Chapter 7

Energy-Efficient Communication

7.1 Introduction

Most conventional communication strategies utilize energy based transmission (EbT) schemes, which require energy expenditure for transmitting both 0 and 1 bit values. Depending on the transmission strategy, either both 0 and 1 are represented by non-zero voltage levels, or one of the bit values is represented by a zero voltage level while a non-zero voltage level is used to distinguish the other bit value. An example of the latter is the polar return-to-zero (polar-RZ) transmission scheme, where a 0 corresponds to a zero voltage level, while a 1 is represented by a non-zero voltage level. However, most existing transmission schemes utilize non-zero voltage levels for both 0 and 1 so as to distinguish between a silent and a busy channel. Communication strategies that require energy expenditure for transmitting both 0 and 1 bit values are known as energy based transmission (EbT) schemes. For example, in order to communicate a value of 278, a node will transmit the bit sequence \(<1, 0, 0, 0, 1, 0, 1, 0, 0>\), consuming energy for every bit it transmits. Thus, if the energy required per bit transmitted is \(e_b\), the total energy consumed to transmit the value 278 would be \(9e_b\).

In this chapter, we propose a communication technique that first recodes a binary coded data
using a redundant radix based number representation [128] and then uses silent periods to communicate the bit value of ‘0’. We show that by using the redundant binary number system (RBNS) that utilizes the digits from the set \{-1, 0, 1\} to represent a number with radix 2, we can significantly reduce the number of non-zero digits that need to be transmitted, without reducing the data transmission rate. The transmission time remains linear in the number of bits used for data representation, as in the binary number system. Considering an \(n\)-bit data representation, we have derived the average case reduction in the number of non-zero digits to be transmitted using the redundant binary radix, assuming that each of the \(2^n\) binary strings are equally likely to occur. The fraction of energy savings obtained on an average, by recoding the binary string using RBNS and using our proposed transmission protocol \(RBNSiZeComm\), is 

\[
1 - \frac{(n + 2)2^{n-2} - 12}{n \cdot 2^n}.
\]

To the best of our knowledge, there does not exist any benchmark suite to evaluate the savings in energy generated by a communication protocol for real-life data communication scenarios. For this, we have developed a benchmark test suite called the energy efficient communication (EEC) benchmark suite, in order to judge the performance of the energy efficient communication strategies. Experimental results show that our proposed transmission protocol achieves on an average, an increase of nearly 69\% in energy savings, when compared to existing energy based transmission schemes, without any degradation in the network throughput.

The rest of the chapter is organized as follows. The system model is presented in section 7.2. In section 7.3, we discuss some preliminaries about RBNS and the basic idea behind the proposed low energy communication scheme. Section 7.4 presents the proposed low energy communication protocol. Derivation of the average case energy savings obtained using the redundant binary number (RBN) encoding has been discussed in section 7.5. Experimental results are presented in section 7.6, followed by conclusion in section 7.7.
7.2 System Model

We assume a single channel wireless ad hoc network. A node $j$ is said to be a neighbor of a node $i$, if it is within the transmission range of $i$. A transmission by a node $i$ can be heard by all its neighbors. We assume that for every transmission of data between a pair of nodes, the receiver and the transmitter are synchronized in time during the period of the transmission. A possible way of achieving this would be to synchronize on the packet headers, if we assume that only the data part of a transmitted packet is in RBN format, while all the packet headers and trailers are in binary format.

7.3 Preliminaries and Basic Idea

The redundant binary number system (RBNS) [128] utilizes the digits from the set \{-1, 0, 1\} for representing numbers using radix 2. In the rest of the chapter, for convenience, we denote the digit '-1' by $\bar{1}$. In RBNS, there can be more than one possible representation of a given number. For example, the number 7 can be represented as either 111 or 100$\bar{1}$ in RBNS. In this chapter, we utilize this property of RBNS to recode a message string so as to reduce the number of 1’s in the string while transmitting the message. The original binary message can, however, be obtained at the receiver end by reconverting the received message from RBN to binary number system [128].

The basic idea of our recoding scheme is as follows: Consider a run of $k$ 1’s, $k > 1$. Let $i$ be the bit position for the first 1 in this run, $i \geq 0$ (bit position 0 refers to the least significant bit at the rightmost end). Let $v$ represent the value of this run of $k$ 1’s. Then,

$$v = 2^i + 2^{i+1} + 2^{i+2} + \ldots + 2^{k+i-1}$$ \hspace{1cm} (7.1)

Alternatively, we can rewrite equation 7.1 as,
Equation 7.2 can be represented in RBNS by a ‘1’ at bit position $(k + i)$ and a $\bar{1}$ at bit position $i$, while all the intermediate 1’s between them are converted to 0’s. We formalize this observation in the form of the following reduction rule:

**Reduction Rule 7.1** A run of $k$ 1’s ($k > 1$) starting from bit position $i$, is replaced by an equivalent representation consisting of a ‘1’ at bit position $k + i$ and a $\bar{1}$ at bit position $i$, with 0’s in all intermediate bit positions.

Observe that for a run of $k$ 1’s, $k > 1$, the savings in terms of the number of non-zero digits is $k - 2$. Note that the number of non-zero digits remain unchanged for $k = 2$. Also, if we consider a string, say 110111, with only one ‘0’ trapped between runs of 1’s, then after applying reduction rule 7.1, we would get the string 10$\bar{1}$100$\bar{1}$. Note the presence of the pattern $\bar{1}1$ for this trapped ‘0’. A second reduction can now be applied to the bit pattern $\bar{1}1$ so as to replace it by $0\bar{1}$. Thus, from 10$\bar{1}$100$\bar{1}$, we would get the string $100\bar{1}00\bar{1}$ with further reduced number of non-zero digits. We thus have a second reduction rule:

**Reduction Rule 7.2** Every occurrence of the bit pattern $\bar{1}1$ in a string obtained after applying reduction rule 7.1, is replaced by the equivalent bit pattern $0\bar{1}$.

Most existing communication strategies are *Energy based Transmissions* (EbT). Essentially, the communication of any information between two nodes involves the expenditure of energy for the transmission of data bits. Depending on the transmission strategy, either both 0 and 1 are represented by non-zero voltage levels, or one of the bit values is represented by a zero voltage level while a non-zero voltage level is used to distinguish the other bit value. An example of the latter is the polar return-to-zero (polar-RZ) transmission scheme, where a 0 corresponds to a zero voltage level (or no transmission), while a 1 is represented by a
non-zero voltage level. By keeping the transmitter silent for 0 bit-values, we can reduce the power consumption of the transmitter. Combining this silent-zero transmission scheme with our RBNS-based recoding strategy, a significant reduction in the energy expenditure can be achieved, without compromising on the throughput.

We present our protocol for such a low-energy communication strategy in the next section.

### 7.4 Low Energy Communication Protocol

Our proposed low energy transmission strategy involves the execution of the following two steps:

1. Recode the binary data frame in RBNS using reduction rules 7.1 and 7.2.

2. Send the RBNS data frame, transmitting a non-zero voltage level only for the $1$ and $\bar{1}$ bits, while remaining silent for the 0-bits.

In order to distinguish between a channel with no signal and a zero bit inside an ongoing transmission, a start of transmission signal is sent at the beginning of transmission of the data and another end of transmission signal at the end of the data frame.

We present a two-pass algorithm that applies reduction rules 7.1 and 7.2 in two steps. However, with just a slight modification of this algorithm, we can achieve the same result in a single pass as stated below.

To convert protocol $RBNSiZeComm$ to a single pass routine, we replace procedures $ReductionRule1$ and $ReductionRule2$ by a single procedure and in procedure $ReductionRule1$, wherever we write a $\bar{1}$ to the output buffer (lines 13 and 27), we replace it by the following:

```plaintext
if (lowbit ≠ 0) and (out_buf[lowbit − 1] = 1) then
```

114
Algorithm 1 Protocol RBNSiZEComm

1: procedure RBNSiZEComm(IN bit vector data_frame, IN integer len) ∆ size of len + 1 bits;
2:     /* The data frame is converted to RBNS */
3:     /* by applying reduction rules 7.1 and 7.2 */
4:     bit vector out_buf : array [0..len]
5:     boolean runflag : false;
6:     integer runlengthcount, lowbit : 0;
7:     Call ReductionRule1(data_frame, len, out_buf); ∆ Reduction Rule 7.1
8:     Call ReductionRule2(out_buf, len); ∆ Reduction Rule 7.2
9:     for i = 0 to len do
10:         if out_buf[i] ≠ 0 then
11:             transmit bit out_buf[i]; ∆ Either 1 or ̅1
12:         else
13:             remain silent; ∆ No transmission of bit value ’0’
14:         end if
15:     end for
16: end procedure
procedure REDUCTIONRULE1(IN data_frame, IN len, OUT out_buf)
for i = 0 to len − 1 do  // Test each bit of data frame
    if data_frame[i] = 1 then
        if runlengthcount = 0 then
            lowbit ← i;  // Start index of run of 1’s
        end if
        runlengthcount ← runlengthcount + 1;  // Run length
        if runlengthcount > 1 then
            runflag ← true;
        end if
    end if
    else  // data_frame[i] = 0
        if runlengthcount > 1 then  // Apply Reduction Rule 7.1
            out_buf[lowbit] ← 1;
            for j = lowbit + 1 to i − 1 do
                out_buf[j] ← 0;
            end for
            runflag ← false;
        else
            out_buf[lowbit] ← 1;
            out_buf[i] ← 0;
        end if
        runlengthcount ← 0;  // Reset run length count
    end if
end for
if runflag = true then  // run of 1 ends at msb
    out_buf[lowbit] ← 1;
    for j = lowbit + 1 to len − 1 do
        out_buf[j] ← 0;
    end for
    out_buf[len] ← 1;  // out_buf of length len + 1
    runflag ← false;
end if
end procedure
```plaintext
1: procedure REDUCTIONRULE2(INOUT out_buf, IN len)
2:     for i = 1 to len do
3:         if (out_buf[i] = 1) and (out_buf[i - 1] = 1) then  ▷ Reduction Rule 7.2
4:             out_buf[i - 1] ← 1;
5:             out_buf[i] ← 0;
6:         end if
7:     end for
8: end procedure

out_buf[lowbit - 1] ← 1;
out_buf[lowbit] ← 0;
else
    out_buf[lowbit] ← 1;
end if
```

On the receiver side, the reverse process has to be done in order to correctly receive the data and then convert it into its binary equivalent representation for further processing by the different layers of the network stack. We note that the application of the reduction rules 7.1 and 7.2 on the RBN coded data during transmission ensures that there are no occurrences of the bit patterns $1\bar{1}$ and $\bar{1}1$ in the transmitted data. Also, if the original data was an $n$-bit binary data frame, RBN encoding can result in a frame of size $n + 1$ RBN bits. Protocol $RBNSizeRecv$ outlines the steps involved in correctly receiving and reconverting an RBNS encoded data frame to its binary equivalent.

### 7.5 Analysis of the Energy Savings

We first compute the average number of non-zero digits after applying reduction rule 1 on a binary string of length $n$. For this, let us first find out the total number of occurrences of runs of 1’s of length $k$, $1 \leq k \leq n$ in all possible binary strings of 0’s and 1’s of length $n$. 

Algorithm 2 Protocol RBNSiZeRecv

1: \textbf{procedure} RBNSiZeRecv(IN RBN vector integer \textit{len}, OUT bit vector \textit{out}_\textit{buf})
2: /* First receive an RBNS data frame of length \textit{len} + 1 */
3: /* Then converted it to its equivalent binary value */

4: \textbf{RBN vector} \textit{recv}_\textit{buf} : array [0..\textit{len}]; \quad \triangleright \text{received RBNS data frame}
5: \textbf{bit vector} \textit{out}_\textit{buf} : array [0..\textit{len}]; \quad \triangleright \text{Output binary frame}

6: /* Receive the data frame */
7: \textbf{Call} ReceiveRBNDataFrame(\textit{recv}_\textit{buf}, \textit{len});
8: \textbf{Call} ConvertRBNtoBinary(\textit{recv}_\textit{buf}, \textit{len}, \textit{out}_\textit{buf});
9: 
10: /* The binary data in \textit{out}_\textit{buf} is \textit{len} + 1 bits. The msb */
11: /* can be ignored as it will always be 0, and only the */
12: /* lower order \textit{len} bits needs to be returned */
13: \textbf{end procedure}

1: \textbf{procedure} RECEIVERBNDataFrame(INOUT \textit{recv}_\textit{buf}, IN \textit{len})
2: /* \textit{null} indicated no signal in channel */
3: /* time\_slot[i] refers to the \textit{i}th time slot */

4: \textbf{for} \textit{i} = 0 \textbf{to} \textit{len} \textbf{do}
5: \quad \textbf{if} time\_slot[\textit{i}] = \textit{null} \textbf{then}
6: \quad \quad \textit{recv}_\textit{buf}[\textit{i}] \leftarrow 0; \quad \triangleright \text{channel status \textit{null} during \textit{i}th time slot}
7: \quad \textbf{else}
8: \quad \quad \textit{recv}_\textit{buf}[\textit{i}] \leftarrow time\_slot[\textit{i}]; \quad \triangleright \text{received 1 or \textbar\textit{i} in \textit{i}th time slot}
9: \quad \textbf{end if}
10: \textbf{end for}
11: \textbf{end procedure}
**procedure** CONVERTRBNtoBINARY(IN recv_buf, IN len, INOUT out_buf)

/* Convert received RBNS data frame to binary data */

/* The final output binary data will be in out_buf */

boolean runflag : false;

runflag ← false;  // To decode the run of 1’s

for i = 0 to len do

  repeat

    if recv_buf = ¬1 then
      if runflag = false then
        runflag ← true;
        out_buf[i] ← 1;  // output i\textsuperscript{th} bit = 1
      else
        out_buf[i] ← 0;  // result of reduction rule 7.2
      end if
    end if

  until runflag = false;

until runflag = true then
  out_buf[i] ← 1;  // single 1 in data
end if
end if

if recv_buf = 1 then
  if runflag = true then
    out_buf[i] ← 0;  // output i\textsuperscript{th} bit = 0
    runflag ← false;  // end of a run of 1’s
  else
    out_buf[i] ← 1;  // single 1 in data
  end if
end if

if recv_buf = 0 then
  if runflag = true then
    out_buf[i] ← 1;  // inside a run of 1’s
  else
    out_buf[i] ← 0;
  end if
end if

end for

end procedure
We denote a run of 1’s of length $k$ by $R_k$. Clearly, two consecutive runs of 1’s of length $k_1$ and $k_2$, $1 \leq k_1, k_2 \leq n$ in a bit string will be separated by at least one zero. Let us append a zero on left of each such $R_k, 1 \leq k \leq n$ and denote the symbol 0 $R_k$ by $y_k$. We also denote a single zero by the symbol $y_0$. Then each such $y_k, 0 \leq k \leq n$, will be a string of length $k+1$. We describe two different techniques for solving this counting problem - one using generating functions and the other using recurrence relations, as given below.

### 7.5.1 Generating Functions Approach

To find out the total number of occurrences of $R_k$’s, $1 \leq k \leq n$, in all possible $2^n$ strings of length $n$, we first compute the total number of occurrences of exactly $i_k$ number of $y_k$’s. Let this number be denoted by the symbol $N_{n,k}^{i_k}$. Thus, to compute $N_{n,k}^{i_k}$, we consider only those bit strings of length $n + 1$ which contains exactly $i_k$ number of $y_k$’s. In effect, we aim at partitioning the integer $n + 1$ in all possible ways where order counts, such that there are exactly $i_k$ number of blocks, each of size $k+1$. Each such ordered partition will correspond to a distinct string of length $n$ (because of the appended zero on the left of each $y_k, 1 \leq k \leq n$), in which there are exactly $i_k$ number of $R_k$’s. We find this number of possible partitions in the following three steps:

**Step 1:** First we take out $i_k$ blocks of size $k+1$ each, $1 \leq k \leq n$, from the integer $n + 1$ and find the total number of ordered partitioning of $m = (n + 1) - (k + 1)i_k$ in exactly $r$ blocks ($r \geq 1$ for $k \leq n$ and $r = 0$ for $k = n$), such that there is no block of size $k+1$ in the partition.

**Step 2:** Next, on each such ordered partition obtained in step 1 above, we insert the $i_k$ blocks of size $(k + 1)$ each and find the total number of such distinct possible ways. This step is similar to putting $i_k$ identical balls into $r + 1$ boxes, where some boxes may remain empty.

120
Step 3: Finally, we sum over the expression obtained in step 2 above for all possible values of \( r \) and multiply it by \( i_k \) to get \( N^{i_k,k}_{n} \).

Note that for \( k = n \), there is only one such run in the whole string and hence, \( N^{1,n}_{n} = 1 \). For \( k \leq n \), we note that for step 1 above, if we scan the bit string of length \( n + 1 \) from left to right, then the occurrence of a \( y_i, 0 \leq i \leq n \), would correspond to a generating function \((x + x^2 + x^3 + \cdots)\). To exclude the occurrence of \( y_k \)'s, we subtract \( x^{k+1} \) from it. Hence, for \( r \) such blocks, we consider the generating function

\[
G(x) = (x + x^2 + x^3 + \cdots - x^{k+1})^r,
\]

such that the co-efficient of \( x^m \) in \( G(x) \) will be the required number of partitions for step 1.

Now, \( G(x) \) can be written as

\[
G(x) = x^r (1 + x + x^2 + \ldots - x^k)^r,
\]

\[
= x^r \left[ 1 - x^k (1 - x) \right]^r
\]

\[
= x^r \left[ 1 - x^k (1 - x) \right]^r (1 - x)^{-r}
\]

\[
= x^r \left[ \binom{r}{0} - \binom{r}{1} x^k (1 - x) + \binom{r}{2} x^{2k} (1 - x)^2 - \ldots \right]
\]

\[
+ (-1)^q \left[ \binom{r}{q} x^{kq} (1 - x)^q + \ldots \right] \left[ 1 + \binom{r}{1} x + \left( \frac{r + 1}{2} \right) x^2 + \ldots \right]
\]

\[
= x^r \left[ \binom{r}{0} - \binom{r}{1} x^k (1 - x) + \binom{r}{2} x^{2k} (1 - x)^2 - \ldots \right]
\]

\[
+ (-1)^q \left[ \binom{r}{q} x^{kq} (1 - x)^q + \ldots \right] \left[ 1 + \binom{r}{1} x + \left( \frac{r + 1}{2} \right) x^2 + \ldots \right]
\]

\[
= x^r \sum_{q=0}^{r} (-1)^q \binom{r}{q} x^{kq} \sum_{j=0}^{q} (-1)^j \binom{q}{j} x^j \sum_{p=0}^{\infty} \binom{r + p - 1}{p} x^p \quad (7.3)
\]

121
Let $m = n + 1 - (k + 1)i_k - r$. Hence the coefficient of $x^{n+1-(k+1)i_k}$ in $G(x)$ is the coefficient of $x^m$ in equation 7.3 which is equal to

$$N_m = \sum_{q=0}^{r} \sum_{j=0}^{q} (-1)^{q+j} \binom{r + m - 1 - kq - j}{m - kq - j} \binom{r}{q} \binom{q}{j}$$

Hence, the required number $N^{i_{k},k}_n$ for $k \geq 1$ is given by

$$N^{i_{k},k}_n = \sum_{r=1}^{i_k} \frac{n+1-(k+1)i_k}{i_k} \binom{(r+1)+i_k-1}{i_k} N_m$$

$$= \sum_{r=1}^{i_k} \frac{n+1-(k+1)i_k}{i_k} \sum_{q=0}^{r} \sum_{j=0}^{q} (-1)^{q+j} \binom{r + m - 1 - kq - j}{m - kq - j} \binom{r}{q} \binom{q}{j}$$

**Example 7.1** For $n = 8$, $k = 2$ and $i_k = 2$, we get the number:

$$N^{2,2}_8 = 2 \sum_{r=1}^{9-6} \binom{r + 2}{2} \sum_{q=0}^{r} \sum_{j=0}^{q} (-1)^{q+j} \binom{2 - 2q - j}{3 - r - 2q - j} \binom{r}{q} \binom{q}{j}$$

$$= 2 \left[ \binom{3}{2} \left( \binom{2}{2} \binom{1}{0} \binom{0}{0} - \binom{0}{0} \binom{1}{1} \binom{1}{0} + \binom{-1}{1} \binom{1}{1} \binom{1}{1} \right) + 2 \binom{4}{2} \left( \binom{2}{1} \binom{2}{0} \binom{0}{0} - 0 + 0 \right) + 2 \binom{5}{2} \left( \binom{2}{0} \binom{3}{0} \binom{0}{0} - 0 + 0 \right) \right]$$

$$= 2(-1) + 2(1) + 2(0) = 44.$$ 

For a given $k \geq 1$, if we now sum the expression $N^{i_{k},k}_n$ for all possible values of $i_k$, $1 \leq i_k \leq \lfloor (n+1)/(k+1) \rfloor$, then we get the total number of occurrences of $R_k$ in all possible strings of length $n$. Tables 7.1 and 7.2 show the number of occurrences of all possible runlengths of 1’s in a binary string of length 8 and 16 bits respectively. Tables 7.1 and 7.2 along with figure 7.1 demonstrates that the number of occurrences of run of 1’s of length $k$ in a binary string of length $n$, $1 \leq k \leq n$, decreases exponentially as $k$ increases.
Table 7.1: Number of occurrences of runlengths for $n = 8$

<table>
<thead>
<tr>
<th>Runlength size ($k$)</th>
<th>Number of possible values of $i_k$</th>
<th>Number of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>320</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>144</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 7.1: Number of occurrences of different runlengths for $n = 8$
Table 7.2: Number of occurrences of runlengths for $n = 16$

<table>
<thead>
<tr>
<th>Runlength size ($k$)</th>
<th>Number of possible values of $i_k$</th>
<th>Number of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>147456</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>69632</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>32768</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15360</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>7168</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3328</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1536</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>704</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>144</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
7.5.2 Recurrence Relations Approach

Our objective is to successively generate all possible strings containing $R_{n-k}$’s, $1 \leq k \leq n - 1$, starting from all possible strings containing $R_{n+1}$. First we note that there is only one run of ones of length $n$, i.e., the number of $R_n$’s in all possible strings is equal to 1 for any value of $n$. Also, there are only two possible strings containing $R_{n-1}$’s. Consider the example for $n = 8$. The only string containing $R_8$ is 1111 1111, and there are only two possible strings containing $R_7$ given by 0111 1111 and 1111 1110. For $R_6$, there are 5 possible strings: 0011 1111, 1011 1111, 1111 1100, 1111 1101 and 0111 1110. We now try to find a systematic way to generate the strings containing $R_6$ from the two strings containing $R_7$. For this we define the following rules.

**Left Replacement Rule (LRR):** If there is a string containing $R_{n-k+1}$, $k \geq 2$, then by applying this rule, we can generate two possible strings containing $R_{n-k}$ from that $R_{n-k+1}$ by replacing the leftmost ‘1’ in $R_{n-k+1}$ by a ‘0’ and its immediate preceding ‘0’ (if it exists) by either a ‘0’ or a ‘1’.

**Remark 7.1** If there is no ‘0’ preceding $R_{n-k+1}$, then only one string containing $R_{n-k}$ is generated by replacing the leftmost ‘1’ bit.

**Right Replacement Rule (RRR):** If there is a string containing $R_{n-k+1}$, $k \geq 2$, then by applying this rule, we can generate two possible strings containing $R_{n-k}$ from that $R_{n-k+1}$ by replacing the rightmost ‘1’ in $R_{n-k+1}$ by a ‘0’ and its immediate succeeding ‘0’ (if it exists) by either a ‘0’ or a ‘1’.

**Remark 7.2** If there is no ‘0’ following $R_{n-k+1}$, then only one string containing $R_{n-k}$ is generated by replacing the rightmost ‘1’ bit.

**Example 7.2** Thus, from $R_7$ in 0111 1111, we can generate two strings containing $R_6$ by Left replacement rule (LRR) as 0011 1111 and 1011 1111. Also, by Right replacement rule...
Table 7.3: Runlength 5 strings from strings containing runs of length 6

<table>
<thead>
<tr>
<th>Source string</th>
<th>Generated string by LRR</th>
<th>Generated string by RRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011 1111</td>
<td>0001 1111</td>
<td>0011 1110</td>
</tr>
<tr>
<td></td>
<td>0101 1111</td>
<td></td>
</tr>
<tr>
<td>1011 1111</td>
<td>1001 1111</td>
<td>1011 1110</td>
</tr>
<tr>
<td></td>
<td>1101 1111</td>
<td></td>
</tr>
<tr>
<td>1111 1100</td>
<td>0111 1100</td>
<td>1111 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111 1010</td>
</tr>
<tr>
<td>1111 1101</td>
<td>0111 1101</td>
<td>1111 1001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111 1011</td>
</tr>
<tr>
<td>0111 1110</td>
<td>0011 1110</td>
<td>0111 1100</td>
</tr>
<tr>
<td></td>
<td>1011 1110</td>
<td>0111 1101</td>
</tr>
</tbody>
</table>

(RRR) on $R_7$ in 1111 1110, we can generate the two strings containing $R_6$ as 1111 1100 and 1111 1101. By LRR on $R_7$ in 1111 1110, or by RRR on $R_7$ in 0111 1111, we get the same string 0111 11110, i.e., $0R_60$ (duplicate generation).

From all these five strings containing $R_6$, by applying the above two rules LRR and RRR, we can generate all possible strings containing $R_5$’s as shown in table 7.3. However, by this process, duplicate strings are generated, as we see that the strings generated from 0011 1111 and 1011 1111 by RRR are the same as those generated from 0111 1110 by LRR. Also, the strings generated from 1111 1100 and 1111 1101 by LRR are the same as those generated from 0111 1110 by RRR. To avoid such duplicate counting of runs of ones, we would follow a biased replacement strategy as discussed below.

First we note that, in the string 0111 1110, the substring $R_6$ starts at the second bit position from left. By applying RRR on this string, we get two strings $0R_500$ and $0R_501$ in which the substring $R_5$ also starts at the second bit position from left. Hence, neither of these two
strings can be generated by successively applying $RRR$ on any string in which a run of 1’s starts at the leftmost bit position. In general, if we apply $RRR$ on a string $0R_k0\ast\cdots\ast$, where a * may take any value (0 or 1), then these strings cannot be duplicated by successively applying $RRR$ on any string of the form $R_j0\ast\cdots\ast$, $j \geq k$. Also, successive applications of $LRR$ on a string of the form $\ast\cdots\ast0R_j$ will never be identical with any string generated from $0R_k0\ast\cdots\ast$ by $RRR$ only. Thus to avoid duplicate string generation, we formulate the biased replacement rule as follows:

**Biased Replacement Rule**: To generate new strings containing $R_{n-k}$ from a string $S$ containing $R_{n-k+1}$, $2 \leq k \leq n-1,$

i. Apply $LRR$ on the substring $R_{n-k+1}$ unless $R_{n-k+1}$ starts at the leftmost bit position of the string $S$.

ii. If $R_{n-k+1}$ starts at the leftmost bit position of $S$, then apply $RRR$ on $S$.

iii. If $R_{n-k+1}$ starts at the second bit position from left in $S$, then apply both $LRR$ and $RRR$.

We now show below that by using the biased replacement rule, we can avoid duplicate counting of $R_{n-k}$’s generated from $R_{n-k+1}$’s in all possible strings.

In the following discussions, unless otherwise mentioned, by a substring $R_k$ in a string $S$, we always refer to the $R_k$ at a given position in $S$.

The effect of $LRR$ on a substring $R_k$ in $S$ starting at a bit position, say $b$, will be denoted by the function $L(R_k, S)$, whose value is the substring $R_{k-1}$ (starting at the bit position $b+1$) obtained from the $R_k$ by $LRR$. Similarly, the effect of $RRR$ on a substring $R_k$ in $S$ ending at a bit position, say $b$, will be denoted by the function $R(R_k, S)$, whose value is the substring $R_{k-1}$ (ending at the bit position $b-1$) obtained from the $R_k$ by $RRR$.

The result of successive $m$ number of $LRR$ operations on a substring $R_k$ in a string $S$ will be denoted by $L^m(R_k, S)$. Similarly, the result of successive $m$ number of $RRR$ operations
on a substring $R_k$ will be denoted by $\mathcal{R}^m(R_k, S)$. The result of an application of $LRR$ on a substring $R_k$ in $S$ followed by the application of a $RRR$ on the resulting substring will be denoted by $\mathcal{L}\mathcal{R}(R_k, S)$, while application of $RRR$ followed by $LRR$ will be denoted by $\mathcal{R}\mathcal{L}(R_k, S)$.

Hence, we note that
\[ L^m(R_{n-1}, R_0) = \ast^m 0 R_{n-m-1}, \]
and
\[ R^m(R_{n-1}, 0 R_{n-1}) = R_{n-m-1} \ast^m 0, \]
where the symbol \( \ast \) stands for a 0 or 1.

We now have the following results:

**Lemma 7.1** $L^m \mathcal{R}(R_{n-1}, 0 R_{n-1}) = L^m(R_{n-2}, 0 R_{n-2}0)$.

**Proof:** Follows from the definitions of $L$ and $R$ functions.

**Lemma 7.2** $\mathcal{R}^m L(R_{n-1}, R_{n-1}0) = \mathcal{R}^m(R_{n-2}, 0 R_{n-2}0)$.

**Proof:** Follows from the definitions of $L$ and $R$ functions.

In view of lemma 7.1, $L^m \mathcal{R}$ on $R_{n-1}$ in $0 R_{n-1}$ can be equivalently replaced by $m$ successive $L$ functions on $R_{n-2}$ in $0 R_{n-2}0$. Also, due to lemma 7.2, $\mathcal{R}^m L$ on $R_{n-1}$ in $R_{n-1}0$ can be equivalently replaced by $m$ successive $R$ functions on $R_{n-2}$ in $0 R_{n-2}0$.

**Lemma 7.3** $L \mathcal{R}(R_{n-2}, 0 R_{n-2}0) = R \mathcal{L}(R_{n-2}, 0 R_{n-2}0)$.

**Proof:** Follows from the definitions of $L$ and $R$ functions.

Because of lemma 7.3, we can reverse the order of application of $LRR$ and $RRR$ on the string $0 R_{n-2}0$, and repeated application of this property may result in applying all $LRR$ (or
functions on strings generated by \( L \) write, \( L^2R^3 = L(LR)R^2 = LRLRRL = LR(LR)R = LRLRRLR = LRLRRLL = \cdots = RRLRLRC. \) Hence, we get the following result.

**Lemma 7.4** \( L^pR^q(R_{n-2}, 0R_{n-2}) = R^qL^p(R_{n-2}, 0R_{n-2}). \)

In view of lemma 7.4, \( L^mR \) on \( R_{n-2} \) in \( 0R_{n-2} \) may be equivalently replaced by only \( L \) functions on strings generated by \( R^m \) on \( R_{n-2} \) in \( 0R_{n-2} \).

By repeated applications of \( LRR \) on \( R_{n-1} \) in \( 0R_{n-1} \), we may get a resulting string \( S \) given by \( S = * \cdots * 0R_k0 * \cdots * 0R_{n-m-1}, \) for some value of \( k. \) We claim that application of \( LRR \) or \( RRR \) on this resulting \( R_k \) in \( S \) is not needed to generate any new substring \( R_{k-1}, \) as this \( R_{k-1} \) will also be generated by \( LRR \) and \( RRR \) on \( R_{n-2} \) in \( 0R_{n-2} \). This follows from the following lemma.

**Lemma 7.5** Given that \( S = *^{k_1}0R_k0 *^{k_2}0R_{n-k-k_1-k_2-3}, \) we have,

\[
L(R_k, S) = R^{n-k-k_1-2}(k+1) (R_{n-2}, 0R_{n-2}).
\]

**Proof:** \( L(R_k, S) \) is the substring \( R_{k-1} \) in \( *^{k_1+1}0R_k0 *^{k_2}0R_{n-k-k_1-k_2-3}. \)

On the other hand, the string \( R^{n-k-k_1-2}(R_{n-2}, 0R_{n-2}) \) is the substring \( R_{n-2-n+k_k+1+2} \)

\( (= R_{k+1} \) \) in the string \( 0R_{n-2-n+k_k+1+2} * n-k-k_1-2, \) which is \( 0R_{k+1} * n-k-k_1-2. \)

Hence, \( R^{n-k-k_1-2}(R_{n-2}, 0R_{n-2}) = *^{k_1+1}0R_{k-1}0 *^{n-k-k_1-2}, \) which contains the string \( *^{k_1+1}0R_{k-1}0 *^{k_2}0R_{n-k-k_1-k_2-3}. \)

Note that we have \( L^m(R_{n-1}, 0R_{n-1}) \) is the substring \( R_{n-m-1} \) in \( *^m0R_{n-m-1}, \) while the string \( R^m(R_{n-1}, R_{n-1}) \) is the substring \( R_{n-m-1} \) in \( R_{n-m-1}0 *^m. \) Hence, although \( *^m0R_{n-m-1} \) may be identical with \( R_{n-m-1}0 *^m \) when \( m = \frac{(n+1)}{2} \) for odd \( n, \) these two appearances of \( R_{n-m-1} \) (one at the leftmost position and the other at the rightmost position) in the same
Theorem 7.1 The biased replacement rule always produces all possible distinct appearances of $R_{n-k}$’s in all possible strings without any duplicate counting.

Proof: Follows from all the above lemmas.

Now we see that, by LRR on a substring $R_{n-k+1}$, we generate two possible distinct substrings $R_{n-k}$. By RRR on a substring $R_{n-k+1}$ in each string of the form $0R_{n-k+1}0^k-3$, we always two different substrings of the form $0R_{n-k}0^k-2$, whose number is thus doubled on each application of RRR. Since for $k = 2$, the number of such strings of the form $0R_{n-k}0^k-2 = 1$, we get the following theorem about the total number $N_{n-k}$ of $R_{n-k}$’s in all possible strings of length $n$.

Theorem 7.2 $N_n = 1, N_{n-1} = 2,$ and for $k \geq 2,$

$$N_{n-k} = 2N_{n-k+1} + 2^{k-2}$$

Theorem 7.3 $N_n = 1, N_{n-1} = 2,$ and for $k \geq 2,$

$$N_{n-k} = (k + 3)2^{k-2}$$

Proof: Follows by solving the recurrence relation given in theorem 7.2.

Hence, the total number of runs of ones of all possible lengths $(n - k), 0 \leq k \leq n - 2,$ is given by,
\[ S = \sum_{k=2}^{n-2} (k+3)2^{k-2} + 3 \]
\[ = 5 \cdot 2^0 + 6 \cdot 2^1 + 7 \cdot 2^2 + \cdots + (n+1) \cdot 2^{n-4} + 3 \]

Hence, \( 2S = 5 \cdot 2^1 + 6 \cdot 2^2 + \cdots + n \cdot 2^{n-4} + (n+1) \cdot 2^{n-3} \), from which we get, \( S = n \cdot 2^{n-3} - 6 \).

The total number of 1’s and \( \bar{1} \)’s in the RBN coded message obtained after applying the reduction rule 7.1 (considering also the presence of \( R_1 \)'s) is equal to \( 2S + (n+2) \cdot 2^{n-3} = (3n + 2) \cdot 2^{n-3} - 12 \).

Hence the fraction of energy savings over EbT schemes obtained by applying reduction rule 7.1 is given by,

\[ \gamma_{e1} = 1 - \frac{(3n + 2) \cdot 2^{n-3} - 12}{n \cdot 2^n} \quad (7.4) \]

For large \( n \), equation 7.4 is approximately equal to \( 1 - \frac{(3n+2)}{8n} \). Typically, for \( n = 8 \), \( \gamma_{e1} \) is nearly equal to 60\% and for \( n = 1024 \), \( \gamma_{e1} \simeq 63\% \).

Let us now consider the effect of applying reduction rule 7.2 on the RBN coded string (after applying reduction rule 7.1). Every appearance of the pattern \( \bar{1}1 \) in the RBN coded string will be replaced by \( 0\bar{1} \) after applying reduction rule 7.2, and this appearance corresponds to every single '0' in between two runs of 1’s in the original binary string. To compute the total number of such appearances of singleton 0’s trapped between two runs of 1’s, we proceed as follows.

Note that the total number of singleton 0’s in all possible binary strings of length \( n \) is same as the total number of appearances of \( R_1 \)'s, and is equal to \( \mathcal{N}_1 = (n+2)2^{n-3} \). From this, we have to subtract the total number of appearances of singleton 0’s occurring at either end of the string, i.e., in strings of the form \( 01^*^{n-2} \) or \( ^*^{n-2}10 \), whose number is equal to \( 2^{n-2} + 2^{n-2} \).
Figure 7.2: Plot of theoretical savings vs. data frame size
Thus, the total number of 1’s and \( \bar{1} \)’s in the RBN coded message after applying both reduction rules 1 and 2 is equal to 
\[
[(3n + 2) \cdot 2^{n-3} - 12] \cdot (n + 2) 2^{n-3} + 2^{n-1} = (n + 2) \cdot 2^{n-2} - 12.
\]

If we assume that in the original message each bit, whether 0 or 1, consumes \( e_b \) units of energy for communication, then in the proposed communication technique, the fraction of energy saving obtained by applying reduction rule 7.1 followed by reduction rule 7.2 is given by,

\[
\gamma_{e_2} = 1 - \frac{(n + 2)2^{n-2} - 12}{n \cdot 2^n}
\]  

(7.5)

For large \( n \), equation 7.5 is approximately equal to 
\[
1 - \frac{(n+2)}{4n}.
\]

Typically, for \( n = 8 \), \( \gamma_{e_2} \) is nearly equal to 69\% and for \( n = 1024 \), \( \gamma_{e_2} \simeq 75\% \). Figure 7.2 shows the plot of theoretical energy savings (\( \gamma_{e_1} \) and \( \gamma_{e_2} \)) in data transmission against the data frame size \( n \). From the graph we see that if we apply only reduction rule 7.1 on binary encoded data, the maximum possible energy savings in transmitting data, as obtained theoretically, is 63\%. Application of both reduction rules 7.1 and 7.2, increases the maximum possible theoretical savings to 75\%.

### 7.6 Experimental Results

Experimental results demonstrate that protocol \( RBNSiZeComm \) significantly reduces the energy consumption required for transmission, for different types of application scenarios. For the purpose of testing our proposed protocol, we have developed a new benchmark suite, as we could not find any suitable test-suite to measure the savings in energy for real-life scenario data communications.
7.6.1 Test Suite Description

Our proposed *Energy Efficient Communication* (EEC) Benchmark Suite consists of 1100 files, categorized into 10 different file types. The file categories and the test files have been chosen so as to reflect as closely as possible the different types of wireless data communication that can be seen in today’s world. The internet is by far the most heavily used medium for sharing of information and wireless access to the internet has become quite popular. Most people tend to use it to share data, search information and to download data such as music, videos, images, publications, utilities etc. Besides the internet, wireless communication is beginning to play a vital role in sensor networks, such as wireless video surveillance. The EEC benchmark has been designed keeping in mind these two main application areas of wireless networks. Table 7.4 shows the various file category types and the number of files in each category. Below we provide a description of each of the categories:

1. **PS and PDF files**: The postscript and pdf files are predominantly research papers and patent documents. Most of them have been obtained from the *IEEE*, *ACM*, *Springer* and *MicroPatent* digital libraries.

2. **Music files**: The music files in the EEC benchmark suite are in MP3, AAC or WAV encoded form. The majority of the files are in Apple’s protected AAC format, obtained from the online Apple’s iTunes Store. The remainder of the files are either .mp3, .wma or .wav files and have different bit rates to reflect different audio qualities.

3. **Streaming video**: The streaming video data set is a collection of frames of real-time streaming video obtained from a surveillance camera. These files are in the *bitmap* format.

4. **Plain text files**: These files are a mix of log files, configuration files and program source codes written in different languages such as C, C++ and Java.
5. **HTML files**: The HTML files in the suite have been selected to reflect the common activity patterns of users on the internet in today’s world, such as surfing, news reading, searching and internet shopping. They consist of pages from sites such as Yahoo, MSN, Amazon, etc., pages from the popular web based email sites, search results from Google, Yahoo, MSN and other search engines and pages from auction sites such as Ebay.

6. **Image files**: These are a mix of indoor and outdoor pictures in JPEG format with a higher percentage of outdoor pictures.

7. **Video files**: These are either MPEG, WMV or ASX encoded files and have different bit rates to reflect different video qualities.

8. **Binary files**: The binary files are a mix of Microsoft Windows dynamic link library (DLL) files, object files, Linux and Microsoft’s executable binaries, Java class files and Linux and Microsoft’s library files.

9. **Document files**: The files in this category reflect the various commonly used document classes from Microsoft such Word documents, PowerPoint presentations and Excel spreadsheets.

### 7.6.2 Results

For the purpose of the experiments, we assumed a data frame size of 1024 bits. All the reported values in the tables and graphs (tables 7.5 to 7.9 and figures 7.2 and 7.3) are with respect to energy based transmission schemes where the transmission of both ’0’ and ’1’ bit values require the expenditure of energy. The SiZe communication scheme mentioned in the tables and figures, refers to using silence for communicating the ’0’ bit values in a binary coded files, while spending energy for the transmission of the ’1’ bits. RR1 and RR2 represent
Figure 7.3: Energy Savings for the EEC Benchmark Test Suite
the RBN encoded files obtained after applying reduction rules 7.1 and 7.2 respectively on the binary file. It can be recalled that using the RBNSiZeComm transmission protocol for transmitting RBN encoded data, energy expenditure occurs only for transmitting the ’1’ and \( \bar{1} \) bit values, while the ’0’ bit values are communicated as silent periods.

The results of our experiment on the EEC benchmark suite are given in table 7.5 and figure 7.2. The results show that on an average, a binary encoded file consists of 41.6% zeroes, which thus translates into an increased energy savings of 41.6% by using the SiZe communication scheme, as compared to an EbT transmission scheme. Application of reduction rule 7.1 on a binary encoded file to create an RBN encoded file causes an increased savings in energy from 41.6% to 63.1%. Applying reduction rule 7.1 followed by reduction rule 7.2 generates a net transmission energy savings of 68.9% more on an average, with our proposed RBNSiZeComm transmission protocol.

Experimental results also showed that increasing the data frame size increases the fractional savings in energy as longer runs of ones can then be reduced. It increases steeply with the increase in frame size, when the size of the frames is small (8, 16, 32, 64, \ldots \) bits) and plateaus out for larger frame sizes. We observed that in general, for frame sizes larger than 1024 bits, the increase in fractional savings is either very small or none.

We also tested our RBNSiZeComm transmission protocol on several popular compression benchmark test suites [154, 155]. The results for these test suites are presented in tables 7.6 to table 7.9. Figure 7.3 demonstrates the increase in transmission energy savings for each of the test suites, obtained by using the RBNSiZeComm protocol instead of the SiZe communication scheme. The maximum increase in energy savings is obtained for the Maximum Compression test suite (34.4%) [155] while the minimum is for the Large Canterbury suite (21.8%) [154], which consists of only plain text files and hence is not really a very good representative test suite to demonstrate the different types of data that are communicated in real world scenarios. It was included in order to study the performance of the protocol on large plain text files. From the results in figure 7.3 we see that there is an increase of 26.7% in transmission energy
Figure 7.4: Comparison of Average Energy Savings for the Benchmark Test Suites
Table 7.4: Energy Efficient Communication (EEC) Benchmark Suite

<table>
<thead>
<tr>
<th>File Type</th>
<th>Number of Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Postscript Documents</td>
<td>60</td>
</tr>
<tr>
<td>2. Adobe Acrobat Documents</td>
<td>78</td>
</tr>
<tr>
<td>3. Music Files</td>
<td>107</td>
</tr>
<tr>
<td>4. HTML Files</td>
<td>71</td>
</tr>
<tr>
<td>5. Image Files</td>
<td>71</td>
</tr>
<tr>
<td>6. Binaries</td>
<td>82</td>
</tr>
<tr>
<td>7. MS Documents</td>
<td>112</td>
</tr>
<tr>
<td>8. Plain Text Files</td>
<td>122</td>
</tr>
<tr>
<td>9. Video Files</td>
<td>52</td>
</tr>
<tr>
<td>10. Streaming Video</td>
<td>345</td>
</tr>
</tbody>
</table>

savings, when averaged over all benchmark suites considered in the figure, by using the proposed RBNSiZeComm protocol over the SiZe protocol. This average increase is quite close to the 27.3% increase in energy savings observed for the EEC benchmark suite alone that we have proposed. The experimental results also match well with the theoretical average values as given by equations 7.4 and 7.5.

7.7 Conclusion

The redundant binary number system can be used instead of the binary number system in order to reduce the number of zero bits in the data. Coupled with this, the use of silent periods for communicating the 0’s in the bit pattern provides a remarkable amount of energy savings in data transmissions, without reducing the data transmission rate and with nominal hardware overhead. The transmission time remains linear in the number of bits used for data representation, as in the binary number system. Considering an $n$-bit data represen-
Table 7.5: Results for the EEC Benchmark Suite

<table>
<thead>
<tr>
<th>File Type</th>
<th>Percentage Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Silent-Zero Only</td>
</tr>
<tr>
<td>Postscript Documents</td>
<td>49.02</td>
</tr>
<tr>
<td>Adobe Acrobat Documents (PDF)</td>
<td>36.32</td>
</tr>
<tr>
<td>Plain Text Files</td>
<td>47.92</td>
</tr>
<tr>
<td>Binaries</td>
<td>51.98</td>
</tr>
<tr>
<td>Music Files</td>
<td>34.99</td>
</tr>
<tr>
<td>HTML Files</td>
<td>42.16</td>
</tr>
<tr>
<td>Image Files</td>
<td>32.76</td>
</tr>
<tr>
<td>MS Documents</td>
<td>44.72</td>
</tr>
<tr>
<td>Video Files</td>
<td>38.27</td>
</tr>
<tr>
<td>Streaming Video</td>
<td>37.92</td>
</tr>
<tr>
<td>Test File</td>
<td>Percentage Energy Savings</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td></td>
<td>Silent-Zero Only</td>
</tr>
<tr>
<td>Bitmap File (BMP)</td>
<td>19.82</td>
</tr>
<tr>
<td>Dynamic Link Library (DLL)</td>
<td>36.69</td>
</tr>
<tr>
<td>MS Word (DOC)</td>
<td>18.48</td>
</tr>
<tr>
<td>Executable (EXE)</td>
<td>37.94</td>
</tr>
<tr>
<td>Windows Help (HLP)</td>
<td>40.42</td>
</tr>
<tr>
<td>JPEG Image (JPG)</td>
<td>32.86</td>
</tr>
<tr>
<td>Log file (LOG)</td>
<td>45.44</td>
</tr>
<tr>
<td>Adobe Acrobat Document (PDF)</td>
<td>37.72</td>
</tr>
<tr>
<td>Alphabetically Sorted Word-list (DICT)</td>
<td>39.65</td>
</tr>
<tr>
<td>English Text (TXT)</td>
<td>44.91</td>
</tr>
</tbody>
</table>
Table 7.7: Results for the Large Canterbury Test Suite

<table>
<thead>
<tr>
<th>Test File</th>
<th>Percentage Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Silent-Zero Only</td>
</tr>
<tr>
<td>Complete Genome of E.Coli Bacterium (E.coli)</td>
<td>37.37</td>
</tr>
<tr>
<td>King James Version of Bible (bible.txt)</td>
<td>44.42</td>
</tr>
<tr>
<td>CIA World Fact Book (world92.txt)</td>
<td>46.25</td>
</tr>
</tbody>
</table>

Table 7.8: Results for the Canterbury Test Suite

<table>
<thead>
<tr>
<th>Test File</th>
<th>Percentage Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Silent-Zero Only</td>
</tr>
<tr>
<td>Excel Spreadsheet (kennedy.xls)</td>
<td>76.40</td>
</tr>
<tr>
<td>SPARC Executable (sum)</td>
<td>54.47</td>
</tr>
<tr>
<td>English Text (alice29.txt)</td>
<td>45.28</td>
</tr>
<tr>
<td>GNU Manual Page (xargs.1)</td>
<td>46.71</td>
</tr>
<tr>
<td>CCITT test set (ptt5)</td>
<td>30.50</td>
</tr>
<tr>
<td>Shakespeare (asyoulik.txt)</td>
<td>43.82</td>
</tr>
<tr>
<td>Poetry (plrabn12.txt)</td>
<td>43.94</td>
</tr>
<tr>
<td>HTML Source (cp.html)</td>
<td>38.41</td>
</tr>
<tr>
<td>Technical Writing (lcet10.txt)</td>
<td>43.48</td>
</tr>
<tr>
<td>LISP Source (grammar.lsp)</td>
<td>52.47</td>
</tr>
<tr>
<td>C Source (fields.c)</td>
<td>49.82</td>
</tr>
</tbody>
</table>
Table 7.9: Results for the Calgary Test Suite

<table>
<thead>
<tr>
<th>Test File</th>
<th>Percentage Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Silent-Zero Only</td>
</tr>
<tr>
<td>Bibliography (bib)</td>
<td>46.73</td>
</tr>
<tr>
<td>Fiction Book (book1)</td>
<td>42.96</td>
</tr>
<tr>
<td>Non-fiction Book troff format (book2)</td>
<td>42.90</td>
</tr>
<tr>
<td>Geophysical Data (geo)</td>
<td>58.95</td>
</tr>
<tr>
<td>USENET Batch File (news)</td>
<td>43.71</td>
</tr>
<tr>
<td>Object Code for VAX (obj1)</td>
<td>44.86</td>
</tr>
<tr>
<td>Object Code for Apple Mac (obj2)</td>
<td>36.44</td>
</tr>
<tr>
<td>Technical Paper (paper1)</td>
<td>43.43</td>
</tr>
<tr>
<td>Technical Paper (paper2)</td>
<td>42.33</td>
</tr>
<tr>
<td>B&amp;W Fax Picture (pic)</td>
<td>30.50</td>
</tr>
<tr>
<td>C Source Code (progc)</td>
<td>46.85</td>
</tr>
<tr>
<td>LISP Source Code (progl)</td>
<td>45.90</td>
</tr>
<tr>
<td>PASCAL Source Code (progp)</td>
<td>48.32</td>
</tr>
<tr>
<td>Transcript of Terminal Session (trans)</td>
<td>46.54</td>
</tr>
</tbody>
</table>
tation, we have derived the average case reduction in the number of non-zero digits to be transmitted using the redundant binary radix, assuming that each of the $2^n$ binary strings is equally likely to occur. The fraction of energy savings obtained on an average, by recoding the binary string using RBNS and using the proposed RBNSizComm transmission protocol is 

$$1 - \frac{(n+2)2^{n-2} - 12}{n \cdot 2^n}.$$ 

We have also proposed a benchmark test suite called the energy efficient communication (EEC) benchmark suite in order to test the performance of different energy efficient communication strategies, as we could not find any suitable test suite to measure the energy savings for real-life scenario data communications. Experimental results show that without degrading the network throughput, our proposed RBNSizComm transmission protocol offers a reduction in energy consumption by nearly 69% on an average, when compared to existing energy based transmission schemes.