Communication Protocol for a Swarm Based Routing Algorithm Using the IEEE 802.11p Standard

Christian Stolcis, Steve Zakrzowsky and Wilhelm R. Rossak
Chair of Software Technology
Friedrich Schiller University
Jena, Germany
Email: {Christian.Stolcis, Steve.Zakrzowsky, Wilhelm.Rossak}@uni-jena.de

***Abstract*—Within the next few years, cars will be able to communicate with their surrounding infrastructure, with other cars and even drive autonomously. This allows a new generation of applications to improve security and a better use of space and resources. One of these applications is represented by the *Clustered Swarm* algorithm. The *Clustered Swarm* algorithm is a live swarm based algorithm for vehicles which pursues the aim of a global traffic optimisation by performing a massive load balancing of all road participants to improve the individual routes of each user. Hereby it represents a potential solution for the traffic jam problem. This paper introduces the current communication protocol used by the *Clustered Swarm* algorithm which is designed to ensure the integrity of the data, as well as to reduce the amount of transmitted data to enable the use of current and future vehicle-to-vehicle technologies such as the IEEE 802.11p standard.**

Keywords-IEEE802.11p; data compression; swarm intelligence, traffic optimisation.

I. INTRODUCTION

Current technologies in the area of direct vehicle-to-vehicle communication, most notably the IEEE 802.11p [1] standard, have some limitations regarding their bandwidth and their data rate. Especially for applications where a large amount of data needs to be transmitted to many communication partners within a short time window, the current characteristics of the 802.11p standard are insufficient. And so they are for the *Clustered Swarm* algorithm [2], which uses direct vehicle-to-vehicle communication to perform a load balancing of all road participants. For best communication results, regarding the amount of data transmitted, the *Clustered Swarm* algorithm uses its own communication protocol, which will be presented in the present paper.

In Section II, we will give a short overview of the *Clustered Swarm* algorithm for a better understanding, followed by the used communication model in Section III and the communication protocol in Section IV. Finally in Section V, we will discuss the results and in Section VI, we will give a brief outlook for our next steps regarding further improvements of the communication protocol to comply even more with the requirements of the *Clustered Swarm* algorithm.

II. CLUSTERED SWARM

Looking at traffic on a microscopic level, traffic consists of many individual participants. But during route calculation, only the personal and individual aims of the different drivers are considered, which are mostly represented by reaching the destination as fast as possible. So, current navigation systems perform an individual *local optimisation* during route calculation. Nevertheless the current traffic situation is considered, the resulting route is only optimised for the single user. If too many local optima are too similar in their characteristics, this can have negative effects on the whole traffic, particularly in combination with general traffic influencing measures. If too many drivers take the same diversion because they follow their navigation systems, a new traffic jam can form very fast. Many local optima can therefore counteract a common *global optimum* under some circumstances.

The *Clustered Swarm* algorithm copes with this deficiency by distributing all participating vehicles on the entire road network based on its capacity without the need of a central instance. In this case, the capacity of a road corresponds to the maximum traffic density K_{max} [3], which is the amount of vehicles on the road at the same time, causing a congestion. Depending on the amount of high weight vehicles and the type and construction state of a road, K_{max} is typically about 150 vehicles/km [3]. This relation between the traffic density and the traffic speed is explained in the Fundamental Diagram of Traffic Flow shown in Fig. 1 and expressed in a few words, the more vehicles the less the traffic speed. The *Clustered Swarm* algorithm takes advantage of this dependence between the amount of vehicles and the traffic speed to perform the load balancing.

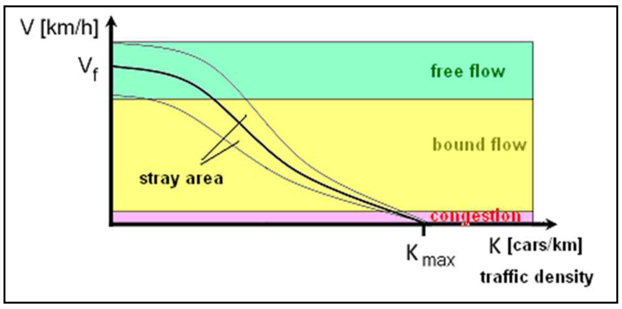


Figure 1. The Fundamental Diagram of Traffic Flow [3]

To accomplish the desired *Emergent Behaviour* [4] following the *Clustered Swarm* algorithm, each vehicle aggregates the estimated traffic density of the road network by exchanging its own route and all other considered routes every time a vehicle is in reach. The first time a vehicle communicates it can only transmit its own route but receives a number of routes from its counterparts. The next time it encounters another vehicle, it transmits its own route and all newly acquired routes. Thereby, it can help spreading knowledge within the swarm, which will be used for route calculation to adapt the routes to the estimated traffic density.

III. COMMUNICATION MODEL

The key function of the *Clustered Swarm* algorithm is the knowledge and use of the estimated traffic density of the road network. As this information needs to be aggregated, communicated and used in route calculation, a common model is needed to meet the requirements of the three main tasks, especially for communication. As already mentioned, vehicles try to transmit all considered routes. So, the main object is the *Route,* which at the same time is the result of the route calculation. Each *Route* consists of different *PathElements,* which decompose the *Route* into different segments and conform to the edges of the graph used for route calculation. Fig. 2 shows both objects.

The attributes of the *Route* object have the following purpose:

- **VehicleId**: Identifies the vehicle to which the route belongs.
- **VersionNumber**: Current version of the Route. Each time a vehicle recalculates its own route the version number is incremented.
- **NavigationDuration**: Is the total duration the vehicle needs to drive on its route.

The attributes of the *PathElement* object have the following purpose:

- **Id**: Identifies the road within the digital road map used for route calculation.
- **NavigationTimestamp**: Represents the time stamp of the day, when the road will be navigated.
- **DrivingDirection**: Stores the direction in which the vehicle will drive on that road segment.
- **NavigationDuration**: Is the duration the vehicle needs to drive on that road segment.

With the information of the *PathElements,* and therefore the route of a vehicle, each vehicle can estimate how many vehicles will use a road at a given point in time and with this information the vehicle can adapt its own route if necessary due to high traffic density on certain roads [2].

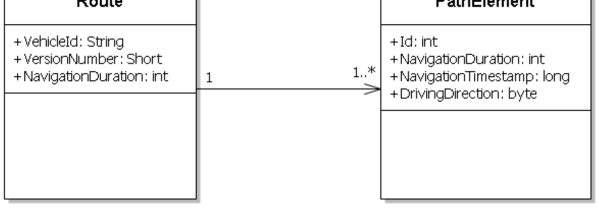


Figure 2. Communication model of the *Clustered Swarm* algorithm

Assuming that only 20% of all vehicles will be able to use the *Clustered Swarm* algorithm, in the near future in Berlin 154,000 vehicles will communicate using a vehicle-to-vehicle technology (in 2012 60.9 % of all registered vehicles in Germany were on the road on working days [5]; Berlin has 1.1 million registered vehicles [6], so, each working day 770.000 vehicles are on the road). If each vehicle communicates with only 50% of all possible vehicles capable of *Clustered Swarm*, each vehicle would save and communicate a maximum of 77,000 routes. A typical route in inner cities has an average amount of 80 road segments (assuming an average length of one trip of 12.3 km [5] and an average length of one road segment of 153.8m for Germany [7]), which corresponds to the *PathElement*. Each *PathElement* has a size of 17 bytes resulting in an average size of 1378 bytes for a Route. In a best case scenario, where the maximum transfer rate of 27 Mbit/s of the IEEE 802.11p standard [8] could be used (which is unlikely), it would take about 30 seconds to transmit the 101 MB of data. In realistic scenarios, where the data rate decreases due to the amount of parallel communication partners, the distances between them, as well as other interferences [8] a transfer of the huge amount of *PathElements* would not be guaranteed. As the vehicles move during communication, the time window for communication could limit the amount of data transmitted even more. This means that there is a big discrepancy between the amount of data to transmit to ensure the functionality of the *Clustered Swarm* algorithm and the transmission rate of the 802.11p standard. To cope with these limitations, we have implemented a communication protocol which reduces the amount of data to be sent while ensuring the integrity of the transmitted data, so that each received data packet contains complete information usable by the *Clustered Swarm* algorithm.

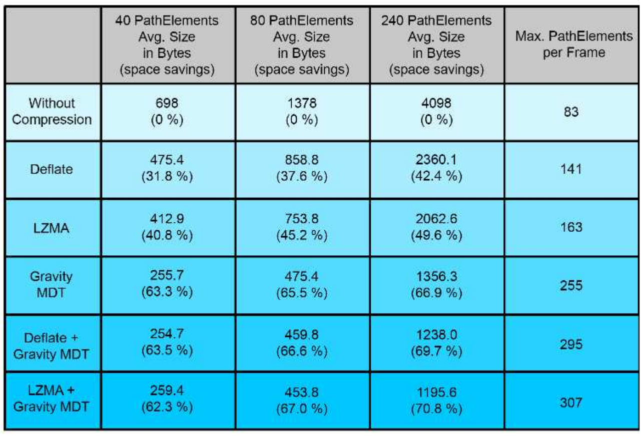
IV. COMMUNICATION PROTOCOL

A. Integrity

As the IEEE 802.11p standard is situated in the data link layer of the OSI model [9], the used protocol for transmission has to be defined by the applications using the IEEE 802.11p standard. Given the limitations of the IEEE 802.11p standard compared to the requirements of the *Clustered Swarm* algorithm, a communication protocol is needed, which allows to transmit as much information as possible within a short time window ensuring that received data is usable by the receiver. The integrity of the data is especially important in this case. Since the amount of transmitted data is restricted by the short communication time of two vehicles passing by each other, it is even more necessary that the actual transmitted data can be used by the receiver. That means, the information needs to be sent in a way that each received data packet contains sound data independent of other data packets and thereby usable by the *Clustered Swarm* algorithm.

In common communication protocols (e.g., TCP/IP), the data to be transmitted is distributed among different packages

TABLE I. SPACE SAVINGS OF DIFFERENT COMPRESSION ALGORITHMS



	40 PathElements Avg. Size in Bytes (space savings)	80 PathElements Avg. Size in Bytes (space savings)	240 PathElements Avg. Size in Bytes (space savings)	Max. PathElements per Frame
Without Compression	698 (0 %)	1378 (0 %)	4098 (0 %)	83
Deflate	475.4 (31.8 %)	858.8 (37.6 %)	2360.1 (42.4 %)	141
LZMA	412.9 (40.8 %)	753.8 (45.2 %)	2062.6 (49.6 %)	163
Gravity MDT	255.7 (63.3 %)	475.4 (65.5 %)	1356.3 (66.9 %)	255
Deflate + Gravity MDT	254.7 (63.5 %)	459.8 (66.6 %)	1238.0 (69.7 %)	295
LZMA + Gravity MDT	259.4 (62.3 %)	453.8 (67.0 %)	1195.6 (70.8 %)	307

depending on the Maximum Transfer Unit (MTU) [10]. By specifying the Maximum Transfer Unit size for a network compatible device, including header and protocol-meta-information, the MTU defines the amount of data for one frame [11]. Data packages exceeding the size defined by the MTU are fragmented and distributed over multiple frames. In order to obtain complete and usable data this procedure requires all frames to be received. If one frame is lost, all others have to be resent [10]. This behaviour can lead to major radio transmission interferences, wherefore a fragmentation should be avoided to meet the requirements of the *Clustered Swarm* algorithm.

As described in Section III, the information transmitted by each vehicle is represented by the *Route* object. Since one *Route* represents a complete set of data usable by the receiver, we decided to follow a "one route one frame" approach to guarantee data integrity. This means, one data packet represented by one *Route* should completely fit into the payload of one transmitted frame. To provide enough space to hold a *Route* and at the same time avoid too much data loss in case of a communication failure, we use a size of 1500 bytes as the maximum data packet size. Also, the fact that this size represents the standard for transmitting data over Ethernet since over 30 years, fortified our decision [10]. If a frame is lost, it does not affect other frames and hereby other *Routes*, so that the small time window for communication can be used in the best possible way.

Since the average route size is 1378 bytes (see Section III), longer routes would not fit into the 1500 bytes. To be able to store even long routes, a reduction of the data was needed.

B. Compression

Thinking about possibilities to reduce the amount of data, a suggestive approach is the compression of the data. Many different algorithms exist (e.g., LZ77 [12], Deflate [13] and LZMA [14]), which are available for various programming languages and allow an easy integration into own applications. But in our special case, the existing algorithms showed low compression ratios (see Table I and Fig. 4), too low for the use in the *Clustered Swarm* algorithm. This mostly relies on the "black box" approach of existing compression algorithms where no semantic information about the data is considered [14]. As the structure of the communication model introduced in Section III offers many possibilities to reduce the data on a logical level, we developed our own compression algorithm called *Gravity MDT Compression*. Basically, the *Gravity MDT Compression* (**Gr**oup-**V**ar-**I**nt **m**inimal **data** compression) combines three approaches to reduce the amount of data without changing its content: **Group-Var-Int-Encoding** [15], **Delta-Encoding** [14] and **Elimination of redundancy**.

The **Group-Var-Int-Encoding** was developed by Google Inc. on the basis of the *Var-Int-Encoding* [16] which stands for "variable integer" and represents an integer data type that only occupies as many bytes as needed to represent the value. For example, a 32-Bit Integer with the value 1 only needs 1 byte instead of 4 bytes to be stored.

Since the normal *Var-Int-Encoding* stores some extra information to be able to decompress the value, the maximum number of bits to be used for storing a value is 30 [16]. *The Group-Var-Int-Encoding* adds an additional byte to store this extra information which allows to use the full 32-bit-integer value range. The *Gravity MDT Compression* uses the *Group-Var-Int-Encoding* to compress and store the *Ids* and the *NavigationTimestamps* of the *PathElements*.

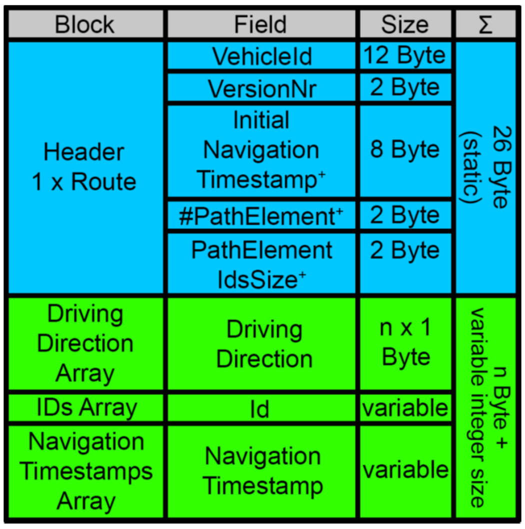
The **Delta Encoding** or differential encoding is a simple data compression method used to reduce correlating or sequential data [14]. The idea behind the delta encoding is that not the information itself is stored but the difference from an initial state to the current state. Table II shows the *Delta-Encoding* applied to sample *Ids*. So, the Delta Encoding helps to trim the possible big integer values to much smaller values. As the compression ratio of the *Group-Var-Int-Encoding*

TABLE II DELTA ENCODING APPLIED ON SAMPLE PATHELEMENT IDS

Encoding	Id *PathElement* #1	Id *PathElement* #2	Id *PathElement* #3	Id *PathElement* #4	Id *PathElement* #5
none	5890234	5839494	5839274	5897947	5897366
Delta-Encoding	5890234	50740	220	-58673	581

increases with the decreasing size of the values to store, the combination of the *Delta-Encoding* with the *Group-Var-Int-Encoding* allows to reach a very high compression ratio. In the *Gravity MDT Compression* algorithm, it is also used to store the *Id* and *NavigationTimestamp* of the *PathElements*.

Elimination of redundancy: Taking a closer look at the communication model, it turns out that the *PathElements* save some redundant information given by the *NavigationDuration* and the *NavigationTimestamp*. Both values are needed by the *Clustered Swarm* algorithm but for communication, one of the values becomes obsolete as both can be calculated considering the other. Since the *PathElements* are stored in the order they are driven during route guidance, the chronological accumulation of the *NavigationDuration* allows the calculation of the *NavigationTimestamp* of all *PathElements* and through the difference of the *NavigationTimestamps* of two following *PathElements* the *NavigationDuration* can be calculated. Also, considering the advantages of the *Group-Var-Int-Encoding* and the *Delta-Encoding*, we decided to store the *NavigationTimestamps* to calculate the *NavigationDuration*. The final structure of the communication protocol considering the three mentioned approaches is visualized in Fig. 3. Each route to be transmitted is converted into this structure and at the same time compressed by applying the three methods.
The first 26 bytes represent a header, which saves information of the *Route* object and some additional meta information used for compression like the *InitialNavigationTimestamp*, the *PathElementCount* and the *PathElementIdsOffset*. The *InitialNavigationTimestamp* is needed by the *Delta-Encoding* as a start value. The *PathElementCount* and the *PathElementIdsOffset* are required since after compressing the *Ids* and the *NavigationTimeStamps* with the *Group-Var-Int-Encoding* the block size is variable. The header is followed by information about the *PathElements,* which are stored in



Block	Field	Size	Σ
Header 1 x Route	VehicleId	12 Byte	26 Byte (static)
	VersionNr	2 Byte	
	Initial Navigation Timestamp+	8 Byte	
	#PathElement+	2 Byte	
	PathElement IdsSize+	2 Byte	
Driving Direction Array	Driving Direction	n x 1 Byte	n Byte + variable integer size
IDs Array	Id	variable	
Navigation Timestamps Array	Navigation Timestamp	variable	

Figure 3. Communication protocol structure

arrays for best compression rate, and save the *DrivingDirections* the *Ids* and the *NavigationTimestamps* of the *PathElements*.

C. Further compression improvements

A simple way to improve the quality of an algorithm is applying two or more algorithms to the same problem. This is also applicable for the compression of data, by combining the advantages of different approaches to reach a better compression. However, a potential improvement can only be reached in case of the compression ratio since the coding and decoding times increase with the number of the applied algorithms.

Nevertheless, we evaluated the combination of the Deflate and the LZMA with the Gravity MDT algorithm to get the most out of the compression. The results are shown in Table I and in Fig. 4 and Fig. 5, and will be discussed in the next Section.

V. EVALUATION

The evaluation of the quality of the presented compression algorithms has been determined using JUnit tests. For this purpose we generated a set of *Route* objects with random data. To be able to generate realistic data, the random generator has been restricted. In case of the *NavigationDuration* we used a Gaussian distribution with a mean value of 30 which represents the average duration in seconds to navigate a *RoadElement* retrieved from the used Navteq maps [7]. In addition, the generated number has an upper limit of 3600 (1 hour), so that the range of values for the *NavigationDuration* is [0, 3600].

Fig. 4 shows the determined compression ratios of the different algorithms and combinations of them. The x-axis shows the amount of *PathElements* of a random generated *Route*, and the y-axis shows the arithmetic mean over all generated *Routes* (100,000) with x *PathElements*. The lowest compression ratio of 1.8 has been reached by the Deflate algorithm (blue). This corresponds to a space saving of 45.2% (see Table I). The LZMA algorithm (yellow) reached a compression ratio of 2 which corresponds to a space saving of 53%. The evaluation of the Gravity-MDT algorithm (green) showed a compression ratio of 3.1 and therefore a compression of 68% which is an improvement of 15% compared to the LZMA algorithm.

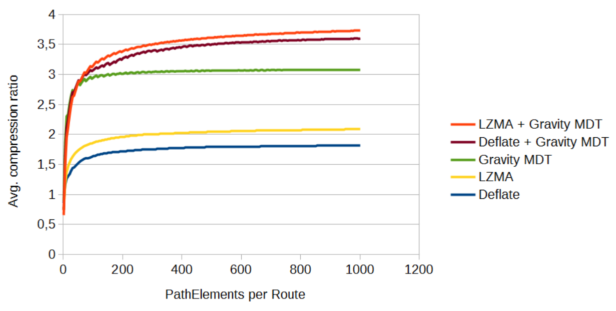


Figure 4: Compression ratios

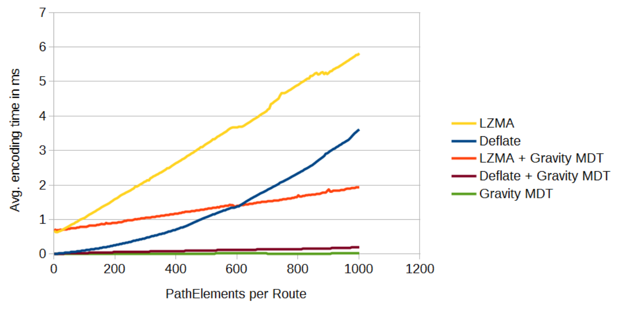


Figure 5: Compression encoding times

As expected, the combined algorithms showed the highest compression ratios, where the LZMA combined with the Gravity MDT algorithm (red) reached the best space saving of 74%.

Taking a look at the encoding times of the different algorithms in Fig. 5, with the amount of *PathElements* on the x-axis and the coding time in milliseconds on the y-axis, the LZMA (yellow) followed by the Deflate (blue) algorithm achieved the slowest encoding times and showed the fastest increase depending on the amount of *PathElements*. The fastest algorithm regarding the encoding times reached the Gravity-MDT algorithm (green), followed by the Deflate combined with the Gravity-MDT algorithm (brown). Since in the combined approaches the Gravity-MDT algorithm is executed first, the reduced amount of bytes to compress lead to very good encoding times for the Deflate-Gravity-MDT combination.

Considering these results for the compression ratio and the encoding times, we decided to use the combination of the Deflate with the Gravity-MDT algorithm in the communication protocol. Despite the slightly inferior results compared to the best algorithms in the two categories, the approach represents the best choice considering both characteristics.

VI. Conclusion And Future Work

Current mobile communication technologies, especially the IEEE 802.11p standard enable a broad field of new applications and services to improve different aspects of the current road traffic. However, the IEEE 802.11p standard with its current characteristics is not completely suitable for all future applications, and so it is not for the *Clustered Swarm* algorithm. As the quality of the load balancing performed by the *Clustered Swarm* algorithm is highly depending on the amount and the actuality of the transmitted *PathElements,* it is crucial that within the short communication window as much information as possible is transmitted and that the integrity of the data is guaranteed. To achieve this goal, we build a communication protocol that compresses the routes to be transmitted in a way that one route fits into one 1500 bytes sized frame. Since every received frame contains complete and usable data, the communication is failsafe without the need of additional integrity checks, even if a package is lost.

Beside these advantages, the current implementation of the protocol offers some possibilities for improvement. The longer the route, the more *PathElements* and so the more bytes to transmit, regardless the compression ratio. If a route exceeds the size of 1500 bytes, at the moment the route would be truncated and thereby incomplete. However, not all *PathElements* are really necessary for all vehicles in range. A vehicle driving in the opposite direction does not need the information where all other vehicles intend to go. It only needs information about a smaller area regarding its position and its target. To adapt the transmitted information to the receivers' needs and at the same time reduce the amount of *PathElements*, we started elaborating a filtering mechanism. Based on the position of the receiver and a defined radius, the sender transmits only *PathElements* within the given perimeter. Because this directly affects the quality of the calculated routes and hereby the load balancing, the filter will be evaluated to guarantee a maximum reduction for each route to fit into the 1500 bytes while still maintaining the impact on the traffic. Nevertheless this approach also implies that most routes will not need all the 1500 available bytes so that the capacity of one frame is not fully used. Basically, this corresponds to a knapsack problem where a subset of routes needs to be chosen without exceeding the size of 1500 bytes. Since the communication represents the key mechanism of the *Clustered Swarm* algorithm, a suitable solution for the knapsack problem is already in development to get the most out of it.

References

[1] G. R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X. P. Costa, and B. Walke, "The IEEE 802.11 universe", Communications Magazine, IEEE (Volume:48, Issue: 1), pp. 62-70, January 2010.

[2] C. Stolcis and E. Pfannerstill, “Clustered Swarm – A live swarm based traffic load balancing algorithm against traffic jams”, “in press”, ITS World Congress, Bordeaux 2015.

[3] Leutzbach, W., “Introduction into the theory of traffic flow”, Springer Berlin, 1972.

[4] J. Kennedy and R. C. Eberhart, „Swarm Intelligence“, Morgan Kaufmann, March 1997.

[5] „Motor vehicle traffic in Germany 2010 (KiD 2010)“, Federal Ministry of Transport, April 2012.

[6] „Vehicle registrations, inventory of motor vehicles and vehicle trailers by town”, Federal Ministry of Transport, January 2014.

[7] „GDF 3.0 Reference Manual v42.0“, Nokia, April 2012.

[8] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments", Vehicular Technology Conference, May 2008, pp. 2036-2040.

[9] “IEEE Std. 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications”, Institute of Electrical and Electronics Engineers Inc., 2007.

[10] K. R. Fall and W. R. Stevens, “TCP/IP Illustrated”, Volume 1, The Protocols, Second Edition, Addison Wesley 2012.

[11] D. Murray, T. Konziniec, K. Lee, and M. Dixon, “Large MTUs and Internet Performance”, IEEE 13th International Conference, June 2012, pp. 82-87.

[12] S. Kreft and G. Navarro, “LZ77-like Compression with Fast Random Access”, Data Compression Conference (DCC), 2010, pp.239-248.

[13] P. Deutsch, “DEFLATE Compressed Data Format Specification”, version 1.3 in RFC 1951, May 1996.

[14] E. Leavline and D. Singh, “Hardware Implementation of LZMA Data Compression Algorithm”, International Journal of Applied Information Systems (IJAIS), March 2013, pp. 51-56.

[15] J. Dean, “Challenges in building large-scale information retrieval systems: invited talk.”, Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM), 2009.

[16] A. Stepanov, A. Gangolli, D. Rose, R. Ernst, and P. Oberoi, “SIMD-Based Decoding of Posting Lists in CIKM”, International Conference on Information and knowledge management, 2011, pp. 317-326.