§4 Заключительный этап: командная часть

Участники решают задачу безопасной передачи данных в канале с помехами и низкой скоростью передачи данных. Каждой команде необходимо разобраться с форматом передаваемых файлов, изучить свойства канала передачи данных, и способы восстановления поврежденной информации, научиться добавлять необходимые элементы конструкции в цифровой образ детали.

4.1 Описание стенда

Стенд-тренажер «Технологии беспроводной связи» представляет собой систему, предназначенную для изучения и отработки навыков работы с передачей данных в низкоскоростных, зашумленных каналах связи и использования помехозащитных кодов с исправлением ошибок, ознакомления с форматами данных и принципами организации аддитивного производства.

В основу стенда-тренажера входят: передающий и приемный комплексы. Передающий комплекс состоит из планшетного сканера и персонального компьютера, приемный комплекс – из компьютера и 3D-принтера MakerBot Replicator+. Координирует работу комплекса центральный сервер. В качестве объектов для передачи и печати используется макет коробки передач, состоящий из 6 шестеренок разного диаметра.

Макет коробки передач и 3D-принтер MakerBot Replicator+

На компьютерах стенда установлено следующее программное обеспечение (ПО): операционная система Ubuntu 16.04, компиляторы gcc и g++, интерпретаторы python, perl с основными библиотеками.

Эмуляция низкоскоростного, зашумленного канала связи сделана в стенде на основе протокола HTTP. Для приема и зашумления файла используется отдельный компьютер-сервер с установленными: веб-сервером Apache2, интерпретатором языка php и базой данных Mysql. Для отправки файла – любой компьютер с веб-браузером, в данном случае Firefox.

Общий принцип работы стенда-тренажера

Работа на стенде состоит из следующих этапов:

- выбранная шестеренка сканируется в планшетном сканере на передающем конце стенда;
- с помощью специального программного обеспечения, образ переводится в текстовый формат;
- текстовый файл передается на сервер по низкоскоростному, зашумленному каналу связи;
- на сервере этот файл переводится в формат, пригодный для печати на 3D-принтере;
- печать объекта на 3D-принтере.

Основные, затратные по времени операции на стенде — это передача файлов по каналу связи и печать шестеренки на 3D-принтере, для оценки в таблице 1 для данных операций приведены требуемые затраты времени.

№ шестеренки	Диаметр (мм)	Объем образа (МБ)	Время на передачу образа шестеренки (чч:мм)	Ориентировочное время печати (чч:мм)
1	110	30	01:00	2:10

2	89	20	00:40	1:40
3	78	15,5	00:31	1:10
4	74	13,5	00:27	1:07
5	65	11,5	00:23	0:57
6	63	10	00:20	0:51
	479	**100,5**	**03:21**	**7:55**

Таблица 1. Временные затраты при работе на стенде

4.2 Задание

Цель работы: Создать рабочую копию механизма средствами аддитивного производства с использованием технологий беспроводной связи в распределенных физико-технических системах.

Задача: Оцифровать шестеренки, передать полученные файлы в канале с помехами и ограничениями по скорости и распечатать на приемном конце максимальное количество шестеренок из макета коробки передач за отведенное на работу время.

Макет коробки передач с эталонными шестеренками

Макет коробки передач с 3D-печатными копиями шестеренок

4.3 Порядок работы команд на стенде

Все участники объединяются в 3 команды, в каждой команде по 5 – 7 человек.

В своем распоряжении команды имеют одинаковые объекты для оцифровки, передачи и печати - рабочий макет коробки передач, состоящий из 6 шестеренок разного диаметра; ноутбуки с предустановленным ПО и одинаковые 3D-принтеры MakerBot Replicator+ с разным цветом пластика для каждой команды.

Все программное обеспечение, разработанное командами в процессе решения задачи, должно предоставляться в виде исходных текстов с последующей компиляцией и запуском на компьютерах стенда. Команды могут пользоваться сторонним программным обеспечением.

Этапы работы на стенде:

Блок-схема работы на стенде «Технологии беспроводной связи»

1. **Выбор шестеренки.** Команда выбирает любую шестеренку и передает ее организаторам.

2. **Сканирование**. В результате сканирования шестеренки создается ее графический образ с разрешением 50 точек на миллиметр, который далее переводится в текстовый файл, сохраняющий образ шестеренки в виде последовательности символов '0' и '1'. В первой строке файла перечисляются: разрешение (кол-во точек на миллиметр), кол-во столбцов, кол-во строк, толщина шестеренки в мм. Время сканирования и создание текстового файла составляет около 1 минуты, все команды пользуются одним сканером. Полученный текстовый файл передается команде на usb-флешке для

последующего анализа.
Цифровой образ шестеренок с номерами: 2, 4, 5 выдается командам с заполненным отверстием под вал! Для данных шестеренок командам требуется самостоятельно сформировать в цифровом образе отверстие под вал.

3. **Сжатие образа и защита от помех.** На данном этапе команды самостоятельно исследуют возможности более компактного представления цифрового образа шестеренки, продумывают стратегию защиты от помех, разрабатывают соответствующее программное обеспечение. В результате подготавливается файл, который команда на флешке предоставляет организаторам.

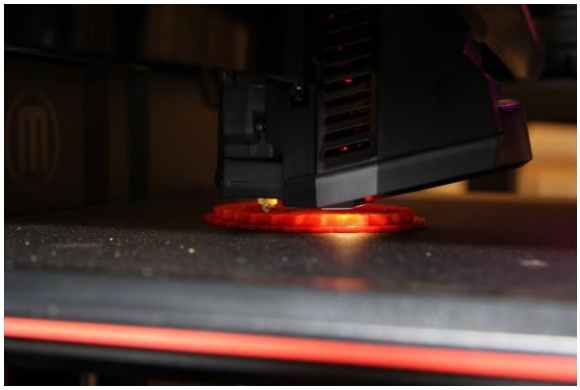
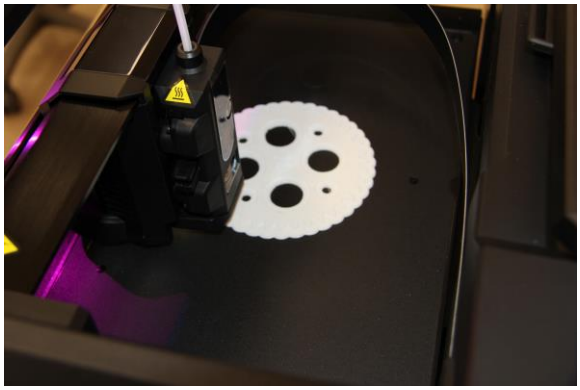
4. **Передача данных по каналу связи.** Организаторы передают полученный файл по медленному каналу связи, в котором имитируются шумы и помехи. Время, затраченное на передачу данных, можно оценить из таблицы 1, причем приведенные значения справедливы в случае, если канал связи использует одна команда, в случае, когда канал связи используется несколькими командами, скорость падает пропорционально количеству пользователей. После завершения передачи данных команде на флешку копируется полученный файл с ошибками.

5. **Анализ сбоев и исправление ошибок.** Команда проводит анализ свойств шума в канале передачи данных. На основе проведенного анализа команды разрабатывают/дорабатывают методики и алгоритмы, позволяющие исправить сбои в данных, разрабатывают соответствующее программное обеспечение, дорабатывают методики и алгоритмы созданные на этапе 3.

6. **Создание 3D модели шестеренки.** Команды передают организаторам на флешке программное обеспечение, созданное на этапе 5, которое проверяется и запускается для обработки файла со сбоями, полученного на этапе 4, в результате должен создаться корректный цифровой образ шестеренки, причем формат восстановленного файла должен соответствовать формату, описанному в пункте 2, разрешение менять не разрешается. Имя файла может содержать только латинские буквы.

7. **Предварительный контроль восстановленного образа**. На данном этапе проводится контроль разработанного образа, если образ содержит не устраненные ошибки, либо не соответствует требуемому формату (пункт 2), тогда команда возвращается работать над ошибками. Если файл соответствует формальным требованиям, тогда строиться 3D образ шестеренки в формате STL, с которым работает 3D принтер, в случае если образ не соответствует номеру шестеренки, с которой работает команда, файл возвращается для работы над ошибками.

8. **Печать шестеренки на 3D принтере**. Прошедший предварительный контроль файл отправляется на печать, время печати, в зависимости от диаметра шестеренки занимает от 50 минут до 2 часов. Если во время печати команду не устраивает получаемый результат, то организаторы могут остановить процесс печати. Если шестеренка не успевает намечаться до окончания рабочего дня, печать не останавливается, но на следующий день команда имеет меньшее время для работы с принтером ровно на тот интервал, которого им не хватило в предыдущий день.

Печать шестеренок на 3D принтере

9. **Итоговый контроль**. Напечатанная шестеренка устанавливается на стенд (незначительные дефекты разрешается исправлять надфилем) взамен эталонной. Если ее невозможно насадить на вал или она с него сваливается или конструкция не прокручивается минимум на 2 оборота, команды возвращаются продолжать работать с цифровым образом шестеренки. Если напечатанная шестеренка выдержала все испытания, она остается на стенде и команда может взять следующую, т.е. переходит к пункту 1.

4.4 Решение

Для решения поставленной задачи командам следовало:

1. Изучить свойства шума и характер помех в канале.
2. Сжать файл. Можно написать собственный алгоритм сжатия или воспользоваться любым архиватором (zip, rar, 7zip и тд.). Это необходимо для уменьшения времени передачи.
3. Защитить архив восстанавливающим кодом. Можно использовать готовую библиотеку.
4. Передать файл.
5. После передачи файла необходимо избавиться от последствий воздействия периодичного шума. Для этого нужно провести инверсию каждого второго блока по 1024 байт – данные о размере блока и характере шума получены на этапе 1.
6. Следующим этапом нужно избавиться от случайных ошибок. Для этого проводится декодирование восстанавливающего кода.
7. Заключительный этап - разархивация переданного файла.

Ниже приведен пример написанной на языке C утилиты, использующей библиотеку кода Рида-Соломона.

```
#include "rscode.h"
#include "ecc.h"

int main(int argc, char *argv[])
{
    FILE *fp_in, *fp_out;
    unsigned char msg[MSG_LEN];
    unsigned char buff[BUFF_LEN];
    int i, ret, mode, inv, fl_inv, idx_start, idx_end;
    char fname_in[PATH_MAX], fname_out[PATH_MAX];

    if(argc != 4) {
```

```
      printf("Usage:\n%s   -e|-d   in_file   out_file\n\t-e   -   encode
data\n\t-d - decode data\n", argv[0]);
      return 1;
       }
       strcpy(fname_in, argv[2]);
       strcpy(fname_out, argv[3]);
       mode = 0;
       if(strncmp(argv[1], "-e", 2) == 0)
      mode = MODE_ENC;
       if(strncmp(argv[1], "-d", 2) == 0)
      mode = MODE_DEC;
       if(mode == 0) {
      printf("Error! Invalid parameter.\n");
      return 1;
       }

       printf("NPAR=%d   BUFF_LEN=%d   MSG_LEN=%d\n",   NPAR,   BUFF_LEN,
MSG_LEN);

       initialize_ecc();


       fp_in = fopen(fname_in, "rb");
       if(!fp_in) {
      printf("Error! Can't open file for read: %s\n", fname_in);
      return 1;
       }
       fp_out = fopen(fname_out, "wb");
       if(!fp_out) {
      printf("Error! Can't open file for write: %s\n", fname_out);
      return 1;
       }

       if(mode == MODE_ENC) {
      do {
          ret = fread(msg, 1, MSG_LEN, fp_in);
          encode_data(msg, ret, buff);
          fwrite(buff, 1, ret+NPAR, fp_out);
      } while(ret == MSG_LEN);
       }
       if(mode == MODE_DEC) {
      inv = 0;
      fl_inv = FALSE;
      do {
          ret = fread(buff, 1, BUFF_LEN, fp_in);
          inv += ret;
          if(fl_inv) {
            idx_start = 0;
            idx_end = ret;
          }
          else {
            idx_start = 0;
            idx_end = 0;
          }
          if(inv >= INV_LEN) {
            inv -= INV_LEN;
            if(!fl_inv) {
                idx_start = ret - inv;
```

```
            idx_end = ret;
            fl_inv = TRUE;
          }
          else {
            idx_start = 0;
            idx_end = ret - inv;
            fl_inv = FALSE;
          }
        }
        if(idx_start > 0 || idx_end > 0) {
          for(i=idx_start; i<idx_end; i++) {
            buff[i] = ~buff[i];
          }
        }

        decode_data(buff, ret);
        if(check_syndrome() != 0) {
          printf("Some error(s)!\n");
          correct_errors_erasures(buff, ret, 0, NULL);
          fwrite(buff, 1, ret-NPAR, fp_out);
        }
        else {
          fwrite(buff, 1, ret-NPAR, fp_out);
        }
    } while(ret == BUFF_LEN);
    }

    fclose(fp_in);
    fclose(fp_out);

    return 0;
}
```