Implementazione Accordo Bizantino – Cattaneo Kevin – S4944382
Protocollo Las Vegas

Per l'implementazione dell'accordo bizantino, protocollo Las Vegas si è utilizzato il linguaggio C++.

Di seguito il codice C++

```
#include <iostream>
#include <random>
#include <time.h>
#include <vector>

const int t = 1; // numero processi inaffidabili
const int numProc = 8*t+1; // numero processi totali, l'ennesimo è inaffidabile
const int numRun = pow(2,10);
const int L = 5*t + 1;
const int H = 6*t + 1;

int LVByzantine(int bit[], int& round){
  int a[numProc][numProc-t];

  while (true){
    round++;
    // Trasmetto bit (fra righe), tranne quello inaffidabile
    for(int k=0; k<numProc-t; k++)
      for(int i=0; i<numProc-t; i++)
          a[k][i] = bit[k];

    // Controllo se si è raggiunto il consenso
    /***
    NOTA: il controllo è posto qui per catturare anche i casi in cui
    la distribuzione casuale di bit dia valori identici
    (e quindi impiega solo un round per trovare il consenso)
    ***/
    bool consenso = true;
    for(int i=0; i<numProc-t-1; i++)
      if(a[i][0] != a[i+1][0])
        consenso = false;
    if(consenso) return a[0][0];

    // Trasmissione della confusione da parte degli inaffidabili
    for(int k=0; k<numProc-t; k++){
      for(int i=numProc-t; i<numProc; i++)
        a[i][k] = 1-bit[k];
    }

    int numZero[numProc-t];
    int numOne[numProc-t];
    int maj[numProc-t];
    int tally[numProc-t];
    int moneta = rand()%2; // moneta globale: esito uguale x tutti ad ogni round

    // Conta dei valori maggioritari (solo per processi affidabili)
```

```
    for(int k=0; k<numProc-t; k++){
      numZero[k] = 0;
      numOne[k] = 0;

      for(int i=0; i<numProc; i++){
        if(i != k){
          if(a[i][k] == 0)
            numZero[k]++;
          else numOne[k]++;
        }
      }

      // Includo il proprio bit trasmesso
      if(bit[k] == 0)
        numZero[k]++;
      else numOne[k]++;

      if(numZero[k] > numOne[k]){
        maj[k] = 0;
        tally[k] = numZero[k];
      }
      else{
         maj[k] = 1;
         tally[k] = numOne[k];
      }

      // Confronti
      int soglia;
      if(moneta==0)
        soglia = L;
      else soglia = H;
      if(tally[k] >= soglia)
        bit[k] = maj[k];
      else bit[k] = 0;
      if(tally[k] >= 7*t+1)
        bit[k] = maj[k];
    }
  }
}

int main(){
  srand(time(NULL));
  int bit[numProc-t]; // bit trasmessi da processi affidabili
  int round;
  std::vector<int>roundTot;
  std::vector<int>freq(numRun+1,0);

  for(int i=0; i<numRun; i++){
    for(int j=0; j<numProc-t; j++)
      bit[j] = rand()%2;

    LVByzantine(bit, round=0);
    roundTot.push_back(round);
  }
```

```
  // Freq[0] = numero di run in cui l'accordo è raggiunto in 1 round
  for(int i=0; i<numRun; i++)
    for (int e : roundTot)
      if(e == i+1)
        freq[i]++;
  std::cout << "Round | Frequenza consensi\n";
  double medio = 0.0;
  for(int i=0; i<numRun; i++)
    if(freq[i] != 0){
      medio += (double)((i+1)*freq[i]);
      std::cout << i+1 << ": " << freq[i] << std::endl;
      if(accumulate(freq.begin(),freq.begin()+i,0) == numRun) break;
    }

  medio /= numRun;
  std::cout << "Valore atteso round per ottenere il consenso = " << medio << "\n";
}
```

Screenshot dell'output ottenuto

```
Round | Frequenza consensi
1: 7
2: 856
3: 161
Valore atteso round per ottenere il consenso = 2.15039
```

Commento

Si osserva che il valore atteso del numero di round necessari al raggiungimento dell'accordo configurando casualmente i bit iniziali è pari a 2.15039. In accordo con la teoria, tale algoritmo converge in un numero atteso di round che non dipende dal numero di processi inaffidabili, grazie all'introduzione di due soglie. Difatti il numero di round per il raggiungimento della convergenza è costante e pari a 2.

Come si puntualizza nel codice, il controllo è posto subito dopo la trasmissione dei bit (riempimento della matrice) per catturare anche i casi in cui la distribuzione casuale di bit dia valori identici e quindi il consenso viene trovato subito, ovvero in un numero di round pari a 1.