

Parse Guide

I nuläget samlas data in från en simulering genom att omvandla .pcap filerna som skapas för varje nod till .txt filer och därefter plockas information från dessa med hjälp av jämförelsesträngar och sparas i .dat filer. Klassen NSPlot.cc skickar sen dessa .dat filer till Gnuplot för att rita upp grafer.



I nuläget går man dock endast igenom .pcap filen från server-noden, vilket man kan se i koden som ligger sist i **RunSimulation()**:

```
220 // Parse packet capture file
221 std::string server_pcap = output_filename + "-0-0";
222 std::string simtotal = sim_settings.output_dir + protocol_name_str + "_simtotal";
223 cout << left << setw(28) << "Protocol totals file:" << simtotal << endl;
224 cout << left << setw(28) << "Server pcap:" << server_pcap << ".pcap" << endl;
225 start_data_parser(protocol_name_str, sim_settings.number_of_clients, sim_settings.transfer_data_bytes * sim_settings.num_cycles, sim_settings.num_sockets_streams, server_pcap, simtotal, "-print");
```

Efter varje simulering skapar man alltså två strängar: **server_pcap** som håller namnet på servers .pcap fil som man kan se från nod numret "-0-0" som läggs till på slutet samt **simtotal** som håller namnet på .dat filen för slutgiltiga datan. Efter att man printat filnamnen till konsolen kallar man till sist på **start_data_parser()** från DataParser.cc för att börja processen.

```
399 void start_data_parser(string protocol, int numClients, int dataBytesPerClient, int numberOfStreamsPerClient, string sourceFile, string summaryTargetFile, string print) {
400     string convert = "tshark -V -r " + sourceFile + ".pcap > " + sourceFile + ".txt"; tshark kommando
401     system(convert.c_str());
402
403     string targetFile = sourceFile + "-parse.txt";
404     summaryTargetFile += ".dat"; output .dat
405     sourceFile += ".txt"; input .txt
406     int expectedData = numClients * dataBytesPerClient;
407
408     DataParser parser(sourceFile, protocol, targetFile, summaryTargetFile, print, expectedData, numClients, numberOfStreamsPerClient);
409     parser.GUI();
410     parser.packetExtractor();
411 }
```

Här skapar man först en sträng med ett tshark kommando (tshark är wireshark för command line interface) som omvandlar .pcap filen till en .txt fil och kör sedan kommandot genom att kalla på **system()**.

För att få en känsla av hur .txt filen är formaterad körde jag kommandot på .pcap filen för server-noden från tutorial filen second.cc och början av output kan ses i bilden på nästa sida. Här kan det finnas möjlighet att använda andra filter-flags då man kör kommandot eller alternativt använda något annat än tshark för att parse .pcap filerna på ett bättre sätt och att bara få ut den information vi är intresserad av och kanske i ett mer lätthanterat format.

En nackdel med nuvarande lösning är användningen av jämförelsesträngar för att lyfta ut information från .txt filerna vilket är långsamt och kan skapa problem om man även måste gå igenom .pcap filerna för klient-noderna för att få den information vi behöver. En alternativ lösning kan vara att använda ett program som **libpcap** för att parse .pcap filerna.

Efter att ha kört kommandot så .txt filen skapats gör man två nya strängar: **summaryTargetFile** som bara är simtotal med ".dat" ändelse och **sourceFile** som är server_pcap med ".txt" ändelse så den hänvisar till den nyskapade text filen. Det skapas även en tredje sträng **targetFile** med "-parse.txt" på slutet men denna fil verkar aldrig användas och tas bort längre fram i processen.

Output från kommandot: `tshark -V -r sourceFile.pcap > sourceFile.txt`

```
Frame 1: 1054 bytes on wire (8432 bits), 1054 bytes captured (8432 bits)
  Encapsulation type: PPP (4)
  Arrival Time: Dec 31, 1969 16:00:02.000000000 PST
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 2.000000000 seconds
  [Time delta from previous captured frame: 0.000000000 seconds]
  [Time delta from previous displayed frame: 0.000000000 seconds]
  [Time since reference or first frame: 0.000000000 seconds]
  Frame Number: 1
  Frame Length: 1054 bytes (8432 bits)
  Capture Length: 1054 bytes (8432 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: ppp:ip:udp:data]
Point-to-Point Protocol
  Protocol: Internet Protocol version 4 (0x0021)
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 10.1.2.101
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
  .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport
  Total Length: 1052
  Identification: 0x0000 (0)
  Flags: 0x00
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    --0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 64
  Protocol: UDP (17)
  Header checksum: 0x0000 [validation disabled]
    [Good: False]
    [Bad: False]
  Source: 10.1.1.1
  Destination: 10.1.2.101
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
User Datagram Protocol, Src Port: 49153 (49153), Dst Port: 9 (9)
  Source Port: 49153
  Destination Port: 9
  Length: 1032
  Checksum: 0x0000 (none)
    [Good Checksum: False]
    [Bad Checksum: False]
  [Stream index: 0]
Data (1024 bytes)
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0280 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0290 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
02f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0300 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0310 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0330 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0340 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0350 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0370 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0380 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0390 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
03f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Data: 00000000000000000000000000000000000000000000...
[Length: 1024]

Frame 2: 1054 bytes on wire (8432 bits), 1054 bytes captured (8432 bits)
  Encapsulation type: PPP (4)
  Arrival Time: Dec 31, 1969 16:00:02.017607000 PST
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 2.017607000 seconds
```

Sen skapas ett DataParser objekt. Konstruktorn för DataParser initierar bara några variabler:

```
21 DataParser::DataParser(string sourceFile, string protocol1, string targetFile, string summaryTargetFile, string print, int expectedData1, int numClients1, int numberOfStreamsPerClient1) {
22
23     //Assign values to the variables within the object
24     filename = sourceFile;
25     protocol = protocol1;
26     file_out = targetFile;
27     sum_file_out = summaryTargetFile;
28
29     timeFix = -1;
30     totalTime = 0;
31     totalUsefulData = 0;
32     totalData = 0;
33     dataChunks = 0;
34     packetCounter = 0;
35
36     expectedData = expectedData1;
37     numberOfStreamsPerClient = numberOfStreamsPerClient1;
38     numClients = numClients1;
39
40     //Handle the printflag
41     if (print == "-printall"){
42         PRINT_PACKET = true;
43         PRINT_ALL = true;
44     }
45     else if (print == "-print") {
46         PRINT_PACKET = false;
47         PRINT_ALL = true;
48     }
49     else if (print == "-noprint") {
50         PRINT_PACKET = false;
51         PRINT_ALL = false;
52     }
53     else {
54         cout << "Argument failure" << "\n";
55         exit(1);
56     }
57 }
```

Och **GUI()** som kallas efteråt skriver ut ett simpelt GUI:

```
60 void DataParser::GUI() {
61     cout << GUI_line() << "\n";
62     cout << "Data Parser for .pcap files (Version 2.0)" << "\n";
63     cout << left << setw(28) << "Input file: " << filename << "\n";
64     cout << left << setw(28) << "Output file: " << sum_file_out << "\n";
65     cout << left << setw(28) << "Protocol: " << protocol << "\n";
66     if (PRINT_PACKET == true) {
67         cout << GUI_line() << "\n";
68         cout << left << setw(7) << "Frame";
69         cout << left << setw(11) << "Sender";
70         cout << left << setw(11) << "Receiver";
71         cout << left << setw(10) << "Time [s]";
72         cout << left << setw(10) << "TTL [ms]";
73         cout << left << setw(12) << "Length [b]";
74         cout << left << setw(12) << "Data [b]";
75         cout << left << setw(10) << "Chunks" << endl;
76         cout << GUI_line() << "\n";
77     }
78 }
```

Därefter kallas **packetExtractor()** som letar upp raden där ny paketinformation börjar:

```
89 //Searches for packets within a .pcap file
90 void DataParser::packetExtractor() {
91     int current_frame = 1;
92     ifstream reader(filename.c_str()); Öppna input .txt fil
93     remove(file_out.c_str());
94     string line;
95     string arr[NUM_OF_WORDS];
96
97     if(reader.is_open()){
98
99         while((getline(reader, line))) { Ta u nästa rad
100             string frame = to_string(current_frame) + ":";
101             stringstream str(line);
102             int i = 0;
103             Ta ut strängar på rad while (str.good() && i < NUM_OF_WORDS) {
104                 str >> arr[i];
105                 i++;
106             }
107             Nytt paket på raden? if(arr[0] == "Frame" && arr[1] == frame) {
108                 dataExtractor(reader, current_frame);
109                 current_frame++;
110             }
111         }
112     }else {
113         cout << filename << " could not be opened" << "\n";
114         exit(1);
115     }
116     cout << GUI_line() << "\n";
117
118     insertTotalData(); Skriver till .dat efter all
119                        data samlats in
120 }
```

Här ser man att .txt filen som skapades med tshark kommandot öppnas och sedan läser man en rad i taget och ser om dom två första strängarna i den är "Frame" samt "Paket#:" vilket man kan se tyder på att ny paketinformation börjar där:

```
Frame 1: 1054 bytes on wire (8432 bits), 1054 bytes captured (8432 bits)
Encapsulation type: PPP (4)
Arrival Time: Dec 31, 1969 16:00:02.000000000 PST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 2.000000000 seconds
```

Här ser man också att **remove()** kallas för att ta bort text filen med "-parse.txt" på slutet utan att den använts.

Efter att man hittat ett paket kallar man på **dataExtractor()** som på samma sätt går igenom efterföljande rader och använder jämförelsesträngar för att identifiera vilken information raden innehåller och sen har man enskilda funktioner för att plocka ut just den datan:

```
122 //Extracts internal parameters from packets
123 void DataParser::dataExtractor(ifstream& reader, int frame) {
124     string line;
125     string arr[NUM_OF_WORDS];
126
127     bool CORRECT_PROTOCOL = true;
128     bool TTL_GET, HEARTBEAT = false;
129
130     double frame_length, epoch_time, time_to_live;
131     int data_length = 0, data_chunk_count = 0;
132     string sender, reciever;
133
134
135     while((getline(reader, line))) { Ta ut nästa rad
136         if(line == "") {
137             break;
138         }
139         if (CORRECT_PROTOCOL == false) {
140             return;
141         }
142         if (HEARTBEAT == true) {
143             return;
144         }
145
146         int words = 0;
147         stringstream str(line);
148         while (str.good() && words < NUM_OF_WORDS) { Ta ut strängarna i raden
149             str >> arr[words];
150             words++;
151         }
152
153         Jämförelsesträngar för att identifiera info på raden
154         if ((arr[0] + " " + arr[1]) == "HEARTBEAT chunk") {
155             HEARTBEAT = getHeartBeat(arr, frame);
156         }
157         else if ((arr[0] + " " + arr[1]) == "HEARTBEAT_ACK chunk") {
158             HEARTBEAT = getHeartBeat(arr, frame);
159         }
160         else if((arr[0] + " " + arr[1] + " " + arr[2]) == "[Protocols in frame:]") {
161             CORRECT_PROTOCOL = getPacketProtocol(arr, frame);
162         }
163         else if ((arr[0] + " " + arr[1]) == "Epoch Time:") {
164             epoch_time = getEpochTime(arr);
165         }
166         else if((arr[0] + " " + arr[1]) == "Frame Length:") {
167             frame_length = getPacketLength(arr);
168         }
169         else if(arr[0] == "Source:") {
170             sender = getSender(arr);
171         }
172         else if(arr[0] == "Destination:") {
173             reciever = getReciever(arr);
174         }
175     }
```

När man tagit ut datan från alla paket kallas **insertTotalData()** för att skriva informationen till .dat filen. Varje simulering leder i nuläget alltså till en enda rad med information i .dat filen:

```
324 void DataParser::insertTotalData() {
325     float dataPercentage = totalUsefulData / totalData * 100;
326     double speed = totalData / totalTime / (1024*1024);
327     double dataChunkAvg = totalUsefulData / dataChunks;
328     double frameSizeAvg = totalData/packetCounter;
329     double dataLossPercent = 100 - (totalUsefulData / expectedData * 100);
330     expectedData = expectedData / (1024*1024);
331     totalData = totalData / (1024*1024);
332     totalUsefulData = totalUsefulData / (1024*1024);
333
334     ofstream myfile;
335     myfile.open(sum_file_out.c_str(), ios::app);
336     if(myfile.is_open()){
337         if(protocol == "sctp"){
338             myfile << packetCounter << " " << totalTime << " " << totalData << " "
339             << totalUsefulData << " " << expectedData << " " << dataPercentage << " " << dataLossPercent << " " << speed << " "
340             << frameSizeAvg << " " << dataChunks << " " << dataChunkAvg << " " << numClients << " " << numberOfStreamsPerClient << "\n";
341         }else{
342             myfile << packetCounter << " " << totalTime << " " << totalData << " "
343             << totalUsefulData << " " << expectedData << " " << dataPercentage << " " << dataLossPercent << " " << speed << " "
344             << frameSizeAvg << " " << "NaN" << " " << "NaN" << " " << numClients << " " << numberOfStreamsPerClient << "\n";
345         }
346     }
347     myfile.close();
348
349     if (PRINT_ALL == true) {
350         cout << "Extraction summary for " << filename << " (" << protocol << " protocol)" << "\n";
351         cout << left << setw(28) << "Number of frames: " << packetCounter << " frames" << "\n";
352         cout << left << setw(28) << "Total transmission time: " << totalTime << " s" << "\n";
353         cout << left << setw(28) << "Data sent (with headers): " << totalData << " MB" << "\n";
354         cout << left << setw(28) << "Data sent (no headers): " << totalUsefulData << " MB" << "\n";
355         cout << left << setw(28) << "Expected amount of data: " << expectedData << " MB" << "\n";
356         cout << left << setw(28) << "Percent data in packet: " << dataPercentage << " %" << "\n";
357         cout << left << setw(28) << "Percent data loss: " << dataLossPercent << " %" << "\n";
358         cout << left << setw(28) << "Transmission speed: " << speed << " MB/s" << "\n";
359         cout << left << setw(28) << "Average frame size: " << frameSizeAvg << " bytes" << "\n";
360
361         if(protocol == "sctp"){
362             cout << left << setw(28) << "Data chunk count: " << dataChunks << " chunks" << "\n";
363             cout << left << setw(28) << "Average data per chunk: " << dataChunkAvg << " bytes" << "\n";
364         }else{
365             cout << left << setw(28) << "Data chunk count: " << "NaN" << "\n";
366             cout << left << setw(28) << "Average data per chunk: " << "NaN" << "\n";
367         }
368         cout << left << setw(28) << "Number of clients: " << numClients << " clients" << "\n";
369         cout << left << setw(28) << "Streams/sockets per client: " << numberOfStreamsPerClient << " streams" << "\n";
370         cout << GUI_line() << "\n";
371     }
372 }
```

Beräkna data

Öppna output .dat fil

Skriv till output filen

Den skapade .dat filen skickas sen till Gnuplot av NSPlot.cc för att rita upp graf.