Computer Networks 69 (2014) 19–34

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Adaptive network-traffic balancing on multi-core software networking devices

Tomaž Buh [a,*], Roman Trobec [b], Andrej Ciglič [a]

[a] Iskratel, Ltd., Kranj, Slovenia
[b] Institut Jožef Stefan, Dep. of Communication Systems, Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 15 May 2013
Received in revised form 2 January 2014
Accepted 21 April 2014
Available online 29 April 2014

Keywords:
Networking device
Linux Bridge
Multi-core architecture
Load balancing
Traffic distribution
Packet reordering

ABSTRACT

Software networking devices running on commercial-off-the-shelf hardware offer more flexibility and less performance than high-end, dedicated, networking devices. However, this lack of performance can be compensated, to some extent, by multi-core processors that can manage network packets in parallel. In order to efficiently utilize multi-core architectures, the processing load and the network traffic must be properly balanced to optimize the inter-core communication. Here, we analyze the traffic distribution on a per-packet and per-flow basis and verify the performance of the Linux Bridge networking device. A new, adaptive, traffic-distribution method is proposed, which combines packet-based and flow-based traffic distributions. The method was experimentally validated by two test cases – the "worst-case" scenario, with one dominant flow, and the "backbone-link" scenario, with a large number of flows that have a similar packet rate. In the case of one dominant flow, the performance in traffic throughput is improved by a factor of 2.8 by engaging four processing cores. In the case of a large number of traffic flows, the performance remains similar to the existing flow-based methods.

1. Introduction

Compared to high-end, professional networking devices, which require specialized hardware, software-based networking devices run on commercial-off-the-shelf (COTS) hardware platforms and thus generally offer less performance. On the other hand, they are more flexible and less expensive. As a result software-based networking devices are often used in cloud computing, where they provide an efficient communication between virtual machines by implementing networking functionalities, e.g., bridging. Furthermore, initiatives such as Network Functions Virtualization [1] with the support of the Software Defined Network approach [2] assume the implementation of any network function on a separate virtual machine, where fast communications between the virtualized functions are crucial.

The increased performance of software-based networking devices can be achieved by optimizing the efficiency of the software and by exploiting all the enhancements of the COTS platforms, which are the subject of continuous development. Several approaches, such as multi-level cache, increased system frequency, instruction-level parallelism and branch prediction, have been applied to COTS platforms in order to improve their performance. However, the effects of all these solutions have now reached their limits, with the multi-core architecture being the next step that can offer significantly more performance while consuming less power. To take advantage of this

* Corresponding author. Address: Iskratel, Ltd., Kranj, Ljubljanska c. 24a, 4000 Kranj, Slovenia. Tel.: +386 (0) 4 207 30 96.
E-mail addresses: buh@iskratel.si (T. Buh), roman.trobec@ijs.si (R. Trobec), ciglic@iskratel.si (A. Ciglič).

http://dx.doi.org/10.1016/j.comnet.2014.04.015
1389-1286/

performance, some major adaptations need to be made to the existing networking software, especially with regards to improving its scalability, i.e., the ability to also perform well on a larger number of processing cores.

The Linux operating system (OS) has proved to be a very popular platform for the development of efficient software-based networking devices and so in recent years it has received a lot of interest from the scientific and industrial communities [3] for the following three reasons: (1) the open-source nature of the Linux OS, (2) the licensing policy of the Linux source code, which allows its usage in commercial products, and (3) the availability of the Linux OS for several micro-architectures.

Several initiatives have been undertaken recently on the Linux OS to make the use of multi-core COTS platforms more efficient. The Symmetric Multiprocessing (SMP) Linux kernel (LK) introduced the SMP-aware process scheduler, which is able to distribute efficiently the software processes across the available cores. Unfortunately, this solution does not apply to Linux networking devices, which are based on the interrupt (IRQ) driven mechanism. The network-traffic reception, processing and transmission in the LK are therefore realized as IRQ routines to provide low latency of the network packets. Because of additional mechanisms in the SMP LK, which must provide the protection of shared data, e.g., data locks and inter-process communications, the Linux networking devices running on the SMP kernel may not perform as well as networking devices running on the Uni-processor version of the LK [4,5].

A methodology with the efficient utilization of the multi-core architecture by retaining scalability must provide on-chip parallelism, which can be achieved in two ways: (1) by parallelization of the network-processing tasks and (2) by an efficient network-traffic distribution among the processing cores. The first approach is based on the functional composition concept [6] by compelling a concurrent execution of the software tasks on several processing cores, while retaining cache coherency. However, the parallelized tasks are often interdependent, because they share the same system resources; therefore, an effective method of synchronization between them has to be provided. The benefits of this approach are limited by Amdahl's law. The second approach, used in our work, relies on network-traffic distribution, which is not limited by Amdahl's law, because it achieves parallelism through the domain-decomposition concept [6], by implementing multiple network-packet queues that are used for the traffic distribution. The queues are mapped to available processing cores by using the SMP Affinity mechanism [7], which provides a mapping of the IRQ lines to the selected cores. The traffic distribution can be packet-based or flow-based [8].

The packet-based traffic distribution can balance network packets, irrespective of their contents. Consequently, the network packets of a connection, i.e., the network packets' flow from a particular source to a particular destination, can take different paths through the networking device and thus become reordered, which causes significant problems for some network protocols. On the other hand, the network traffic can be equivalently distributed between different paths by using packet-based distribution, which is often implemented with the round-robin traffic distribution (RRTD) [8]. In this case, each available core is used in turn to process the packets, which means the processing load is equivalently distributed among all the cores. An improved version of the RRTD method is the weighted RRTD [8], which assigns a weight to each processing core, allowing an enhanced distribution of the processing load.

The flow-based traffic distribution balances the network traffic by examining the contents of each network packet and ensuring that packets of the same flow do not use different paths when they cross a network device. A common implementation of this approach is the hash-based traffic distribution (HBTD) [8], where the network packets are assigned to flows according to hashes calculated from the packets' headers. More specifically, a 5-tuple combination of protocol identifier, destination IP address, destination port, source IP address and source port is normally used as an input to the Toeplitz hash function [9], which is used to calculate the packet's hash. Network packets with identical hashes are processed on the same processing core, so the packets cannot be reordered. However, the load distribution is relatively limited, especially if the network traffic contains a small number of asymmetric flows with significantly different packet rates. In this case each high-packet-rate flow is processed on a single processing core and, consequently, the overall utilization of system resources is limited.

By default the Linux OS uses the HBTD approach, which can be implemented entirely in software or can be assisted by hardware engines incorporated into Network Interface Cards (NICs). The software-based implementations, such as the Receive Packet Steering (RPS), Receive Flow Steering (RFS) and Transmit Packet Steering (XPS) [10], are incorporated into the LK, which allows a flexible network-traffic distribution among processing cores with any type of NIC. The behavior of these mechanisms can be modified relatively easily by altering the LK source code. However, if traffic-distribution engines are integrated into the NICs, then mechanisms such as the Receive Side Scaling (RSS) and Extended Message Signaled Interrupts [11] are used to calculate the hashes and distribute the network traffic accordingly. In this case the flexibility is reduced because the control over these mechanisms is limited.

A new adaptive Token-Based Traffic-Distribution (TBTD) method, which contributes to improved performance in the throughput of the packets on multi-core hardware platforms is proposed. It combines the flow-based traffic distribution with the packet-based distribution and is able to adapt to the network traffic. As long as the performance is not affected, the network traffic is distributed among the processing cores on a per-flow basis, similar to the HBTD approach, by using RPS and RFS mechanisms. However, the proposed method allows network packets of the same flow to be processed on several cores if the characteristics of the network traffic limit the utilization of the system resources. The cores that process the packets are carefully selected with respect to the properties of the hardware architecture in order to maximize the throughput and minimize the imposed packet reordering.