QUALITY OF SERVICE PACKET MARKING DRIVER FOR WINDOWS BASED TERMINAL DEVICES

Radko Krkos

Doctoral Degree Programme (1), FEEC BUT

E-mail: krkos@phd.feec.vutbr.cz

Supervised by: Vit Novotny

E-mail: novotnyv@feec.vutbr.cz

Abstract: This paper describes the design and implementation of a driver for network traffic marking to ensure Quality of Service provision on Microsoft Windows based terminal devices. Requirements are specified by analysis of the intended usage and considering details specific to this exact type of software driver. Key parameter in design is the least possible impact on network traffic performance and the running system itself. DSCP marking is performed using commonly used criteria based on network and transport layer metadata; source and destination addresses and ports are used. In addition to the driver, user mode application programming interface for marking rule specification and driver management followed by a simple application using this interface are designed so the proposed system as a whole completely ensures DSCP marking of network traffic.

Keywords: QoS, driver, Windows, DSCP, marking

1. INTRODUCTION

Quality of Service traffic marking is typically used in autonomous systems at the Internet service provider level and up, in the Internet backbone system. Traffic is marked at border gateways and the queuing rules are applied at intermediate routers, but the end-to-end support is rarely seen. The reason is that in typical applications, the terminal devices and access network is managed by distinctive authorities and for small customers, no level of trust is established between terminal device administrator and service provider administrator. In special applications, where terminal devices and network infrastructure is administered by the same authority, this traditional model could be altered by moving the traffic marking to terminal devices. Support for this is limited in the Windows family of operating systems [1]; therefore a custom solution is proposed to accomplish this goal.

Windows driver development is described in [2] and architecture of modern Windows operating systems is described in [1]. Deep study of this architecture is essential to gather required understanding to develop kernel mode software. Network stack has been rewritten from scratch in Windows kernel version 6.x (Windows Vista, Windows 7), so the approach to writing network drivers is different than in earlier versions [3]. The new network stack natively supports IPv6 in addition to IPv4 and also adds new interfaces such as Windows Filtering Framework, designed especially to simplify and speed up traffic filtering and header and data payload manipulation [3].

There are two significant existing solutions for QoS provisioning in Windows kernel version 6.x, Quality Windows Audio Video Experience (qWave) [4] and Policy-based QoS [5]. Both have their limitations. While qWave is designed for wireless home audio and video streaming and is disabled in enterprise applications, Policy-based QoS is used in enterprise systems and requires a domain joined Windows computer to work. Proposed solution is targeted to work in all scenarios, but lacks some special functionality of both systems. Policy-based QoS allows DSCP marking by originating program in addition to options provided by presented solution, but on the other hand performs marking only for TCP or UDP protocols. Then qWave, as a system supporting streaming over unstable channel, provides live quality measurement, which is not present in proposed solution.

2. DRIVER DESIGN AND IMPLEMENTATION

Driver development is divided into four separated but overlapping parts:

- Requirements specification;
- Architecture design;
- Implementation;
- Support software design and implementation.

2.1. DRIVER REQUIREMENTS SPECIFICATION

The driver has to be able to assign appropriate DSCP marks to network traffic belonging to correct network flows. For unique representation of network flow the sum of these parameters can be used:

- Source network layer address (source IP address);
- Destination network layer address (destination IP address);
- Transport layer protocol;
- Source transport layer port (for protocols using ports on transport layer);
- Destination transport layer port (for protocols using ports on transport layer).

Some of this information can be omitted, when a more general rule is desired. Integral part of every rule is the DSCP mark value that should be assigned to network traffic belonging to the flow that the rule matches. In fact, the DSCP mark value is the only mandatory part of rule, effectively creating the default marking rule.

The case when rule matches multiple flows must be handled. The way used by the driver is that rules are tested in order they were entered and first matching rule is used to mark the flow. Therefore the order of rules is very important and must be considered when rule set is designed. More specific rules should be granted priority before more general ones.

2.2. DRIVER ARCHITECTURE DESIGN

Based on the requirements specification, driver architecture has been proposed. Simplified model of the proposed architecture is displayed in figure 1.

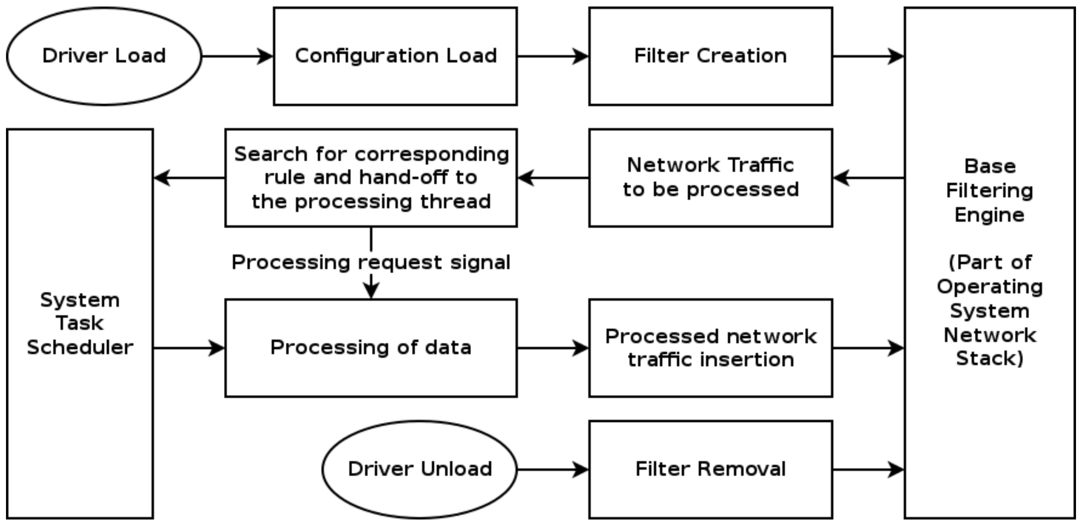

Driver Load
Configuration Load
Filter Creation
Base Filtering Engine (Part of Operating System Network Stack)
Network Traffic to be processed
Search for corresponding rule and hand-off to the processing thread
System Task Scheduler
Processing request signal
Processing of data
Processed network traffic insertion
Driver Unload
Filter Removal

Figure 1: Simplified model of proposed driver architecture

Important parts required for network driver according to the Windows Filtering Platform are [2]:

- Initialization function called on the driver load, named DriverEntry();

- Unloading function called on stopping the driver, named DriverUnload();
- Classification function called to process the traffic if a rule is matching, named ClassifyFn();
- Notification function called to inform the driver about events, named NotifyFn().

Except these mandatory functions there are other support functions used to simplify the code and improve its logical structure. Management functions are separated from the filtering, manipulation and reinsertion logic.

2.3. DRIVER IMPLEMENTATION

Building on requirements specification and architecture proposal, the driver for marking network traffic was implemented using Windows Filtering Platform. Driver implements all required functions described in 2.2 and also some additional support logic. Driver is implemented in C language, standard for Windows kernel mode software. Microsoft build system, same that is used to build the Windows operating system, is used to assemble the driver.

Processing is split in two threads. The primary thread matches flow with a rule; removes packet from network stack queue; signals the worker thread and returns control to operating system to minimize the delay for unprocessed traffic. When the worker thread is scheduled to run, it processes all packets queued by primary thread. During the processing packet header is altered to contain matching DSCP mark and new packet is injected into the network stack. If no other packets are remaining, the thread sleeps until more work is queued and signaled to process.

Driver code consists of three files:

- QoS_drv.c – contains the implementation of driver support logic (rule loading and validation, driver loading and unloading);
- QoS_marking.h – contains common declarations (also used in user mode API);
- QoS_marking.c – contains filtering and marking logic (filter addition and removal, DSCP marking, packet reinsertion,).

2.4. DRIVER PERFORMANCE

One of the basic requirements for a Quality of Service support driver is low impact of the driver on network traffic, either that processed by the driver, or the ignored traffic. Important parameters are:

- Low additional delay of packet due to processing;
- Low jitter due to processing (constant processing time for single flow on one processor);
- No packet loss incurred by the driver;
- Inhibition of packet order change due to processing.

Extra delay is added to the flows matching any of the rules, as can be seen in figure 2, but traffic not matching the rules is processed without additional delay. The processing time for every packet and thus the jitter cannot be known because Windows is not real-time operating system and thus no maximal time to complete an operation is guaranteed. Under normal load however, system response times are minimal and jitter is therefore approximately constant. For traffic not matching any rule, the processing time does not depend on the driver and jitter is not impacted.

Figure 2 displays extreme testing case when the first test was conducted under common load with no demanding applications running. The outcome is that a small delay is added by the driver. The second test displays case when processor was overloaded. Dramatic increase in packet assembly time can be seen, but driver induced relative delay is smaller than is the first case. That means that the driver scales good, driver caused additive delay grows slower than operating system processing caused additive delay.

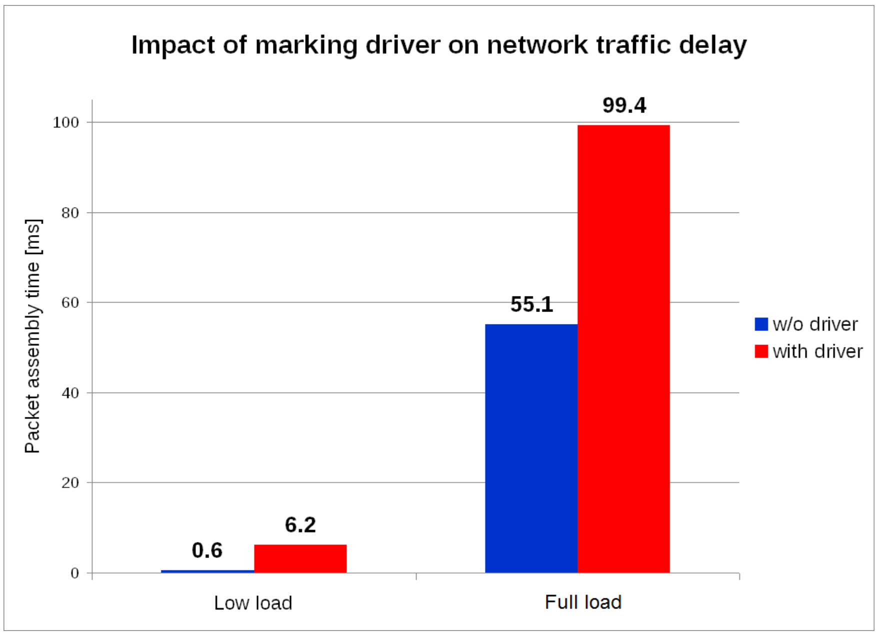


Figure 2: Impact of marking driver on network traffic delay

Packet loss occurs only in extreme conditions, when system resources are depleted. Such highly loaded system would have serious stability issues and could not operate normally. Packet order change is guaranteed not to happen on single processor systems, no guarantee can be made for multiprocessor systems when several processing threads are used. Processing of packets from a flow however happens on the same processor as just context is switched from user mode to kernel mode, so flow is often processed in order even on multiprocessor systems.

3. USER MODE SUPPORT CODE

For the driver to be useful, there has to exist an easy way to manage it. Driver has two anticipated use cases. First one is incorporation in a larger system for Quality of Service support. In this case, driver is used to mark the network traffic with DSCP marks according to rules provided by external application using the management API. The second case is standalone use, when driver marks the network traffic according to internally defined rules. For this case, application for rule management has been created. The rule set can be prepared either locally and written directly to the system registry, or prepared centrally and distributed to the terminal devices in the form of .REG files.

3.1. DRIVER SUPPORT FILES

Driver has separate packages for 32 bit and 64 bit systems. Driver .SYS file containing the driver code is accompanied by an .INF file for installation. The package is not signed, so use on production systems would require changes, but this form of distribution is sufficient for testing purposes.

Additional .REG file containing marking rules for target system can be distributed with package. This has to be added to the registry by administrator. No other files are required for proper installation and function of the driver, but can be included to enable more advanced features.

3.2. MANAGEMENT API

Application interface has been designed to make access to whole functionality of the driver in any programming language of choice. This is enabled by using a DLL file, which can be used in all languages on Windows platform. For C/C++ additional C header file is supplied to simplify use.

Management API functions are divided in these categories:

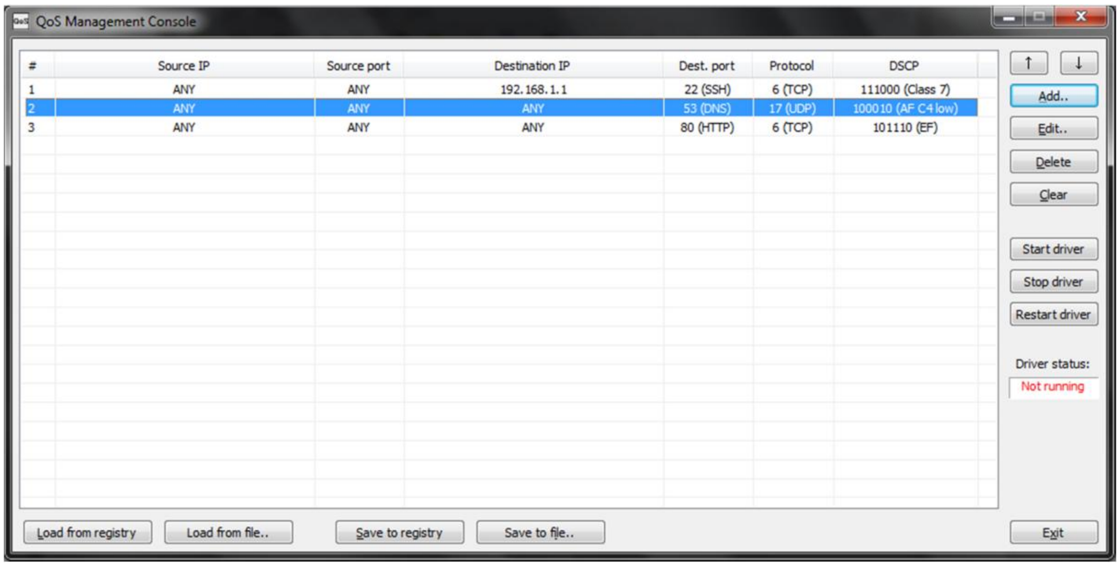
- functions for loading and saving rule sets (to and from both system registry an .REG files);

- functions for rule management (alteration, order change);
- functions for driver runtime control;
- functions for user data processing (input validation);
- functions for reading default values and list of possible values for parameters.

3.3. Management Application

The management application uses all of the support functionality of the API to provide complete and simple to use solution for controlling the DSCP marking driver functionality. It enables user to manage rules and monitor or alter driver status.

The graphical user interface of the application is displayed in figure 3. Application is written in C++ using Microsoft Foundation Classes library.



#	Source IP	Source port	Destination IP	Dest. port	Protocol	DSCP
1	ANY	ANY	192.168.1.1	22 (SSH)	6 (TCP)	111000 (Class 7)
2	ANY	ANY	ANY	53 (DNS)	17 (UDP)	100010 (AF C4 low)
3	ANY	ANY	ANY	80 (HTTP)	6 (TCP)	101110 (EF)

Figure 3: QoS Management Console graphic user interface

4. CONCLUSION

Paper proposes a basic design of driver for filtering network traffic and DSCP marking according to definable rules. Driver enables the move of network traffic classification for Quality of Service support from access network routers to terminal equipment devices, lowering the load on network infrastructure and offering better scalability. Reference implementation has also been developed, including API for controlling the behavior of driver and a simple stand-alone control application.

REFERENCES

[1] Russinovich, Mark E. and Solomon, David A.: Windows Internals, 5th edition. Redmond, Microsoft Press, 2009. ISBN 978-0-7356-2530-3.

[2] Microsoft Corporation: Windows Driver Kit Documentation. 2009.

[3] Microsoft Corporation: Windows Filtering Platform. MSDN. [Online] 2012-03-21. http://msdn.microsoft.com/en-us/library/windows/desktop/aa366510%28v=vs.85%29.aspx

[4] Microsoft Corporation: Quality Windows Audio-Video Experience – qWave. [Online]. 2012-03-21. http://msdn.microsoft.com/en-us/windows/hardware/gg463351

[5] Microsoft Corporation: Policy-based Quality of Service (QoS) [Online] 2012-03-21. http://technet.microsoft.com/en-us/library/dd919203(v=ws.10).aspx