Improving Resource Management in Software Defined Networks

Marek Galinski[*]
Institute of Computer Engineering and Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 842 16 Bratislava, Slovakia
marek.galinski@stuba.sk

Abstract
For real-time multimedia sessions, it is not possible to perform per-packet load balancing as these services are sensitive to delay and the delivery order of individual fragments. These data flows must be considered unsplittable and therefore we must use the load-balancing mechanism on a per-flow basis. If the sum of the free transmission capacities of all paths is greater than the transmission capacity required by this new data flow, there is a possibility that by rearranging the existing data flows between paths, sufficient free capacity will be available on one of the paths to meet the requirements of the new data flow while avoiding congestion. In this work, we express this problem by using the Knapsack problem, respectively by defining a new type of problem by combining existing known subtypes. This view allowed us to design the QFLA algorithm, looking for an improved solution reallocations are needed. However, unlike other known solutions, the main criterion in our case is not maximizing capacity but minimizing calculation time and minimizing the number of reallocations required to achieve an improved solution. The proposed algorithm was verified in two ways - by simulating the calculations themselves with different numbers and sizes of backpacks and objects, but also by simulating the implementation of the algorithm in the SDN environment.

Categories and Subject Descriptors
C.2.0 [**Networks**]: General; C.2.1 [**Networks**]: Network Architecture and Design; C.2.3 [**Networks**]: Network Operations

Keywords
Software Defined Networking, Load Balancing, Knapsack Problem, Optimization Problem, Network Monitoring, Network Management

[*]Recommended by thesis supervisor: Prof. Ivan Kotuliak
To be defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on June XX, 2020.

1. Introduction
Multimedia over Internet protocol (usually covered also as Voice over IP) has developed as a mainstream platform. SIP protocol has been standardized by IETF in RFC3261 [20] and is commonly accepted as a standard. Even the 3GPP has adopted it in IMS (IP Multimedia Subsystem) as a signalling protocol (with some further information in RFC4083 [8]). Despite this technology is mature and widely used it still faces problems, among which are load balancing and quality of service of multimedia sessions in multipath VoIP networks. IP is designed as best-effort protocol, it does not implement or guarantee any kind of Quality of Service mechanism.

Standard approaches of VoIP architectures in similar scenarios are not optimal, when it comes to load balancing and quality management and this problem has been analyzed and solved by many resereach teams throughout the years (for example in [23]). As a solution we propose a new architecture capable of active management of ongoing multimedia sessions.

Software Defined Networks (SDN) and Network Function Virtualization (NFV) in the world of computer networks open new possibilities for managing and optimizing network traffic. The load balancing mechanisms are well known since years ago, and they are naturally part of the SDN environment. Nevertheless, most of these mechanisms are insufficient these days, since most of the current network traffic consists of real time multimedia sessions. These data flows cannot be (or at least are not supposed to be) load balanced per-packet. In the same manner, it is not sufficient to handle congestion after it happens, instead of avoiding it to happen. This thesis sets its objective to continue to work on the Quick Forward Lookup Algorithm (QFLA) brought in [7] in the way of extending the algorithm for the SDN environment, since it was intended for use in the SIP Single Port networks, and to evaluate and formalize the QFLA, comparing performance of the QFLA with existing load balancing mechanisms in SDN environment.

2. State of the Art
The Internet itself is designed in the manner, where the end nodes are responsible for most of the tasks concerning the communication. The network by design does not actively maintains the quality of service, nor handles the congestion or other impairments. This design has two main advantages: Firstly, it allows a unified best-effort service for any type of data at the network layer where

service definitions are made at the upper layers (hosts). Secondly, it reduces the overhead and the cost at the network layer without losing reliability and robustness. [11]

This type of architecture fits perfectly to data transmission where the primary requirement is the delivery of the data. Yet, in multimedia transmission, timely delivery is crucial. Multimedia streaming applications have strict delay requirements which cannot be guaranteed in the best-effort Internet. [11]

2.1 Differentiated Services

Differentiated services (DiffServ) is an architecture, that specifies simple mechanism for classifying and managing IP network traffic in order to provide QoS. It's common usage is to provide low-latency to critical traffic such as voice or another streaming media, while providing best-effort delivery for other, non-critical services such as web or file transfers.

DiffServ is defined in IETF RFC 2474 and later updated by RFC 3168 and RFC 3260 as an extension of well-known IP architecture based on packet routing and forwarding. On top of that, “the differentiated services architecture contains two main components. One is the fairly well-understood behavior in the forwarding path and the other is the more complex and still emerging background policy and allocation component that configures parameters used in the forwarding path.” [17]

2.2 Resource Reservation Protocol

Another tool commonly used in several QoS approaches is the Resource Reservation Protocol (RSVP), defined by RFC 2205 [4]. This protocol has been designed for an Integrated services within the Internet. Before we will discuss Integrated Services (IntServ), we take a closer look onto RSVP, since IntServ relies on this protocol.

The main purpose of RSVP is, as stated in the protocol name already, to reserve resources for service, in order to avoid congestions, since the requested resources are fully dedicated for the service, so no one else on the network can use them.

RFC 2205 states that “RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows, with good scaling and robustness properties.” Further, “RSVP is also used by routers to deliver quality-of-service (QoS) requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path.” [4] RSVP is not a routing protocol, thus it can only work in cooperation with any configured routing protocol in the network.

2.3 Software Defined Networks

As could have been spotted in the previous section, SIP Single Port in combination with MPLS-TE separated the network between two logical layers - upper layer where the network “intelligence” and the decision making is located, and the bottom layer that is only responsible for forwarding the actual data.

The conventional networks consisted of mostly closed and proprietary dedicated hardware devices with a stand-alone configuration and mostly the devices could only control their own behaviour, not considering what is good for the network as a whole entity. This resulted in the very exhausting network management, since the network configuration was distributed between practically all network devices, each of them having part of the overall network behaviour implemented. In such network, any on-demand or real-time changes are a non-trivial task, furthermore, making a configuration mistake in any of the network devices may led to overall network failure.

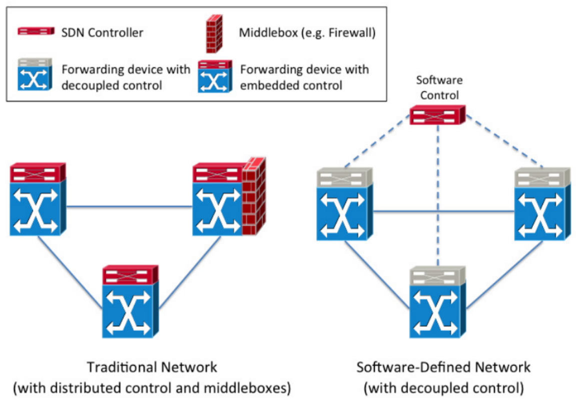
SDN Controller
Middlebox (e.g. Firewall)
Forwarding device with decoupled control
Forwarding device with embedded control
Software Control
Traditional Network (with distributed control and middleboxes)
Software-Defined Network (with decoupled control)

Figure 1: SDN architecture compared to conventional network. [18]

Software defined networking (SDN) is an approach to have the networks programmable in order to have the capacity to initialize, control, change and manage network behaviour dynamically via open interfaces. This approach emphasizes the role of software in the computer networks by abstracting the network for the data forwarding plane and the separate control plane. The layers and architecture of SDN is described in RFC 7426 [9] by IETF. This definition is very similar to what we have mentioned by SIP Single Port - the below layer, in SDN called data forwarding layer (DFL) is not capable of doing any logical decision - it is only responsible to deliver the actual data hop-by-hop via the network, while the upper layer, in SDN called control layer (CL) is a software component that controls the entire network from a logically centralized system. In this way, SDN makes it easier for network operators to evolve the network capabilities.

In the paper [18] author state that ”the idea of programmable networks has recently re-gained considerable momentum thanks to the emergence of the Software-Defined Networking paradigm”. SDN, often referred to as a “radical new idea in networking”, promises to dramatically simplify network management and enable innovation through network programmability.

2.4 OpenFlow Protocol

Driven by the SDN principle of decoupling the control and data forwarding planes, OpenFlow [14] standardizes information exchange between the two planes.

In the OpenFlow architecture illustrated in Figure 2, the forwarding device, or OpenFlow switch, contains one or more flow tables and an abstraction layer that securely communicates with a controller via OpenFlow protocol. [18]

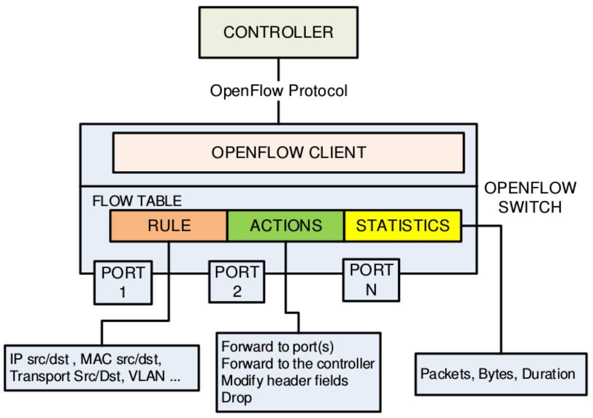

Figure 2: OpenFlow protocol architecture. [18]

Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined by the table-miss flow entry. Every flow table must contain a table-miss entry in order to handle table misses.

2.5 Unsplittable Flow Problem

At this point, the reader might consider applying the maximum flow algorithms like Ford-Fulkerson algorithm [6] to solve the optimization. When talking about jitter and delay sensitive data flows such as multimedia, we cannot perform the load balancing on a per-packet base. Instead, we must consider all such data flows unsplittable. Splitting these sessions will lead to a quality degradation due to different delivery times and therefore cause a jitter to rise or packets to be delivered out of order. [16]

2.6 Knapsack Problem

In general, the problem, where we want to distribute multiple data flows of given bandwidth between the multiple disjoint network paths of given capacity without exceeding the capacity of any path, can be expressed as a Knapsack Problem. According to [13], Knapsack is not a single specific problem, but a group of similar problems of the combinatorial optimization, where all af them belong to the class of NP-hard. We call these problems *the Knapsack problem modifications.*

The Knapsack Problem (KP) is the task of combinatorial optimization and is ranked as the task of integer linear programming. The Knapsack Problem has many applications in real life, and is therefore one of the most challenging issues in this area. For this reason, there have been many variations on this issue.

In the original KP, we have a set of items that all have some weight and some price (the weight and the price are not dependant on each other). Then, we have a knapsack of a certain loading capacity. The task is to select the subset of the items to be packed into the knapsack in order to carry the highest possible value without exceeding the knapsacks loading capacity. [5]

2.7 Online Knapsack Problem

The (classical) knapsack problem is given a set of items with profits and sizes, and the capacity value of a knapsack, to maximize the total profit of selected items in the knapsack satisfying the capacity constraint.

In the name Online Knapsack Problem the word ”online” means that items are given over time, i.e., after a decision of rejection or acceptance is made on the current item, the next item is given, and once an item is rejected or removed, it cannot be considered again. The goal of the online knapsack problem is the same as the offline version, i.e., to minimize or maximize the total cost. [10]

The authors in [10] further state, that by the observation: if all the items have the same size, then a simple greedy algorithm (called Lowest Cost First strategy) of picking items with the lowest cost first provides an optimal solution.

In another paper that deals with Online Knapsack Problem [3], the authors state that when the weights are uniform and equal to the weight constraint, the problem reduces to the famous secretary problem, or the problem of selecting online an element of maximum value in a randomly-ordered sequence. This problem was first introduced by Dynkin [21] in 1963.

2.8 Conversion Between Different Knapsack Problem Modifications

As mentioned above, some modifications of Knapsack Problems are very similar. So for example all solving methods for 0-1 Knapsack Problem are also applicable to some other Knapsack Problems. Most resources are devoted to 0-1 Knapsack Problem. The main reason is probably the possibility of transforming different Knapsack Problems into equivalent 0-1 Knapsack Problem form with only a generally limited increase in the number of variables. So it is possible to effectively solve this problem by using algorithms for 0-1 Knapsack Problems. [13]

Value independent Knapsack Problem can also be solved by any method for 0-1 Knapsack Problem, since we can simply transform it into 0-1 Knapsack Problem, so that $p_i = w_i$. However, in this case, it is much better to deal with it by specialized methods because they usually provide better results [13].

2.9 Brute Force Method

Brute force can be used to solve all the mentioned modifications of Knapsack Problem. It is a solution that simply examines all solution possibilities and chooses the best.

With 0-1 Knapsack Problem this algorithm has a complexity of $O(2^n)$, which is the number of all combinations. Obviously, with more complex variations of the problem, e.g. with more knapsacks, the number of combinations is even greater. Because of this complexity, this algorithm can only be used for small instances of the problem, and even then it is still slow and inefficient.

2.10 Branch & Bound Method

This method is an improvement of the brute force method. The principle of the method of branches and boundaries is to systematically go through all the potential solutions represented by the tree.

The algorithm passes through the branches of this tree that represent subsets of the set of solutions. Before an algorithm is embedded in a branch, it is always checked whether it is necessary to scan this branch at all. This is done by comparing the current branch with the lower and upper bounds of the optimal solution. If a branch can no longer deliver a better solution than the best solution found so far, it simply stops searching. The algorithm relies on an effective lower and upper bound estimate. By this method we can achieve an optimal solution, but not necessarily. It depends precisely on the way of calculating lower and upper bounds [22].

The complexity of this solution in the worst case is the same as in the brute force method, $O(2^n)$. However, in practice, the speed of this solution is significantly lower [24].

The Branch & Bound method is probably the most used and most effective method for obtaining an accurate solution for Knapsack Problems in general. It can be used in all the variations of the problem mentioned in this work. Of course, for each problem variation, boundaries are calculated differently, and although it provides significantly lower time than the brute force method, this time is still too long for this method to be used for real-time calculations.

2.11 Greedy Algorithms

These algorithms rely on a fact that in each step they select a local optimal solution in the hope that this solution will also be globally optimal. Heuristics are used to select a local optimal solution. Due to the combination of hope and heuristics, they do not need to always find the optimal solution.

It is a very simple and fast solution that consists of two phases. In the first phase, we sort the objects downward by price / weight ratio $\frac{p_i}{w_i}$. In the second phase, the objects are gradually stored in the knapsack until they exceed the capacity of the knapsack.

Thus, the most challenging part of this algorithm is the sorting of the elements that have minimal complexity of $O(n * log(n))$, for example, using *quicksort*. Inserting into the knapsack has a linear complexity. Thus, it is clear that this algorithm is much faster than the exact algorithms mentioned so far, but it will not always bring us the optimal solution. However, as the test results shown in [15] [2], the average deviation from the optimal solution decreases rapidly as the set of objects increases.

Therefore, this heuristics is useful in larger instances of the problem where the deviation is not so significant and exact methods would be too time consuming.

3. Value Independent Multiple Knapsack Problem

In this Section we introduce new Knapsack Problem modification, that combines two problem modifications mentioned earlier - Value Independent Knapsack Problem, where the objects do not have its weight and its value, but the value is equal to their weight, and the aim is to carry as much weight as possible, and the Multiple Knapsack Problem, where multiple containers (knapsacks) exist with each having its own capacity. The task remains bivalent - we either put the object into any of the knapsacks or we do not. Each object can be put into knapsack only once.

The mathematical model can be displayed by assuming that $p_i = w_i$ in the mathematical model of Multiple Knapsack Problem, as follows:

maximization:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{ij} \tag{1}$$

when:

$$\sum_{j=1}^{n} w_j x_{ij} \leq c_i, i \in M = \{1, ..., m\} \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, i \in N = \{1, ..., n\} \tag{3}$$

$$x_{ij} \in 0, 1, i \in M, j \in N \tag{4}$$

$$w_j, c_j \in N; i, j, m, n \in N^0 \tag{5}$$

where:

x_{ij} is a binary variable expressing whether the object j has been inserted into the knapsack i

w_j is the ratio of $j-$object to knapsack capacity

c_i is the capacity of the $i-$knapsack

m is the amount of knapsacks

n is the amount of objects

3.1 Problem Description Using Value Independent Multiple Knapsack Problem

Let L be a set of all multimedia clients:

$$L = \{l_i : i \in \{1, 2, 3, ..., m\}\} \tag{6}$$

Let P be a set of all disjunct paths in the network:

$$P = \{p_k : k \in \{1, 2, 3, ..., r\}\} \tag{7}$$

Let ε be a function assigning 1 if a multimedia client is assigned to an path, otherwise 0, where:

$$\varepsilon : L \times P \rightarrow \{0, 1\} \tag{8}$$

All clients of the set L are interpreted as items. Paths of the set P are interpreted as knapsacks. The weight of item l, w_l, is the required bandwidth of a client and the size of a knapsack p, c_p, is the bandwidth capacity of the path.

For this problem modification we bring an approach made by us we call Quick Forward Lookup Algorithm.

4. Quick Forward Lookup Algorithm

Quick Forward Lookup Algorithm (QFLA) has been developed to address the problem of assigning clients to paths, in managing multimedia sessions. Based on the interpretation as Value Independent Multiple Knapsack Problem, where clients are objects and paths are knapsacks. QFLA as well as greedy algorithms does not guarantee finding the best solution.

The algorithm works on a model of gradual adding of items, and thus objects cannot be pre-arranged. In this case, it is an optimization algorithm that allows you to exchange items between different knapsacks. It allows us to increase the free capacity of some knapsacks so that another item can be placed into them, which otherwise would not be placed into any knapsack.

This algorithm has been first introduced as a concept in [7], where we provided proof of concept, the formal verification as well as some deeper research were still missing.

4.1 Algorithm Design

For the purpose of the algorithm, we define a knapsack problem UFKP (Unsplittable Flow Knapsack Problem), an instance of Value Independent Multiple Knapsack Problem, as follows:

$$x_{lp} \begin{cases} 1, & \text{if a client } l \text{ is assigned to a path } p \\ 0, & \text{if a client } l \text{ is not assigned to a path } p \end{cases} \tag{9}$$

In a formulation of this problem, we want to maximize:

$$\sum_{p=1}^{n}\sum_{l=1}^{m} w_l x_{lp}, \tag{10}$$

where n is a number of paths, m is a number of clients, x_{lp} is defined in (9) and w_l is the bitrate of all client's multimedia sessions.
While trying to maximize (10) following conditions have to be satisfied:

$$\sum_{l=1}^{m} w_l x_{lp} \leq c_p, \forall p \in \{1, ..., n\}, \tag{11}$$

$$\sum_{p=1}^{n} x_{lp} = \{0, 1\}, \forall l \in \{1, ..., m\}, \tag{12}$$

$$x_{lp} = 0 \text{ or } 1, l \in \{1, \ldots, m\}, p \in \{0, \ldots, n\}. \tag{13}$$

where c_p is a capacity of a path and x_{lp} is defined in (9).

At this point we would like to point to the situation where a client l has no multimedia sessions. In such situation w_l is equal to zero and therefore client l is not a subject to the UFKP.

There are several possibilities to solve clients that are not assigned to any path after an optimization process and their w is nonzero. Client's sessions either can be disconnected from the network with notification about the possible congestion in the network or client can be assigned to a path even though this will decrease its quality.

5. Algorithm Evaluation

In this Section we will evaluate algorithm formally as well as by testing its performance compared to known Knapsak Problem approaches.

5.1 Formal Verification of QFLA

The QFLA has not yet been formally verified. In the work [19] we prove that this algorithm is correct using direct proof.

Let m be the number of given knapsacks, R_i be the set of items assigned to knapsack i, R_{ia} be item a assigned to knapsack i, c_i be the capacity of knapsack i and we are supposed to assign given item with weight w. For every given R_i the capacity constraint $\sum_{a=1}^{|R_i|} R_{ia} \leq c_i$ must be satisfied.

$$\sum_{j=1}^{|R_i|} R_{ij} \leq c_i \tag{14}$$

must be satisfied. [19]

To prove the correctness of this algorithm we have to prove that the algorithm does not violate the capacity constraint, neither changes the overall number of assigned items otherwise than increasing it by one. We can interpret this statement as following theorem: [19]

Theorem A:

$$\sum_{k=1}^{m} |R_k| \leq \sum_{k=1}^{m} |R'_k| \leq \sum_{k=1}^{m} |R_k| + 1 \bigwedge \forall i \in 1, 2, 3, ..., m, \sum_{a=1}^{|R'_i|} R'_{ia} \leq c_i \tag{15}$$

where R' represents the sets after the assignment.

The algorithm consists of 3 stages, each of them represented by a lemma, in which k_l represents weight of items moved to knapsack l and x represents the knapsack where the item is supposed to be added: [19]

Lemma 1: In the first stage, the greedy knapsack algorithm stage, the item may only be assigned to a knapsack, but only if:

$$\sum_{a=1}^{|R_i|} R_{ia} + w \leq s_i \tag{16}$$

Lemma 2: In the second stage an item may be selected to be moved from one knapsack to another, but only if:

$$\sum_{a=1}^{|R_x|} R_{xa} + w - k_y \leq s_x \bigwedge \sum R_y + k_y \leq s_y \tag{17}$$

Lemma 3: In the last stage multiple items from one knapsack may be selected to moved from one knapsack to one or many other knapsacks, but only if:

$$\sum_{a=1}^{m} R_{xa} + w - \sum_{l=1}^{m} k_l \leq s_x \bigwedge \forall y \in \{1, 2, ..., n\} - i \tag{18}$$

$$\sum_{b=1}^{|R_y|} R_{yb} + k_y \leq s_y \tag{19}$$

Therefore we can conclude:

$$Lemma\, 1 \bigwedge Lemma\, 2 \bigwedge Lemma\, 3 \implies Theorem\, A$$

Table 1: Utilisation of Knapsacks and Required Calculation Time on Different Approaches [12]

	util. [%]	time [ms]
Greedy algorithm	99.93%	0.1 ms
Branch & Bound	100%	1165.92 ms
QFLAv1	99.81%	0.4 ms

and so prove that Quick Forward Lookup Algorithm is correct as for we proved that no item has been removed from knapsacks, only on may be added and that all capacity constraints are satisfied. [19]

5.2 Algorithm Performance Evaluation

In this test, all evaluated algorithms have been tested for a set of 100 items of different sizes, on top of three knapsacks with different, but predetermined sizes (6000, 4000 and 10000).

For items, we used data from dataset [1]. This resource provides a large amount of data to test 0-1 Knapsack Problem in csv format. However, since we are dealing with a modification of the 0-1 Knapsack Problem, we can use this data for any other Knapsack Problem. The sizes of the items ranged from 1 to 1000. The price of the item was not considered.

We ran all tests repeatedly 5 times (with random objects from datasets) and averaged the results we got.

When testing the Branch & Bound algorithm, the algorithm was limited to only one backtrack. In the table we can see that the branch & bound algorithm has found the maximum solution (total sum of all knapsacks - 20000).

The more detailed test results presented in [12] show that the calculation time does not change noticeably for larger instances and we can see that larger instances achieve better results. It was also unusual for the one case the Branch and Bound algorithm found a solution with a profit of only 19803, in other cases the final profit of this algorithm was repeatedly 20000. This solution is the only one of all the solutions worse than the greedy algorithm. As far as the calculation times are concerned, the times are clearly increasing exponentially with a larger number of objects.

Branch & Bound, which has the longest solution times, clearly stands out from the compared algorithms. It has the best results on average, but in real time applications, we require different algorithm.

The greedy algorithm and QFLA had quite similar results in testing, but the greedy algorithm has always found a slightly better solution in a shorter average time. The only, but in our case very important advantage of QFLA, is that it takes into account also the amount of needed reallocations, and it tries to find the solution in the way that minimises number of reallocations will be needed.

6. Conclusion

We described the basic principles of multimedia sessions, mechanisms on quality of service in IP networks, and the principles and advantages of the SDN approach. We described existing SDN protocols and controllers, and we mentioned the related projects that deal with the problems of quality of multimedia services. Furhter we described the problems with load balancing, when the per-packet load balancing approach is useless and we consider whole data flow unsplittable.

We explained the problem of distributing these flows of given required bandwidth between the disjoint network paths of given capacity using the well known Knapsack Problem, where we brought up our own modification of this problem we call Value Independent Multiple Knapsack Problem. We described the known methods on solving various modifications of Knapsack Problem, such as brute force, branch and bound, dynamic programming or greedy algorithms.

There exist several approaches on how to solve the problems related to Online Value Independent Multiple Knapsack Problem, all of the however do not take into account the number of steps required to achieve the improved items-to-knapsack distribution. Since in the real network every step equals a configuration change that may affect QoS, one the the requirements was that the algorithm must look for the improved solution with minimum reallocations (configuration changes).

For this purpose, we took the prototype of QFLA algorithm which has been designed by us in [7], that in case, when new client (flow) cannot be inserted directly into any of the paths, tries to move existing flows to different paths in order to relax one path enough for the new flow to be inserted without exceeding the capacity of the path.

The most important areas for the future work that can be done this project are following: In all the scenarios, we were only dealing with knapsacks and with items, without any other constrain. That means, any of the items can be packed into any of the knapsack if there is free enough capacity (the size matters only). Yet remains unsolved, how would be the solution proposed in this thesis be suitable for the scenarious where the items would have not only their size, but also their price, thus we would need to consider some priorities of packing certain item into the knapsack with higher importance than some other.

Acknowledgements. This research was supported by the Ministry of Education, Science, Research and Sport of the Slovak Republic, Incentives for Research and Development, Grant No.: 2018/14427:1-26C0. The work was also supported by the Operational Program Research and Innovation for the project: Research of advanced methods of intelligent information processing (ITMS code: NFP313010T570), co-financed by the European Regional Development Fund, and the Research and Development Operational Program for the project: Support of Center of Excellence for Smart Technologies, Systems and Services II (ITMS 26240120029), co-financed by the European Regional Development Fund. It was also partially supported by the grants APVV-15-0731, KEGA 011STU-4/2017, and VEGA 1/0836/16.

References

[1] Pisinger, d.: Small coefficients. Accessed Online: http://www.diku.dk/ pisinger/smallcoeff_pisinger.tgz, 2019.

[2] D. Antonín. Řešení problému batohu metodou hrubé síly a jednoduchou heuristikou, 2013. Accessed Online: http://data.antonindanek.cz/paa-du1.pdf.

[3] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. A knapsack secretary problem with applications. In M. Charikar,

K. Jansen, O. Reingold, and J. D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 16–28, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. RFC 2205, RFC Editor, September 1997. http://www.rfc-editor.org/rfc/rfc2205.txt.

[5] E. ŠEMNICKÁ. Praktické řešení úlohy batohu [online], 2008 [cit. 2019-07-08].

[6] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[7] M. Galinski. Multimedia sessions optimization using sip single port. Master's thesis, Slovak University of Technology in Bratislava, 2016.

[8] M. Garcia-Martin. Rfc 4083: Input 3rd-generation partnership project (3gpp) release 5 requirements on the session initiation protocol (sip). *Internet Engineering Task Force*, 2005.

[9] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou. Software-defined networking (sdn): Layers and architecture terminology. RFC 7426, RFC Editor, January 2015. http://www.rfc-editor.org/rfc/rfc7426.txt.

[10] X. Han and K. Makino. Online minimization knapsack problem. In E. Bampis and K. Jansen, editors, *Approximation and Online Algorithms*, pages 182–193, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] K. T. B. A. M. T. Hilmi E. Egilmez, S. Tahsin Dane. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. *Koc University Istanbul Turkey*.

[12] P. Hlavaty. Knapsack problem, 2018. Slovak University of Technology in Bratislava, Bachelor's Thesis, Supervisor: M. Galinski.

[13] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[15] T. Michal. Problém batohu, 2009. Accessed Online: https://woq.nipax.cz/sobaka/01_problem_batohu.php.

[16] J. Muranyi. *Optimization of Multimedia Flows in Multipath Networks*. PhD thesis, Slovak University of Technology in Bratislava, 2015.

[17] K. Nichols, S. Blake, F. Baker, and D. L. Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. RFC 2474, RFC Editor, December 1998. http://www.rfc-editor.org/rfc/rfc2474.txt.

[18] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[19] F. Pavkovcek. Load balancing as a modified knapsack problem. Master's thesis, Slovak University of Technology in Bratislava. Supervisor: M. Galinski, In progress, not published yet.

[20] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. RFC 3261, RFC Editor, June 2002. http://www.rfc-editor.org/rfc/rfc3261.txt.

[21] S. S. Seiden. An optimal online algorithm for bounded space variable-sized bin packing. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2000.

[22] A. Shaheen and A. Sleit. Comparing between different approaches to solve the 0/1 knapsack problem. *International Journal of Network Security*, 16:1–10, 07 2016.

[23] S. S.Kandula, D.Katabi and A.Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2):51–62, 2007.

[24] K. Tomáš. 3. úloha - problém batohu ii., 2005. Accessed Online: https://woq.nipax.cz/sobaka/01_problem_batohu.php.

Selected Papers by the Author

M. **Galinski**, T. Vrtal and I. Kotuliak, "Network Controller Extension for Unsplittable Data Flows in the SDN Environment," 2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA), Starý Smokovec, Slovakia, 2019, pp. 197-203, doi: 10.1109/ICETA48886.2019.9039979.

M. **Galinski**, J. Volko, I. Kotuliak, "Binary Search Tree as an approach on solving Modified Knapsack Problem", Sborník příspevku PAD 2019 – elektronická verze PAD2019, 2019, pp. 39-42, ISBN 978-80-88214-20-5

L. Mastilak, M. **Galinski**, I. Kotuliak and M. Ries, "Improved Smart Gateway in IoT," 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2018, pp. 349-354, doi: 10.1109/ICETA.2018.8572057.