \log_2 x'_s} t_g(\log_2 x_r, \log_2 x_s)$$

For any selected k (g') the optimal block ranges x'_r, x'_s can be found by setting the derivative of the above function to zero. As x'_r and x'_s can be cyclically permutated in the formula, the minimum must be of the form $[x'_{opt}, x'_{opt}]$. It is therefore advisable to rewrite the transmission time as a function of a single variable x' which constitutes the block range of both types of blocks. After finding the minimum $[x'_{opt}, x'_{opt}]$, we only have to try four nearby integer solutions by replacing x'_{opt} to $\lfloor x'_{opt} \rfloor$ and $\lceil x'_{opt} \rceil$. Us-

			Transmission time per bit		
k	$\mathbf{x'_r}$	$\mathbf{x'_s}$	**Plain**	**Total**	**Total w/o correlations**
1	6	6	6.518	8.427	6.518
2	9	9	4.622	7.160	6.585
3	13	13	3.959	6.625	6.625
4	18	18	3.645	6.871	6.631
5	22	22	3.384	6.625	6.625
6	25	25	3.155	6.444	6.620
7	30	30	3.050	6.611	6.611
8	34	34	2.934	6.487	6.604
9	38	38	2.838	6.383	6.598
10	43	43	2.781	6.505	6.592

Table 2.4: Transmission time achieved by the Modulo method for $k \in \{1, \ldots, 10\}$, with and without the use of error-correction bits.

ing small granularity (large k) typically leads to plain bit transmission times small enough to avoid the ringing limit l. If we are unlucky, however, this process does not work. We would then have to find x'_r compliant with l first, by partial derivation, then with fixed x'_r we would derive x'_s. We will not state this process formally, as the steps are similar to those taken in the analysis of Ringing Length protocol and the non-sped-up version of this protocol. The computation is technical and the results themselves are too complex. In the following, we will find the optimal values of x'_s and x'_r for our parameters experimentally.

We will now explore the bit transmission time for speedup with $k \in \{1, \ldots, 10\}$. To compare the results, remember that the transmission time we achieved for the non-sped-up version was $t(2, 2) = 6.225s$. For every value of k, the table in Figure 2.4 will show the optimal values of x'_r and x'_s and

- Transmission time of a plain bit $t_{g'}(\log_2 x'_r, \log_2 x'_s)$
- Total transmission time $t_{g'}(\log_2 x'_r, \log_2 x'_s) + \frac{\log_2 k+1+2\left\lceil\frac{\Delta h}{g'}\right\rceil}{x'_r+x'_s} t_g(\log_2 x_r, \log_2 x_s)$
- Total transmission time if we did not use the error-correction bit correlations $t_{g'}(\log_2 x'_r, \log_2 x'_s) + \frac{2\log_2 k}{x'_r+x'_s} t_g(\log_2 x_r, \log_2 x_s)$

We can see in the results that $1 + 2\left\lceil\frac{\Delta h}{g'}\right\rceil \geq \log_2 k$ for some of the first values, but for large enough k it becomes advantageous (this was illustrated in 2.15). However, similarly as in the Ringing Length protocol, we did not observe speedup for our values of parameters. Just as in the case of Ringing Length protocol, we have to conclude that the speedup would be better observable if the l limit did not allow us to choose optimal block ranges x_r, x_s, as the most important consequence of using lower granularity is being able to use larger block sizes under a fixed value of l.

Chapter 3

Channel capacity

In this chapter, we are going to find an upper bound for the bitrate, or equivalently the lower bound for the transmission time. Obviously, the amount of information that can be transferred between the sender and receiver is related to the number of different sequences of events that the receiver can distinguish and the sender can induce. Let us first compute the latter, the number of possible sequences of dialing and hanging up from the sender's perspective.

Suppose that time is discrete, divided into intervals of size g and that the participants' clocks are synchronized. We will express the number of possible sequences from the sender's viewpoint in terms of two recurrences, $S(n)$ and $T(n)$ for $n > 0$. Let $S(n)$ be the number of sequences that can occur in n intervals if we start dialing in the (beginning of the) first interval. Similarly, let $T(n)$ be the number of sequences that can occur in n intervals if we hang up in the beginning of the first one. We get

$$S(n) = \sum_{i=\left\lceil \frac{d_{max}}{g} \right\rceil}^{\min\left\{\left\lfloor \frac{d_{min}+l-h_{max}}{g} \right\rfloor, n-1\right\}} T(n-i)$$

because ringing may take as much as d_{max} time, covering $\left\lceil \frac{d_{max}}{g} \right\rceil$ intervals, so we may hangup no sooner than in the $\left(\left\lceil \frac{d_{max}}{g} \right\rceil + 1\right)$-th. The ringing may only take d_{min}, in which case the ringing limit elapses as soon as in $d_{min} + l$ time. As the sender is supposed to induce the events that the receiver perceives, we do not want the receiver to hang up on their own (as it would be unclear whether that happens sooner or later than when the sender's hangup signal arrives). We therefore allow h_{max} extra time for the sender's hangup signal to reach the receiver. We also assume that the sender always has to hang up, at latest in the last interval, as the communication cannot end with ringing. Hence the upper bound of the sum. For any feasible choice of i, the number of intervals for which the sender is offhook, we can hang in the i-th interval, which brings us to the situation described by the other recurrence T, with i less intervals at our disposal.

$$T(n) = 1 + \sum_{i=\left\lceil \frac{h_{max}}{g} \right\rceil}^{n-1} S(n-i)$$

In case of T, we do not have to worry about the l limit. The only condition is that we do not dial again sooner than h_{max} time elapses (the computation of S ends where sender hangs up; therefore, we must always allow this much time for the hangup signal). Furthermore, we always add +1 for the special case that we decide to not actually dial again and remain idle. Note that we do not need initial conditions for these reccurences. For small values of n, the sum in the reccurent formula for $S(n)$ is empty, so the result is 0. This is correct, as these cases represent situations with too few intervals so there is not enough time to dial, and thus there are no valid sequences for S. Similarly, we do not need an initial condition for T, as for small n the empty sum evaluates to zero, leading to $S(n) = 1$ which is correct, as there is exactly one valid sequence - the one where no event occurs.

Since the sender may ring for the first time at any point or none at all, the total number of sequences the sender can produce is (denoted $R(n)$)

$$R(n) = 1 + \sum_{i=1}^{n} S(i)$$

It follows that the number of bits transmittable by the sender in n intervals is $\log_2 R(n)$. If we use $g = d_{max}$, then the receiver always experiences the matching event in the same interval - i.e., if the sender starts dialing, the receiver's phone will start ringing within the same interval and if the sender hangs up, the receiver's phone will also stop ringing in the same interval. This means that every sequence of events produced by the sender is exactly mirrored on the receiver's side. The receiver can distinguish exactly what sequence the sender produced, and thus they also receive the same number of $\log_2 R(n)$ bits.

Let us simplify the recurrences. First, let us define $T(n)$ for $n \leq 0$ as $T(n) = 0$. This can be done by rewriting it as

$$T(n) = [n > 0] + \sum_{i=\left\lceil \frac{h_{max}}{g} \right\rceil}^{n-1} S(n-i)$$

Note that $S(n) = 0$ for $n \leq 0$ already holds with our current definition, as the sum is then empty. Now we may notice that the expression $\min\{\ldots, n-1\}$ in the definition of $S(n)$ is only a sentinel that prevents using $T(n)$ with non-positive arguments. Since we have already defined those to be zero, we can remove this sentinel.

$$S(n) = \sum_{i=\left\lceil \frac{d_{max}}{g} \right\rceil}^{\left\lfloor \frac{d_{min}+l-h_{max}}{g} \right\rfloor} T(n-i)$$

To simplify further computations, let us define the abbreviations

$$a = \left\lceil \frac{d_{max}}{g} \right\rceil$$
$$b = \left\lfloor \frac{d_{min} + l - h_{max}}{g} \right\rfloor$$
$$c = \left\lceil \frac{h_{max}}{g} \right\rceil$$
$$z = b - a + 1$$

Let us now substitute the definition of $T(n)$ into $S(n)$

$$S(n) = \sum_{i=a}^{b} T(n-i) = \sum_{i=a}^{b} \left([n - i > 0] + \sum_{j=c}^{n-i-1} S(n-i-j) \right) =$$
$$= min\,\{b - a + 1, n - a\} + \sum_{i=a}^{b} \sum_{j=c}^{n-i-1} S(n-i-j) = min\,\{z, n-a\} + \sum_{i=n-b}^{n-a} \sum_{j=c}^{n-i-1} S(i-j) =$$
$$= min\,\{z, n-a\} + \sum_{i=n-b}^{n-a} \sum_{j=1-n}^{i-c} S(j) = min\,\{z, n-a\} + \sum_{i=n-b}^{n-a} \sum_{j=1}^{i-c} S(j)$$

Before we try to evaluate $S(n)$, we must realize that it is based on the parameter g. Naturally, the lower g we pick, the more precisely we are able to measure time and so we are able to distinguish more sequences per a fixed time unit. As we want to measure the maximum bitrate of the channel, independent on the participants' of g, we must use as low value of g as is possible. It seems that the correct choice is $g \to 0$, but there are two limitations to this. First, the impossibility of infinite precision of time measurement means that there is a technical limit of g. The second problem arises from the variability of d and h. If g is lower than Δd or Δh, it is unpredictable how many intervals will a particular dialing or hangup span. Thus, we lose the bijection between sequences of events perceived by sender and those perceived by receiver. For example, let us consider $g = \frac{1}{2}\Delta d$. When the sender starts dialing, they cannot know when exactly does the receiver's phone start ringing - it could be one of two possible intervals, depending on whether the actual realization of d is above or below $d_{min} + \frac{1}{2}\Delta d$. The same ambiguity is seen by the receiver - if a phone starts ringing in this interval, there are two possible intervals in which the sender could have started dialing. Therefore, with low values of g we are transferring data faster, but having sent an n-bit message, less than n bits of entropy are actually received on the other side due to the uncertainty. For the message to be decoded, extra error-correcting bits will have to be sent. We therefore expect the transmission time to grow again. However, we will not go on to examine and prove this, as the computations for low g are rather complicated. The reader will pardon the simplification made if we just use the lowest feasible value of g that does not induce uncertainty, $g = \max\{\Delta d, \delta h\} = 3s$ and proclaim this choice of g to be optimal. We obtain

$$a = 3 \qquad b = 20 \qquad c = 2 \qquad z = 18$$

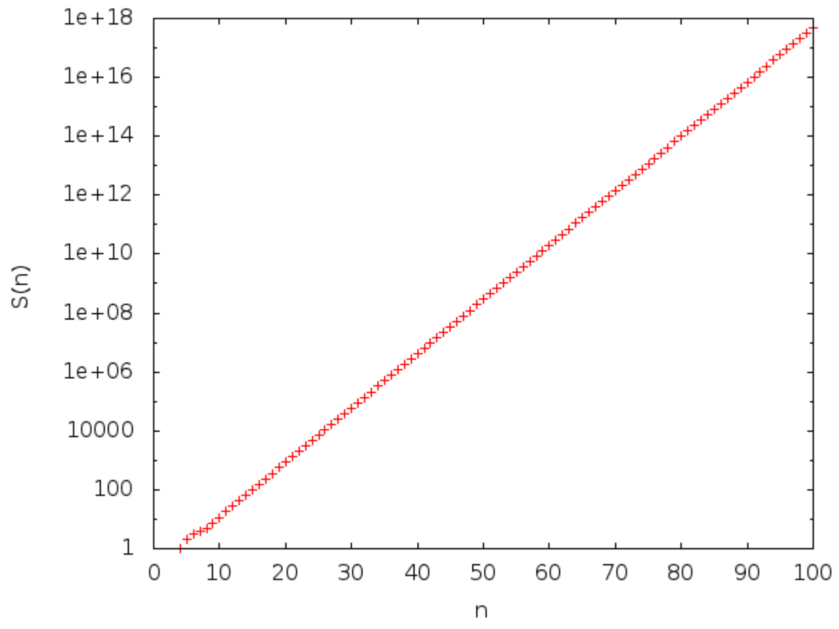


Figure 3.1: The values of $S(n)$ for our values of parameters and $g = 3$.

$$S(n) = min\left\{18, n-3\right\} + \sum_{i=n-20}^{n-3} \sum_{j=1}^{i-2} S(j)$$

In Figure 3.1 we can see $S(n)$ plotted in logarithmic scale as almost a straight line, which suggests that $S(n)$ grows exponentially. We will try to find an upper bound in the form $S(n) \leq c\alpha^n$ for some $c > 0, \alpha > 1$. We want the bound to be as tight as possible, which means that we have to minimize c and α. The condition trivially holds for $S(0)$ with any c, α. Now suppose that it holds for all $i < n$, we get

$$S(n) \leq 18 + \sum_{i=n-20}^{n-3} \sum_{j=1}^{i-2} c\alpha^j = 18 + \sum_{i=n-20}^{n-3} \sum_{j=1}^{i-2} c\frac{\alpha^{i-1} - 1}{\alpha - 1} =$$

$$= 18 + c\frac{\frac{\alpha^{n-3}-1}{\alpha-1} - \frac{\alpha^{n-21}-1}{\alpha-1} - 18}{\alpha - 1} = 18 + c\frac{\alpha^{n-3} - \alpha^{n-21}}{(\alpha-1)^2} - c\frac{18}{\alpha - 1}$$

To prove that $S(n) \leq c\alpha^n$, we have to show that

$$18 + c\frac{\alpha^{n-3} - \alpha^{n-21}}{(\alpha-1)^2} - c\frac{18}{\alpha-1} \leq c\alpha^n$$

$$\frac{18}{\alpha^n} + c\frac{\alpha^{-3} - \alpha^{-21}}{(\alpha-1)^2} - c\frac{18}{\alpha^{-n}(\alpha-1)} \leq c$$

$$\frac{18}{\alpha^n}\left(1 - \frac{c}{\alpha - 1}\right) + c\frac{\alpha^{-3} - \alpha^{-21}}{(\alpha-1)^2} \leq c$$

Let us add a constraint that $c \geq \alpha - 1$ (we are trying to minimize c, so this condition may lead us to loose, but still valid upper bound on $S(n)$). Now, if the above inequality is to hold for any value of n, zero is a tight upper estimate for the first term. We obtain

$$c\frac{\alpha^{-3}-\alpha^{-21}}{(\alpha-1)^2} \leq c$$

$$c\left(\frac{\alpha^{-3}-\alpha^{-21}}{(\alpha-1)^2}-1\right) \leq 0$$

$$\frac{\alpha^{-3}-\alpha^{-21}}{(\alpha-1)^2}-1 \leq 0$$

Using numerical methods to solve this inequality, we obtain that $\alpha \approx 1.5289$. We still do not know the value of c and we will soon see that we do not even need it. With our upper estimate of $S(n)$, we can easily find the upper estimate on $R(n)$.

$$R(n) = \sum_{i=1}^{n} S(n) = \sum_{i=1}^{n} c\alpha^n = c\frac{\alpha^{n+1}-1}{\alpha-1}$$

And thus the upper estimate of the number of bits transmittable over n intervals is

$$\log_2 R(n) \leq log_2 c\frac{\alpha^{n+1}-1}{\alpha-1} = \log_2\frac{c}{\alpha-1} + \log_2(\alpha^{n+1}-1) \leq$$
$$\leq \log_2\frac{c}{\alpha-1} + \log_2(\alpha^{n+1}) = \log_2\frac{c}{\alpha-1} + (n+1)\log_2\alpha$$

It follows that the upper estimate of the number of bits transmittable in a single interval is

$$\lim_{n\to\infty}\frac{\log_2\frac{c}{\alpha-1}+(n+1)\log_2\alpha}{n} = \log_2\alpha \approx 0.6125$$

We computed the values of $\frac{log_2 R(n)}{n}$ for $n \in \{0\ldots100\}$. In Figure 3.2, we can see that our upper estimate α is quite precise. Finally, the lower estimate for the bit transmission time can be computed as

$$\frac{g}{\log 2\alpha} = \frac{3s}{0.6125} \approx 4.8980s$$

We achieved bit transmission time of $\sim 5s$ in the Ring-or-Silence protocol, which means that this protocol uses the capacity of the channel in an almost optimal way. It is peculiar that we achieved this result using $g = 4s$, not $g = 3s$.

Note that we computed the bound in this chapter for the synchronized case. We can achieve the same result without having the participants' clocks synchronized, though not deterministically - only with high probability. However, this will require us to first introduce calibration methods, what is done in the next chapter.

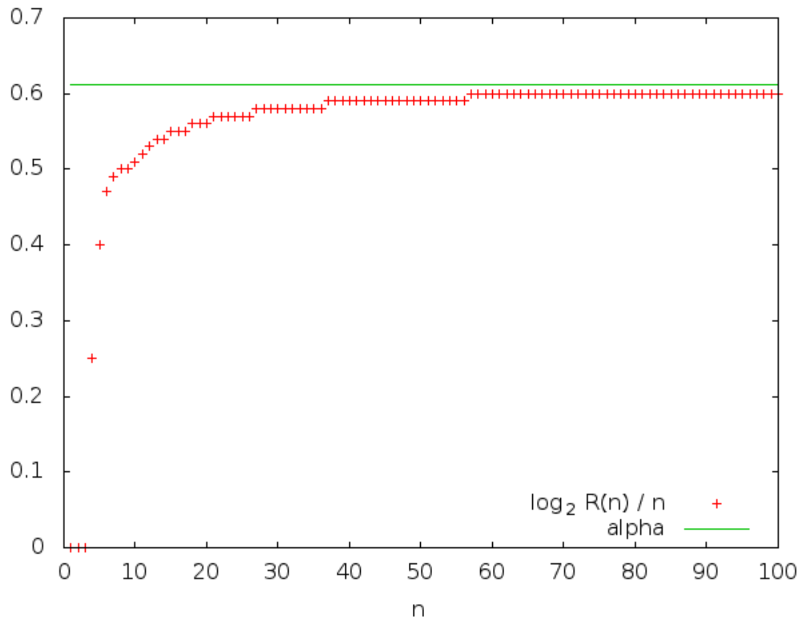


Figure 3.2: The values of $\frac{\log_2 R(n)}{n}$ for our values of parameters and $g = 3, n \in \{1 \dots 100\}$.

Chapter 4

Parameter calibration

While devising the protocols, we have always assumed that both the sender and receiver already know and have agreed upon the parameter values. We suggested that even if they have never communicated before, they may exchange this information in some kind of preamble before transmitting the message itself. However, as the parameter values vary from case to case, there must also exist a way to find them.

In this chapter, we propose methods for parameter calibration. The participants may include a calibration phase in the beginning of each communication or omit it if they are confident that the things that influenced them (e.g. cell phone devices, network carrier, distance) have not changed significantly. The calibration phase can take a considerable amount of time and in practice, it may even top the actual transmission time for short messages. However, it is still only a finite process, so its amortized time cost for message length $n \to \infty$ is zero.

We will sometimes need both participants to transmit data to each other. This is in contrast with the overall idea in our model that one of the participants only sends data and the other only receives them. We will still retain the names sender and receiver, where by sender we mean the participant that wants to send a message and initiated the calibration. We may therefore say that "receiver sends data to the sender", which means that the participant designated as receiver behaves as a sender in context of that one particular data transmission.

Before we start the callibration process, we need to have some prior upper estimation of the parameters. We will assume that

$$h_{max}, h'_{max}, d_{max}, d'_{max} < e$$

where e is some implicit (i.e. hardcoded in the application) value and d', h' represent the d, h parameters in the opposite direction (when the roles are reversed and the receiver is calling the sender).

Calibrating l

The calibration phase starts by agreeing on the value of l. This value is set in the receiver's mobile phone. The receiver should set l to the maximum possible value allowed by their device. Then, they must inform the sender of their choice. As the parameter l is typically settable as a multiple of $5s$, it suffices to inform the sender of a single natural number $\frac{l}{5s}$. This value can be sent using the Calibration data transfer protocol.

Calibration data transfer protocol (CDTP)

The Calibration data transfer protocol (CDTP) is a protocol used by the participants during the calibration phase, when no other protocols are utilizable due to the lack of knowledge of parameter values. CDTP is a special case of the One-or-Two protocol with parameter values

$$\begin{aligned} d_{max} &= e \quad d_{min} = 0 \\ h_{max} &= e \quad h_{min} = 0 \end{aligned}$$

We overestimated d_{max} and h_{max} by setting them to e, which by our assumption is guaranteedly larger than both of them. Similarly, we underestimated d_{min} and h_{min} by setting them to 0, which is guaranteedly less than both them, as dialing and hanging up must take at least some time. Note well that all our protocols still work if we overestimate the ranges d and h and slightly underestimate l. However, we know no lower estimate of l other than $l = 0s$, and that would render all protocols except the One-or-Two protocol inoperable. This is precisely why it is the One-or-Two protocol that was chosen as the basis of CDTP.

Calibration of d and h

The participants will dial and hang up multiple times to measure the values of d and h. First, they must agree on a number of measurements they will make. This number may be either implicit or agreed via the CDTP. Let us denote this number k. The sender will dial the receiver. As soon as the reciever's phone starts ringing, the receiver hangs up. When the hangup signal reaches the receiver, the sender will dial again. The whole process is repeated k times. Let us use the notation s_i for the timestamp when the sender starts dialing and/or receives the hangup signal for the i-th time and r_i for the timestamp when the receiver's phone starts ringing for the i-th time[1]. The timestamps s_i are measured by the sender's clock and r_i by the receiver's one, so it does not necessarily hold that $s_i < r_i < s_{i+1}$ for any $i \in \{1 \dots k\}$. The receiver uses CDTP to send all the values r_i to the sender.

If the participants' clocks are synchronized, the variables $r_i - s_i$ and $s_{i+1} - r_i$ are obviously realizations of d and h'. If they are not synchronized, let δ represent the difference between sender's and receiver's clocks (i.e. the receiver's clock is δ too early). Clearly, $r_i - s_i$ and $s_{i+1} - r_i$ are then instead realizations of random variables $d - \delta$ and $h' + \delta$, respectively. We will use the minimum and maximum measurements

[1]This means that there are $k + 1$ measurements of s_i, while only k of r_i.

of $r_i - s_i$ to establish the lower and upper bounds of $d - \delta$, i.e. $d_{min} - \delta$ and $d_{max} - \delta$ (and analogously for h'). When we find the value of δ, this will allow use to derive d_{min} and d_{max}. However, since we defined d_{min} and d_{max} as non-inclusive bounds (i.e. $d \in (d_{min}, d_{max})$), there is zero probability of a random sample having such value. Even if we did not, there is no guarantee that our k measurements of $d - \delta$ will also contain the actual extreme values $d_{min} - \delta$ and $d_{max} - \delta$. Therefore, we must expect that $\min\{r_i - s_i\} > d_{min} - \delta$ and $\max\{r_i - s_i\} < d_{max} - \delta$. In other words, our maximum is probably underestimated and the minimum overestimated. A solution to this is to add two more r_i, s_i pairs, one of which will create new maximum and the other to create a new minimum. They will serve as sentinels to stretch our range of values of $d - \delta$ and possibly include values from $(d_{min} - \delta, \min\{r_i, s_i\}$ and $(max\{r_i, s_i\}, d_{max} - \delta)$ that we did not have the luck to draw during our random sampling of $d - \delta$. These sentinels may be established by a simple guess - e.g. to stretch the interval to double its size, we will add new measurements r_i, s_i for $i = k+1, k+2$ such that

$$r_{k+1} - sk + 1 = \frac{3}{2} max\{r_i, s_i \mid i \leq k\} - \frac{1}{2} min\{r_i, s_i \mid i \leq k\}$$

$$r_{k+2} - sk + 2 = \frac{3}{2} min\{r_i, s_i \mid i \leq k\} - \frac{1}{2} max\{r_i, s_i \mid i \leq k\}$$

If we have some a priori knowledge about the distribution of d, we can do better. For example, suppose that we know that d is uniform on interval (d_{min}, d_{max}), then $d - \delta$ must also be uniform on $(d_{min} - \delta, d_{max} - \delta)$. Now if we did k measurements, there is $> 50\%$ probability that our measurements did not span the entire interval, but only an inner part of it - say $x\%$, while $(100 - \frac{x}{2})\%$ is left on both sides, i.e.

$$(d_{min} - \delta, d_{min} - \delta + (100 - \frac{x}{2})\%\Delta d) \cup (d_{max} - \delta - (100 - \frac{x}{2})\%\Delta d, d_{max} - \delta)$$

from where we sampled no value. A fitting value of x for this statement can be computed as

$$(x\%)^k > 50\%$$

$$(x\%) > \sqrt[k]{\frac{1}{2}}$$

So to reiterate, there is a $> 50\%$ (rather high) probability that all measurements of d were from the inner $\sqrt[k]{\frac{1}{2}}$ part of the interval (d_{min}, d_{max}). To cope with this, perhaps we should choose the $(k+1)$-st and $(k+2)$-nd measurement so that the maximum and minimum are moved by $(100 - \frac{x}{2})\%$ outwards. A similar reasoning can be done for other distributions of d and the whole process should be repeated analogously for h as well. Of course, other threshold probability than 50% can be chosen for safer results.

Let us now use the measurements to determine the ranges of d and h. To do this, we must find the value of δ. In general, the sender's and receiver's clocks may not be synchronized and the value δ may be unknown. Notice that even if the receiver attempted to inform the sender of their current time via CDTP, the sender would still not know how long did it take for this message to travel to them. The sender cannot estimate this with higher precision than is allowed by $\Delta d, \Delta d', \Delta h, \Delta h'$ - the inherent imprecision brought by dialing and ringing as the basic blocks of any communication. As a side note, even if we leave our model and look into actual Android applications, the possibilities are not great. We could make the participants synchronize over the internet, but no third-party application can obtain permission to set the system time (unless the phone is rooted)[spe14]. Furthermore, the users themselves cannot set time manually with precision higher than minutes, which is unacceptably coarse-grained for our purposes. It is therefore left to sender to guess the value of δ. Let us use the notation d_{max} and d_{min} to represent the upper and lower estimates of δ. In the synchronized case, we simply have $\delta_{min} = \delta_{max} = 0$; we will later show how to obtain these estimates in the general case.

We can now estimate the parameters d and h' as follows:

$$d_{max} = \max\{r_i - s_i + \delta\} = \max\{r_i - s_i\} + \delta_{max}$$

$$d_{min} = \min\{r_i - s_i + \delta\} = \min\{r_i - s_i\} + \delta_{min}$$

$$h'_{max} = \max\{s_{i+1} - r_i - \delta\} = \max\{s_{i+1} - r_i\} - \delta_{min}$$

$$d'_{min} = \min\{s_{i+1} - r_i - \delta\} = \min\{s_{i+1} - r_i\} - \delta_{max}$$

Although it may not always hold that $r_i \in [s_i, s_{i+1}]$ if the clocks are too off, objectively the receiver's measurement of r_i took place between the sender's measurements of s_i and s_{i+1}, so it must hold that $r_i + \delta \in [s_i, s_{i+1}]$ for any i. It follows that

$$\forall i : s_i \leq r_i + \delta \leq s_{i+1}$$

$$\forall i : s_i - r_i \leq \delta \leq s_{i+1} - r_i$$

$$\max\{s_i - r_i\} \leq \delta \leq \min\{s_{i+1} - r_i\}$$

$$\delta_{min} = \max\{s_i - r_i\} \wedge \delta_{max} = \min\{s_{i+1} - r_i\}$$

$$\delta_{min} = -\min\{r_i - s_i\} \wedge \delta_{max} = \min\{s_{i+1} - r_i\} \tag{4.1}$$

We narrowed δ down into an interval described by the minima of our realizations of random variables $d - \delta$ and $h' + \delta$. We can substitute the obtained values of δ_{min} and δ_{max} into the above formulae for d and h'. We receive valid estimates of d and h', although if the interval $[\delta_{min}, \delta_{max}]$ is too large, they are still very coarse.

If we assume that the network communication is symmetric so that dialing and hangup takes rougly the same time in both directions, we can get a better estimate of δ. That is, with high probability we can find δ with high precision. If we make a large amount of measurements of r_i, s_i, by the Law of large numbers[Ré68]

$$\frac{1}{n}\sum_{i=1}^{k}(r_i - s_i) \to E[d - \delta] = d_{avg} - \delta \tag{4.2}$$

Now let us reverse the roles and make the receiver call the sender and make similar measurements. Since the sender's clock is δ later than the receiver's one, these measurements will be realizations of the random variable $d'_{avg} + \delta$. The participants can then use CDTP to exchange these values and compute their difference

$$(d'_{avg} + \delta) - (d_{avg} - \delta) = (d'_{avg} - d_{avg}) + 2\delta \tag{4.3}$$

By our assumption of symmetricity, it follows that $d_{avg} \approx d'_{avg}$ and the value we obtained is actually an approximation of 2δ. Dividing it by 2, we obtain δ. An analogous process can be done with h and h'. This means that we actually only need one of the parameters to be symmetric.

Note that equation (4.2) can be also used in the opposite way. We can use our estimates of δ (equations (4.1)) to approximate d_{avg} (and by taking $s_{i+1} - r_i$ measurements instead, h_{avg}). If the assumption about symmetricity holds, we can replace subtraction by addition in equation (4.3) to obtain

$$(d'_{avg} + \delta) + (d_{avg} - \delta) = d'_{avg} + d_{avg}$$

and consequently get an approximate value of d_{avg} by dividing by two.

It seems that we cannot do better than this. We first used the knowledge that d, h' must have non-negative duration to find coarse estimates. Then, we used the assumption of symmetricity to compare the communication times in opposite direction. However, without assumptions like this, there is no clue that would lead to one value of δ over another. For the participants, the situation with parameters d, h, d', h', δ is indistinguishable from any situation $d - \xi, h + \xi, d' + \xi, h' - \xi, \delta + \xi$ for $\xi \in \mathbb{R}$.

Relation to synchronization

We ended the previous chapter on Channel capacity with a remark that our calibration methods can be used for synchronization. To synchronize clocks of the participants means the same as to find the value δ. However, in the context of the previous chapter we assumed that the participants already know the values of parameters. This is the crucial difference - we were unsuccessful in finding a good estimate of δ in the calibration phase, because we had no knowledge of d, h, d', h'. If the participants have this knowledge, it is an easy task.

All that is necessary is to subtract formula (4.2) from d_{avg}, which we assume to already know, to obtain the value of δ. In fact, we only obtain a high-probability high-precision approximation thereof, since the value in formula (4.2) is computed through the Law of large numbers. But this is not a problem, because the only protocol that strictly required synchronization, the Ring-or-Silence protocol, can take advantage of any approximation. If we know δ with a possible deviation ε, then the checking time window caused by the clock misalignment (described in subsection 2.3.3) only needs to be of size ε. By increasing the number of measurements we get ever better approximations, i.e. $\varepsilon \to 0$, so the time window can become arbitrarily small and we can get arbitrarily close to the synchronized-case bitrate of the protocol.

Chapter 5

Implementation

In this chapter, we are going to present our implementation of the described protocols. We created a mobile application for the Android operating system, which can be downloaded from

`https://sourceforge.net/projects/ringingmessenger/`

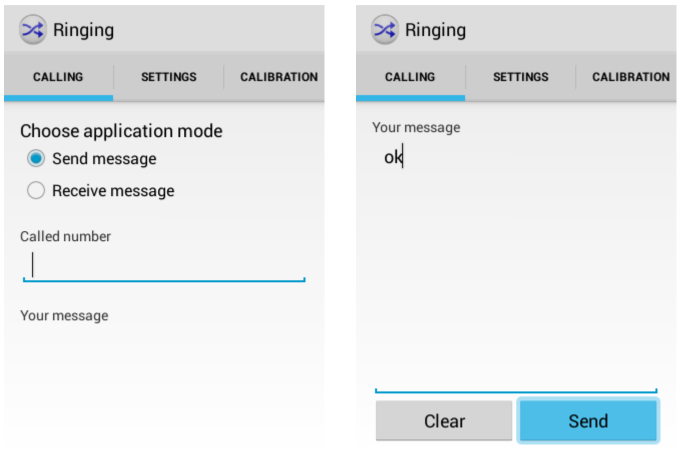
The application is licensed under the Simplified BSD License and its source code is freely available at the same site. The target system version is Android 2.1 (API level 7) and higher.

We implemented the protocols in the following manner.

- One-or-Two protocol with the Receiver Hangup method.
- Ringing Length protocol without speedup. The block size is computed automatically and the message is padded with 0 bits to its nearest multiple.
- Ring-or-Silence protocol is implemented without speedup. The communication always starts with one ringing for synchronization. Either the dialing or the hangup part of the ringing is used to synchronize, depending on which of Δd and Δh is smaller.
- Alternating Ringing protocol without speedup. The block size is computed automatically and the message is padded with 0 bits to its nearest multiple. An extra block of 0 bits is added if the message does not end with a ringing block.

As an input method, we allow three encodings.

- **Plain bits.** The message is typed as a sequence of 0 and 1 characters.
- **Alphanumeric encoding.** We developed this encoding to provide alphabet and numbers in as few bits as is necessary (which is 6 bits, since there are 26 letters and 10 numerals, $2^5 < 36 \leq 2^6$)



Ringing

CALLING | SETTINGS | CALIBRATION

Choose application mode

[✓ Send message]

Receive message

Called number

Your message

Ringing

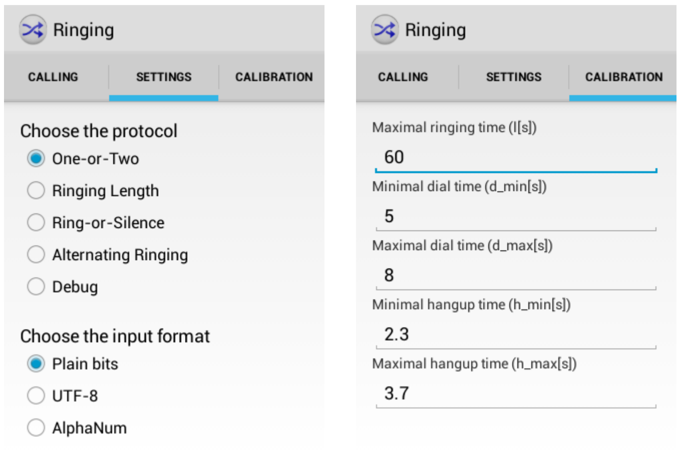
CALLING | SETTINGS | CALIBRATION

Your message

ok

Clear | Send

Figure 5.1: The Calling tab.



Ringing

CALLING | SETTINGS | CALIBRATION

Choose the protocol

[✓ One-or-Two]

Ringing Length

Ring-or-Silence

Alternating Ringing

Debug

Choose the input format

[✓ Plain bits]

UTF-8

AlphaNum

Ringing

CALLING | SETTINGS | CALIBRATION

Maximal ringing time (l[s]): 60

Minimal dial time (d_min[s]): 5

Maximal dial time (d_max[s]): 8

Minimal hangup time (h_min[s]): 2.3

Maximal hangup time (h_max[s]): 3.7

Figure 5.2: The Settings and Callibration tabs.

to avoid the necessity of using traditional 8 bits per character. We used the remaining space given by 6 bits for lowercase letters (resulting in total 62 characters) and the last two characters were used for space (" ") and for period (".") to enable basic interpunction. The exact mapping may be arbitrary, but a reasonable choice is that the 6-bit value of 0 should represent the space. This is because the Ring-or-Silence protocol has no explicit end, which means that after the sender has finished transmitting, the receiver will keep interpreting the silence as 0 bits. It is therefore practical to make the application output whitespace instead of any particular visible character.

- **UTF-8.**

The interface is divided into three tabs:

- **Calling.** The user can switch between the sending and receiving mode. In the former, the user can

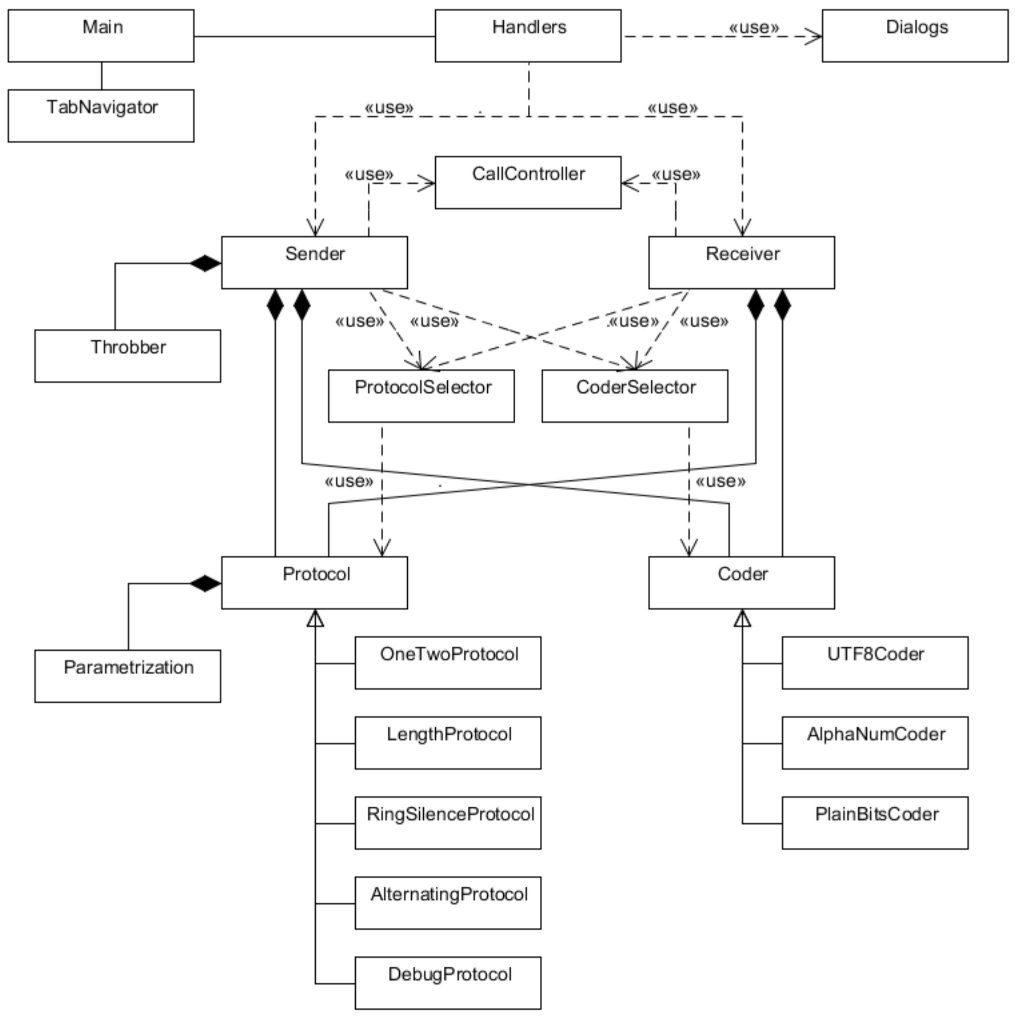


Figure 5.3: The UML class diagram of our application.

type a message and send it. In the latter, the application awaits any incoming communication.

- **Settings.** The user can choose between five protocols - the four listed above and the "Debug protocol" which outputs time measurements of dialing and hangup events for both participants.
- **Calibration.** This tab allows the user to set the values of $d_{min}, d_{max}, h_{min}, h_{max}$ and l. An automated calibration is not provided in the current version of the application.

The UML class diagram of the application can be seen in Figure 5.3. A brief description of the classes follows.

- ***CallControler*** Listens to the telephony events and calls back *Sender* or *Receiver*, depending on which one is active.
- ***Coder*** An abstract class, descendants of which are classes for the encoding methods. Used by the *Sender* and *Receiver* classes to encode input and decode received transmission.
- ***CoderSelector*** A helper class for selecting *Coder* and keeping the selection.
- ***Dialogs*** Encapsulates info and modal error dialogs displayed by the application.
- ***Handlers*** Provides event handlers as a connection between the user interface and program logic.
- ***Main*** The main class which provides the entry point to the program. Initializes the UI classes *Handlers* and *TabNavigator*.
- ***Parametrization*** Stores the parameter values specified by the user and provides them to *Protocol*.
- ***Protocol*** An abstract class, descendants of which are classes representing the protocols. The descendants define the callback function for telephony events caught by *CallControler*, to which it is connected by *Sender* or *Receiver*. The class itself encapsulates the dialing and hangup functionality.
- ***ProtocolSelector*** A helper class for selecting *Protocol* and keeping the selection.
- ***Main*** The main class which provides the entry point to the program. Initializes the UI classes *Handlers* and *TabNavigator*.
- ***Receiver*** Activated if the user selects the receiving mode. Listens to incoming transmissions, uses *Protocol* to interpret them as bits, *Coder* to decode those bits into the message, which it outputs.
- ***Sender*** Used if the user selects the sending mode. Uses *Coder* to encode the message and *Protocol* to send it.
- ***TabNavigator*** Provides the tab navigation functionality.
- ***Throbber*** Encapsulates the throbber that appears while sending is in progress.

Conclusion

We presented several protocols for transferring data through ringing a cellphone.

1. The One-or-Two protocol is slow, but very easy to implement. Its advantage over the faster protocols is that it does not depend on precise timing and that it does not require the knowledge of ringing time limit, what made it very useful for the calibration phase.
2. The Ringing Length protocol makes better use of long dialing times, which makes it significantly faster than One-or-Two. It served primarily as a template for the more advanced Alternating Ringing protocol, which is based upon it. Unlike its faster cousin, Ringing Length protocol does not require continual measurement of time, as the timing is only ever computed from the beginning of the block and thus can be reset between them. This property would make this protocol desirable if we only had unreliable clock at our disposal.
3. The Ring-or-Silence protocol is even faster than its predecessors, but requires participants to have precisely synchronized clocks.
4. The Alternating Ringing protocol is an improvement of the Ringing Length protocol.

For our measured values of parameters, We achieved the following bit transmission times.

One-or-Two	$16.45s$
Ringing length	$9.0\overline{3}s$
Ring-or-Silence (with synchronization)	$\sim 5s$
Ring-or-Silence (without synchronization)	$\sim 6.375s$
Alternating Ringing	$6.225s$

Relying on the assumption that the participants' clocks are synchronized, the Ring-or-Silence protocol achieves the best bit transmission time with $\sim 5s$. In the unsynchronized case, Ring-or-Silence becomes less efficient and is outperformed by the fastest Alternating Ringing protocol with $6.225s$.

We estimated the maximum possible bit transmission time in the ringing communication channel to be $\sim 4.898s$. The Ring-or-Silence protocol almost achieves this value, which means that our goal to design efficient protocols was a success.

To provide a proof of concept and show that our protocols are viable in practice, we implemented them in an Android text messenger application. This application has been already tested multiple times and thus successfully demonstrated that short text messages can indeed be transferred by ringing between real cellphones. The application is released as open source to be used or built upon by everyone.

There are other opportunities for the usage of cellphones' side channels that were not explored in this thesis, as they are beyond our simple sender-receiver model. One that comes into mind would be using two senders (or two-SIM-card phones) to encode extra information by switching between the caller's two telephone numbers. Another would be to take advantage of the "missed call" messages sent to the participants automatically by operator if the line was busy. More advanced protocols may be designed with multiple participants in mind. This creates room for future improvement of our work.

Bibliography

[CGH$^+$96] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[Gra53] F. Gray. Pulse code communications. *U.S. Patent 2632058*, 1953.

[MVM09] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Hamming Code: Parity Bit, Two- Out- Of- Five Code, Hamming(7,4), Reed-Muller Code, Reed-Solomon Error Correction, Turbo Code, Low- Density Parity- Check Code, Telecommunication, Linear Code*. Alpha Press, 2009.

[pho14] Android developers reference, TelephonyManager, 2014.

[Ré68] Pál Révész. The laws of large numbers. Probability and Mathematical Statistics. A Series of Monographs and Textbooks. 4. New York-London: Academic Press. 176 p. (1968)., 1968.

[spe14] Android developers reference, Manifest.permission, 2014.

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, June 1987.