Flow Classification in Delay-Aware NUM-Oriented Wireless Mesh Networks

Przemyslaw Walkowiak, Maciej Urbanski, Mateusz Poszwa, Radoslaw Szalski
Institute of Control and Information Engineering
Poznan University of Technology
Poznan, Poland
Email: {przemyslaw.walkowiak, maciej.urbanski, mateusz.poszwa, radoslaw.szalski}@put.poznan.pl

***Abstract*—The Network Utility Maximisation (NUM) framework is one of the most widely investigated approaches for designing the resource management system for wireless mesh networks. In order to perform the NUM-oriented per-flow network resource management, data flows have to be recognised and classified. Relevant solutions that are constituents of the state-of-the-art NUM frameworks are insufficient, since they are able to differentiate between various network flows only according to the transport layer protocol used. The paper describes improvements introduced to an existing DANUM System (DANUMS) implementation. They provide means for flexible flow classification enabling more accurate utility estimation for more diverse types of flows. The solution improves DANUMS' ability to assign appropriate utility functions suitable for different types of traffic. The experiments show that the enhanced framework enables improving the performance of the DANUMS.**

Keywords-DANUMS; wireless mesh networks; Network Utility Maximisation; traffic classification

I. INTRODUCTION

As the wireless network access is becoming more and more widespread, the needs of its users grow. When the demands exceed the network's capacity, not all flows can be served equally well. NUM [1] aims to manage network resources in an optimal way, which ensures maximal satisfaction of network users. DANUMS [2] provides NUM functionality by identifying and classifying flows, as well as by measuring and acting on changes of their utility. It is an application of and an enhancement to the NUM model providing delay awareness. The framework improves the fairness of the resource allocation among flows with different delay requirements. DANUMS has been designed to work in wireless mesh networks [2].

DANUMS is a part of the architecture developed within the *Carrier-grade delay-aware resource management for wireless multi-hop/mesh networks* (CARMNET) project [3] referred to as CARMNET architecture. This architecture consists of multiple components (see Figure 1): a routing component in the form of Optimised Link State Routing Protocol daemon (OLSRd), a custom SIP User Agent integrated with a Linux Loadable Kernel Module (LKM), a user interface (WebUI) and an IP Multimedia Subsystem (IMS) platform. SIP User Agent is responsible for asynchronous communication between LKM and the IMS. The user interface is a WWW application that allows users to bind utility functions to various types of traffic. The WebUI also provides insight into statistics about transmitted traffic and network usage cost.

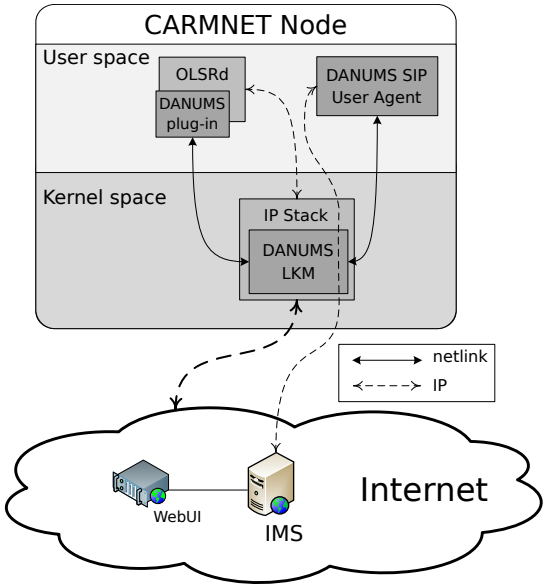

Figure 1. Architecture of the CARMNET network [3].

In NUM, network resource allocation is performed on per-flow basis. The flow classification is necessary for recognition of flow-to-application associations. Once the flow's category is identified, its utility can be computed according to its characteristics. Bidirectional flows, which can be referred to as request-response flows, also should be properly identified as their performance may affect each other's utility. TCP flows' rate and responsiveness depend on the timely delivery of ACK segments. Thus, it is desirable to prioritise request and response flows similarly in order to avoid unnecessary congestion window reductions due to excessive ACK segments queueing. Moreover, recognition of response flows is beneficial from the business perspective, which is important in the CARMNET architecture integrating Authentication, Authorisation and Accounting (AAA) and charging functionality [3]. Each node that generates traffic should have control over both request and response flows' *virtual prices* (see Section III).

The basic DANUMS implementation [2] differentiates flows according only to the transport layer protocol. While effective for basic scenarios, this solution is not versatile enough to control media streams with different needs. This paper describes a practical implementation of a more robust

 ISBN: 978-1-61208-299-8

method for flow classification and recognition of response flows.

The paper structure is as follows. Related work is described in Section II. Section III introduces the Delay-aware Network Utility Maximisation (DANUM) model and provides its main purposes. Section IV discusses the problem of flow classification. Methods of recognition of response flows are described in Section V. Experiments and their results are described in Sections VI and VII, followed by conclusion in Section VIII.

II. Related work

Many NUM systems determine utility of flows according to the flows' throughput only [4]–[7]. Such an approach is not sufficient to effectively measure the utility of delay-sensitive flows. DANUMS, on the other hand, takes the delay into consideration as well [2].

The work [6] presents a policy ensuring constant worst-case delay, however, the utility function used in a maximisation scheme is based only on throughput. In [4], it is assumed that the mechanism based on providing inelastic flows with bandwidth exceeding their injection rate ensures satisfying their end-to-end delay requirements. The framework presented in [5] considers only TCP flows. Solutions presented in [7] require modification of a network card driver, which does not comply with the basic assumptions of CARMNET [3].

In order to estimate the flow's utility accurately, its type has to be determined. Advanced techniques, such as payload examination [8], machine learning algorithms [9], [10] or solutions based on neural networks [11], have been used for this purpose. However, DANUMS is also aimed at serving mobile nodes, which may be power-constrained. For this reason, classification methods for DANUMS should not be computationally complex.

III. DANUM System

The aim of the DANUM model is to provide an optimal packet scheduling policy regarding the maximisation of the network users' satisfaction. It targets the maximum of the network utility (a sum of utility of all flows within the network):

$$\max \sum_{r \in S} U_r(x_r, d_r), \tag{1}$$

where S denotes a set of flows within the network; x_r – rate of flow r; d_r – delay of flow r; U_r – the utility function of flow r. In other words, DANUMS aims at solving the NUM problem in a delay-aware way.

The relation between measurable flow transmission quality parameters and its utility is modelled by means of a utility function. Each function corresponds to flows of a given type or, more precisely, to flows with specific network requirements. In DANUMS the utility is determined not only according to the flow's throughput, but also to its end-to-end delay. Each flow may have a distinct utility function since it may prioritise different network performance parameters. Assigning utility functions to flows is a task of the Flow Classifier described in Section IV.

It has been proven that the Max-Weight Scheduling (MWS) algorithm is a solution to the standard throughput-oriented NUM problem formulation [12]. The DANUMS applies the MWS algorithm to virtual queue levels in order to determine the next flow queue to transmit a packet from.

A virtual queue is defined as a product of flow's packet backlog level and a *virtual price* of a single packet. Packet's virtual price is a value of the derivative of a utility function assigned to the flow. In other words, the more utility a flow would gain from improving its network performance parameters (e.g., by lowering its delay), the higher is the virtual price. The virtual price plays an important role in packet scheduling as well as influences the cost of CARMNET network usage.

DANUMS LKM is responsible for packet queueing, measuring flows' characteristics, as well as applying utility functions and the backpressure scheduling algorithm. Packets scheduled by DANUMS are relayed to the network interface output buffer, the level of which is kept low as a result of using Layer-2 Queue level Estimation [13]. Possible routes acquisition and explicit signalling of virtual queues is done through modified OLSRd. The details concerning the DANUM and its implementation can be found in [2], [13]–[15].

IV. Flow Classification Framework for DANUMS

Flows can be divided into two general groups: throughput-demanding and delay-sensitive. They roughly correspond to the TCP- and UDP-based traffic, respectively. Such a division was used in DANUMS prior to implementation of the Flow Classifier presented in this section. However, for some scenarios, this simple classification is insufficient. The application of the classification subsystem in DANUMS allows a more fine-grained flow classification. The more traffic classes a given NUM system is able to recognise, the more accurately the utility functions may reflect the requirements of different types of traffic.

Flow Classifier used in DANUMS is a cascade of simple filters (see Figure 2). Flow's properties are checked against rules defined for each of the filters. Each rule is a pair composed of filter-specific constraints and a flow type. If any of the rules matches the flow, i.e., the flow's properties meet the rule's constraints, the classification yields a flow type assigned to the matching rule as a result. An unspecialised utility function is assigned if all the filters fail to classify the flow. Using this utility function is equivalent to setting a constant virtual price for each packet, i.e., excluding the flow from the evaluation of NUM.

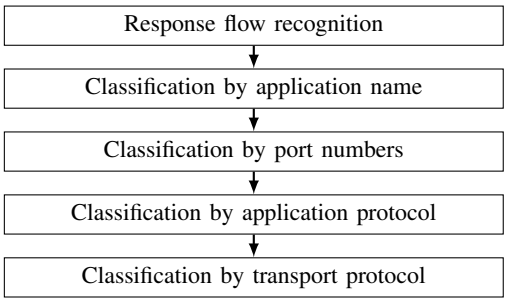

Figure 2. Overview of the Flow Classification Framework for DANUMS. Arrows denote the order of applying the filters.

 ISBN: 978-1-61208-299-8

A. Classification by a transport layer protocol

Classification by the transport layer protocol is the simplest solution, nevertheless, it lacks the ability to differentiate specific uses of each of the protocols. An example of different TCP protocol applications can be provided: the comfort of Web navigation depends on low delay, whereas comfort of sending an e-mail does not.

B. Classification by source and destination port numbers

A common way to determine flow's type is to assume that the traffic of a certain service or an application is bound to a predefined port. The advantage of this approach is that the port numbers are already known to the classifier, so no additional processing is needed to determine them. However, this assumption usually holds true only for services using the *well-known ports* provided they were not configured to use non-standard ports. Moreover, some application layer protocols can be used with random ports, or even different transport protocols. For the given reasons, the method assuming that the traffic of a certain service or an application is bound to a predefined port would require constant reconfigurations to ensure an optimal classification of flows.

C. Classification by an application layer protocol

In order to address the limitations of the above-described approaches, classification by the application layer protocol has been considered. The transport layer and network layer protocols' headers do not provide any indication of the application layer protocol used. Therefore, it has to be determined by a direct analysis of the payload data, which is a complex task and should be delegated to an external program such as l7-filter [8]. However, its kernel-space implementation is known to cause problems on SMP-enabled processors [16], and the use of the user-space implementation in critical systems is discouraged by its authors [16]. Due to these disadvantages as well as lack of well tested alternatives, this classification method was not yet implemented.

Even if it was possible to use some version of the l7-filter to classify flows, some of its patterns return false results [17] and the cost of this method is considerably higher than the cost of other methods discussed here. This may have an influence both on performance and energy consumption. The latter aspect is of great importance for the use of mobile nodes, which DANUMS is designed for [2].

D. Classification by filename of sending application

It is possible to use locally available information to automatically determine the filename of the application sending the flow. Moreover, it is much more probable that the node's user is able to state the name of the application she uses, than that she is able to determine the port numbers bound to flows sent by the application. Thus, this method supports associating flows with desired utility functions chosen by the user through the WebUI.

On the other hand, some applications send many types of flows simultaneously. A VoIP client, for example, is responsible for setting up sessions, sending multimedia streams and reporting statistics by means of SIP, RTP and RTCP protocols respectively. It is essential to choose a utility function meant for the protocol whose transmission quality impacts the application usability the most (in this case – RTP). In this approach, heterogeneous flows sent by a single application are assigned the same utility function, which is, obviously, not the optimal assignment.

E. Combining the classification methods

Each of the aforementioned methods has disadvantages, which render each of them insufficient when used separately. Some of the disadvantages may be avoided or minimised by combining several classification methods.

Response flows are already classified by their destination nodes (referred to as "owners"). They are treated specially and should be filtered out first.

The most desired classification criterion is the application layer protocol used for the flow payload. Unfortunately, as discussed above, it is computationally expensive to determine. For this reason, classification based on regular expressions should be preceded by less complex methods. Classification by the sending application filename is unreliable in case of applications that send multiple flows of various types. Nevertheless, it reflects the end user's needs most strictly, so it should be the first filter for request flows. Classification by the transport layer protocol can serve as a fallback mechanism for flows which fail to be classified by any other criteria. The final order of filters is illustrated in Figure 2.

V. Recognition of request and response flows

A request flow originating from one node and addressed to the other is usually accompanied by a response flow transmitted in the opposite direction. These two flows provide a duplex point-to-point connection between two nodes.

However, as far as DANUMS is concerned, a node which initiates a request flow should also be charged for the response flow. Such a node is marked as the "owner" of both flows. Information about flow "ownership" is propagated by OLSRd along the flow's path and allows the flow classifier to differentiate request and response flows.

If the flows were considered separately, a utility function would be assigned to each of them by the source node. Such a scheme would have undesirable consequences. Let us consider a scenario in which the requesting node assigns a utility function demanding a very low delay to a certain type of flow, but the responding node user does not require such low delay for that type of flow. Even though the requests could be sent quickly, thanks to the assigned utility function, the perceived utility of network may not be satisfying for requesting node's user, as the response flow may fail to be prioritised by the replying node.

Another example of undesirable consequence of mismatching utility functions is related to the CARMNET business model [3]. When the source node is outside the CARMNET network, the destination node is charged for the transmission of the flow. Were the flows considered separately, their utility would be decided by the node at the border of the CARMNET network (an Internet-sharing node), which forwards the flow to the destination node inside the CARMNET network. Since the

 ISBN: 978-1-61208-299-8

utility of a flow is closely related to its virtual price, it should not be set by a node other than the one which is charged for transmitting the flow.

For the aforementioned reasons, a mechanism for recognising response flows has been implemented that enables two methods for response flows' virtual price adjustment. Their performance has been evaluated in Section VI.

1) Copying the request flow's virtual price: The information about the virtual price of each flow, encapsulated in the Queue Report Message (QRM) packets [15], is propagated through the network along the flow's path. Therefore, it is available for the replying node and can be applied to the response flow. This simple method does not require the replying node to analyse the flow to which virtual price is applied or to fetch requesting node's profile. However, the potential issues this method introduces need to be considered.

In DANUMS, characteristics of a certain path are measured on per-flow basis. When the network is congested, throughput and delay measured at both endpoints of a flow may differ significantly. Undesired consequences of using this method may also arise when request and response flows' requirements differ. Such situation is illustrated by the first experiment described in Section VI-B1.

2) Calculating the request flow's new virtual price by applying a utility function at the replying node: The other method of controlling the virtual price of a response flow is to assign a utility function chosen by the requesting node. This information can be retrieved from IMS by sending the *Get Profile* message [3]. While flows' *owner* node can assign the same utility function to both request and response flows, its parameters will differ from those measured on each endpoint of respective flows. Network characteristics perceived at both nodes may be influenced by factors such as congestion, asymmetry of links or choice of routes. Therefore, it is more accurate to calculate the flow's virtual price at the transmitting node, whether or not it is the flow's "owner".

VI. Experiments

A. Testbed

Experiments have been performed in a wireless network testbed called wnPUT [18]. The wnPUT testbed deployment approach has been influenced by the Distributed Embedded System Testbed (DES-Testbed) [19] architecture. Currently our testbed consists of about 20 PC-class machines, each equipped with two network interfaces, wired and wireless. The wired network is used for out-of-band management. The wireless connections are used for experimentation purposes only. Each testbed node runs a Debian GNU/Linux distribution.

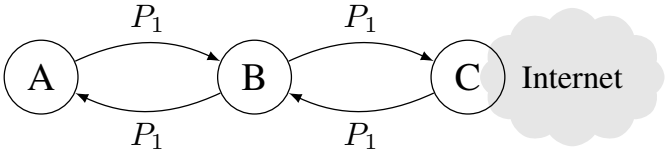

Figure 3. Copying the flow's virtual price. **A** – CARMNET node, **B** – CARMNET Relaying node, **C** – CARMNET Internet Sharing node, $\mathbf{P_1}$ – Virtual price calculated at Node A

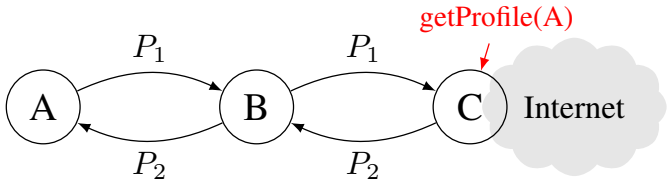

Figure 4. Applying utility function at replying node. **A** – CARMNET node, **B** – CARMNET Relaying node, **C** – CARMNET Internet Sharing node, $\mathbf{P_1}$ – Virtual price calculated at Node A, $\mathbf{P_2}$ – Virtual price calculated at Node C

The testbed allows for an easy and automated experiment execution. It handles parsing of the experiment description files, setting up wireless network, configuring topology, executing specified commands and, finally, gathering results. Experiments are described using the scenario files written in XML. The syntax of those files is an extension of the DES-CRIPT [20] language used in DES-Testbed [19]. The unified format of experiment description files has many benefits such as portability and expressiveness, as well as allowing the experiments to be performed on different testbeds. Although the testbed framework was heavily modified, the phases of experimentation remain as defined in the previous work [18], [19]. Status of performed commands and the DANUMS LKM is acquired in real-time by means of `rsyslogd` and visualised by a monitoring system in order to make the analysis easier.

Due to space constraints, currently all nodes are directly connected to each other in a wireless mesh network sharing the same collision domain. Taking the wireless networks characteristic into account, in which even nodes separated by 2 hops might interfere with each other, it has been assumed that 2-hop topology can be simulated by blocking traffic on software level. Thus, the wnPUT testbed framework allows user to specify desired topology, which is attained with `iptables` rules generated automatically during experiment initialisation.

B. Experiment scenarios

In order to illustrate the benefits of recognising response flows, two experiment scenarios were prepared. Their purpose is to show possible undesirable outcome of miscalculating flows' virtual price, which may result from taking wrong measurements under consideration. For both scenarios, the linear topology consisting of three nodes was used (see Figure 3).

In both experiments, Node A initiates communication by sending data to Node C. Since Nodes A and C are not directly connected, Node B forwards the flow in order to provide connection between them. Node C responds with a reverse flow addressed to Node A.

In the first experiment, Node C marks the destination of the response flow (Node A) as its owner in order to inform relaying nodes (Node B) that the flow's virtual price has to be copied from Queue Information Block (QIB) blocks corresponding to the request flow. In the second experiment, Node C fetches the profile of Node A and applies the utility function corresponding to the served flow in order to determine response flow's virtual price. These two experiments correspond to the methods of adjusting the virtual price described in Section V.

 ISBN: 978-1-61208-299-8

1) Experiment 1: The first experiment illustrates a possible undesired consequence of using the method based on virtual price copying described in Section V-1. This scenario models a VoIP call (labelled as "RTP C→A" in Figure 5 and Figure 6) made during a HTTP file transfer (labelled as "HTTP C→A"). The experiment starts with a HTTP request (labelled as "HTTP A→C") sent from Node A to Node C. Its size was artificially enlarged to 5MB for experiment clarity purposes. Node C responds to the requester with a 25MB HTTP response, one second after receiving the request. While the response is being transmitted, Node C initiates a 35-second long RTP flow at constant rate of 2.5Mbit/s addressed to Node A. The experiment ends when both flows originating from Node C are terminated.

2) Experiment 2: In the second experiment, timing and characteristics of flows are the same as in Experiment 1. The only difference is the virtual price of the response flow, which is now calculated at Node C according to the method described in Section V-2.

C. Utility functions assignment

For RTP flows, the following utility function was used [15]:

$$U_U(x,d) = \frac{w_u}{\left(1+e^{a(x_t-x)}\right)\left(1+e^{b(d-d_t)}\right)} \quad (2)$$

where $w_u = 10^6$ is an aggressiveness parameter; $x_t = 2.5 \cdot 10^6$ and $d_t = 300$ denote desired bitrate and delay respectively; $a = b = 0.01$ are parameters controlling the slope of utility function; The utility function assigned to RTP flows aims to maintain their delay below 300ms, i.e., the flow's virtual price peaks when its delay equals 300ms.

Utility function used for HTTP flows is as follows:

$$U_T(x,d) = w_t log(x) \quad (3)$$

To ensure desired assignment of the utility functions, appropriate classification rules have been configured. They were based on the application filename criterion (discussed in Section IV-D).

VII. Results

Virtual price values in both experiments indicate that the flow classifier was able to differentiate flows correctly. Delay-sensitive utility function has been assigned to the RTP flow (which may be observed in Figure 5 near $t = 55$s, when the flow's virtual price drops due to high delay) and throughput-oriented utility function has been assigned to HTTP flows (whose virtual price rises when its rate drops considerably).

The virtual price values also show that the response flows have been properly recognised by the classifier. Therefore, methods for response flows' virtual price adjusting described in Section V could be applied and evaluated.

In Experiment 1, the transmission of the RTP flow ended prematurely, because the HTTP response flow had its virtual price set inappropriately high. The virtual price was calculated at Node A for a low-throughput sequence of TCP acknowledgements and was not meant to be used with high-throughput TCP flows. The HTTP response flow overwhelmed the RTP flow despite having lower product of utility derivative and packet queue level, causing the RTP flow's delay to rise beyond the d_t threshold value, which led to lowering the RTP flow's virtual price. Such a behaviour is an undesirable outcome of copying the virtual price calculated for accompanying request flow when its characteristics differ considerably.

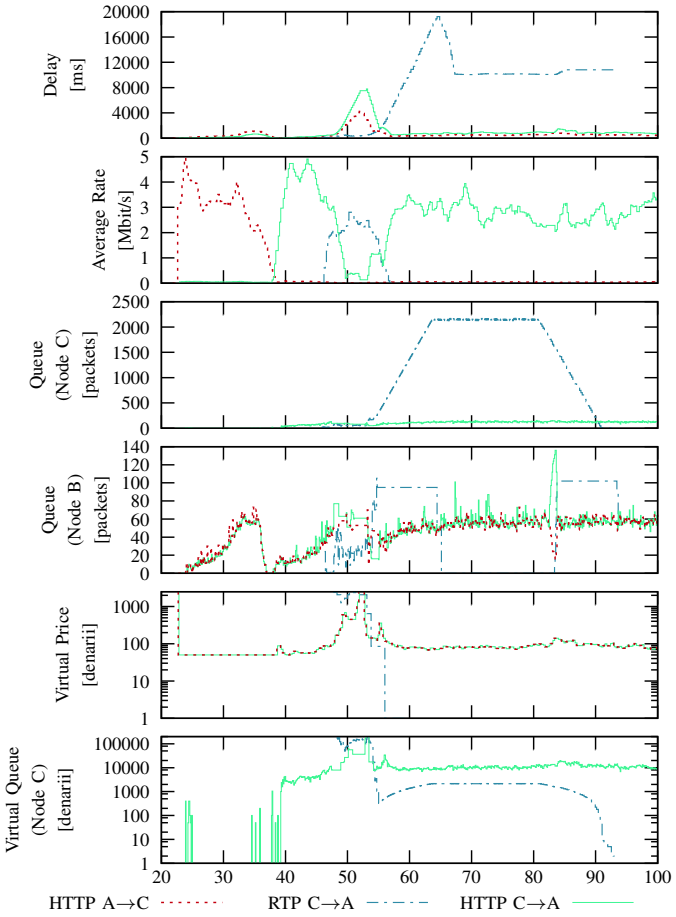

Figure 5. Results of Experiment 1.

In Experiment 2, the virtual price of HTTP response flow was calculated locally on Node C, resulting in much lower virtual price of the response flow since the derivative of HTTP flows' utility declines with the growth of throughput. As a result, the virtual queue level of the RTP flow was high enough to successfully compete with HTTP flows while maintaining a satisfiable delay.

VIII. Conclusion and Future Work

Two ways of dealing with request/response flows were presented. The first one is based on copying the flow's virtual price between requesting and replying nodes, the second forces utility recalculation on both nodes. The copying-based approach is a less demanding solution since it involves sending the virtual price using CARMNET-specific protocol. However, as the experiments showed, applying this simplification may destabilise DANUMS. On the other hand, the virtual price recalculation using actual parameters at the replying node, results in a better stability of the system. Nonetheless, the local resource requirements are higher, as this method requires

 ISBN: 978-1-61208-299-8

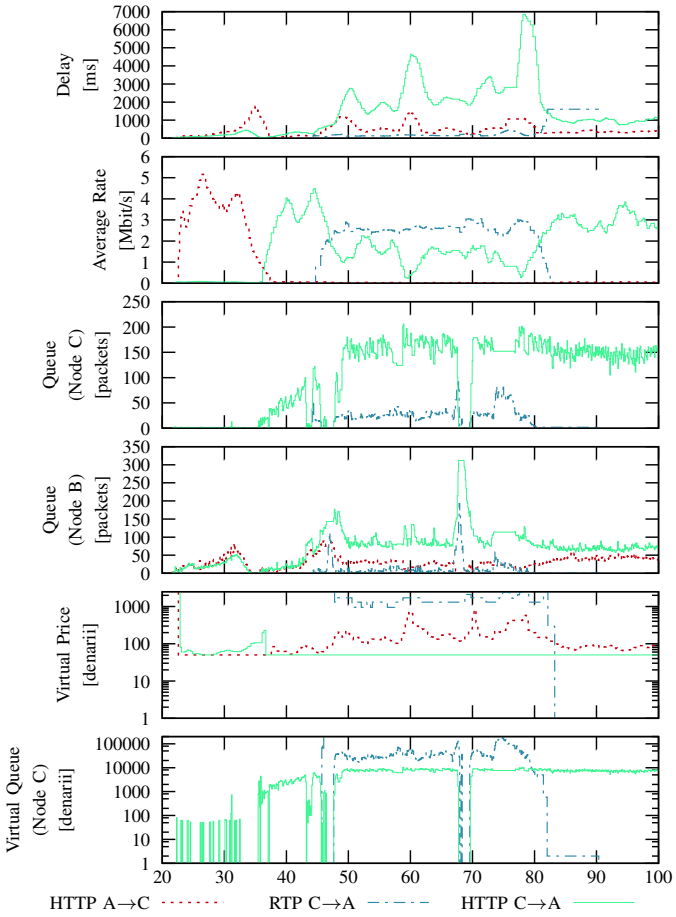

Figure 6. Results of Experiment 2.

nodes to acquire and store users' profiles, as well as to perform additional calculations. Most importantly, after all, this approach achieves the highest stability.

The implementation of the flow classifier improved flexibility of DANUMS by enabling the use of utility functions adapted to specific applications' requirements. However, the classification could likely be enhanced even further by introducing more reliable or more fine-grained, but still power-efficient (in terms of the battery power consumption caused by necessary computations) filters. Adding the possibility of combining multiple criteria into a single rule may also be beneficial to the quality of flow classification.

ACKNOWLEDGEMENT

Supported by a grant from Switzerland through the Swiss Contribution to the enlarged European Union (PSPB-146/2010, CARMNET).

REFERENCES

[1] F. Kelly, "Charging and rate control for elastic traffic," European Transactions on Telecommunications, vol. 8, no. 1, Jan. 1997, pp. 33–37. [Online]. Available: http://doi.wiley.com/10.1002/ett.4460080106

[2] A. Szwabe, P. Misiorek, and P. Walkowiak, "Delay-Aware NUM system for wireless multi-hop networks," in European Wireless 2011 (EW2011), Vienna, Austria, Apr. 2011, pp. 530–537.

[3] M. Glabowski and A. Szwabe, "Carrier-Grade Internet Access Sharing in Wireless Mesh Networks: the Vision of the CARMNET Project," The Ninth Advanced International Conference on Telecommunications, Jun. 2013, in print.

[4] U. Akyol, M. Andrews, P. Gupta, J. D. Hobby, I. Saniee, and A. Stolyar, "Joint scheduling and congestion control in mobile ad-hoc networks," in The 27th IEEE International Conference on Computer Communications (INFOCOM 2008), Apr 2008, pp. 619–627.

[5] B. Radunović, C. Gkantsidis, D. Gunawardena, and P. Key, "Horizon: Balancing TCP over multiple paths in wireless mesh network," in Proceedings of the 14th ACM international conference on Mobile computing and networking, MobiCom 2008, 2008, pp. 247–258.

[6] M. Neely, "Delay-based network utility maximization," In Proc. IEEE INFOCOM 2010, 2010, pp. 1–9.

[7] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: Practical differential backlog congestion control for wireless networks," in The 28th IEEE International Conference on Computer Communications (INFOCOM 2009), Apr. 2009, pp. 262–270.

[8] Application Layer Packet Classifier for Linux. [Online]. Available: http://l7-filter.clearfoundation.com [retrieved: Jun., 2013]

[9] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," ACM SIGCOMM Computer Communication Review, vol. 36, no. 5, Oct. 2006, pp. 5–16. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1163593.1163596

[10] I. Anantavrasilp and T. Schöler, "Automatic flow classification using machine learning," in Software, Telecommunications and Computer Networks, 2007. SoftCOM 2007. 15th International Conference on. IEEE, 2007, pp. 1–6.

[11] M. Ilvesmäki, M. Luoma, and R. Kantola, "Flow classification schemes in traffic-based multilayer IP switching–comparison between conventional and neural approach," Computer Communications, vol. 21, no. 13, Sep. 1998, pp. 1184–1194. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366498001637

[12] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," IEEE Transactions on Automatic Control, vol. 37, no. 12, Dec. 1992, pp. 1936–1949.

[13] A. Szwabe, P. Misiorek, and P. Walkowiak, "Protocol Architecture for DANUM Systems," Poznan University of Technology, Institute of Control and Information Engineering, Tech. Rep. IAII-595, Apr. 2010.

[14] A. Szwabe, "DANUMS: The First Delay-Aware Utility Maximization System for Wireless Networks," in Proc. of NEM Summit - Towards Future Media Internet. NEMS 2009, Sep. 2009, pp. 59–64.

[15] A. Szwabe, P. Misiorek, and P. Walkowiak, "DANUM System for Single-hop Wireless Mesh Networks," In Proceedings of 2010 International Conference on Future Information Technology (ICFIT 2010), volume 1, Changsha, China, IEEE Press, Dec. 2010, pp. 365–369.

[16] Application Layer Packet Classifier for Linux – Getting started. [Online]. Available: http://l7-filter.clearfoundation.com/docs/readme#getting_started [retrieved: Jun., 2013]

[17] L7-filter supported protocols. [Online]. Available: http://l7-filter.sourceforge.net/protocols [retrieved: Jun., 2013]

[18] A. Nowak, P. Walkowiak, A. Szwabe, and P. Misiorek, "wnPUT Testbed Experimentation Framework," in Distributed Computing and Networking, ser. Lecture Notes in Computer Science, L. Bononi, A. Datta, S. Devismes, and A. Misra, Eds. Springer Berlin Heidelberg, 2012, vol. 7129, pp. 367–381. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25959-3_27

[19] The Distributed Embedded Systems Testbed (DES-Testbed) Webpage. [Online]. Available: http://www.des-testbed.net [retrieved: Jun., 2013]

[20] M. Güneş, F. Juraschek, B. Blywis, and O. Watteroth, "DES-CRIPT - A Domain Specific Language for Network Experiment Descriptions," in Next Generation Wireless Systems 2009 – Proceedings, N. Chilamkurti, Ed., Mar. 2010.

 ISBN: 978-1-61208-299-8