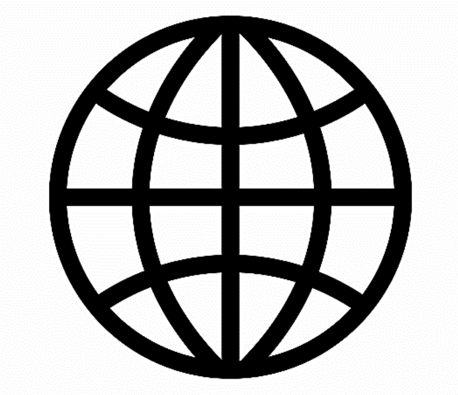
Atma Ram Sanatan Dharma College
University of Delhi



# Computer Networks

Practical Assignment for Paper Code 32341303


Submitted By
Sudipto Ghosh
College Roll No. 19/78003
BSc (Hons) Computer Science

Submitted To
Ms Uma Ojha
Department of Computer Science

# PRACTICAL 1

## Objective

Simulate Cyclic Redundancy Check (CRC) error detection algorithm for a noisy channel.

## Code

```
/**
 * Simulate Cyclic Redundancy Check (CRC) error detection
 * algorithm for a noisy channel.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <iostream>

using namespace std;

int main()
{
  // Message
  int mSize;
  int message[255];
  cout << "Enter Message Size: ";
  cin >> mSize;
  cout << "Enter Message: ";
  for (int i = 0; i < mSize; i++)
    cin >> message[i];

  // Generator
  int gSize;
  int generator[64];
  cout << "Enter Generator Size: ";
  cin >> gSize;
  cout << "Enter Generator: ";
  for (int i = 0; i < gSize; i++)
    cin >> generator[i];

  if (!(generator[0] == 1 &&
        generator[gSize - 1] == 1))
  {
    cerr << "\nERROR: MSB and LSB of the Generator must be 1\n";
    return -1;
  }

  cout << "\nSENDER\n======\n";
  cout << "Message: ";
  for (int i = 0; i < mSize; i++)
    cout << message[i];
```

```
cout << endl;
cout << "Generator: ";
for (int i = 0; i < gSize; i++)
  cout << generator[i];
cout << endl;

// Message + r 0's
int codeword[mSize + (gSize - 1)];
for (int i = 0; i < mSize; i++)
  codeword[i] = message[i];
for (int i = mSize; i < mSize + (gSize - 1); i++)
  codeword[i] = 0;

// Binary Division
int temp[mSize + (gSize - 1)];
for (int i = 0; i < mSize + (gSize - 1); i++)
  temp[i] = codeword[i];
for (int i = 0; i < mSize; i++)
{
  int j = 0, k = i;
  if (temp[k] >= generator[j])
    while (j < gSize)
      temp[k++] ^= generator[j++];
}

// CRC
int crc[64];
for (int i = 0, j = mSize; i < (gSize - 1); i++, j++)
  crc[i] = temp[j];

cout << "CRC: ";
for (int i = 0; i < (gSize - 1); i++)
  cout << crc[i];
cout << endl;

// Codeword + CRC
for (int i = 0, j = mSize; i < (gSize - 1); i++, j++)
  codeword[j] = crc[i];

cout << "Transmitted Codeword: ";
for (int i = 0; i < mSize + (gSize - 1); i++)
  cout << codeword[i];
cout << endl;

cout << "\nNOISY CHANNEL SIMULATION\n========================\n";
int nb, n;
cout << "Enter Number of Bits to Flip: ";
cin >> nb;
```

```
if (nb > 0 && nb < mSize + (gSize - 1))
{
  if (nb == 0)
    cout << "Codeword Not Changed.\n";
  for (int i = 0; i < nb; i++)
  {
    cout << "Enter Bit Position to Flip: ";
    cin >> n;
    if (n > 0 && n < mSize + (gSize - 1))
      codeword[n - 1] = codeword[n - 1] == 0 ? 1 : 0;
    else
      cout << "Invalid Position. Codeword Not Changed.\n";
  }
}
else
  cout << "Invalid Request. Codeword Not Changed.\n";

cout << "\nRECEIVER\n========\n";
cout << "Received Codeword: ";
for (int i = 0; i < mSize + (gSize - 1); i++)
  cout << codeword[i];
cout << endl;

// Binary Division
int temp2[mSize + (gSize - 1)];
for (int i = 0; i < mSize + (gSize - 1); i++)
  temp2[i] = codeword[i];
for (int i = 0; i < mSize; i++)
{
  int j = 0, k = i;
  if (temp2[k] >= generator[j])
    while (j < gSize)
      temp2[k++] ^= generator[j++];
}

// Remainder
int rem[64];
for (int i = mSize, j = 0; i < mSize + (gSize - 1); i++, j++)
  rem[j] = temp2[i];

cout << "Remainder: ";
for (int i = 0; i < (gSize - 1); i++)
  cout << rem[i];
cout << endl;

// Checking Error
int flag = false;
for (int i = 0; i < (gSize - 1); i++)
```

```
      if (rem[i] != 0)
        flag = true;

    // Declare Result
    cout << endl;
    if (!flag)
      cout << "TRANSMISSION OK!" << endl;
    else
      cout << "TRANSMISSION ERROR DETECTED!" << endl;

    return 0;
}
```

## Output

```
Enter Message Size: 8
Enter Message: 1 0 0 1 1 1 0 1
Enter Generator Size: 4
Enter Generator: 1 0 0 1

SENDER
======
Message: 10011101
Generator: 1001
CRC: 100
Transmitted Codeword: 10011101100

NOISY CHANNEL SIMULATION
========================
Enter Number of Bits to Flip: 0
Invalid Request. Codeword Not Changed.

RECEIVER
========
Received Codeword: 10011101100
Remainder: 000

TRANSMISSION OK!
```

```
Enter Message Size: 8
Enter Message: 1 0 0 1 1 1 0 1
Enter Generator Size: 4
Enter Generator: 1 0 0 1

SENDER
======
Message: 10011101
Generator: 1001
CRC: 100
Transmitted Codeword: 10011101100

NOISY CHANNEL SIMULATION
========================
Enter Number of Bits to Flip: 1
Enter Bit Position to Flip: 3

RECEIVER
========
Received Codeword: 10111101100
Remainder: 100

TRANSMISSION ERROR DETECTED!
```

```
Enter Message Size: 8
Enter Message: 1 0 0 1 1 1 0 1
Enter Generator Size: 4
Enter Generator: 1 0 1 0

ERROR: MSB and LSB of the Generator must be 1
```

# PRACTICAL 2

## Objective

Simulate and implement stop and wait protocol for noisy channel.

## Code

```
/**
 * Simulate and implement stop and wait protocol for noisy channel.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

// protocol.hpp
#include <cstdio>
#include <string>
#define MAX_PKT 4

using namespace std;

typedef enum
{
  dat,
  ack,
  nak
} frameKind;

typedef enum
{
  wait,
  frameArrival
} eventType;

typedef struct
{
  unsigned char data[MAX_PKT];
} packet;

typedef struct
{
  packet *info;
  frameKind kind;
  unsigned int seq;
  unsigned int ack;
} frame;

class Protocol
{
public:
```

```
int sentSeq;
int receivedSeq;

packet dataPacket;
frame senderFrame, receiverFrame;

Protocol()
{
  sentSeq = receivedSeq = -1;
}

int waitForEvent(eventType e)
{
  return e == frameArrival;
}

string showkind(frameKind k) //display the event type
{
  switch (k)
  {
  case dat:
    return "data";
    break;
  case ack:
    return "ack";
    break;
  case nak:
    return "nak";
    break;
  }
  return "";
}

// SENDER: Network -> Data Link Interface
void fromNetworkLayer(packet &i)
{
  printf("\nEncapsulating Packet<data='%s'> ...", i.data);
  senderFrame.seq = ++sentSeq;
  senderFrame.kind = dat;
  senderFrame.info = &i;
}

// SENDER: Data Link -> Physical Interface
void toPhysicalLayer(frame &f)
{
  if (f.kind == dat)
    printf("\nSending DataFrame<kind=%s, sequence=%i> to Physical Layer ..."
,
```

```
            showkind(f.kind).c_str(), f.seq);
    else
      printf("\nSending ControlFrame<kind=%s, ack=%i> to Physical Layer ...",
            showkind(f.kind).c_str(), f.ack);
  }

  // RECEIVER: Physical -> Data Link Interface
  void fromPhysicalLayer(frame &f)
  {
    printf("\nReceived DataFrame<kind=%s, sequence=%i> from Physical Layer ...
",
          showkind(f.kind).c_str(), f.seq);
    printf("\nValidating Sequence Number ... ");
    if (receivedSeq != f.seq)
      printf("\nDecapsulating Frame ...");
    else
    {
      printf("\nDuplicate Frame Encountered ...");
      printf("\nDiscarding Frame ...");
    }
  }

  // RECEIVER: Data Link -> Network Interface
  void toNetworkLayer(packet &p)
  {
    printf("\nSending Packet<data='%s'> to Network Layer ...", p.data);
    receivedSeq = senderFrame.seq;
    receiverFrame.seq = 0;
    receiverFrame.kind = ack;
    receiverFrame.ack = senderFrame.seq + 1;
  }
};

// main.cpp
#include <cstring>
#include <cstdlib>
#include "protocol.hpp"

class stopAndWait : public Protocol
{
public:
  string buf;
  eventType event;

  bool flag, start;
  int coeff, count, len, lim;
```

```
  stopAndWait(string s, int t)
  {
    buf = s;
    coeff = t;
    lim = 1e6;
    flag = false;
    start = false;
    count = 0;
  }

  void sender();
  void receiver();
};

void stopAndWait::sender()
{
  if (!start)
  {
    lim = buf.length() % MAX_PKT == 0
            ? buf.length() / MAX_PKT
            : buf.length() / MAX_PKT + 1;
    printf("\nDividing Data into Groups of %d-bytes Each ...", MAX_PKT);
    start = !start;
  }

  printf("\n\nSENDER\n======");
  if (count > 0)
  {
    if (count % coeff == 0)
    {
      printf("\nERROR: SIMULATED TIMEOUT ...");
      flag = true;
    }
    else
    {
      printf("\nReceived ControlFrame<kind=%s, ack=%d> ...",
             showkind(receiverFrame.kind).c_str(),
             receiverFrame.ack);
      if (flag)
      {
        count--;
        flag = !flag;
      }
    }
    if (receiverFrame.kind == nak || flag)
    {
      printf("\nResending Previous Frame ...");
      count--;
```

```
      flag = true;
    }
  }

  if (count == lim)
  {
    printf("\n\nData '%s' Sent Successfully ...", buf.c_str());
    exit(0);
  }

  while (count < lim)
  {
    while (event != wait)
    {
      if (!flag)
      {
        printf("\nEncapsulating Data D%d into a Packet ...", count + 1);
        for (int i = 0; i < MAX_PKT; i++)
          dataPacket.data[i] = buf[i + count * MAX_PKT];
        printf("\nPassing Packet to Data Link Layer ...");
        event = frameArrival;
      }
      if (waitForEvent(event))
      {
        if (!flag)
          fromNetworkLayer(dataPacket);
        toPhysicalLayer(senderFrame);
        event = wait;
      }
      receiver();
    }
  }
}

void stopAndWait::receiver()
{
  printf("\n\nRECEIVER\n========");
  if (event == wait)
  {
    fromPhysicalLayer(senderFrame);
    if (!flag)
    {
      toNetworkLayer(dataPacket);
      count++;
    }
    else
      count += 2;
    toPhysicalLayer(receiverFrame);
```

```
      event = frameArrival;
      sender();
    }
}

int main()
{
  char temp[50];
  printf("Enter Data: ");
  scanf("%s", temp);

  int temp2;
  printf("Enter Simulation Noise (>=2): ");
  scanf("%i", &temp2);

  stopAndWait *obj = new stopAndWait(string(temp), temp2);
  obj->sender();
  delete obj;

  return 0;
}
```

## Output

```
Enter Data: ARSD78003
Enter Simulation Noise (>=2): 2

Dividing Data into Groups of 4-bytes Each ...

SENDER
======
Encapsulating Data D1 into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='ARSD'> ...
Sending DataFrame<kind=data, sequence=0> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=0> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='ARSD'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=1> to Physical Layer ...

SENDER
======
Received ControlFrame<kind=ack, ack=1> ...
Encapsulating Data D2 into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='7800'> ...
Sending DataFrame<kind=data, sequence=1> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='7800'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=2> to Physical Layer ...
```

```
SENDER
======
ERROR: SIMULATED TIMEOUT ...
Resending Previous Frame ...
Sending DataFrame<kind=data, sequence=1> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Duplicate Frame Encountered ...
Discarding Frame ...
Sending ControlFrame<kind=ack, ack=2> to Physical Layer ...

SENDER
======
Received ControlFrame<kind=ack, ack=2> ...
Encapsulating Data D3 into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='3'> ...
Sending DataFrame<kind=data, sequence=2> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=2> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='3'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=3> to Physical Layer ...

SENDER
======
Received ControlFrame<kind=ack, ack=3> ...

Data 'ARSD78003' Sent Successfully ...
```

# PRACTICAL 3

## Objective

Simulate and implement the selective repeat sliding window protocol.

## Code

```cpp
/**
 * Simulate and implement the selective repeat sliding window protocol.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

// protocol.hpp
#include <cstdio>
#include <string>
#include <iostream>
#define MAX_PKT 5

using namespace std;

typedef enum
{
  dat,
  ack,
  nak
} frameKind;

typedef enum
{
  timeout,
  checksumError,
  frameArrival,
  networkLayerReady
} eventType;

typedef struct
{
  unsigned char data;
} packet;

typedef struct
{
  packet *info;
  frameKind kind;
  unsigned int seq;
  unsigned int ack;
} frame;
```

```
class Protocol
{
public:
  eventType event;
  bool noNak, errorDetected;
  int MAX_SEQ, flag, err, buf;
  int frameExpected, frameToSend;

  packet dataPacket;
  frame senderFrame, receiverFrame;

  Protocol()
  {
    noNak = true;
    err = buf = -1;
    flag = frameToSend = frameExpected = 0;
    errorDetected = false;
  }

  int waitForEvent(eventType e)
  {
    return e == frameArrival;
  }

  string showkind(frameKind k)
  {
    switch (k)
    {
    case dat:
      return "data";
      break;
    case ack:
      return "ack";
      break;
    case nak:
      return "nak";
      break;
    }
    return "";
  }

  // Network -> Data Link Interface
  void fromNetworkLayer(packet &i)
  {
    printf("\nEncapsulating Packet<data='%c'> ...", i.data);
    senderFrame.seq = frameToSend;
    senderFrame.kind = dat;
    senderFrame.info = &i;
```

```
    frameToSend = (frameToSend + 1) % (MAX_SEQ + 1);
  }

  // Data Link -> Physical Interface
  void toPhysicalLayer(frame &f)
  {
    if (event == timeout)
    {
      cout << "\nTimeout period expired. Resending frame with sequence no. " <
< err;
      f.seq = err;
      err = -1;
      frameToSend = (err + 1) % (MAX_SEQ + 1);
      event = frameArrival;
    }
    else if (f.kind == dat)
      printf("\nSending DataFrame<kind=%s, sequence=%i> to Physical Layer ..."
,
             showkind(f.kind).c_str(), f.seq);
    else
    {
      if (err != -1)
      {
        if (!noNak)
        {
          f.kind = nak;
          f.ack = err;
          noNak = true;
        }
        else
        {
          f.kind = ack;
          f.ack = err - 1;
        }
      }
      else if (buf != -1)
      {
        f.ack = buf;
        frameExpected = (buf + 1) % (MAX_SEQ + 1);
        frameToSend = frameExpected;
        buf = -1;
      }
      printf("\nSending ControlFrame<kind=%s, ack=%i> to Physical Layer ...",
             showkind(f.kind).c_str(), f.ack);
    }
  }

  // Data Link -> Network Interface
```

```
  void toNetworkLayer(packet &p)
  {
    printf("\nSending Packet<data='%c'> to Network Layer ...", p.data);
    receiverFrame.seq = frameToSend - 1;
    receiverFrame.kind = ack;
    receiverFrame.ack = frameExpected;
    frameExpected = (frameExpected + 1) % (MAX_SEQ + 1);
  }

  // Physical -> Data Link Interface
  void fromPhysicalLayer(frame &f)
  {
    printf("\nReceived DataFrame<kind=%s, sequence=%i> from Physical Layer ...
",
           showkind(f.kind).c_str(), f.seq);
    printf("\nValidating Sequence Number ... ");
    {
      if (frameExpected == f.seq)
      {
        if (f.seq == 1 && flag == 0) // Error Simulation
        {
          cout << "\nError in received frame ...";
          flag = 1;
          noNak = false;
          errorDetected = true;
          err = f.seq;
        }
        else
        {
          printf("\nDecapsulating Frame ...");
          noNak = true;
          toNetworkLayer(dataPacket);
        }
      }
      else
      {
        printf("\nFrame out of order. Storing in buffer ...");
        buf = f.seq;
      }
    }
  }
};

// main.cpp
#include <cstring>
#include <cstdlib>
#include <cmath>
#include "protocol.hpp"
```

```
void getch()
{
  cin.ignore();
  cin.get();
  return;
}

void clrscr()
{
#ifdef _WIN32
  system("cls");
#elif __unix__
  system("clear");
#endif
  return;
}

class selectiveRepeatSlidingWindow : public Protocol
{
public:
  string in_buf;

  selectiveRepeatSlidingWindow(int n, string s)
  {
    MAX_SEQ = n;
    in_buf = s;
  }

  void sender();
  void receiver();
};

void selectiveRepeatSlidingWindow::sender()
{
  event = frameArrival;
  printf("\n\nSENDER\n======");
  if (frameToSend ==
          (err + (MAX_SEQ / 2)) %
              (MAX_SEQ + 1) &&
      errorDetected == true &&
      err >= 0)
  {
    event = timeout;
    frameToSend = err;
    errorDetected = 0;
  }
  else if (frameToSend == MAX_SEQ &&
```

```
            frameToSend != frameExpected &&
            errorDetected == true)
  {
    fromNetworkLayer(dataPacket);
    frameToSend = frameExpected;
    errorDetected = false;
  }
  else if (event == frameArrival)
  {
    printf("\nEncapsulating Data '%c' into a Packet ...", in_buf[frameToSend])
;
    dataPacket.data = in_buf[frameToSend];
    printf("\nPassing Packet to Data Link Layer ...");
    fromNetworkLayer(dataPacket);
  }
  toPhysicalLayer(senderFrame);
  receiver();
}

void selectiveRepeatSlidingWindow::receiver()
{
  printf("\n\nRECEIVER\n========");
  fromPhysicalLayer(senderFrame);
  toPhysicalLayer(receiverFrame);
  getch();
  clrscr();
  sender();
}

int main()
{
  int n;
  cout << "\nEnter bits needed to identify window: ";
  cin >> n;

  char temp[50];
  printf("Enter Data: ");
  scanf("%s", temp);

  selectiveRepeatSlidingWindow *obj =
      new selectiveRepeatSlidingWindow(
          pow(2, n) - 1,
          string(temp));
  obj->sender();
  delete obj;

  return 0;
}
```

## Output

```
Enter bits needed to identify window: 3
Enter Data: Sudipto


SENDER
======
Encapsulating Data 'S' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='S'> ...
Sending DataFrame<kind=data, sequence=0> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=0> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='S'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...

SENDER
======
Encapsulating Data 'u' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='u'> ...
Sending DataFrame<kind=data, sequence=1> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Error in received frame ...
Sending ControlFrame<kind=nak, ack=1> to Physical Layer ...

SENDER
======
Encapsulating Data 'd' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='d'> ...
Sending DataFrame<kind=data, sequence=2> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=2> from Physical Layer ...
Validating Sequence Number ...
Frame out of order. Storing in buffer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...
```

```
SENDER
======
Encapsulating Data 'i' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='i'> ...
Sending DataFrame<kind=data, sequence=3> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=3> from Physical Layer ...
Validating Sequence Number ...
Frame out of order. Storing in buffer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...

SENDER
======
Timeout period expired. Resending frame with sequence no. 1

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='i'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=3> to Physical Layer ...

SENDER
======
Encapsulating Data 'p' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='p'> ...
Sending DataFrame<kind=data, sequence=4> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=4> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='p'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=4> to Physical Layer ...
```

```
SENDER
======
Encapsulating Data 't' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='t'> ...
Sending DataFrame<kind=data, sequence=5> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=5> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='t'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=5> to Physical Layer ...

SENDER
======
Encapsulating Data 'o' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='o'> ...
Sending DataFrame<kind=data, sequence=6> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=6> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='o'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=6> to Physical Layer ...

SENDER
======
Encapsulating Data '' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data=''> ...
Sending DataFrame<kind=data, sequence=7> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=7> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data=''> to Network Layer ...
Sending ControlFrame<kind=ack, ack=7> to Physical Layer ...
```

# PRACTICAL 4

## Objective

Simulate and implement the Distance Vector Routing algorithm.

## Code

```
/**
 * Simulate and implement the distance vector routing algorithm.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <cstdio>
#include <climits>
#include <iomanip>
#include <iostream>
#define MAX_NODES 10

using namespace std;

class Graph
{
public:
  int edges;
  int vertices;
  int nextHop[MAX_NODES][MAX_NODES];
  int distances[MAX_NODES][MAX_NODES];
  int adjMatrix[MAX_NODES][MAX_NODES];

  void input(int v, int e)
  {
    edges = e;
    vertices = v;

    // initialize the adjacency matrix
    for (int i = 0; i < v; i++)
      for (int j = 0; j < v; j++)
        adjMatrix[i][j] = 0;

    int src, dest, weight;

    // populate the adjacency matrix
    for (int i = 0; i < edges; i++)
    {
      cout << "\nEDGE " << (i + 1)
           << "\n======\n";
      cout << "Enter Source: ";
      cin >> src;
```

```
    cout << "Enter Destination: ";
    cin >> dest;
    cout << "Enter Weight: ";
    cin >> weight;
    adjMatrix[src - 1][dest - 1] = weight;
    adjMatrix[dest - 1][src - 1] = weight;
  }
}

void display()
{
  for (int i = 0; i < vertices; i++)
  {
    for (int j = 0; j < vertices; j++)
      cout << setw(5) << adjMatrix[i][j] << " ";
    cout << endl;
  }
}

void distanceVector()
{
  // populate additional data structures
  for (int i = 0; i < vertices; i++)
    for (int j = 0; j < vertices; j++)
    {
      if (i == j)
        distances[i][j] = 0;
      else if (adjMatrix[i][j] == 0)
        distances[i][j] = INT_MAX / 2;
      else
        distances[i][j] = adjMatrix[i][j];
      nextHop[i][j] = -1;
    }

  cout << "Initial Distance Matrix\n";
  cout << "======================\n";
  for (int i = 0; i < vertices; i++)
  {
    for (int j = 0; j < vertices; j++)
      if (distances[i][j] == INT_MAX / 2)
        cout << setw(5) << right << "INF"
             << " ";
      else
        cout << setw(5) << distances[i][j] << " ";
    cout << endl;
  }

  // iterate if a more efficient route exists
```

```cpp
bool flag;
do
{
  // assume a shorter route does not exist
  flag = false;
  // iterate over routers considering them to be src
  for (int i = 0; i < vertices; i++)
    // iterate over all neighbours
    for (int j = 0; j < vertices; j++)
      // iterate over possible destination routers
      for (int k = 0; k < vertices; k++)
        // check if the cost of sending packet to kth router
        // is less than the current cost
        if (distances[i][j] > (distances[i][k] + distances[k][j]))
        {
          // update the cost i.e. distance
          distances[i][j] = distances[j][i] =
              distances[i][k] + distances[k][j];
          // update router for the next hop
          nextHop[i][j] = nextHop[j][i] = k;
          // declare that a shorter route was found
          flag = true;
        }
} while (flag);

cout << "\nFinal Distance Matrix\n";
cout << "======================\n";
for (int i = 0; i < vertices; i++)
{
  for (int j = 0; j < vertices; j++)
    cout << setw(5) << distances[i][j] << " ";
  cout << endl;
}

// display router configurations
for (int i = 0; i < vertices; i++)
{
  cout << "\nRouting Table for Router " << (i + 1) << ":";
  cout << "\nDest Router \t Via \t\t Distance";
  cout << "\n=========== \t ======== \t ========\n";
  for (int j = 0; j < vertices; j++)
  {
    if (i == j)
      continue;
    cout << (j + 1)
         << " \t\t ";
    if (nextHop[i][j] == -1)
      cout << "-";
```

```
        else
          cout << (nextHop[i][j] + 1);
        cout << " \t\t "
             << distances[i][j] << endl;
      }
    }
  }
};

int main()
{
  int v, e;
  int link1, link2;
  Graph graph;

  cout << "Enter No. of Nodes: ";
  cin >> v;
  cout << "Enter No. of Edges: ";
  cin >> e;

  graph.input(v, e);

  cout << "\nGRAPH\n=====\n";
  graph.display();
  cout << endl;

  graph.distanceVector();

  cout << "\nSimulating Link Failure\n";
  cout << "=======================\n";
  cout << "Enter Routers to Break Link Between: ";
  cin >> link1 >> link2;
  cout << endl;

  graph.adjMatrix[link1 - 1][link2 - 1] =
      graph.adjMatrix[link2 - 1][link1 - 1] = 0;

  graph.distanceVector();

  return 0;
}
```

## Output

```
Enter No. of Nodes: 5
Enter No. of Edges: 6

EDGE 1
======
Enter Source: 1
Enter Destination: 2
Enter Weight: 7

EDGE 2
======
Enter Source: 1
Enter Destination: 5
Enter Weight: 1

EDGE 3
======
Enter Source: 2
Enter Destination: 5
Enter Weight: 8

EDGE 4
======
Enter Source: 2
Enter Destination: 3
Enter Weight: 1

EDGE 5
======
Enter Source: 5
Enter Destination: 4
Enter Weight: 2

EDGE 6
======
Enter Source: 3
Enter Destination: 4
Enter Weight: 2

GRAPH
=====
    0     7     0     0     1
    7     0     1     0     8
    0     1     0     2     0
    0     0     2     0     2
    1     8     0     2     0

Initial Distance Matrix
======================
    0     7   INF   INF     1
    7     0     1   INF     8
  INF     1     0     2   INF
  INF   INF     2     0     2
    1     8   INF     2     0

Final Distance Matrix
======================
    0     6     5     3     1
    6     0     1     3     5
    5     1     0     2     4
    3     3     2     0     2
    1     5     4     2     0

Routing Table for Router 1:
Dest Router      Via             Distance
===========      ========        ========
2                3               6
```

```
3               4               5
4               5               3
5               -               1

Routing Table for Router 2:
Dest Router     Via             Distance
==========      =======         =======
1               3               6
3               -               1
4               3               3
5               4               5

Routing Table for Router 3:
Dest Router     Via             Distance
==========      =======         =======
1               4               5
2               -               1
4               -               2
5               4               4

Routing Table for Router 4:
Dest Router     Via             Distance
==========      =======         =======
1               5               3
2               3               3
3               -               2
5               -               2

Routing Table for Router 5:
Dest Router     Via             Distance
==========      =======         =======
1               -               1
2               4               5
3               4               4
4               -               2

Simulating Link Failure
=======================
Enter Routers to Break Link Between: 1 5

Initial Distance Matrix
======================
    0     7   INF   INF   INF
    7     0     1   INF     8
  INF     1     0     2   INF
  INF   INF     2     0     2
  INF     8   INF     2     0

Final Distance Matrix
======================
    0     7     8    10    12
    7     0     1     3     5
    8     1     0     2     4
   10     3     2     0     2
   12     5     4     2     0

Routing Table for Router 1:
Dest Router     Via             Distance
==========      =======         =======
2               -               7
3               2               8
4               3               10
5               4               12

Routing Table for Router 2:
Dest Router     Via             Distance
==========      =======         =======
1               -               7
```

```
3              -              1
4              3              3
5              4              5

Routing Table for Router 3:
Dest Router      Via          Distance
===========    ========       ========
1              2              8
2              -              1
4              -              2
5              4              4

Routing Table for Router 4:
Dest Router      Via          Distance
===========    ========       ========
1              3              10
2              3              3
3              -              2
5              -              2

Routing Table for Router 5:
Dest Router      Via          Distance
===========    ========       ========
1              4              12
2              4              5
3              4              4
4              -              2
```

# PRACTICAL 5

## Objective

Simulate and implement Dijkstra Algorithm for shortest path routing.

## Code

```
/**
 * Simulate and implement Dijkstra algorithm for
 * shortest path routing.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <cstdio>
#include <climits>
#include <iomanip>
#include <iostream>
#define MAX_NODES 10

using namespace std;

class Graph
{
public:
  int edges;
  int vertices;
  int path[MAX_NODES];
  int distances[MAX_NODES];
  int adjMatrix[MAX_NODES][MAX_NODES];

  void input(int v, int e)
  {
    edges = e;
    vertices = v;

    // initialize the adjacency matrix
    for (int i = 0; i < v; i++)
      for (int j = 0; j < v; j++)
        adjMatrix[i][j] = 0;

    int src, dest, weight;

    // populate the adjacency matrix
    for (int i = 0; i < edges; i++)
    {
      cout << "\nEDGE " << (i + 1)
           << "\n======\n";
      cout << "Enter Source: ";
```

```
    cin >> src;
    cout << "Enter Destination: ";
    cin >> dest;
    cout << "Enter Weight: ";
    cin >> weight;
    adjMatrix[src - 1][dest - 1] = weight;
    adjMatrix[dest - 1][src - 1] = weight;
  }
}

void display()
{
  for (int i = 0; i < vertices; i++)
  {
    for (int j = 0; j < vertices; j++)
      cout << setw(5) << adjMatrix[i][j] << " ";
    cout << endl;
  }
}

void dijkstra(int src)
{
  bool visited[MAX_NODES];

  for (int i = 0; i < vertices; i++)
  {
    visited[i] = false;     // mark node as not processed
    distances[i] = INT_MAX; // set distance from src as infinity
  }

  // mark the src node
  path[src] = -1;
  distances[src] = 0;

  // iterate over all vertices
  for (int i = 0; i < vertices - 1; i++)
  {
    // find the nearest unprocessed node
    int u = minDistance(visited);
    // mark node as processed
    visited[u] = true;
    // iterate over all nodes
    for (int v = 0; v < vertices; v++)
      // update distance for unprocessed node if there
      // exists an edge(u,v) and new distance is lesser
      // also add the node to the shortest path
      if (visited[v] == false &&
          adjMatrix[u][v] &&
```

```
            distances[u] != INT_MAX &&
            distances[u] + adjMatrix[u][v] < distances[v])
        {
          path[v] = u;
          distances[v] = distances[u] + adjMatrix[u][v];
        }
    }

    // print distances and shortest paths
    cout << "\nDest Node \t Distance \t Shortest Path";
    cout << "\n========= \t ======== \t ============";
    for (int i = 0; i < vertices; i++)
    {
      cout << endl
           << (i + 1)
           << " \t\t " << distances[i]
           << " \t\t " << (src + 1);
      printShortestPath(i);
    }
  }

  int minDistance(bool *visited)
  {
    int min = INT_MAX, min_index;
    for (int v = 0; v < vertices; v++)
      if (visited[v] == false &&
          distances[v] <= min)
      {
        min = distances[v];
        min_index = v;
      }
    return min_index;
  }

  void printShortestPath(int node)
  {
    if (path[node] == -1)
      return;
    printShortestPath(path[node]);
    cout << " -> " << (node + 1);
  }
};

int main()
{
  int v, e;
  Graph graph;
```

```
  cout << "Enter No. of Nodes: ";
  cin >> v;
  cout << "Enter No. of Edges: ";
  cin >> e;

  graph.input(v, e);

  cout << "\nGRAPH\n=====\n";
  graph.display();
  cout << endl;

  cout << "Enter Source Node: ";
  cin >> v;

  graph.dijkstra(v - 1);

  return 0;
}
```

## Output

```
Enter No. of Nodes: 5
Enter No. of Edges: 7

EDGE 1
======
Enter Source: 1
Enter Destination: 2
Enter Weight: 10

EDGE 2
======
Enter Source: 1
Enter Destination: 3
Enter Weight: 2

EDGE 3
======
Enter Source: 1
Enter Destination: 5
Enter Weight: 100

EDGE 4
======
Enter Source: 2
Enter Destination: 4
Enter Weight: 3

EDGE 5
======
Enter Source: 3
Enter Destination: 2
Enter Weight: 5
```

```
EDGE 6
======
Enter Source: 4
Enter Destination: 3
Enter Weight: 15

EDGE 7
======
Enter Source: 4
Enter Destination: 5
Enter Weight: 5

GRAPH
=====
    0    10     2     0   100
   10     0     5     3     0
    2     5     0    15     0
    0     3    15     0     5
  100     0     0     5     0

Enter Source Node: 1

Dest Node        Distance        Shortest Path
=========        ========        =============
1                0               1
2                7               1 → 3 → 2
3                2               1 → 3
4                10              1 → 3 → 2 → 4
5                15              1 → 3 → 2 → 4 → 5
```

# THEORY 1

## Question

The maximum number of frames that may be outstanding at any instant is not the same as the size of the sequence number space in Go-Back-N. Why?

## Answer

- Consider frames/packets ~~to~~ be identified by n-bit sequence numbers. There can be $2^n$ distinct sequence numbers $\{0, 1, \ldots, 2^n - 1\}$. This is referred to as the sequence number space.
- The term outstanding frame refers to a frame whose acknowledgment has not yet been received, wherein maybe the forward channel is more reliable than the reverse channel from the receiver to the sender.
- The GO-Back-N sliding window protocol allows the sender to send N outstanding frames before receiving acknowledgements (ACKs). The receiver is allowed to buffer a single frame.
- Consider the case if 2-bit sequence numbers are used. Allowed sequence nos. = $\{0, 1, 2, 3\}$

Case(1): Maximum number of outstanding frame is the same as the size of the sequence no. space.

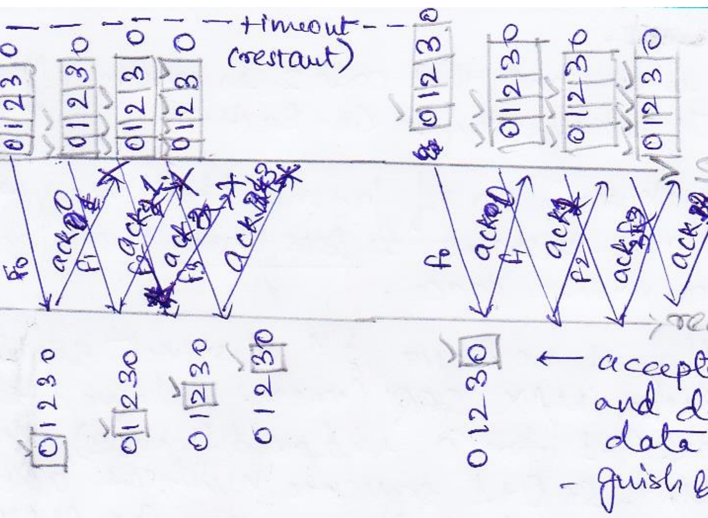
i.e. max. no. of outstanding frames = sender window size $= 4 = 2^2 = 2^n$ where $n = 2$

(a) The sender sends 4 frames without worrying about their acknowledgement. Suppose that the ACKs are not received or the frames are lost in transit.

(b) After some time, the frames timeout and the sender resends the 4 frames. If the receiver had already received the frames, this was a spurious re-transmission. However, the receiver will not be able to distinguish the duplicate frames as it expects to receive ~~packet~~ the frames with same sequence nos in the next cycle. The erroneous, duplicate frame is ~~(c)~~ accepted and passed on to the next layer.

(c) In case the last frame was successfully received and ACK3 was sent, and ACKs for the other packets are not received, then the sender resends all the 4 frames only to receive an ACK3. There is no way to determine if all the frames were successfully received or lost and in which specific batch.

If initially all frames were lost due to ACK not received




← accepted erroneously and delivered as new data, fails to distinguish b/w old & new pkt (frame)

Case (2): Maximum no. of outstanding frames is lesser than the size of the sequence number space.

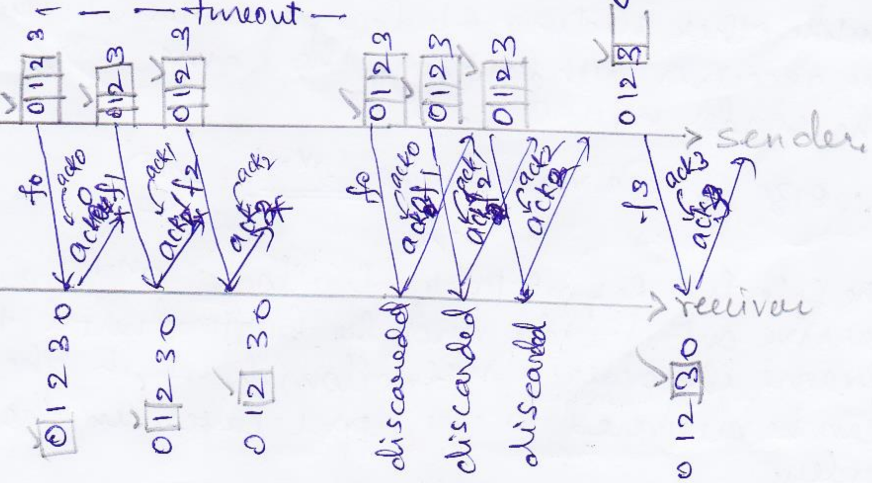
i.e. suppose max no. of outstanding frames < sender window size ($= 4 = 2^n$ where $n = 2$)

and say max no of outstanding frames is one less than 4 i.e. 3 or $2^n - 1$.

(a) The sender sends 3 frames without worrying about acknowledgements. Suppose they are lost or the ACKs are not received.

(b) After some time, the timer times out and the frames 0, 1, 2 are resent. The receiver expects frame 3 to be sent and can correctly discard the duplicate frames and send back ACKs as necessary.

(c) The sender can then comfortably 'slide' the window.



Therefore, it is evident that case 2 is more efficient and expected behaviour for Go-Back-N.

Hence, the maximum no. of frames that may be outstanding at any instant is <u>less than</u> the size of the sequence number space.

conventionally if we have $2^n$ distinct sequence numbers where MAX_SEQ (max. allowed sequence number) is $2^n - 1$ (for 0-indexed frames), then we have MAX_SEQ+1 sequence numbers and thus max. no. of outstanding frames can be MAX_SEQ in the Go-Back-N protocol.

# THEORY 2

## Question

What is the size of sending window and receiving window in Selective Repeat? Give reason of choosing the window size.

## Answer

In the Selective Repeat Sliding window protocol, the size of the sender window is $\frac{(MAX\_SEQ + 1)}{2}$ where MAX_SEQ refers to the maximum allowed sequence number for frames that can be identified using n-bits. This simplifies the expression as follows.

As $MAX\_SEQ = 2^n - 1$,

$$\therefore \text{size of sender window} = \frac{(2^n - 1) + 1}{2} = \frac{2^n}{2}$$

$$= \underline{\underline{2^{n-1}}} \quad -(1)$$

Unlike other sliding window protocoles like GoBackN, the receiver window is of the same size as the sender window.

$$\therefore \text{size of receiver window} = \underline{\underline{2^{n-1}}} \quad -(2)$$

The Selective Repeat protocol allows $2^{n-1}$ frames to arrive out-of-order and be buffered until the next frame in-order arrives following which the frames can be delivered to the upper layer in the correct order.

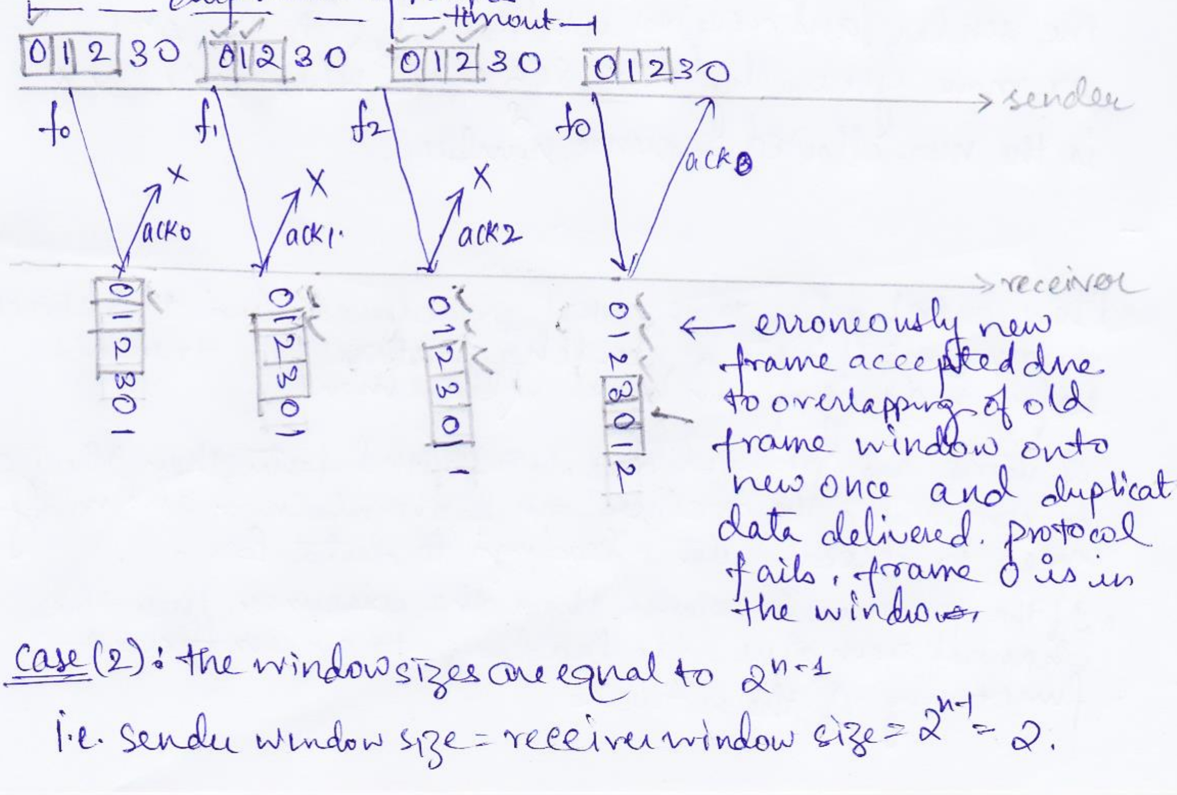
Consider 2-bit sequence numbers.
Now Allowed sequence numbers = {0, 1, 2, 3}
~~Therefore~~ the sender and receiver window sizes must be equal. assuming two different cases,

<u>case (1)</u>: the window sizes are greater than $2^{n-1}$

say sender window size = receiver window size $= (2^{n-1}) + 1$
$= 2^{2-1} + 1 = 2^1 + 1 = 2+1$
$= 3$

(a) sender sends 3 frames without worrying about their acknowledgements. consider these frames are lost or ACKs are not received.

(b) The receiver already moves the window in case the frame was actually not lost but ACK was not received by sender. After time out, sender resends the frames. This overlaps with the valid sequence number space for the next cycle and results in erroneous delivery of duplicate frames.

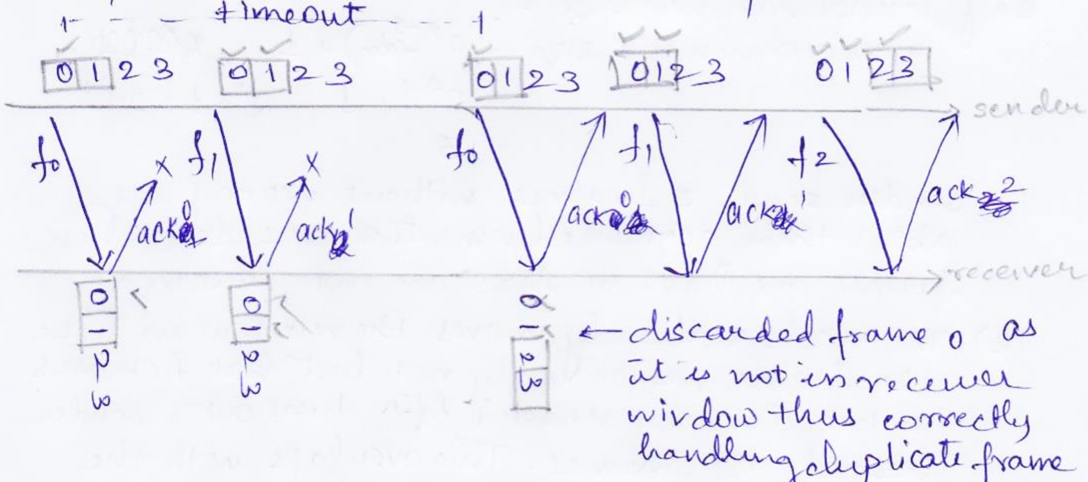



<u>case (2)</u>: the window sizes are equal to $2^{n-1}$

i.e. sender window size = receiver window size $= 2^{n-1} = 2$.

(a) sender sends 2 frames without worrying about acknowledgements. consider these are lost or ACKs are not received.

(b) In case frames are not actually lost but ACKs are not received, the sender resends the same window frames. the receiver does not expect ~~the pro some~~ the frames as it has already moved its window. so, these are discarded as is expected.




Hence, it is evident that the window size that we choose must not be greater than $2^{n-1}$ therefore, we chose the sender and receiver window sizes to be $2^{n-1}$ or more generally $\frac{(MAX\_SEQ+1)}{2}$ where MAX_SEQ is the max. allowed sequence number.

# THEORY 3

## Question

How does the sliding window protocol take care of flow control in the network? Explain using the Go-Back-N protocol.

## Answer

- Flow control refers to the set of procedures used to restrict/coordinate the amount of data a sending station can transmit before waiting for an acknowledgement.
- If items are produced and transmitted faster than they can be consumed, the receiver can be overwhelmed and may need to discard items leading to data loss.
- If the producer is slower than the consumer, then the channel remains idle leading to an inefficient functioning of the network.

- Sliding window ~~alto~~ protocols allow multiple frames/packets to be "in flight" at the same time using allowed sequence numbers (n-bit) to identify these entities.
- In a network, the sliding window protocols place a buffer on each station - the sender and receiver both. The data received in the network is stored in the buffer from where the frames/packets can be sent or be stored in, for other layers to read it. A window is the amount of data that can be read from such buffers.
- Frames and packets need to be in accordance ~~to~~ with the window. At the sender side, frames in a certain range (called the sender window) are allowed to be sent.
- The receiver occasionally sends acknowledgements for the frames/packets received, following the receipt of which, both stations adjust their windows before going to the next chunk of data.

- Flow control using Go Back-N Protocol.

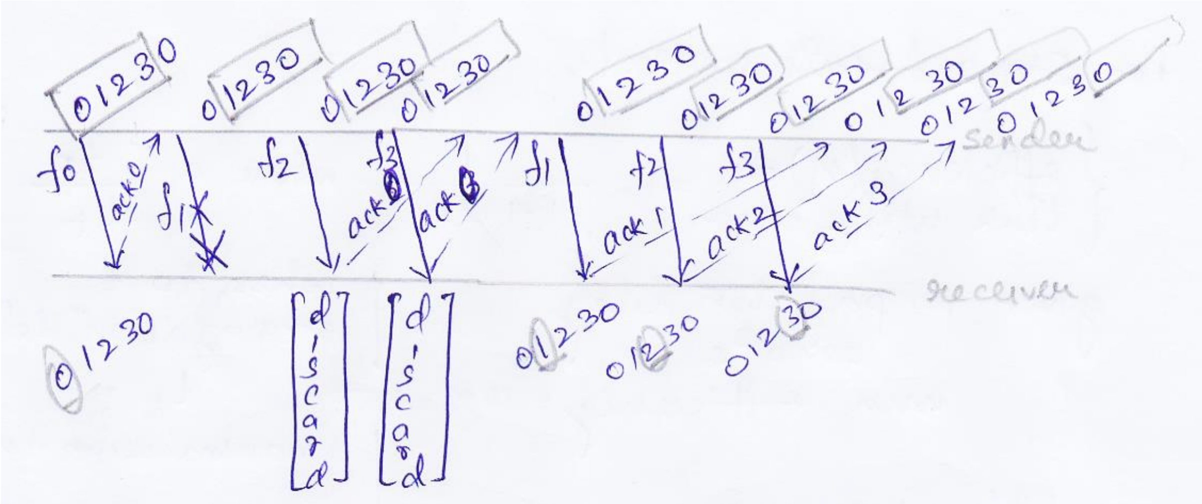
  In the GOBackN sliding window protocol, the sender can transmit $2^n-1$ frames/packets before receiving acknowledgements, but, the receiver can only keep one entity in buffer. The sender keeps unacknowledged entities in its buffer until acknowledgements arrive.

Suppose the sender window size is 4, frames numbered 0, 1, 2, 3 are kept in flight. assume frame 1 is lost in transit.

→ frames following the errored / lost frame are discarded. The windows do not move.

→ ACKs are sent for the last frame correctly received.

→ After the expected frame is recent, the receiver slides its window and sends an acknowledgment following the receipt of which, the sender also moves the window.



# THEORY 4

## Question

A channel has a bit rate of 4 kbps and a propagation delay of 20 ms. For what range of frame sizes does stop-and-wait give an efficiency of at least 50 percent?

## Answer

for Stop-and-wait protocol,

$$\left\{ \text{efficiency}(\eta) \text{ or link utilization} = \frac{1}{1+2a} \quad \text{where } a = \frac{t_p}{t_t} \right\}$$

where $t_p$ = propagation delay

$t_t$ = transmission time

given propagation delay $= 20 \times 10^{-3}$ s

min. efficiency $= 50\% = 0.5$

Now,

$$\frac{1}{2} \leq \frac{1}{1+2\left(\frac{20\times10^{-3}}{t_t}\right)}$$

$$\Rightarrow \frac{1}{2} \leq \frac{t_t}{t_t + (40\times10^{-3})}$$

$$\Rightarrow t_t + (40\times10^{-3}) \leq 2t_t$$

$$\Rightarrow t_t \geq 40\text{ms}$$

Now,

$$\left\{ \text{transmission time} = \frac{\text{message size}}{(\text{Bandwidth of channel})} \right\}$$

$$\Rightarrow 40\times10^{-3} = \frac{f_s}{4\times10^{3}}$$

$$\Rightarrow f_s = 160 \text{ bits}$$

for $t_t = 40$ ms, $f_s = 160$ bits

for atleast 50% efficiency,

$\boxed{t_t \geq 40\text{ms}}$ and $\boxed{\text{frame size} \geq 160 \text{ bits}}$

$\therefore$ The frame size should be greater or equal to 160 bits