University of Alberta

A Cluster-Based, Scalable and Efficient Router

by

Qinghua Ye

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

©Qinghua Ye
Fall 2010
Edmonton, Alberta

Examining Committee

Mike H. MacGregor, Department of Computing Science, University of Alberta

Paul Lu, Department of Computing Science, University of Alberta

Duncan Elliott, Department of Electrical and Computer Engineering, University of Alberta

Liyan Yuan, Department of Computing Science, University of Alberta, University of Alberta

Kwan L. Yeung, Department of Electrical and Electronic Engineering, University of Hong Kong

To my wife, Fan, and my son, Eric
To my parents, Bisheng and Juming

Abstract

A cluster-based router is a new router architecture that is composed of a cluster of commodity processing nodes interconnected by a high-speed and low-latency network. It inherits packet processing extensibility from the software router, and forwarding performance scalability from clustering.

In this thesis, we describe a prototype cluster-based router, including the design of the cluster-based router architecture and the addressing of critical issues such as the design of a highly efficient communication layer, reduction of operating system overheads, buffer recycling and packet packing. By experimental evaluation, we expose its forwarding capacity scalability and latency variance. We also evaluate and analyze the potential hardware bottlenecks of its commodity processing nodes, and present the correlation between the reception and transmission capabilities of an individual port as well as ports on the same bus. We propose an adaptive scheduling mechanism based on system state information to manage the adverse effect of this correlation on the router performance.

We also investigate internal congestion in the cluster-based router. To manage the internal congestion, we propose two backward explicit congestion notification schemes: a novel queue scheduling method and an optimal utility-based scheme. We show the effectiveness of these schemes either by ns-3 simulation, experimental evaluation, or both. We also analyze the stability of the optimal utility-based BECN internal congestion control scheme through theoretical proof, simulation and experimental evaluation.

Acknowledgements

I am profoundly grateful to my supervisor, Dr. Mike MacGregor, for his continuous help and encouragement throughout my graduate studies. His vigorous pursuit of excellence in research, teaching, advising and every other aspect of his academic work is truly inspiring. His kindness and considerateness also made my graduate life enjoyable.

I would like to thank the members of my candidacy examination committee: Dr. Paul Lu, Dr. Duncan Elliott, Dr. Liyan Yuan, and Dr. Mariusz Klobukowski. Their excellent and helpful suggestions and thought-provoking questions made this thesis much more valuable.

I am grateful to all the members in TRLabs and the communication network research group in the department, especially Dr. Vinod Ratti and Luke Chong, for their assistance and friendship during past years.

I would also like to thank my great friends in Edmonton for making Edmonton an enjoyable and memorable place to spend my time doing graduate studies.

Specially thanks to my wife and my parents. They provided an enormous amount of support and made me who I am today. Their confidence and encouragement drives me forward. I dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AIMD: Additive Increase, Multiplicative Decrease

AMD: Advanced Micro Devices Inc.

API: Application Programming Interface

AQM: Active Queue Management scheme

ATM: Asynchronous Transfer Mode

AVQ: Adaptive Virtual Queue

BECN: Backward Explicit Congestion Notification

BGP: Border Gateway Protocol

CA: Channel Adapter

CCT: Congestion Control Table

COTS: Commercial Off The Shelf

CPU: Central Processing Unit

CSMA: Carrier Sense Multiple Access

DDR: Double Data Rate

DMA: Direct Memory Access

DWDM: Dense Wavelength Division Multiplexing

ECN: Explicit Congestion Notification

FECN: Forward Explicit Congestion Notification

Gbps: Giga bit per second

GHz: Giga Hertz

HCA: Host Channel Adapter

HT: Hyper Transport

HW: Hardware

IBUDCOM: InfiniBand Unreliable Datagram COMmunication

IGMP: Internet Group Management Protocol

IP: Internet Protocol

IPD: Inter Packet Delay

IPoIP: IP over InfiniBand

kpps: Kilo Packets Per Second

MHz: Mega Hertz

MIB: Management Information Base

MLD: Multicast Listener Discovery

MPI: Message Passing Interface

$ms(msec)$: Micro Second

MTU: Maximum Transmission Unit

NIC: Network Interface Card

OEM: Original Equipment Manufacturer

OS: Operating System

OSPF: Open Shortest Path First routing protocol

OSV: Operating System Vendor

PCI: Peripheral Component Interconnect

PCI-Express: Peripheral Component Interconnect Express

PCI-X: Peripheral Component Interconnect eXtended

PIM-SM: Protocol Independent Multicast - Sparse Mode

PIM-SSM: Protocol Independent Multicast - Source-Specific Multicast

QoS: Quality of Service

RAM: Random Access Memory

RDMA: Remote Direct Memory Access

RFC: Request For Comments

REM: Random Exponential Marking queue management scheme

RED: Random Early Detection queue management scheme

RIP: Routing Information Protocol

RIPng: Routing Information Protocol - next generation
RN: Remote Network
RTT: Round Trip Time
SDRAM: Synchronous Dynamic Random Access Memory
SNMP: Simple Network Management Protocol
SM: Subnet Management
SMA/GMA: Subnet Management Agent/ General Service Agent
SMP: Symmetric multiprocessing
SMUX: SNMP Multiplexing Protocol
SRAM: Static Random Access Memory
SRP: SCSI Remote Protocol
SW: Software
TCP: Transmission Control Protocol
uDAPL: User Direct Access Programming Library
$us(usec)$: Macro second
VIPL: Virtual Interface Provider Library
VL: Virtual Lane
VOQ: Virtual Output Queue
VRRP: Virtual Router Redundancy Protocol
XCP: Explicit Control Protocol
XORP: eXtensible Open source Routing Platform project
XRL: XORP Resource Locator

Chapter 1

Introduction

The variety and intensity of Internet applications are increasing rapidly: more and more kinds of real-time and data-intensive applications such as video-on-demand and immersive virtual reality (e.g. network gaming) are deployed on the Internet. Routers are expected to support these applications, and must do more than just traditional packet forwarding. They should be able to integrate a wide variety of packet processing functions without compromising the packet forwarding performance, support changes in packet processing functions by reconfiguration without interrupting services, easily extend new processing functions that are unanticipated at the time of design, and scale forwarding performance and numbers of interfaces linearly by simply adding hardware so that users can protect their investment.

However, most commercial routers are based on customized hardware and software. They support a set of fixed, predefined processing functions, but require software/hardware upgrades or outright replacement to add new functionality. Users cannot select from a menu of features, but rather are restricted to a set of images that include pre-selected bundles of features which may not meet the users' exact requirements. Furthermore, navigating through the somewhat tortuous process of image selection can be time consuming and error-prone. Third party extension of functionality is usually completely infeasible.

On the other hand, significant work has been directed toward software routers that use commodity processors. These projects include the Click Modular Router [20], Router Plugins [7], the Extensible Router [18] for the data plane (where a router actually forwards packets), and XORP (eXtensible Open source Routing Platform project)[13], Zebra [16] for the control plane (where a router exchanges routing information). These packages enable various degrees of extensibility and flexibility of packet processing functions. However, using just one PC, it is impossible for software routers to scale across a range of packet forwarding rates [6].

In order to address the performance scalability of software routers and the extensibility of commercial routers, we propose a cluster-based router framework by deploying the software router on a cluster of commodity processors interconnected by a high-speed and low-latency network. The cluster-based architecture has been widely used in the design of high performance supercomputers. With the development of high-speed interconnection networks and CPUs (Central Processing Units), the cluster-based architecture has become very successful in the field of high performance computing because of its scalability, high availability and outstanding performance/cost ratio. Besides the area of supercomputing, the cluster-based architecture is also widely used in web services and other commercial applications. The challenge of building a cluster-based router is to minimize the latency of the interconnection network while maintaining the scalability of its forwarding rate.

As we will show in this thesis, it is possible to build a cluster-based router using high-speed general-purpose CPUs and a high-throughput / low-latency interconnection network. We describe a prototype cluster-based router, including the design of the cluster-based router architecture and the implementations of critical components. By evaluation, we show that this architecture can be scaled to support higher packet forwarding rates.

In order to study the behaviors of individual nodes in the cluster-based router, we evaluate and analyze several potential hardware bottlenecks that may exist on a PC-based router, and show that there is a correlation between the transmission and reception capabilities of an individual network interface card (NIC) or multiple NICs on the same bus because of NIC/bus bottlenecks. To manage the adverse effects of this correlation, we propose an adaptive scheduling mechanism based on system state information, which balances transmission and reception rates and increases the forwarding rates under overload.

As in any distributed system, congestion control is a critical design issue in the cluster-based router. Congestion can affect both the forwarding rate and the latency of packets travelling through the router. If packets are sent too quickly from multiple ingress nodes to an egress node, the egress node will become overloaded and its forwarding capability will be reduced. At this point, packets will be delayed and perhaps even dropped due to the queue overflow at the egress. To address the congestion problem in the cluster-based router, we investigate the congestion schemes that can be applied on the cluster-based router, study their behaviors and analyze their stabilities.

In the rest of this chapter, we review the history of router architecture development as well as the software router, and then introduce the cluster-based router and its benefits. We also introduce the internal congestion problem in the cluster-based router and related work in the area of congestion control. After describing the contributions of this thesis work, we give the organization of the thesis.

1.1 Router architecture

The Internet has developed at an exponential rate, and so has the traffic it carries. This has placed an enormous amount of pressure to continually improve the router performance. The diverse nature of the traffic on the Internet makes the task of improving router performance on the Internet extremely challenging.

In this section, we start by looking at the functionality of routers, and then review the different popular router architectures in use up to now.

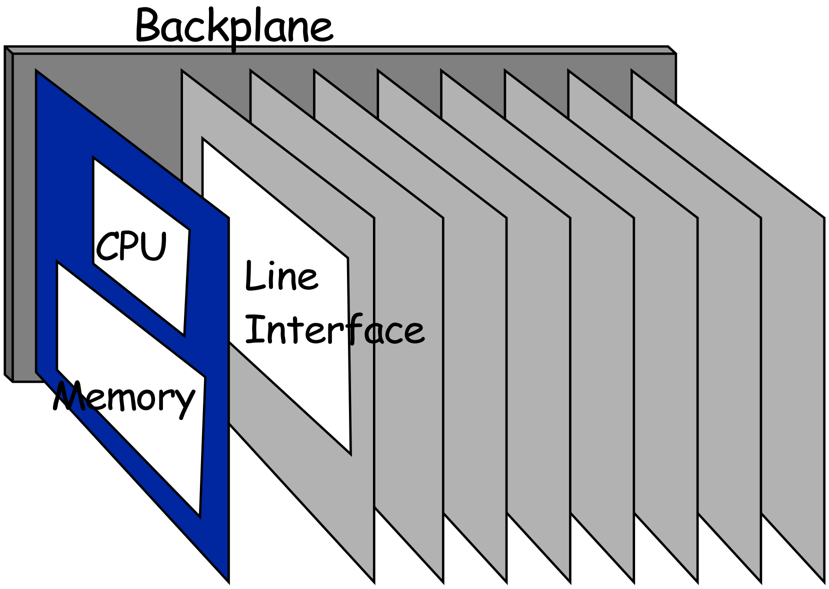


Figure 1.1: General architecture of IP router

1.1.1 Basic IP router components and functionality

Generally, a router consists of the following basic components [2]: several network interfaces to the attached networks, processing module(s), buffering module(s), and an internal interconnection unit (or switch fabric). Typically, packets are received at an inbound network interface, processed by the processing module and, possibly, stored in the buffering module. Then, they are forwarded through the internal interconnection unit to the outbound interface that transmits them to the next hop on the journey to their final destinations. The aggregate packets from all attached network interfaces need to be processed, buffered and relayed. Therefore, the processing and buffer modules may be replicated either fully or partially on the network interfaces to allow concurrent operations.

A generic architecture of an IP (Internet Protocol) router is given in Figure 1.1.

Figure 1.1 shows the basic architecture of a typical router: the controller card (which holds the CPU), the router backplane, and interface cards. The CPU in the router typically performs such functions as path computations, routing table maintenance and reachability propagation. It runs the routing protocols needed in the router. The interface cards consist of adapters that perform inbound and outbound packet forwarding (and may even cache routing table entries or have extensive packet processing capabilities). The router backplane is responsible for transferring packets between the cards. The basic functionality in an IP router can be categorized as: route processing (i.e., path computation, routing table maintenance, and reachability propagation), packet forwarding and router special services.

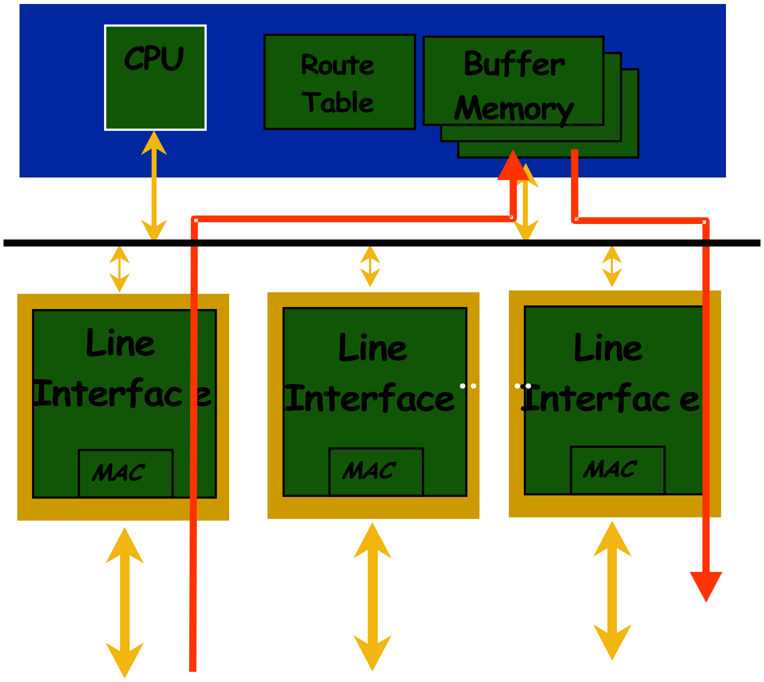


Figure 1.2: Bus-based router architecture with single processor

Router processing includes routing table construction and maintenance using routing protocols (such as RIP (Routing Information Protocol) or OSPF (Open Shortest Path First routing protocol)) to learn about and create a view of network's topology. Updates to the routing table can also be done through management actions where routes are added and deleted manually.

Packet forwarding does the following work: IP packet validation, destination IP address parsing and table look up, packet lifetime control and checksum calculation.

Router special services are becoming more important as new generations of innovative applications such as e-commerce and video streaming are put on the Internet. Special services include packet translation, encapsulation, traffic prioritization, authentication and access services for security purposes.

1.1.2 Bus-based router architecture

The first generation of IP routers is based on software implementations on a single general-purpose central processing unit (CPU). As described in Figure 1.2, these routers consist of a general-purpose processor and multiple network interface cards interconnected through a shared bus. In this architecture, packets arriving at the interfaces are forwarded to the CPU for processing, which determines the next hop addresses and sends them to the appropriate outgoing interfaces. The performance of this router architecture depends heavily on the throughput of the shared bus and on the forwarding speed of the central CPU. All the packet processing and route computing are done solely by CPU,

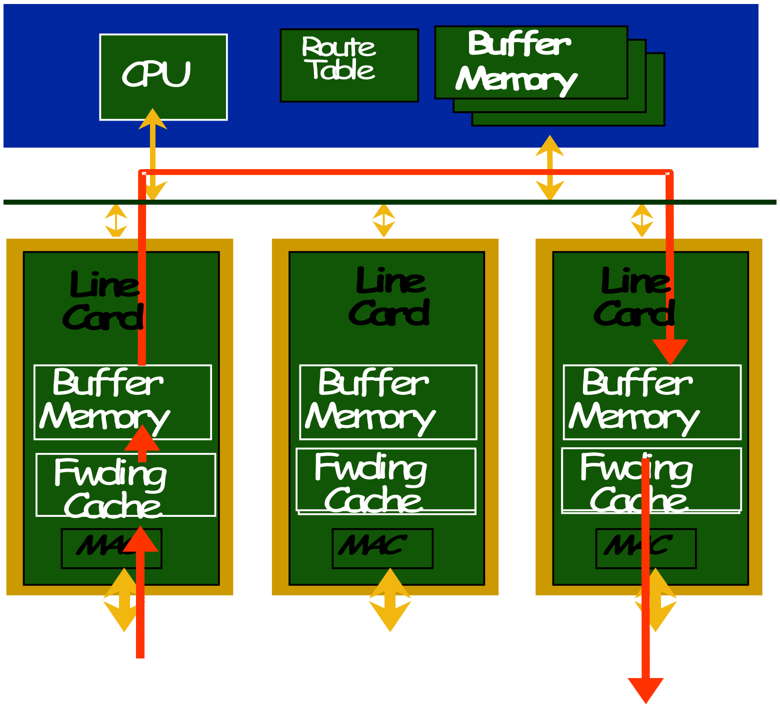


Figure 1.3: Bus-based router architecture with multiple processors

which becomes a bottleneck. As a result, packets are buffered in a centralized data memory waiting for processing, which leads to the disadvantage of having the data cross the bus twice, making the bus a major system bottleneck.

Given the drawbacks of the bus-based router architecture with single processor, improvements in the bus-based router architecture were introduced by distributing the packet forwarding operations. Figure 1.3 presents this improved bus-based architecture. In this architecture, fast processors, buffer memories and routing table caches were distributed into line interface cards, so most packets are processed locally on the line interface cards, and packets can be buffered in local line cards if it cannot be processed on time. Therefore, the load on the system bus and central CPU is reduced because most packets cross the bus only once.

However, because the route table cache on the line interface card can only caches the most recently used routes, packets that cannot be processed locally (the route is not in the local route table cache) will still request routes from the central CPU which keeps the whole route table. As a result, the central CPU is still a potential bottleneck.

If the route table cache on each line interface card can be scaled to store the whole route table, no packet will need to request processing from the central CPU. Thus the CPU bottleneck can be solved. However, each packet still crosses the shared bus, thus the shared bus becomes an unsolvable bottleneck in this architecture.

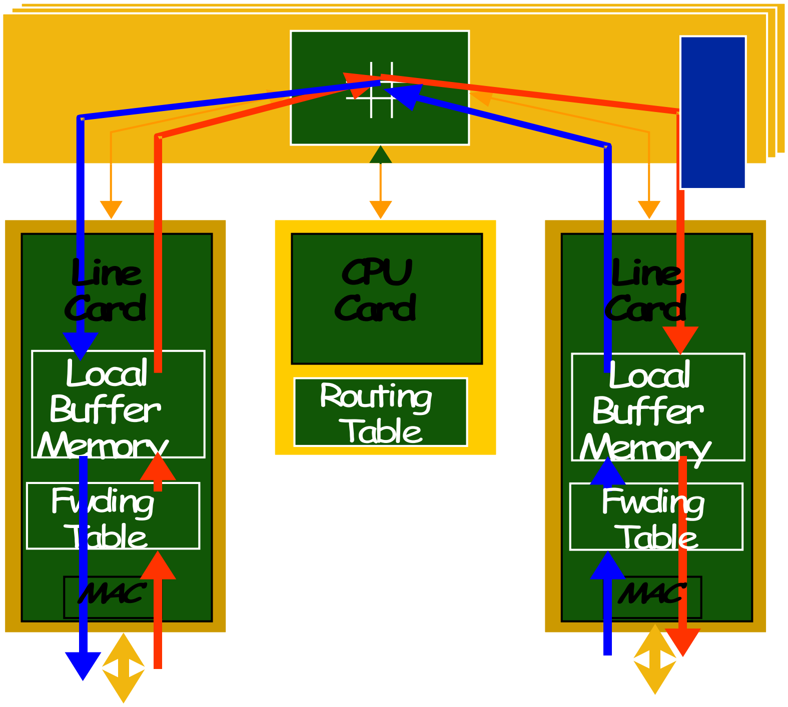


Figure 1.4: Switch-based router architecture

1.1.3 Switch-based router architecture

In order to alleviate the shared bus bottleneck in the bus-based router architecture, the switch backplane fabric was introduced in the router architecture. The switch backplane provides sufficient bandwidth for transmitting packets between line cards, and the overall throughput of the switch-based router can be increased by several orders of magnitude. Figure 1.4 describes the switch-based router architecture.

With more data-intensive applications being put on the Internet, the demand for high-throughput routers is still increasing. The dense wavelength-division multiplexing (DWDM) vastly increases the number of channels available on a single fiber, however, the speed of a single channel does not increase [24]. That means more ports at the same speed are needed at the switch. On the other hand, the SRAM (Static Random Access Memory) clock cycle poses a limit on the line rate in a combination with the minimum packet size. The limit of line rate is reached when the line rate times the clock cycle exceeds the minimum packet size, which requires packet aggregation techniques [24]. Finally, it becomes increasingly difficult and costly to perform line-speed processing in the network processors at these rates. All these facts cause the aggregate throughput to grow by increasing the number of ports and line cards rather than the port speed.

However, the switch cannot be scaled easily. Thus, to increase the number of line cards and the aggregate system bandwidth, architecture innovations are needed. The switch-core based router architectural was proposed to solve the increasing number of line cards and aggregate system band-

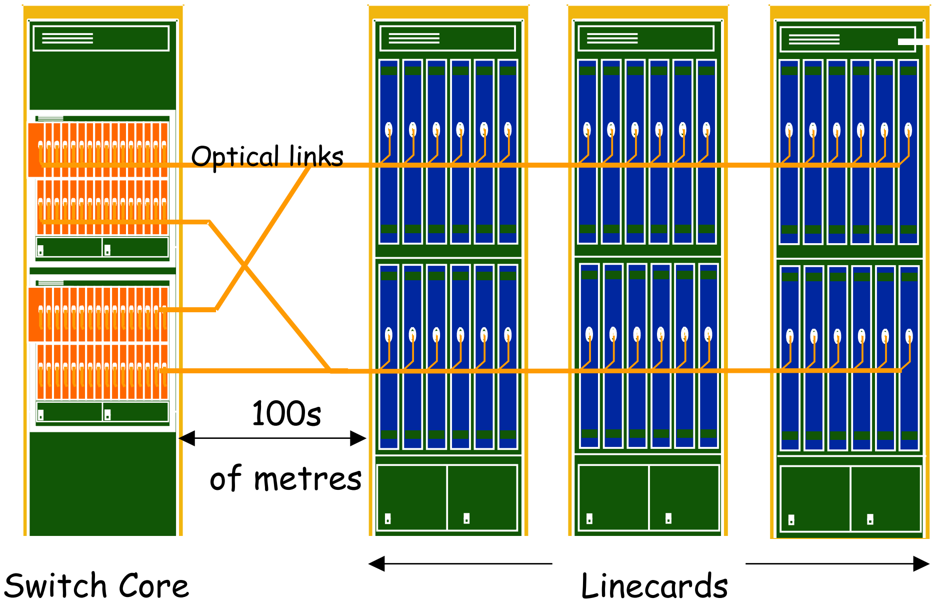


Figure 1.5: Switch-core based router architecture

width needs.

1.1.4 Distributed switch-core based router architecture

Figure 1.5 shows the most recent router architecture, which is based on the switch-core technology. In this architecture [15], the router can have more than one switch to form a tree, or a complete graph. As a result, more line cards can be supported, and higher aggregate throughput can be reached. The other benefit is, we can get simple fault-tolerance by sharing central switching sub-system like Figure 1.5.

1.2 Software routers

With its low cost, flexibility and extensibility, software router based on commodity PC hardware and open source operating systems is gaining more and more interest from both scientific researchers and small business users. It provides opportunities to implement new router operations and modify or extend router functions to meet specific research and business requirements.

Significant research has been directed toward the software routers that use commodity processors. They are MIT's Click Modular Router [20], Princeton's Extensible Router [18], Router Plugins [7] for the data plane, XORP [13] and Zebra [16] for the control plane. There are also two other research groups that have done some previous work on applying cluster to network packet forwarding and computation. These are the groups working on the Scalable IP Router at Michigan State

University and Panama at the State University of New York - Stony Brook.

1.2.1 Click modular router

Click [20] is a modular software architecture for building flexible and configurable routers based upon general-purpose processors. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions such as packet classification, queuing, scheduling and interfacing with network devices. A router configuration is modelled as a directed graph with elements at the vertices: packets flow along the edges of the graph. As a result, Click is very easy to extend and to reconfigure at runtime. Extending Click to support new services simply requires adding elements that implement these functions at the right place in the flow-based context configuration. These elements can be scheduled without modifying the core scheduling.

SMP Click [6] extends Click to run on a symmetric multiprocessing machine(SMP), and provides both scheduling flexibility and high performance. Although SMP Click can exploit the power of SMP machines, as the number of CPUs increases the cost of synchronization and cache misses impact scalability significantly [6].

1.2.2 Router plugins

With the goal to increase router modularity, extensibility, flexibility and performance, the Router Plugins [7] architecture is designed to allow limited extensions to an IP router. Implemented in the NetBSD operating system, Router Plugins allows users to write extensions as code modules called plugins to be dynamically added and configured at well-known points of the router's IP execution at runtime.

One of the novel features of their design is the ability to bind different plugins to individual flows; this allows for distinct plugin implementations to seamlessly coexist in the same runtime environment. They achieve high performance through a carefully designed modular architecture; an innovative packet classification algorithm that is highly efficient; and caching that exploits the flow-like characteristics of Internet traffic. An infrastructure called Crossbow allows users to specify a fixed key with each plugin. However, its fixed classifier does not provide the flexibility to implement forwarders that require different types of classifications.

1.2.3 Extensible router

The Extensible Router [18] project has a design goal of building a prototype router that (1) is easily extended to support new network services (including overlay and peer-to-peer networks, firewalls, media gateways, proxies, and cluster-based servers), and (2) exploits commercially available hardware components (including commodity processors, network processors, and high-speed system area networks). The Extensible Router uses an explicit path abstraction which models data flow from input device to output device, possibly with computation interposed along the way. Moreover, a hierarchy of such paths is proposed with different combinations of hardware, kernel and user-level sub-paths. A classification hierarchy is also proposed to support increasing degrees of complexity of the classifiers. As well, they proposed a scalable architecture based on a tightly coupled cluster of high-end systems [28]. ”Tightly coupled” means that the individual computers connect to a very high-speed crossbar switch via their internal system switch, and each system supports multiple line cards including microprocessors. However, no detailed design, implementation or experimental results have been published so far.

1.2.4 XORP

With the goal of developing an open source software router platform that is stable and fully featured enough for production use, and flexible and extensible enough to enable network research, XORP (eXtensible Open source Routing Platform)[13] implements an IP routing software stack with strong emphases on latency, scaling, and extensibility. Based on an event-driver scheme, XORP aims to respond to routing changes with minimal delay. XORP achieves extensibility and robustness by carefully separating functionality into independent modules running in separate UNIX processes with well-defined APIs between them. XORP can be divided into two subsystems: the user-space subsystem consists of the routing protocols and management mechanisms; the kernel subsystem provides the forwarding path and APIs for the higher-level to access. User-space XORP uses a multi-process architecture with one process per routing protocol, and a novel and flexible interprocess communication mechanism known as XORP Resource Locators (XRLs). By XRL, XORP processes can be run in distributed nodes. The kernel subsystem can use traditional OS kernel forwarding as well as Click modular router.

Up to now, XORP has implemented many routing protocols for IPv4 and IPv6 and a unified mean (xorpsh cli) to configure them. It supports: Unicast Routing Protocols - BGP4+ (Border Gateway Protocol version 4, IPv4 and IPv6), OSPFv2 (IPv4), OSPFv3 (IPv6), RIPv2 (IPv4), RIPng (Routing Information Protocol - next generation, IPv6); Multicast Routing Protocols - PIM-SM (Protocol Independent Multicast - Sparse Mode), PIM-SSM (Protocol Independent Multicast - Source-Specific Multicast,IPv4 and IPv6), IGMP (Internet Group Management Protocol) v1, v2, v3 (IPv4), MLD (Multicast Listener Discovery) v1, v2 (IPv6); Virtual Router Redundancy Protocol (VRRP); Network Management - Command Line Interface, SNMP (Simple Network Management Protocol); Forwarding Path - Traditional UNIX, Click, Windows Server 2003 and MacOS X.

1.2.5 Zebra

Making use of kernel routing table, Zebra [16] is a routing software package that provides TCP (Transmission Control Protocol)/IP based routing services with support of routing protocols such as RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP-4, and BGP-4+. Zebra also supports special

BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, Zebra also supports IPv6 routing protocols. With SNMP daemon which supports SMUX (SNMP Multiplexing Protocol), Zebra provides routing protocol MIBs (Management Information Base, a database of objects that can be monitored by a network management system).

Unlike traditional, monolithic architectures and even the so-called ”new modular architectures” that remove the burden of processing routing functions from the CPU and utilize special ASIC chips instead, Zebra software offers true modularity by its multi-process architecture to provide users with a high quality, multi-server routing engine. Like XORP, Zebra has a process for each protocol. Zebra has an interactive user interface for each routing protocol and supports common client commands. Due to this design, users can add new protocol daemons to Zebra easily. Users can use Zebra library as their program's client user interface.

1.2.6 Scalable IP router project

Michigan State University [33] was probably the first to introduce cluster into the field of routing, with their Scalable IP Router. They used four Pentium PCs running Linux interconnected by a four-port Myrinet switch, and proposed the concept of Remote Network (RN) devices to keep information about egress nodes, egress ports and internal network interfaces. A RN device is a virtual interface to an IP network, and makes a remote network appear to be directly connected to a node. The RN has the advantage of the intelligence of the IP layer to figure out the egress information without any change to the IP layer. All ports in the cluster can be regarded as local ports of the same node, thus all the techniques used in a traditional router can be used in the cluster-based router. Also, they proposed a scheme for parallel routing table computation [34]. However, there are no published performance results about the scalability of their prototype. They only compared the throughput of one-node and two-node routers, and no further experimental results were released on more nodes.

1.2.7 Panama project

Panama [29] is another scalable and extensible cluster-based router architecture, which supports an aggregate route caching scheme, a real-time link scheduling algorithm whose performance is independent of the number of real-time flows, a kernel extension mechanism to safely load networking software extensions dynamically, and an integrated resource scheduler which ensures that real-time flows with additional packet processing requirements still meet their end-to-end performance requirements. The Panama prototype consists of four PCs as router nodes connected by an 8-port Myrinet switch. Each router node had two Myrinet interfaces, each equipped with a Lanai network processor. The Panama kernel running on the ingress Lanai network processor executes packet forwarding and classification tasks, and also performs peer-to-peer DMA (Direct Memory Access) to move packets to an internal interface, where the packets are sent to the egress nodes. In this way, there is no difference between external and internal interfaces in their prototype, which is not practical in reality. They report results on single packet forwarding path between one pair of ingress and egress nodes, but give no results concerning scalability. The focus of their experiments was on the impact of different link scheduling policies.

1.3 Cluster-based router

The switch-core based router architecture seems scalable enough to support many line cards and high throughput as we need. However, as more and more kinds of real-time and data-intensive applications such as Video-On-Demand are put on the Internet, routers are increasingly expected to do more than just forward packets. More extensible packet processing functions are required to be included in the boundary routers, such as packet filtering, translating network addresses, and more complex, supporting differential services. All these facts put a new challenge on the router design. How can we support both high packet forwarding performance and a wide array of packet processing functions at the same time? How can we extend new packet processing functions in a router statically or dynamically without compromising system integrity? How can we scale the system throughput with addition of extra processing hardware? The switch-core based architecture uses custom line cards with network processors, which cannot be easily extended to support more packet processing functions. In order to answer these questions, architectural innovations are needed.

In this thesis, we propose a cluster-based router framework based on deploying the Click modular router on a cluster of commodity processors interconnected by a high-speed and low-latency InfiniBand network. With the development of high-speed interconnection networks and CPUs, the cluster-based architecture has become very successful in the field of high performance computing because of its scalability, high availability and outstanding performance/cost ratio. We believe this cluster-based architecture is also practical and will be a successful architecture in the field of network routing and switching.

Similar to the cluster-based supercomputer, the cluster-based router can achieve many other benefits:

1. It can use off-the-shelf components. That means we can focus on the software design and system issues, and use the best available commodity hardware.

2. It has a good performance/cost ratio compared to commercial routers because commodity components have the best performance/cost ratio.

3. It has good performance scalability and function scalability. The cluster-based router can be scaled almost linearly by adding more processing hardware when higher throughput is needed. And the most significant feature compared to switch-core based router architecture is, the cluster-based router can extend packet processing functions easily.

4. It has high availability. By using redundant interconnection network and processing hardware,

with the help of high availability software, the processing nodes and interconnection network can have standby parts, which improves system availability and maintains sustained packet processing service.

In this thesis, we describe how we build this cluster-based router, including the design of the cluster-based router architecture and the implementations of critical components. Also, we evaluate the performance of the cluster-based router and its potential bottleneck components, and propose techniques to eliminate or reduce the impact of these affects.

1.4 Internal congestion control in the cluster-based router

In any network, when the total demand for a resource is greater than the available capacity in any time interval, the resource is said to be congested for that time interval. The resources required in a router include internal link bandwidth, switch buffer capacity, processor cycles, and etc [21].

As in any distributed system, congestion control is a critical design issue in the cluster-based router. Congestion can affect both the forwarding rate and the latency of packets travelling through the router. If packets are sent too quickly from multiple ingress nodes to a single egress node, the egress node will become overloaded and its forwarding capability will be reduced. At this point, packets will be delayed and perhaps even dropped due to the queue overflow at the egress. (For the detailed description of congestion problem in the cluster-based router, please refer to section 5.3).

Despite many congestion control schemes that have been studied, the uniqueness of the internal congestion problem in the cluster-based router requires a different approach. The internal communication between processing nodes in our cluster-based router is based on the InfiniBand network. The link-layer flow control implemented as a part of InfiniBand standard pushes congestion outward from the internal network to the dependent ingress and egress nodes [35]. Unlike the congestion in TCP, the RTT (Round Trip Time) between the ingress and the egress is small compared to that in the Internet. Different from a crossbar switch that has a complete control over the packet forwarding path, some commodity components (Internal InfiniBand network, PCI (Peripheral Component Interconnect) bus, and etc) in our cluster-based router are not programmable.

In this thesis, we start by considering the following: if the ingress nodes can be signalled to reduce their transmission rates to specific overloaded egress queues then the drop rate of packets on those egress nodes will be reduced. This improves the overall router throughput because it avoids wasting the internal network bandwidth on dropped packets and reduces the proportion of cycles in both ingress and egress nodes which are spent on processing packets that will be dropped due to congestion.

A significant amount of work has been done in the area of network congestion control. In the following sections, we describe some of related works on network congestion control.

1.4.1 Internet congestion control

Internet congestion control is composed of end-to-end flow control and router congestion control. TCP, as the core protocol of the Internet Protocol Suite, adopts [17] an adaptive AIMD (Additive Increase, Multiplicative Decrease) window flow control algorithm depending on the detection of packet loss due to the overflow of buffers in the network.

AQM (Active Queue Management) schemes like REM (Random Exponential Marking) [1] and RED (Random Early Detection) [9] avoid congestion and maintain low queue lengths on the router by dropping packets earlier while allowing occasional bursts of packets in the queues. RED monitors the average queue size and drops (or marks when used in conjunction with ECN) packets based on statistical probabilities. If the buffer is almost empty, all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability reaches 1 and all incoming packets are dropped. Compared with the traditional drop tail algorithm used on routers, RED does not possess a bias against bursty traffic that uses only a small portion of the bandwidth. The more a host transmits, the more likely it is that its packets are dropped. RED also helps avoid global synchronization. Based on RED, ECN (Explicit Congestion Notification)[30] allows end-to-end notifications of network congestions without dropping packets by marking any packet that RED drops.

To address the instability of TCP in high-bandwidth optical networks, XCP (Explicit Control Protocol) [19] generalizes the Explicit Congestion Notification proposal (ECN) to include the degree of congestion at the bottleneck instead of the congestion indication, and it introduces the concept of decoupling the utilization control from the fairness control in the calculating of the congestion notification, enabling routers to quickly make use of available bandwidth while conservatively managing the allocation of bandwidth to flows. XCP is built upon a new principle: carrying per-flow congestion state in packets. XCP packets carry a congestion header through which the sender requests a desired throughput. Routers make a fair per-flow bandwidth allocation without maintaining any per-flow state. Thus, the sender learns of the bottleneck routers allocation in a single round trip and adjusts its transmission rate.

1.4.2 Congestion control in ATM networks

In ATM (Asynchronous Transfer Mode) networks, the information is transmitted using short fixed-length cells, which reduces the delay variance, making it suitable for integrated traffic consisting of voice, video and data.

To avoid large queue length and buffer overflow caused by congestion, Newman[25] proposed a backward explicit congestion notification (BECN) AIMD scheme for ATM local networks, and Kumar[21] developed a fair backward explicit congestion control scheme which monitors the buffer utilization in the network and generates control cells which give the recommended rates the sources should use whenever buffer overflows. In Newman's [25] BECN mechanism, when a queue in an ATM switch exceeds a threshold it sends congestion notification cells back to the sources of the virtual channels currently submitting traffic to it. On receipt of a BECN cell on a particular virtual channel, a source must reduce its transmission rate for the indicated virtual channel. If no BECN cells are received on a particular virtual channel for a certain period of time, a source may gradually restore its transmission rate on that virtual channel. In Kumar's [21] fair backward explicit congestion control scheme, it monitors the buffer space and its utilization in a ATM switch. Whenever the buffers get filled to a peak value, control cells are generated and are sent to the host computers responsible for buffer overflow. The control cells have a field set to the rate that host computer should use. The new rate set in the control cells is based on the current rate of the call. After reducing the rate, the host will not reduce it further for a period of time called the recovery period that is also based on the current call rate. After the recovery period is over, the host computer will again transmit at the previous call rate.

1.4.3 Router internal congestion control

Many high performance routers use crossbars as internal switching fabrics [11] [32], which segment received variable length packets into fixed length cells for transmission through the crossbar and reassemble at the output. This simplifies the centralized scheduler in the crossbar and allows for synchronous operation. However, this requires the scheduler to make decisions at every cell transmission interval. It becomes infeasible with increasing numbers and increasing rates of links in a large router.

To break this limitation, with the addition of buffers to each of the crosspoint in the crossbar, Pappu [27] proposed a distributed queueing algorithm. His algorithm regulates the rates at which traffic is forwarded through the interconnection network from inputs to outputs using asynchronous coarse-grained scheduling instead of iteratively scheduling the transmission of individual packets. In this algorithm, the port processors periodically exchange information about the status of their VOQs (Virtual Output Queues). This information is then used to rate control the VOQs, with the objective of moving packets to the output side of the router as expeditiously as possible while avoiding congestion within the switch fabric. The scheme allows the forwarding of variable length packets, which eliminates the need for segmentation and reassembly and is not subject to bandwidth fragmentation [32].

1.4.4 InfiniBand link level flow control and congestion control

In InfiniBand network, the link layer of each port on a CA (Channel Adapter), router, or switch implements one or more transmit/receive buffer pairs (called Virtual Lanes) that are used to transmit and receive data packets. A packet in the transmit buffer of a VL is transmitted to the corresponding receive buffer in the port at the other end of the virtual link. To avoid the discarding of packets when the receive buffer of destination does not have sufficient room to hold the packets, the link level flow control is applied in the InfiniBand [31].

The link level flow control is a credit based flow control. It is used to manage data flow between two ends of a point-to-point virtual link. On a periodic basis, each VL receive buffer generates a special flow control packet to be transmitted to the corresponding VL transmit buffer in the port at the other end. The flow control packet includes the credit that specifies the amount of packets that can be received on the receive end without loss. Packet is not transmitted unless the receiver advertises credits indicating receive buffer space is available.

With the application of InfiniBand link level flow control, no InfiniBand packet will be lost on the network link. However, in a multi-switch InfiniBand network, the congestion spreading problem[14] may occur. In order to avoid or eliminate the congestion spreading problem, the connection control annex A10 in [14] proposes a congestion control scheme consisting of congestion detection/marking, congestion signalling, injection rate reduction and injection rate recovery[12].

1. Congestion detection/marking: InfiniBand switches detect congestion on a virtual lane for a given port when a relative threshold has been exceeded. When congestion is detected as the root of victim, the switch informs the destination node by setting the Forward Explicit Congestion Notification (FECN) bit in the head of the respective packet.

2. Congestion signalling: upon receipt of a packet with FECN, the destination HCA will inform the respective source HCA by sending a Backward Explicit Congestion Notification (BECN).

3. Injection rate reduction: when the source receives a BECN, an index is incremented by a certain amount into the congestion control table (CCT), whose entries define the inter-packet delays (IPD) used for injection rate control. The injection rate of the respective hot flow is changed accordingly.

4. Injection rate recovery: the CCT index is reduced based on a recovery timer. Whenever this timer expires the index is decremented, so the injection rate recovers accordingly.

InfiniBand link level flow control and congestion control solve the packet loss and the congestion spreading problems in the InfiniBand network (From a source HCA to a destination HCA). However, congestion may still happen in the cluster-based router. If the sender nodes transmit faster than the receiver node can process, in which case the posted buffers on the receiver side will not be available to host the incoming packets, the packets will be discarded by the HCA on the receiver. Also, when congestion happens, excessive packets will be accumulated either on the upstream queues of InfiniBand transmission elements, or on the downstream queues of InfiniBand reception elements. (For more detail, please refer to section 5.3).

1.4.5 Flow control over InfiniBand network

InfiniBand Architecture [14] also provides an end-to-end (or message level) flow control capability for Reliable Connections that can be used by a receiver to optimize the use of its receive resources.

Essentially, as in the InfiniBand link layer flow control scheme, a sender cannot send a message unless it has appropriate credits to do so. The encoded credits are transported from the receiver to the sender in an acknowledge message that returned to the sender.

Two mechanisms are defined for transporting credits from the receiver to the sender. The credits can be piggybacked into an existing acknowledge message, or a special unsolicited acknowledge message can be generated by the receiver. Piggybacked credits are those credits that are carried in the head field of an already scheduled acknowledge message. An unsolicited acknowledge message may be sent by the receiver at any time. The requesters send queue simply recovers the credit field and the message sequence number field from the most recently received acknowledge message.

Liu etc [22] also proposed User-level static and dynamic flow control schemes in MPI (Message Passing Interface) over InfiniBand which are developed based on the InfiniBand Reliable Connection service. As the end-to-end flow control in Reliable Connection, they use the credit-based scheme on the user-level flow control. Initially, the credit for each sender is equal to the number of pre-posted buffers. Whenever a sender sends out a message that will consume a receiver buffer, its credit count will be decremented. When the credit count reaches zero, the sender can no longer post send operations that consume credits. In this case, the send operations are stored in a backlog queue where they will be processed later in FIFO (First In First Out) order when the credits are available. At the receiver side, the receiver will re-post a receive buffer after it has finished processing a message. The credit count for the corresponding sender will then be incremented.

To transfer the credit, two methods are used in their user-level schemes as in the InfiniBand end-to-end flow control for Reliable Connections: piggybacking and explicit credit messages. Piggybacking is to transfer the credit by adding a credit information field to each message. So credit can be transferred by already scheduled messages destined to the sender from receiver. Explicit credit message will be used when there is no message sent by the receiver to the sender on the MPI layer. To avoid possible deadlock, they proposes an optimistic scheme for credit messages. That is, no credit is needed for explicit credit messages. Explicit credit messages are always posted directly without going through the backlog queue even though no credit is available.

In the user-level static flow control scheme, a fixed number of receive buffers are pre-posted for each connection during MPI initialization. In the user-level dynamic flow control scheme, each connection starts with a small number of pre-posted buffers, but the number of pre-posted buffers can be gradually increased or decreased based on the communication pattern using a feedback-based control mechanism. By adding a field to each message indicating whether it has gone through the backlog in the sender side, the receiver is able to know if more pre-posted buffers are needed.

In our cluster-based router, we use InfiniBand unreliable datagram service between processing nodes, which requires different congestion control scheme than those based on reliable services.

1.5 Thesis contributions

In this thesis, we address several issues in the design and implementation of an extensible, scalable, and efficient cluster-based router, especially the internal congestion control in the cluster-based router. The contributions of the thesis are summarized as follows.

1. Most software routers based on COTS (Commercial Off The Shelf) hardware address the issues of functional extensibility, but performance is limited by the underlying hardware. To address the scalability of software routers, we proposed a cluster-based router framework by deploying the Click modular router on a cluster of commodity processors interconnected by a high-speed and low-latency network. The cluster-based router inherits extensibility from Click, and scalability from clustering. Although Scalable IP router project [33] and Panama project [29] proposed the similar architecture, we are the first to evaluate the scalability of this architecture in terms of forwarding capacity. Scalable IP router [33] compared only the throughput of one-node and two-node routers, and Panama [29] reported only results on a single packet forwarding path between one pair of ingress and egress nodes.

   To improve the performance and scalability of the cluster-based router, we developed a highly efficient communication layer above the InfiniBand network stack - IBUDCOM (InfiniBand Unreliable Datagram COMmunication). By disabling of the notification feature of InfiniBand Queue Pair which provides the polling interface to router components, IBUDCOM removes the overhead of software interrupts. Also, IBUDCOM allows the packing of multiple Ethernet packets into one large InfiniBand packet. All these improves the efficiency of internal communication.

2. Although much work has been done to compare different software routers or different techniques used in a software router [3] [4] [5] [10], no work has completely analyzed the potential hardware bottlenecks that may exist on a PC-based software router. We evaluated and analyzed the potential hardware bottlenecks of a PC-based router, and concluded that, by applying the polling extension of network driver and buffer recycling techniques, a modern CPU is more than able to process the minimum-size Ethernet packets received and transmitted at one Gigabit network port as well as multiple Gigabit network ports on the same PCI-X (Peripheral Component Interconnect Extended) bus. We also found that there is a nonlinear correlation between the reception and transmission capabilities of an individual port as well as ports on the same bus.

3. The nonlinear correlation between the reception and transmission capabilities of an individual port as well as ports on the same bus puts adverse effects on the performance of a PC router or the processing nodes in the cluster-based router. The adaptive scheduling mechanism based on system state information we proposed manages this adverse effect and enhances the trans-

mission capabilities of NICs when they are overloaded at the cost of very little additional overhead.

4. The potential internal congestion problem in the cluster-based router may affect the overall forwarding rate and latency of packets forwarded in the cluster-based router. We analyzed the potential congestion problem and proposed a Backward Explicit Congestion Notification scheme combined with a novel queue scheduling method to manage the internal congestion. In this scheme, congestion notification is sent back from egress to ingresses when there is congestion on the egress, while the ingresses reduce corresponding ingress queues' scheduling weights upon the receipt of congestion notification or increase the scheduling weights if no congestion notification is received. Through experiments we showed that this scheme prevents the ingress nodes from overwhelming the egress nodes. It also reduces the waste of internal network bandwidth and CPU cycles by dropping packets as early as possible, so as to maximize the forwarding rate under overload.

5. We also presented a discrete optimal utility-based internal congestion control scheme. Different from the above BECN queue scheduling scheme which sends the congestion notification back to ingress nodes when congestion happens, the optimal scheme sends the congestion prices back to ingress nodes at a regular basis. The ingress nodes adjust the transmission rates whenever they receive the congestion price. Similar to other work on the stability analysis of optimal utility-based congestion control schemes [8] [23] [26], we proved the stability of the corresponding continuous model through approximating a discrete system by a continuous model and designing a Lyapunov function which satisfies Lyapunov's theorem. However, we found that the stability condition of the discrete model is different from the continuous model by plotting the stability region of control parameters via numeric simulation. Simulation of the scheme with ns-3 and experimental testing on our cluster-based router prototype validated our results and showed that the stability region predicted by the discrete model is reliable.

6. Based on above discrete optimal utility-based internal congestion control scheme, we further developed the optimal congestion control model by using the queue status instead of packet rates to calculate the congestion price, and compared its performance to that of an additive increase, multiplicative decrease (AIMD) scheme. Although both schemes can be fair to different incoming flows and effective to improve the overall forwarding rate of the router, the optimal scheme provides much smoother control with higher average forwarding rate, smaller average queue length, and smaller queue variance than the AIMD scheme. To our knowledge, this is the first theoretical and experimental study of optimal congestion control in a router. Moreover, the optimal scheme is based on the mathematical definition of a utility function. The specific form of this function can be changed to suit a wide range of performance requirements.

1.6 Thesis outline

This thesis is composed of our published or submitted papers that describe our work. Chapter 2 presents the design, implementation and evaluation of the cluster-based router. To investigate the behaviors of individual nodes in the cluster-based router, in chapter 3, we evaluate and analyze potential hardware bottlenecks that may exist on a PC-based router. In chapter 4, we propose and evaluate an adaptive scheduling mechanism based on system state information to manage the adverse effects of the correlation between the transmission and reception capabilities of an individual network interface card (NIC) or multiple NICs on the same bus because of NIC/bus bottlenecks. Chapter 5 gives the internal congestion problem that may exist in the cluster-based router and describes a Backward Explicit Congestion Notification combined with a novel queue scheduling scheme. In chapter 6, an optimal utility-based internal congestion control scheme is introduced and its stability analysis is presented. Chapter 7 develops the optimal utility-based internal congestion control model by using the queue status instead of packet rates to calculate the congestion price, and compares its performance to that of an additive increase, multiplicative decrease (AIMD) scheme by ns-3 simulation and prototype evaluation. Chapter 8 concludes our work and outlines the future directions.

Bibliography

[1] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: active queue management. *IEEE Network*, 15:48–53, 2001.

[2] James Aweya. IP router architectures: an overview. *International Journal of Communication Systems*, 14(5):447 – 475, 2001.

[3] Andrea Bianco, Robert Birke, Davide Bolognesi, Jorge M. Finochietto, Giulio Galantet, Marco Mellia, Prashant M.L.N.P.P, and Fabio Neri. Click vs. Linux: two efficient open-source IP network stacks for software routers. In *Workshop on High Performance Switching and Routing*, May 2005.

[4] Andrea Bianco, Jorge M. Finochietto, Giulio Gelante, Marco Mellia, and Fabio Neri. Open-source PC-based software routers: A viable approach to high-performance packet switching. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[5] Raffaele Bolla and Roberto Bruschi. RFC 2544 performance evaluation for a Linux based open router. In *Workshop on High Performance Switching and Routing*, June 2006.

[6] Benjie Chen and Robert Morris. Flexible control of parallelism in a multiprocessor PC router. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 333–346, Berkeley, CA, USA, 2001. USENIX Association.

[7] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernhard Plattner. Router plugins: a software architecture for next generation routers. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 229–240, New York, NY, USA, 1998. ACM.

[8] Xingzhe Fan, Murat Arcak, and John T. Wen. L_p stability and delay robustness of network flow control. In *Proceedings of the IEEE Conference on Decision and Control, Maui*, pages 3683–3688, 2003.

[9] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[10] Calarco Giorgio, Raffaelli Carla, Schembra Giovanni, and Tusa Giovanni. Comparative analysis of SMP click scheduling techniques. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[11] Pankaj Gupta. Scheduling in input queued switches: A survey. Technical report, Department of Computer Science, Stanford University, 1996.

[12] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato. Congestion control in infiniband networks. In *HOTI '05: Proceedings of the 13th Symposium on High Performance Interconnects*, pages 158–159, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Mark Handley, Orion Hodson, and Eddie Kohler. XORP: an open platform for network research. *SIGCOMM Computer Communnication Review*, 33(1):53–57, 2003.

[14] IBTA. *InfiniBand Architecture Specification*. InfiniBand Trade Association, 2002.

[15] PMC-Sierra Inc. A new architecture for switch and router design. White paper, 1999.

[16] Kunihiro Ishiguro. Zebra routing software. Available online. http://www.zebra.org.

[17] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988. ACM.

[18] Scott Karlin and Larry Peterson. VERA: an extensible router architecture. *Computer Networks.*, 38(3):277–293, 2002.

[19] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[20] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[21] Anup Kumar, Arpana Maniar, and Adel S. Elmaghraby. A fair backward explicit congestion control scheme for ATM network. In *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, pages 452–457, Washington, DC, USA, 1999. IEEE Computer Society.

[22] Jiuxing Liu and Dhabaleswar K. Panda. Implementing efficient and scalable flow control schemes in mpi over infiniband. *Parallel and Distributed Processing Symposium, International*, 9:183b, 2004.

[23] Shao Liu, Tamer Basar, and R. Srikant. Controlling the Internet: a survey and some new results. In *Proceedings of the 43nd IEEE Conference on Decision and Control*, pages 3048–3057, Maui, Hawaii USA, December 2003.

[24] Cyriel Minkenberg, Ronald P. Luijten, François Abel, Wolfgang Denzel, and Mitchell Gusat. Current issues in packet switch design. *SIGCOMM Comput. Commun. Rev.*, 33(1):119–124, 2003.

[25] Peter Newman. Traffic management for ATM local area networks. *IEEE Communications Magazine*, 8:44–50, 1994.

[26] Fernando Paganini. A global stability result in network flow control. *Systems & Control Letters*, 46:165–172, 2002.

[27] Prashanth Pappu, Jyoti Parwatikar, Jonathan Turner, and Ken Wong. Distributed queueing in scalable high performance routers. In *Proceedings of IEEE Infocom*, 2003.

[28] Larry L. Peterson, Scott C. Karlin, and Kai Li. Os support for general-purpose routers. In *HOTOS '99: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, page 38, Washington, DC, USA, 1999. IEEE Computer Society.

[29] Prashant Pradhan and Tzi-cker Chiueh. Implementation and evaluation of a QoS-capable cluster-based IP router. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society Press, 2002.

[30] K. K. Ramakrishnan, Sally Floyd, and David Black. The addition of explicit congestion notification (ECN) to IP, 2001.

[31] Tom Shanley. *InfiniBand network architecture*. PC System Architecture series. MindShare, Inc, 2003.

[32] Jonathan S. Turner. Strong performance guarantees for asynchronous buffered crossbar scheduler. *IEEE/ACM Trans. Netw.*, 17(4):1017–1028, 2009.

[33] Vibhavasu Vuppala and Lionel M. Ni. Design of a scalable IP router. In *IEEE Proc. of Hot Interconnects Symposium V*, page 7 pages. IEEE Computer Society Press, 1997.

[34] Xipeng Xiao and Lionel M. Ni. Parallel routing table computation for scalable IP routers. In *CANPC '98: Proceedings of the Second International Workshop on Network-Based Parallel Computing*, pages 144–158, London, UK, 1998. Springer-Verlag.

[35] Qinghua Ye and Mike H. MacGregor. Internal congestion control in the cluser-based router. In *Syposium on High Performance Switching and Routing*, pages 13–20, Shanghai, China, May 2008.

Chapter 2

Cluster-based IP Router: Implementation and Evaluation

2.1 Introduction

The variety and intensity of Internet applications are increasing rapidly: applications requiring real-time services such as video-on-demand and immersive virtual reality (e.g. network gaming) are becoming common. IP routers are expected to support these applications, and must do more than just traditional packet forwarding. Routers must be easily extensible so that they can include new packet processing functions such as network address translation, packet filtering and scheduling, and complex, differential QoS (Quality of Service). Significant issues arise when designing extensible and efficient routers to support both high packet forwarding rates and a wide array of packet processing services. How can a wide variety of packet processing functions be integrated without compromising the packet forwarding performance? How can we support changes in packet processing functions by reconfiguration without interrupting services? Can we scale forwarding performance and numbers of interfaces linearly by simply adding hardware?

Most commercial routers are based on custom hardware and software. They support a set of closed, predefined processing functions, but require hardware upgrades or outright replacement to add new functionality. Most of the predefined functions can be activated or deactivated by the network administrator, but no new functions that were unanticipated at the time of design can be added into the packet forwarding path.

On the other hand, significant research has been directed towards software routers using commodity processors, which increase the extensibility of packet processing functions. In this paper, we describe the implementation and evaluation of an extensible and scalable cluster-based IP router built using a cluster of commodity processors interconnected by an InfiniBand network. This cluster-based router can be scaled linearly to support higher packet forwarding rates and processing performance. The cluster-based architecture has been widely used in the design of high performance su-

[0] This chapter is based on the paper ”Cluster-based IP Router: Implementation and Evaluation” published on IEEE International Conference on Cluster Computing, Barcelona, Spain, September 25th-28th, 2006

percomputers. With the development of high-speed interconnection networks and CPUs, the cluster-based architecture has become very successful in the field of high performance computing because of its scalability, high availability and outstanding performance/cost ratio. Besides the area of supercomputing, the cluster-based architecture is also widely used in Web services. The challenge of building a cluster-based router is to minimize the latency of the interconnection network while maintaining scalability of the forwarding rate.

As we show in this work, it is possible to build a cluster-based router using high-speed general-purpose CPUs, and a high-throughput / low-latency interconnection network.

Compared to the traditional router architecture, the cluster-based router has many different features. The following issues must be addressed when designing the cluster-based router architecture.

1. Efficient communication based on a high-speed / low-latency interconnection network and the related communication layer, such as Myrinet GM or InfiniBand Verbs. The interconnection hardware and software communication layer is the core of a cluster-based router. The primary problems are how to make the best use of low-latency features of the interconnection hardware and software, how to make the fewest data copies, how to reduce or remove operating system (OS) overheads, and how to reduce data buffering and make any remaining buffer maintenance as efficient as possible.

2. Flexible packet forwarding path. One benefit of the cluster-based router architecture is its extensibility of packet processing. Our focus has been on how to organize the forwarding path, such that packet processing functions can be added or deleted easily.

3. Packet processing element scheduling. We must consider how to schedule packet processing elements, how to share and distribute processing loads and how to make the best use of parallel processing features of symmetric multiprocessing (SMP) servers. These aspects affect the overall packet forwarding rate directly.

4. Efficient link scheduling mechanism. Due to the different numbers of external links and internal links, packets from several external links may compete for the same internal communication link, so there must be a buffer mechanism for each forwarding path or applications with different service priorities. Link scheduling for this situation is a complex problem that involves QoS and Diff-service technology.

In this paper, we describe the design, implementation and evaluation of such a cluster-based router. The rest of this paper is organized as follows: section 2.2 introduces the system architecture of our cluster-based router, including the layout of the system architecture, software components, etc. The detailed implementation of our prototype is presented in section 2.3. Section 2.4 describes the experimental setup and initial results of performance evaluation. Related work is mentioned in section 2.5. Finally, we conclude this paper and outline some future work.

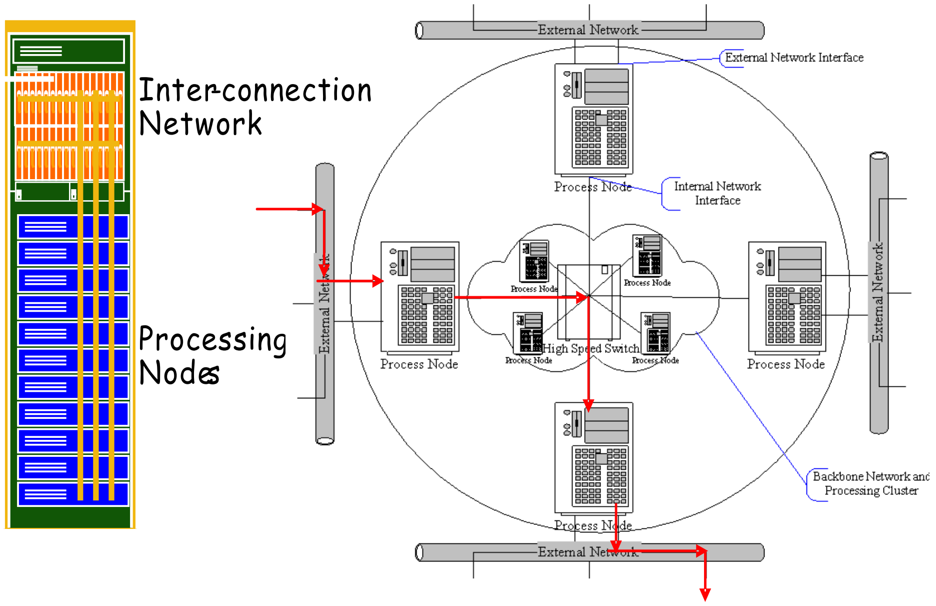


Figure 2.1: Prototype cluster-based IP router architecture

2.2 System architecture

Figure 2.1 shows our cluster-based router architecture. The left side shows the physical deployment view, while the right side shows the internal interconnections.

The cluster-based router architecture is composed of two main parts: the interconnection network and processing nodes. The interconnection network must be a high-throughput / low-latency fabric such as InfiniBand, Quadrics, and Myrinet. The processing nodes are commodity server blades, each carrying one or more interface cards terminating the links to the outside world, and also carrying interfaces for terminating fabric links. Our prototype uses 1 Gbps Ethernet links to the outside world.

The right side of Figure 2.1 shows the interconnection between processing nodes. In this architecture, we can divide the processing nodes into two categories: internal processing nodes and external processing nodes. Internal processing nodes can be used to do control and management tasks such as routing table updates, or other non-critical processing such as authentication. External nodes connect to the attached outside networks, and are responsible for receiving and forwarding packets to other attached outside networks. The external nodes are dedicated to packet forwarding in the critical data path.

All the internal and external nodes are connected to the high-throughput interconnection network. Almost all packets cross the interconnection network for forwarding. Since each of the nodes has its own processor and memory, the packets cross the interconnection network only once if there is no additional processing needed in the internal processing nodes.

In the rest of this section, we will describe several important parts of our prototype, including the InfiniBand interconnection network, the Click modular software processing stack, the software architecture, and the packet forwarding path.

2.2.1 Interconnection network - InfiniBand

A high-throughput and low-latency interconnection network is the central foundation of a cluster-based router: a router must handle hundreds of millions of packets per second (throughput) and it must add very little delay (latency) when forwarding each of these packets. A device that could forward at a very high rate would be utterly useless if every packet was delayed by hundreds of milliseconds. New interconnection technologies such as InfiniBand are focused specifically on reducing latency while providing high throughput. These interconnects are now available as commodity components. The InfiniBand Architecture is an industry standard developed by the InfiniBand Trade Association. It has been proposed as the next generation of I/O and inter-process communications. In InfiniBand, computing nodes are connected to the switch fabric through a Host Channel Adapter (HCA). The InfiniBand Verbs interface is provided to communicate with these Channel Adapters. Four kinds of communication protocols (reliable connection, unreliable connection, reliable datagram, and unreliable datagram) are provided.

Installed on industry standard PCI-Express (Peripheral Component Interconnect Express) bus [3], current PCI-Express InfiniBand host channel adapters (two ports) provide 40 Gbps full duplex bandwidth. In the near future, by integrating the InfiniBand logic into the chipset, PCI-Express can be implemented at higher bandwidths up to 60 Gbps. Current InfiniBand switches [2] have up to 144 ports, and each port can support 20 Gbps bandwidth, for an aggregate switch capacity of 5.76 Tbps. In addition, InfiniBand has much lower latency than Ethernet.

Due to these features, the InfiniBand network is used in our cluster-based prototype as the interconnection fabric to connect all the processing nodes.

2.2.2 Click modular router

Click [4] is a modular software architecture for building flexible and configurable routers based upon general-purpose processors. A Click router instance is assembled from packet processing modules called elements. Individual elements implement simple router functions such as packet classification, queuing, scheduling and interfacing with network devices. A router configuration is modelled as a directed graph with elements at the vertices: packets flow along the edges of the graph. Two important features of this architecture are ”pull” processing and flow-based router context. ”Pull” processing decouples the source and sink of a packet forwarding path by allowing the sink to decide when to pull data from the source. ”Pull” processing can recursively propagate towards the source, if needed. The flow-based router context allows a module to perform a closure operation over the modules in the computation graph that are reachable from it, and to extract important information about them.

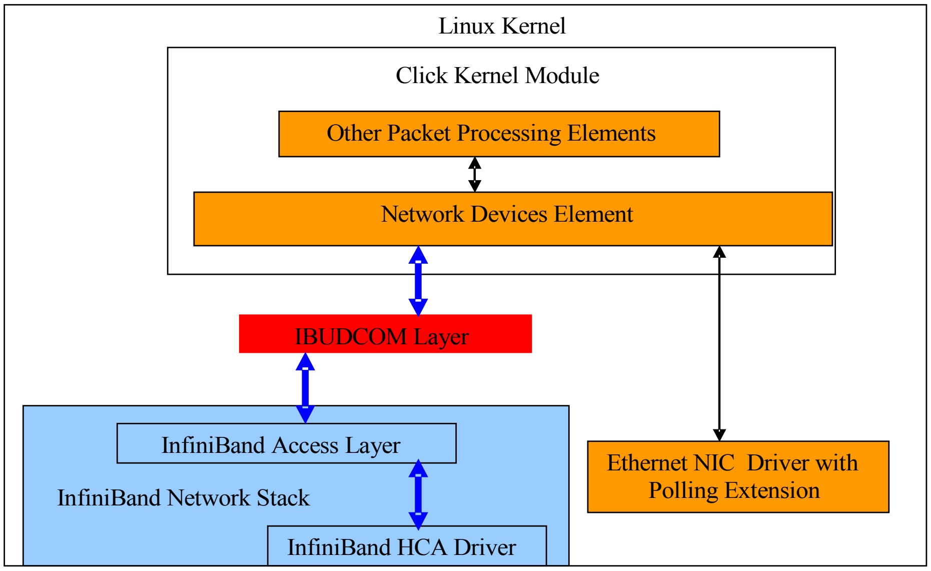


Figure 2.2: Software architecture

The focus of Click is on showing that a modular and low-overhead structure can be built for router software, and the techniques proposed therein can be utilized in the computation model of any router architecture based on a general-purpose computing platform.

In Click, any packet processing function is represented as an element, which is the basic unit of CPU scheduling. As a result, Click is very easy to extend, and to reconfigure at runtime. Extending our prototype to support new services simply requires adding elements that implement these functions at the right place in the flow-based context configuration, and these elements can be scheduled without modifying the core scheduling.

We use Click as our node routing software because of its flexibility and extensibility, and have extended it to run on a cluster. We have added support for the communication between processing nodes by implementing corresponding elements.

2.2.3 Processing node software architecture

We have extended the Click Modular Router to run on a multiprocessor cluster. Figure 2.2 presents the software architecture of each processing node.

The main implementation effort has been to abstract the details of the Infiniband network stack: the Click kernel module relies on InfiniBand unreliable datagram communication (IBUDCOM), which implements all the communication functions between processing nodes. The Click network device elements can send/receive internal packets through this layer without interacting directly with the InfiniBand stack. The detailed implementation of the IBUDCOM layer will be explained in section 2.3.

The Ethernet NIC driver has been modified to use polling. The Click network device elements can disable the NIC interrupt, and use polling to get received packets and clean the transmission buffer ring, which avoids the significant overhead of interrupts. The Click kernel module runs as a kernel thread under Linux. As described above, it is assembled from selected packet processing modules called elements. The combination of these elements implements a flexible router. Based on this flexible architecture, it is easy to extend the router's functions.

2.2.4 Packet forwarding path

In the right hand side of Figure 2.1, a basic packet forwarding path is presented. An IP packet from some external network enters the router through one of its external processing nodes. The processing node performs some basic IP routing functions such as IP table lookup to find the next hop IP address and the egress interface. As well, it also finds the egress processing node. Then it will forward the IP packet to the egress processing node via the internal interconnection network. When it receives the packet, the egress processing node forwards the IP packet to the next hop through the egress interface.

Figure 2.2 details the forwarding path in each processing node. The IP packet is received at the Ethernet driver, where it is captured by the network device element in Click, and then follows the Click processing path according to the defined router configuration. At the end of this processing path, the network device element gives the packet to the IBUDCOM layer, which sends it to the egress node.

When received by the HCA driver on the egress node, the IP packet is transferred to the network device element through the IBUDCOM layer, and then through additional processing, if necessary. Finally, it will be transmitted by the Ethernet driver.

A network administrator can add processing functions in the form of Click elements to any flow's processing path. For example, for the sake of system performance, two or more IP packets may be packed into one InfiniBand packet, which helps to reduce average PCI overhead and hard interrupt of HCA, and results in higher forwarding throughput.

2.3 Implementation

In this section, we describe the implementation of our prototype, including the communication layer, polling extension of network driver, buffer recycling, and InfiniBand elements.

2.3.1 Communication layer

An efficient communication layer is critical to building a low-latency / high-throughput router. Although InfiniBand provides four kinds of communication protocols, considering the overhead of three of these protocols, we decided to use the Unreliable Datagram protocol. We call our communication layer IBUDCOM. Figure 2.3 shows the InfiniBand architecture and the relationship between IBUDCOM and the other InfiniBand components.

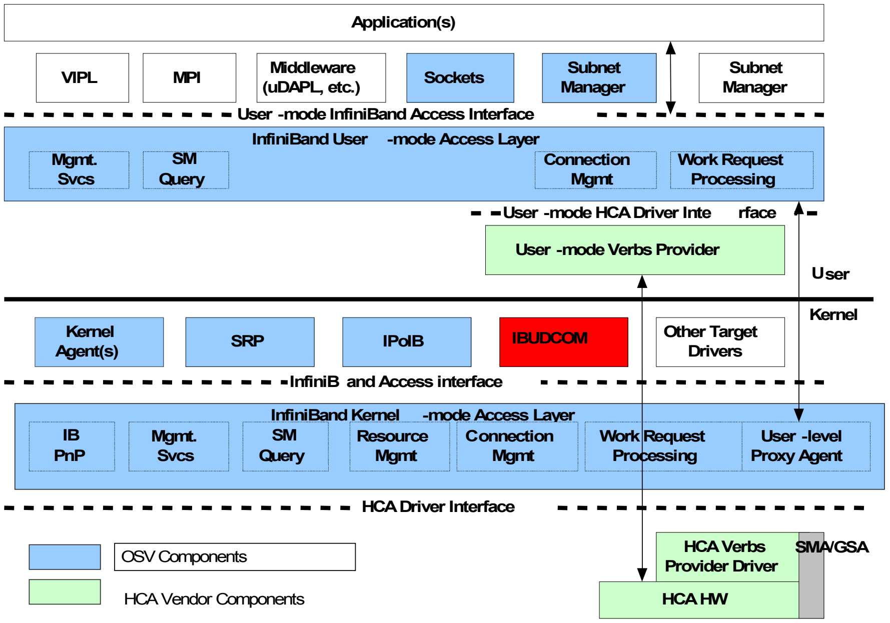


Figure 2.3: InfiniBand architectural component block diagram [1]

InfiniBand software architecture [1] provides two access layers - kernel mode and user mode - which export the full capabilities of the underlying HCA implementations to higher-level software, provide useful services that would otherwise be duplicated in the independent upper-level components, and coordinate the access to shared InfiniBand resources that are needed by upper-level components.

Similar to other middleware such as SRP (SCSI Remote Protocol) and IPoIB (IP over InfiniBand), the IBUDCOM layer is built on the kernel level access layer. It provides some encapsulated APIs (Application Programming Interfaces) for upper level kernel modules to create communication channels, to send packets with different QoS levels, and to receive packets. During the communication channel initialization, it exchanges its local channel information with other nodes using multicast, and creates a local copy. In this way, in the later communication procedure, instead of using the detailed channel information, the upper level kernel module can use the channel name to locate remote nodes. This is very useful since the channel information varies at different initializations.

In order to reduce interrupt overhead, IBUDCOM disables the notification feature of the InfiniBand Queue Pair. It provides a polled interface to the Click network element.

2.3.2 Polling extension of network driver and buffer recycling

Most NICs notify the operating system of packet reception events by generating hardware interrupts to the CPU. The CPU then invokes the interrupt handler, which acknowledges the NIC request, and transfers the packets or raises soft interrupts for later processing. This mechanism reduces the response time of a reception event, however it may cause a receive live-lock at high packet arrival rates. In this case, the NICs generate interrupts at a high rate, with the result that most CPU cycles are consumed simply dealing with these interrupts. In order to solve this problem, the latest Linux kernels disable interrupt generation on the NIC and raise a soft interrupt to poll for packets in the NIC's reception ring. After all these packets are processed, the kernel re-enables the hardware interrupt on the NIC. In this way, the hardware interrupt rate is reduced significantly, and the CPU will not be consumed entirely in handling interrupts.

The use of polling solves the live-lock problem, and therefore increases the maximum forwarding rate. However, even the remaining relatively infrequent interrupts are simply too expensive, and the use of soft interrupts still causes significant context switch overhead. For these reasons, we have extended Click's pure polling support to manage the cluster NICs.

The use of pure polling also helps to implement buffer recycling. In the standard implementation of the Linux network stack, buffer management is performed by the operating system's general-purpose memory management algorithms, which requires expensive CPU operations. This memory allocation / de-allocation overhead is reduced in Click by buffer recycling. Click stores unused packet buffers on a recycling list to speed up subsequent packet buffer allocations, and falls back on the general-purpose memory allocator only when the recycling list is empty.

2.3.3 InfiniBand elements

Starting with the Click modular router, we implemented several elements to enable IBUDCOM to communicate with other processing nodes, and to encapsulate and process packets. These elements include FromIB, ToIB, IBEncap, IBPack and IBUnpack.

FromIB and ToIB belong to the network device category of Click elements. FromIB works in ”push” mode, getting packets from the IBUDCOM layer and pushing them into downstream elements in the Click configuration path, while ToIB works in ”pull” mode, pulling packets from upstream elements and sending them to other processing nodes via the IBUDCOM layer and InfiniBand fabric. Based on the IBUDCOM layer, these two elements implement the pure polling to reduce interrupt overhead. We also implemented a recycling pool for efficiently recycling InfiniBand packet buffers for these two elements.

IBEncap belongs to the encapsulation category. It takes the remote processing node's name and the packet type as its input parameters, and encapsulates them with the InfiniBand packet header, which will be used by the remote processing nodes to classify the incoming InfiniBand packets.

IBPack is used to pack several Ethernet packets into one InfiniBand packet, while IBUnpack is used to unpack an InfiniBand packet to several Ethernet packets. Due to the difference between the MTUs (Maximum transmission units) of InfiniBand and Ethernet, we can pack several Ethernet packets into one InfiniBand packet, which can reduce the average processing cost of the IP packets, thus increasing maximum throughput. Moreover, small packets of size less than 64 bytes account

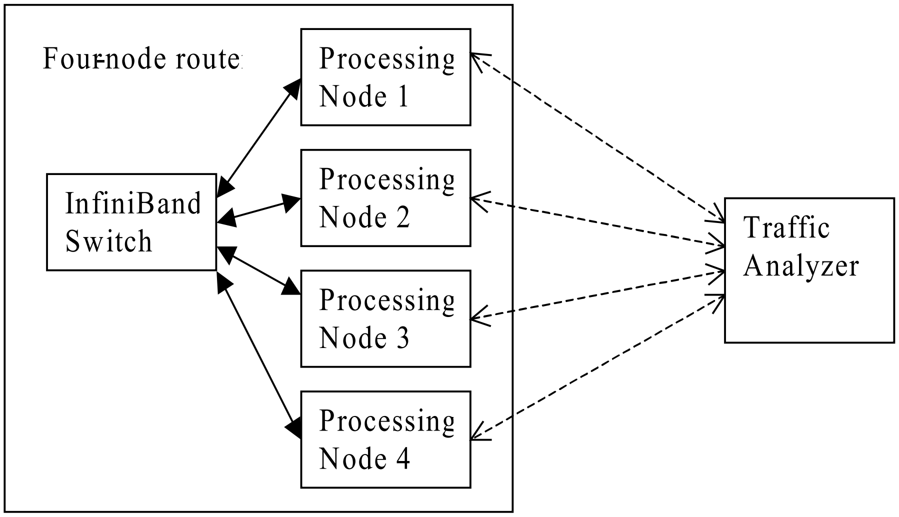


Figure 2.4: Four-node router test setup

for a large part of Internet traffic [6]. Packing several of these small Ethernet packets into a single large InfiniBand packet reduces internal overheads such as PCI arbitration and DMA operations.

IBPack works in ”pull” mode. Normally, it is activated by a downstream ToIB element asking for an InfiniBand packet for transmission. IBPack pulls Ethernet packets from multiple upstream queues as many as possible and packs them into a large InfiniBand packet. The new InfiniBand packet is allocated from the InfiniBand buffer recycling pool, and the Ethernet packets are recycled into the Ethernet buffer pool. Different queue scheduling schemes can be implemented in IBPack, and an index tag is attached to each Ethernet packet, which is used by the IBUnpack element for dispatching Ethernet packets to the correct queues.

IBUnpack works in ”push” mode. When an InfiniBand packet is pushed into this element by upstream elements, it disassembles the InfiniBand packet into Ethernet packets according to the InfiniBand packet header information, and pushes the component Ethernet packets into the correct downstream queues according to their index tags. The InfiniBand packet is recycled to the InfiniBand buffer recycling pool, and the Ethernet packets are allocated from the Ethernet buffer recycling pool.

2.4 Measurements

This section summarizes measurements of the performance of our prototype. Forwarding rates and latency are reported, and the effects of packing multiple Ethernet frames into one Infiniband packet are highlighted.

2.4.1 Experimental setup

We implemented a prototype using a cluster of eight SunFire V20Z nodes interconnected by a Mellanox InfiniBand switch. Each SunFire node has two AMD Opteron 200 Series processors, 4 GB of registered DDR1-333 SDRAM (Synchronous Dynamic Random Access Memory), two BroadCom BCM 5703 Ethernet ports, and one Voltaire InfiniBand 4x PCI-X Host Channel Adapter installed in the 133MHz 64-bit PCI-X slot. A Mellanox MTS2400 24-port Modular InfiniBand Switch System that can support 10 Gbps switching rate at each port is used to connect the HCAs on each node. Packets are generated by a commercial traffic analyzer and sent to the processing nodes of the prototype for forwarding. After being forwarded they return to the analyzer. Figure 2.4 illustrates the test setup of a four-node cluster-based router. The dashed lines represent the traffic going in/out our prototype's Ethernet ports from/to the traffic analyzer's Ethernet ports.

Figure 2.5 shows the details of the Click configuration in the cluster-based router, along a single path from ingress to egress. That is, while we tested routers from two to eight nodes in size, this is the path that would be followed by a packet in any of these configurations. Changing the number of nodes in the router only requires instantiating Click on each node, and then having each node discover the others during initialization. In Figure 2.5, the elements with solid triangle inputs and solid rectangle outputs work in ”push” mode, while the elements with hollow triangle inputs and rectangle outputs work in ”pull” mode. Queue element is special since it is the only element that can store packets. It has a ”push” mode input and a ”pull” mode output. Along the packet forwarding path in the ingress processing node, PollDevice polls Ethernet packets from its associated Ethernet port, Strip strips the Ethernet header from packets, CheckIPHeader validates the IP header, GetIPAddress gets the destination IP address from the packet, and RadixIPLookup uses the destination address to look up the egress port. After IBPack pulls several IP packets from one or more queues into an InfiniBand packet, IBEncap encapsulates the InfiniBand packet with an IBUDCOM header, RoundRobinSched pulls InfiniBand packets from several input flows, and ToIB sends these packets to the egress processing node via IBUDCOM. In the egress processing node, FromIB polls Infiniband packets from IBUDCOM, and IBUnpack disassembles an InfiniBand packet into several IP packets, which are switched into queues of different processing paths. The packets in the queues are encapsulated with an Ethernet header by EtherEncap, and transmitted by ToDevice.

During our tests, we consider both a one-to-one bidirectional traffic pattern and a uniform bidirectional traffic pattern. In the one-to-one traffic pattern, all packets arriving at an ingress node are addressed to one egress processing node, and no egress node gets packets from more than one ingress node. The uniform traffic pattern means packets received by one ingress node are spread over all the other egress nodes uniformly. In both cases, every interface in the cluster both receives and transmits packets, and every packet goes through the InfiniBand switch.

The IP routing table used in all tests is minimal that contains only routes to the class-C sub networks reachable from each port.

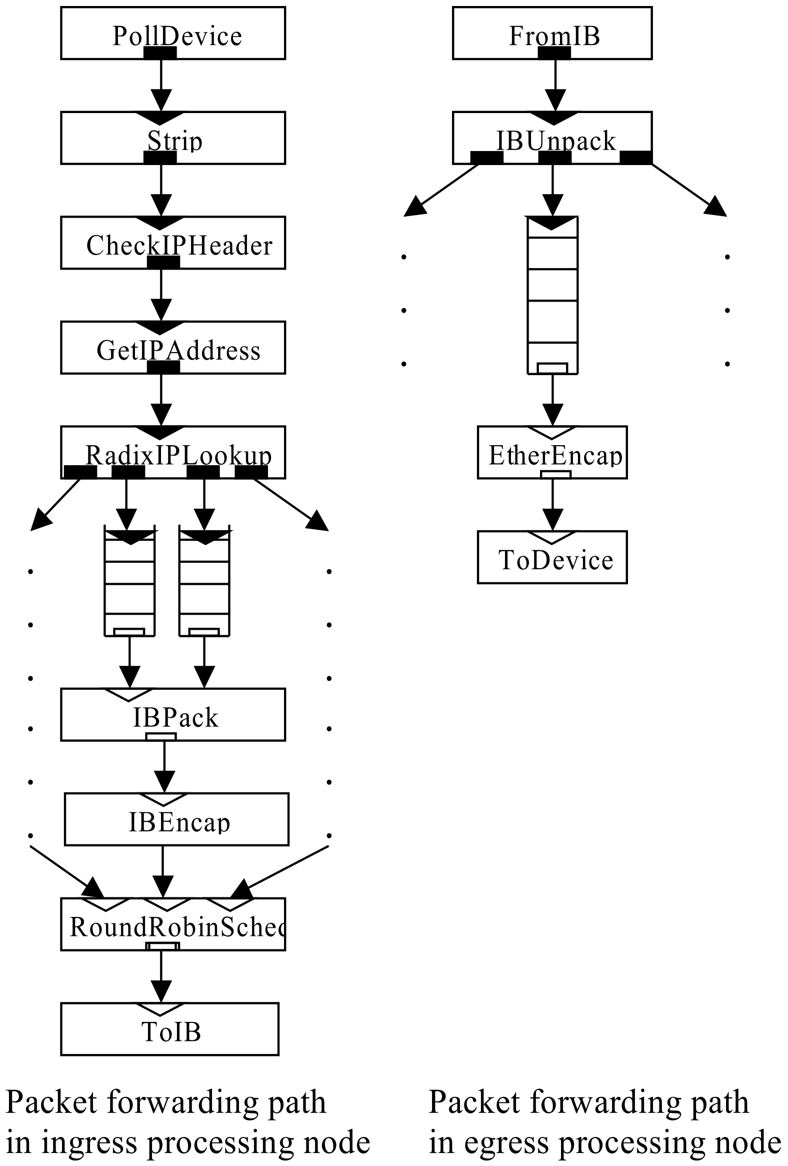


Figure 2.5: Ingress to egress path in cluster

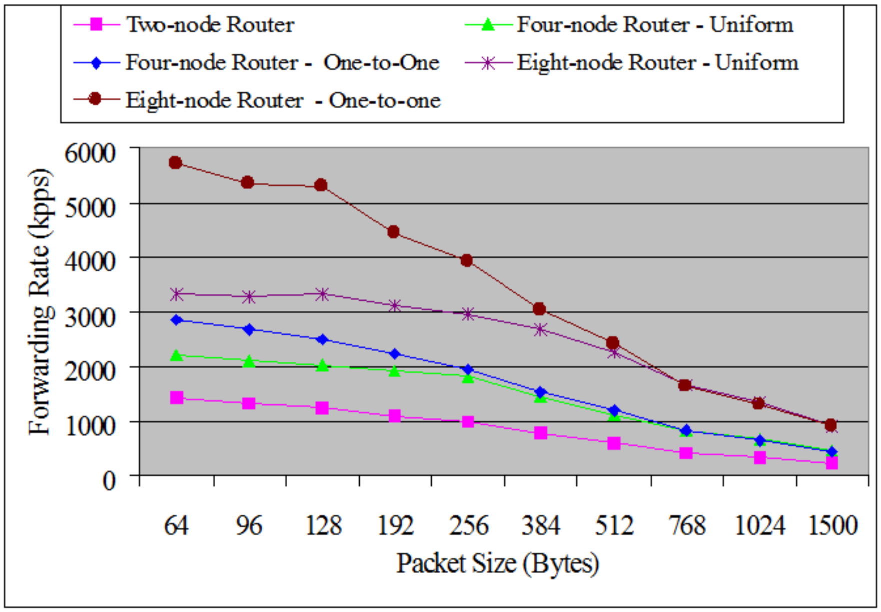


Figure 2.6: Saturated forwarding rates

2.4.2 Forwarding rate

One of the benefits of the cluster-based router is its scalability. That is, the forwarding rate (as well as the number of interfaces) should be easily scaled simply by adding more processing nodes. The goal of our first measurements was to determine whether this was in fact possible. We characterize router throughput by measuring the saturated forwarding rate. This is the forwarding rate at which less than 0.1% of incoming packets are lost. In this case, the offered load saturated some parts of our cluster system, so the offered load at this point is called the saturate traffic load. If the offered load exceeds the saturated load, packets are lost dramatically.

Figure 2.6 shows the saturated forwarding rate as a function of packet sizes for a range of router sizes. Results for both the one-to-one and uniform traffic patterns are given. The two-node cluster router can forward almost 1328k 64-byte packets per second. With the addition of more processing nodes, the total forwarding rate of the cluster-based router increases linearly as a function of number of nodes in the cluster (along a vertical line in Figure 2.6) for the one-to-one traffic pattern. The forwarding rate reaches 5712 kpps (Kilo Packets Per Second) for 64B packets with eight nodes in the cluster, which is 4 times the rate for the two-node cluster. This linear scaling is retained even for larger packets.

With the uniform traffic, the linear scaling is not clear for small packets. There are decreases for both the 4-node and 8-node configurations. However we again see linear scaling when the packet size reaches 512 bytes. We believe that the behavior with small packets is partially caused by the

Table 2.1: Forwarding rates of 64B packets without packing

	Forwarding Rate (kpps)
Two-node Router	714
Four-node Router - Uniform	1428
Four-node Router - One-to-One	1428
Eight-node Router - Uniform	2384
Eight-node Router - One-to-one	2856

packing issues in the implementation of IBPack. With the uniform traffic, packets are distributed evenly into the queues that are used to store packets waiting for different internal links, which means fewer Ethernet packets can be packed in one InfiniBand packet. As a result, the benefit of packing is reduced, leading to lower throughput for small packets. This can be partially validated by our investigation of packing effect in subsection 2.4.4.

Besides the packing issues, the link scheduling and switching properties of InfiniBand stack/network may also cause the decrease of forwarding capability for smaller packets in the 4-node and 8-node clusters. We compared the forwarding rates of 64B packets in clusters of different sizes without any packing (see Table 2.1). With one-to-one traffic, the forwarding rate without packing scales linearly. However, with the uniform traffic more links need to be scheduled and more traffic destinations need to be switched at each node as the size of the cluster increases. Linear scaling breaks down for the eight-node cluster. We can conclude from this that, in large clusters, there are other issues besides packing that affect the scalability of forwarding capability for small packets and the uniform traffic pattern. As part of our future work, we will investigate whether link or switch scheduling can be used to resolve this issue.

2.4.3 Latency

Besides the forwarding rate, latency is another important performance metric for routers, because it shows how efficient the forwarding path is. In our tests, latency is defined as the difference between the time a packet was transmitted by and then subsequently returned to and received by the traffic analyzer. This includes all the transmission delay on the media and processing delay in the router.

Figure 2.7 shows latency measurements for the cluster-based routers of different sizes given saturated traffic loads. These results show, firstly, the benefits of using InfiniBand in the cluster: the latency in the cluster is around 100 μsec for packets of different sizes. Secondly, the cluster-based router scales very well for different packet sizes, with no obvious dependence of latency on the number of router nodes in this range.

Figure 2.8 illustrates the latency for packets of different sizes at light loads. The results again show the benefits of a cluster-based router: the latency of 64B to 1500B packets is less than 70 μsec. Observing the scalability of forwarding capability and low latency for packets of different sizes, we can conclude that a relatively powerful, scalable and efficient router can be built from commodity components very economically.

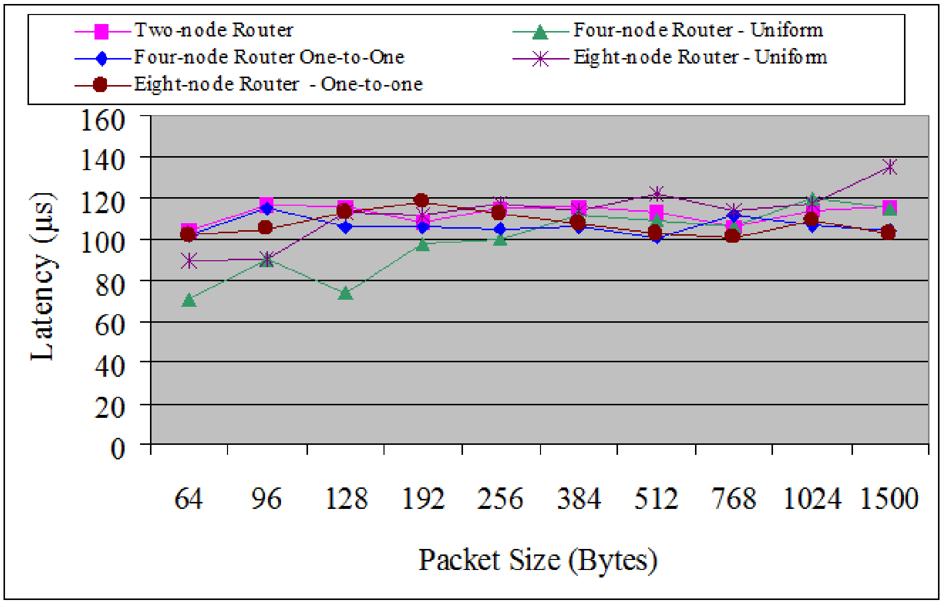


Figure 2.7: Latency at saturated traffic loads

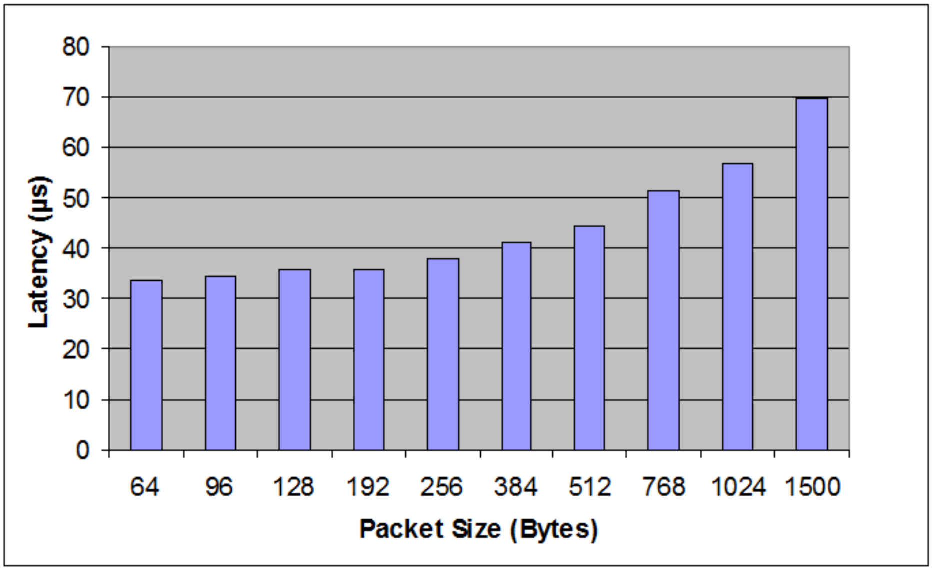


Figure 2.8: Latency at light traffic loads

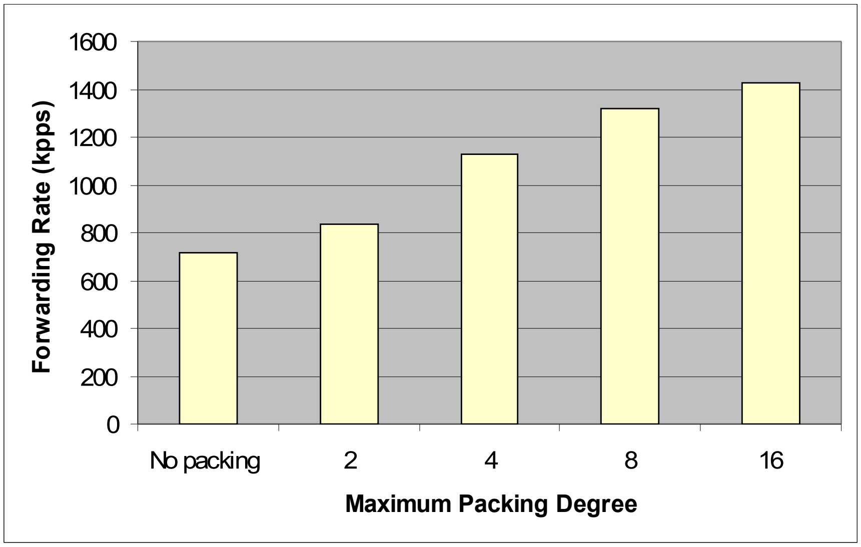


Figure 2.9: Saturated forwarding rates with different packing degrees

2.4.4 Effect of packing

In order to investigate the effect of packing, we measured the performance of forwarding with different degrees of packing and without packing. Packing degree is defined as the maximum number of Ethernet packets that can be packed into one InfiniBand packet. Figure 2.9 gives the saturated forwarding rates for 64 byte packets in the 2-node cluster with different degrees of packing and without packing. With increased packing, the forwarding rate increases dramatically. This demonstrates that packing is an effective way to improve forwarding capability for small packets. Also, we see a drop in forwarding rate for 64B packets when using lower degree of packing. This validates our earlier statement that the decrease of saturated forwarding rates for small packets with the uniform traffic is partially caused by the current implementation of IBPack.

In general, the packing technique in IBPack element improves forwarding rate dramatically, and does not affect packet latency at light loads. Figure 2.10 shows latency for packets of different sizes with and without IBPack at light traffic loads. There is not much difference between these two curves.

2.5 Related work

Two research groups have done some previous work on applying clusters to network packet forwarding and computation. They are the groups working on Michigan State University's Scalable IP

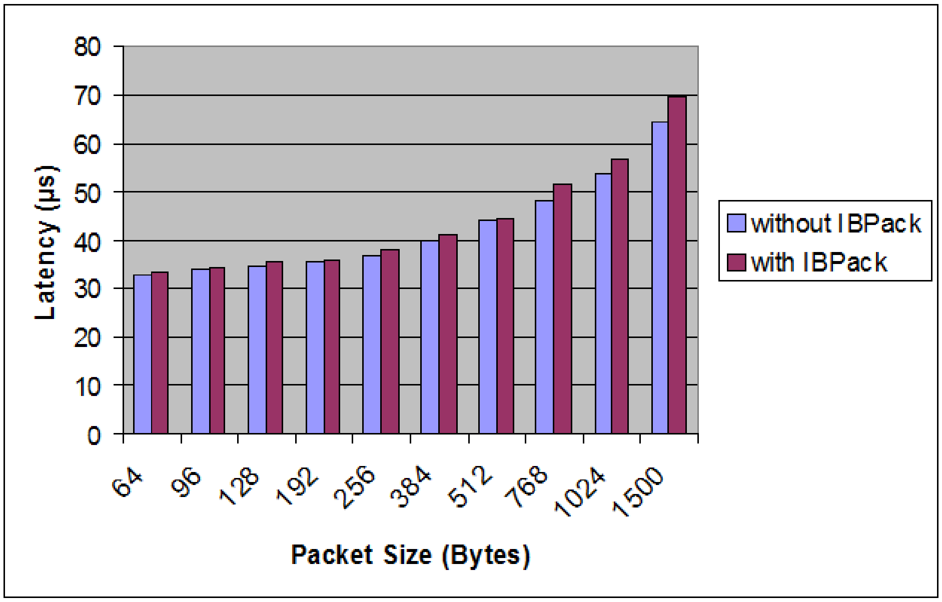


Figure 2.10: Latency at light traffic loads

Router, and State University of New York at Stony Brook's Panama.

2.5.1 Scalable IP router project

Michigan State University [7] was probably the first to introduce clusters into the field of routing, with their Scalable IP Router. They used four Pentium PCs running Linux interconnected by a four-port Myrinet switch, and proposed the concept of Remote Network (RN) devices to keep information about egress nodes, egress ports and internal network interfaces. A RN device is a virtual interface to an IP network, and makes a remote network appear to be directly connected to a node. The RN has the advantage of the intelligence of the IP layer to figure out the egress information without any change to the IP layer. All ports in the cluster can be regarded as local ports of the same node, thus all the techniques used in a traditional router can be used in the cluster-base router. Also, they proposed a scheme for parallel routing table computation [8]. However, there are no published performance results about the scalability of their prototype. They only compared the throughput of one-node and two-node routers, and no further experimental results were released on more nodes.

2.5.2 Panama project

Panama [5] is another scalable and extensible cluster-based router architecture, which supports an aggregate route caching scheme, a real-time link scheduling algorithm whose performance is independent of the number of real-time flows, a kernel extension mechanism to safely load networking software extensions dynamically, and an integrated resource scheduler which ensures that real-time flows with additional packet processing requirements still meet their end-to-end performance requirements. The Panama prototype consisted of four PCs as router nodes, connected by an 8-port Myrinet switch. Each router node had two Myrinet interfaces, each equipped with a Lanai network processor. The Panama kernel running on the ingress Lanai network processor executes packet forwarding and classification tasks, and also performs peer-to-peer DMA to move packets to an internal interface, where the packets are sent to the egress nodes. In this way, there is no difference between external and internal interfaces in their prototype, which is not practical in reality. They report on single packet forwarding paths between one pair of ingress and egress nodes, but give no results concerning scalability. The focus of their experiments was on the impact of different link scheduling policies.

Compared to the previous work, our contributions are the extension of a software router called Click to a 64-bit clean implementation on a multiprocessor cluster platform, and the development of an efficient communication layer called IBUDCOM that abstracts the properties of InfiniBand and presents them to Click as a pure polling mechanism. IBUDCOM makes the high forwarding rate and low latency of InfiniBand available to our cluster-based router. Our router inherits extensibility from Click, and scalability from clustering. In particular, starting from Click has enabled us to implement and test a variety of implementation approaches. While previous work has also addressed the development of a cluster-based router, we have in addition gathered a set of performance measurements to characterize the scaling behavior of our prototype. We believe that such quantitative results are required to validate any design approach, and that this activity has shown where we have succeeded and where further attention is required.

2.6 Conclusions and future work

It is a challenge to design a router architecture to support both extensibility of packet processing services and scalable packet forwarding performance. Most commercial routers support a set of closed, predefined packet processing functions, without offering extensibility. Software routers can be extended to support new packet processing functions, but significant consideration must be given to reduce operating system overheads and mitigate hardware limitations.

In this paper, we report some early results on the scalability and performance of an extensible software router based on the commodity cluster technology. We discuss several key design considerations, focusing on replacing interrupts with polling, ensuring that buffers are recycled economically rather than being discarded, and packing of several Ethernet packets from multiple queues into one InfiniBand packet. Our measurement results show that the forwarding rate of our cluster-based router can be scaled linearly by adding more processing nodes. For example, for 512B packets, the saturated forwarding rate of our 8-node prototype is 3.76 times greater than that of our 2-node machine given the uniform traffic. At a saturated traffic load, latency for packets of different sizes is on the order of 100 μsec, and independent of the number of cluster nodes.

One of the key observations is that a relatively powerful, extensible, scalable router can be built from commodity components very economically. The eight-node instance of our prototype forwards an aggregate 5.7 million 64B packets per second, and attains a maximum aggregate data rate of 10.528 Gbps with 1500B packets. All the nonrecurring costs of creating this particular prototype were in the development of the (reusable and retargetable) software, rather than in the design of custom hardware.

We have investigated the feasibility of this cluster-based router prototype and presented some initial performance results. Many issues including link scheduling and the details of the interconnection abstraction layer remain to be investigated. The bottlenecks in the forwarding path such as processor, interconnection, memory and internal bus are also being analyzed. Moreover, the extension of this cluster-based router to support high rate uplinks is under study.

Bibliography

[1] Intel Corporation. Software architecture specification SAS. White paper, 2002.

[2] Mellanox Technologies Inc. Mtpdk144 switch. Product introduction, Available online, 2006. http://mellanox.com/products.

[3] Mellanox Technologies Inc. Understading PCI Bus, PCI-Express and InfiniBand architecture. White paper, 2006.

[4] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[5] Prashant Pradhan and Tzi-cker Chiueh. Implementation and evaluation of a QoS-capable cluster-based IP router. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society Press, 2002.

[6] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.

[7] Vibhavasu Vuppala and Lionel M. Ni. Design of a scalable IP router. In *IEEE Proc. of Hot Interconnects Symposium V*, page 7 pages. IEEE Computer Society Press, 1997.

[8] Xipeng Xiao and Lionel M. Ni. Parallel routing table computation for scalable IP routers. In *CANPC '98: Proceedings of the Second International Workshop on Network-Based Parallel Computing*, pages 144–158, London, UK, 1998. Springer-Verlag.

Chapter 3

Hardware Bottleneck Evaluation and Analysis of A Software PC-based Router

3.1 Introduction

The Internet technology has been developed fast in the last decades. The variety and intensity of Internet applications are increasing rapidly: applications requiring real-time services such as video-on-demand and immersive virtual reality are becoming common. As the core parts of Internet, IP routers are expected to support these applications and must do more than just traditional packet forwarding. Routers must be easily extensible so that they can include new packet processing functions such as network address translation, packet filtering and scheduling, and complex, differential QoS.

Unfortunately, most commercial routers are based on custom hardware and software. They support a set of closed, predefined processing functions but require hardware upgrades or outright replacement to add new functionalities. Most of the predefined functions can be activated or deactivated by the network administrator, but no new functions that were unanticipated at the time of design can be added into the packet forwarding path.

On the other hand, the software router based on commodity PC hardware increases the extensibility and programmability of packet processing functions. With the feature of low cost, it is an appealing alternative to both scientific researchers and small business users. Significant research has been directed toward the software routers: Click Modular Router [15], Router Plugins [8] , and Extensible Router [14] for the data plane; XORP [11] and Zebra [13] for the control plane. All this research manages to increase the flexibility and manageability and improve the overall forwarding performance by reducing the software overhead. Some new distributed router architectures [18] [20] are also introduced to break the hardware limitations of PC-based software router by utilizing multiple PCs connected by high-speed networks. With the fast development of high end CPUs and other

[0] This chapter is based on the paper ”Hardware Bottleneck Evaluation and Analysis of A Software PC-based Router” published on International Symposium on Performance Evaluation of Computer and Telecommunication Systems, Edinburgh, UK, June 16th-18th, 2008

commodity hardware, the PC-based software routers are promising and continue to be attractive.

The maturity of software routers based on PC hardware provides us an opportunity to explore the potential limitations of software routers. Much work has been done to compare different software routers or different techniques used in a software router. A. Bianco et al [1] [2] evaluated and compared the reception, transmission, and forwarding rates of a Linux based software router and Click modular router. R. Bolla [3] did RFC 2544 performance evaluation including throughput, latency, back-to-back, and loss rate on a dual-processor Linux 2.5.75 standard kernel, a single-processor Linux 2.5.75 optimized kernel, and the Click modular router on a Linux 2.4.21 Kernel. G. Calarco [10] did comparative analysis of different thread scheduling techniques on SMP Click system. However, none of them completely analyzed the potential hardware bottlenecks that may exist on a PC-based software router.

In this paper, we focus on the evaluation and analysis of potential hardware bottlenecks of a PC-based router by exploring the reception, transmission, and forwarding rate of different combinations of configurations of Click modular router. We found that, by applying polling extension of network driver and buffer recycling techniques, the CPU is more than able to process the minimum-size Ethernet packets received and transmitted at one Gigabit network port as well as multiple Gigabit network ports on the same PCI-X bus, reaching around 1.5M packets per second at maximum. However, the Gigabit NICs cannot send the minimum-size Ethernet packets at a full speed. In addition, for both the minimum-size and maximum-size Ethernet packets, there is a potential PCI bus bottleneck in the forwarding path, and the bandwidth utilization of PCI bus is limited to around 70%. The E1000 card and BroadCom behave differently in reception and transmission capabilities. There is a nonlinear correlation between the reception and transmission capabilities of an individual port, as well as multiple ports on the same bus.

This paper is organized as follows. We first describe the hardware architecture and point out some potential hardware bottlenecks in a PC-based router in section 3.2, and then we introduce the Click modular router in section 3.3. Section 3.4 evaluates and analyzes each potential hardware bottleneck by testing and analyzing different Click configurations. Finally, we conclude our work in section 3.5.

3.2 PC-based router architecture and its potential bottlenecks

The commodity PC architecture is a general-purpose unit. Four main parts of a general PC play important roles in a PC-based router: the Central Process Unit (CPU), Random Access Memory (RAM), Network Interface Card (NIC), BUS (Including System Bus, Memory Bus and PCI Bus) and chipsets that glue other three parts together. Figure 3.1 sketched the architecture of a PC-based router and its internal data path.

In Figure 3.1, the solid line gives the data path between different components, and the slashed line depicts a flow path of a packet in a PC-based router. An IP packet arriving at a NIC is forwarded

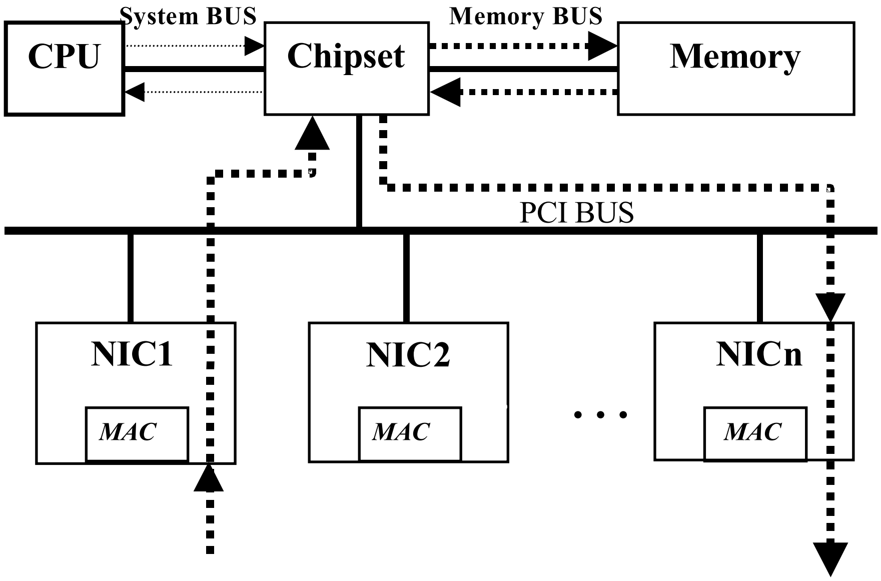


Figure 3.1: PC-based router architecture

to the memory buffer by DMA for temporary storage. Then the CPU fetches the necessary header fields of the IP packet for processing, determines the next hop address of the packet, and tells the appropriate outgoing NIC to transmit the packet. The NIC transfers the packet from host memory to its on-card queue buffer by DMA and sends it out. This is so-called ”store and forward” scheme of IP packets.

In this internal data path of a PC-based router, CPU, memory, buses, and NICs are potential hardware bottlenecks. The system bus in high-end Intel architecture (called Front Side Bus)[7] supports a base system bus frequency of 200 MHz. The address and request interface is double pumped to 400 MHz while the 64-bit data interface (+ parity) is quad pumped to 800 MHz. This provides a matched system bus data bandwidths of 6.4 GB/s. In AMD architecture, the HyperTransport link [5] between the Processor and PCI-X Tunnel is 16 bits wide, supporting transfer rates of 1600 mega-transfers per second, with the achievable maximum bandwidth of 6.4 GB/s. Both architectures support DDR266, DDR333 or DDR2-400 memory, and memory buses are 128-bit wide, which achieve data transfer bandwidth range between 4GB/s and 6.4GB/s. Compared to the capability of CPU processing, PCI bus transfer and NIC processing, the system bus, memory bus, and memory cannot be a bottleneck. We describe three other potential bottlenecks as follows.

3.2.1 CPU

In a PC-based software router, all packets are processed by CPU, so CPU is a potential bottleneck in the path of forwarding packets, especially in the forwarding of small-size packets. The minimum-size packets stress the CPU harder than larger packets. E. Kohler et al [15] measured the cost of forwarding a packet through a Click router in nano-seconds and analyzed the maximum forwarding rate that is consistent with the observed peak forwarding rate. In addition to the processing overhead, cache misses may make the processor idle and wait for memory access, resulting in reduced processing capability.

Due to the complexity of different CPU architectures, it is hard to estimate the processing capability of a CPU only by its frequency.

3.2.2 PCI BUS

The PCI bus [12] was developed in early 1990's by a group of companies with the goal to advance the interface allowing OEM's or users to upgrade the I/O of personal computers. The PCI bus has proven a huge success and has been adopted in almost every PC and server. The PCI bus essentially defines a low level interface between a host CPU and peripheral devices. The PCI architecture utilizes PCI to PCI bridges to extend the number of devices that can be supported on the bus. By the definition a system built from PCI bridges forms a hierarchical tree with a primary bus extending to multiple secondary PCI bus segments. In this kind of shared bus, no more than one device can act as a master at any given time. Therefore, an arbiter is included in the chipset to regulate the access and fairly share of the bandwidth among the connected peripherals.

As other communication protocols, PCI/PCI-X bus protocols also impose penalties on traffic especially on small-size packet traffic. In addition to the actual data transfer, extra control cycles (such as arbitration latency, address phases, attribute phases, and wait states) are required for each bus transaction. This overhead reduces the efficiency of each transaction as well as overall bus performance. For example, a 64-byte data packet requires 8 bus cycles to transfer while the overhead of a transaction requires a minimum of 4 cycles (3 to initiate and 1 cycle to terminate)[6]. In the case of multiple-bus system, this overhead is getting higher [9]. In our evaluation, we find that the PCI-X bus was far less than fully utilized even given Ethernet packets of the maximum size.

3.2.3 NIC

Network interface is another critical element in the PC-based router. Nowadays, most Gigabit Ethernet NICs are high-performance PCI cards equipped with transceiver, media access controller, and one or two on-card buffer memories in size of 64 KB or more to provide storage for data directed to or received from memory. They also come with a Direct Memory Access (DMA) engine that operates as bus-masters to offload the CPU from performing back-and-forth bulk data transfers between their internal buffer memory and the host memory [2].

During the operation of NIC, the transceiver unit stores the packets received at the interface in the reception queue of the on-card buffer memory and sends the packets stored in the transmission queue of the on-card buffer memory out from the interface. At the same time, the DMA engine fetches the packets from DMA transmission queues mapped by the CPU to the transmission queue of the on-card buffer memory and transfers the packets in the reception queue of the on-card buffer memory to the DMA reception queue mapped by the CPU. The memory controller in the network card arbitrates the on-card memory accesses.

In our tests, we find that NICs of different brands behave differently regarding to the reception and transmission capabilities, and the reception and transmission capabilities are correlated in a nonlinear way with each other.

3.3 Click modular router

Click [15] is a modular software architecture for building flexible and configurable routers based upon general-purpose processors. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions such as packet classification, queuing, scheduling, and interfacing with network devices. A router configuration is modelled as a directed graph with elements at the vertices: packets flow along the edges of the graph. As a result, Click is very easy to extend and to reconfigure at runtime. Extending Click to support new services simply requires adding elements that implement these functions at the right place in the flow-based context configuration. These elements can be scheduled without modifying the core scheduling.

Two important features of Click architecture are ”pull” processing and flow-based router contexts. ”Pull” processing decouples the source and sink of a packet forwarding path by allowing the sink to decide when to pull data from the source. ”Pull” processing can recursively propagate towards the source, if needed. The flow-based router context allows a module to perform a closure operation over the modules in the computation graph that are reachable from it and to extract important information about them.

Figure 3.2 shows a simple Click router configuration as a diagram of function calls that are made as a packet moves through the router [15]. In this figure, the elements with solid triangle inputs and solid rectangle outputs work in ”push” mode, while the elements with hollow triangle inputs and hollow rectangle outputs work in ”pull” mode. The Queue element is special because it is the only element that can store packets. It has a ”push” mode input and a ”pull” mode output. During the push, control flow moves forward through the element graph starting at the receiving device; during the pull, control flow moves backward through the graph, starting at the transmitting device. The packet always moves forward.

A widely held misconception is that polling is always less efficient than interrupt-based I/O because of the processor time spent polling. However, a pure interrupt-based approach can cause

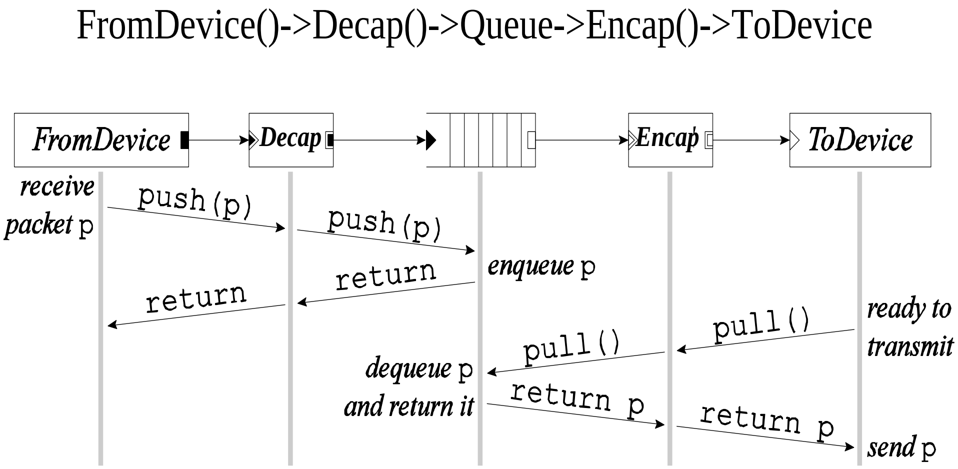


Figure 3.2: A simple Click configuration

receive live-lock problems at high packet arrival rates, in which case the system spend all of its time processing interrupts [17]. In order to solve the receive live-lock problem, the latest Linux kernels disable interrupt generation on the NIC in the interrupt handler and raise a soft interrupt to poll for packets in the NIC's reception ring [19]. After all these packets are processed, the kernel re-enables the hardware interrupt on the NIC. In this way, the hardware interrupt rate is reduced significantly, and the CPU will not be consumed entirely in handling interrupts.

The use of polling solves the live-lock problem, therefore it increases the maximum forwarding rate. However, even the remaining relatively infrequent interrupts are simply too expensive, and the use of soft interrupts still causes significant context switch overhead. For these reasons, Click introduced a pure polling support in the device handling elements called PollDevice and ToDevice and modified the Intel E1000 driver to support this pure polling mechanism [15]. When the pure polling is activated, PollDevice polls its device's receive DMA queue for newly arrived packets; if any are found, it pushes them through the configuration path; ToDevice examines its device's transmit DMA queue for empty slots and filled them by pulling packets from its input. These two elements also refill the receive DMA list with empty buffers and remove transmitted buffers from the transmit DMA list.

The use of pure polling also helps to implement buffer recycling. In the standard implementation of the Linux network stack, buffer management is performed by the operating system's general-purpose memory management algorithm, which requires expensive CPU operations. This memory allocation / de-allocation overhead is reduced in Click by buffer recycling. Click stores unused packet buffers on a recycling list to speed up subsequent packet buffer allocations and falls back on the general-purpose memory allocator only when the recycling list is empty.

SMP Click [4] extends Click to run on a symmetric multiprocessor and provides both scheduling flexibility and high performance. Although SMP Click can exploit the power of SMP machines, as the number of CPUs increases the cost of synchronization and cache misses impact scalability significantly [3].

By applying the techniques mentioned above, Click outperforms other software routers in all the previous studies [1] [2] [3] [15]. In our work, we extended the pure polling support in Click to the BroadCom NetXtreme Gigabit Ethernet driver and use Click as the router software to explore the potential hardware bottlenecks of PC-based routers mentioned in section 3.2.

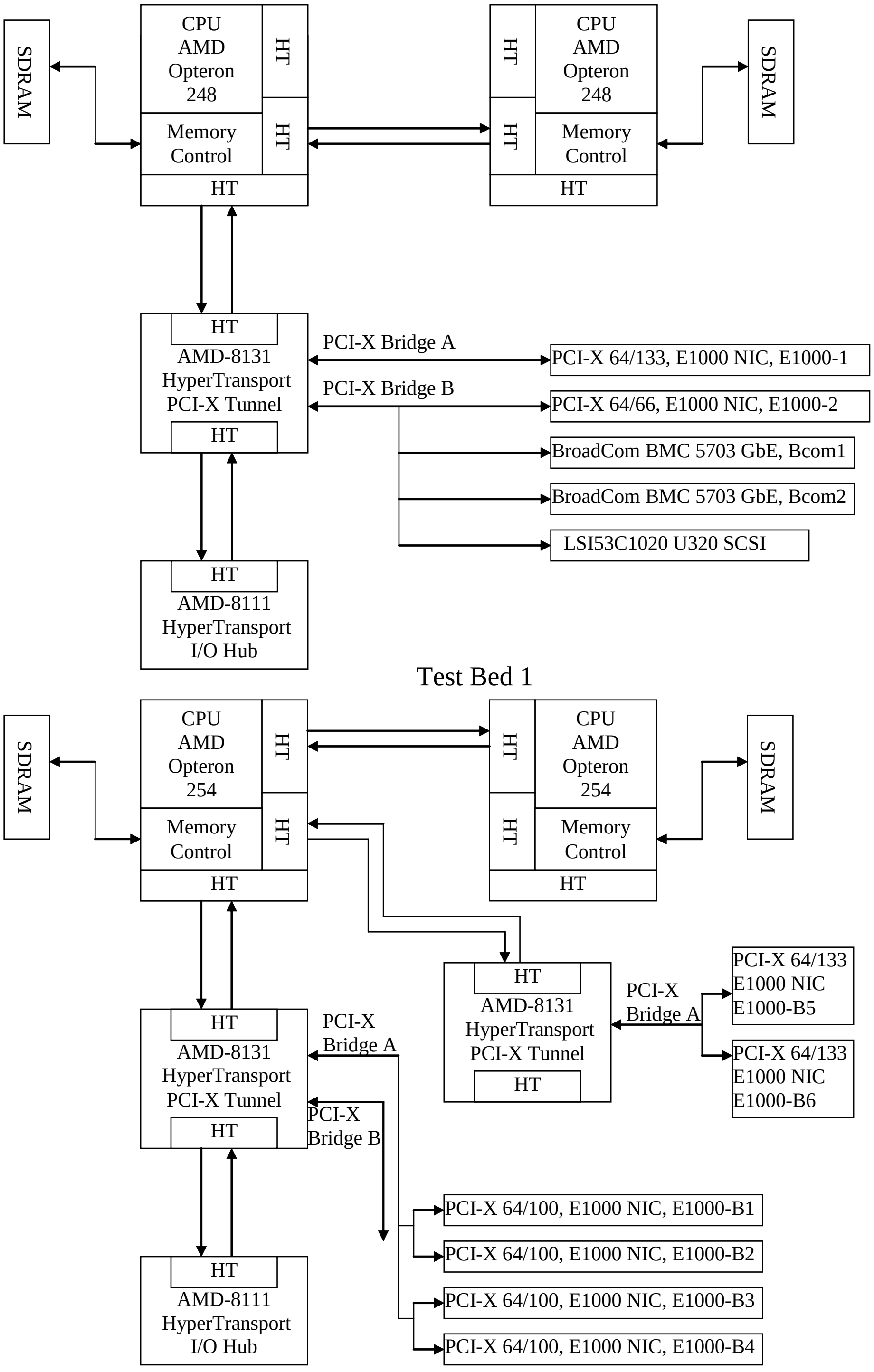
3.4 Evaluation and analysis

The goal of our evaluation is to determine what bottlenecks exist which restrict network throughput in the PCI, NIC or CPU of a PC-based router by several sets of tests. In this section, we introduce our test-bed setups including the test-bed architecture and the deployment of network ports. Then we describe the configurations of each set of tests for investigating the limitation of each potential hardware bottleneck.

3.4.1 Test setup

Our tests are based on two high-end PC servers with AMD HyperTransport architecture. One is equipped with two 2.2 GHz AMD Opteron 248 processors, two on-board BroadCom BMC 5703 Gigabit ports, and two Intel E1000 single-port cards. The other one is equipped with two 2.8GHz AMD Opteron 252 processors, four on-board Intel E1000 ports, and one dual-port Intel E1000 card. AMD-8131 HyperTransport PCI-X Tunnels provide the high-speed data channels between CPUs and PCI buses. As we described in section 3.2 [5] , the uplink of the AMD-8131 HyperTransport PCI-X Tunnel is 16 bits wide, supporting transfer rates of 1600 mega-transfers per second, which can achieve the maximum bandwidth of 6.4 GB per second. Figure 3.3 [16] gives the partial architecture block diagrams of our test-beds with NIC information we deployed.

As we can see from the architecture block diagram, in test-bed 1, two on-board BroadCom BMC 5703 GbE ports are connected to PCI-X bridge B with a PCI-X 66MHz slot. A PCI-X 133MHz slot is connected to the PCI-X bridge A. We install Intel E1000 single-port cards to each of these PCI-X slots, and we name the one in the PCI-X 133MHz slot E1000-1 and the other one in the PCI-X 66MHz slot E1000-2. The on-board BMC ports are named Bcom1 and Bcom2. In test-bed 2, four on-board E1000 ports are connected to the PCI-X bridge A of one AMD-8131 HyperTransport Tunnel, which operate at 100MHz and are named E1000-B1, E1000-B2, E1000-B3, and E1000-B4. We insert an E1000 dual-port card into the PCI-X 133MHz slot connected to the PCI-X bridge A of the other AMD-8131 HyperTransport Tunnel, and we name these two ports E1000-B5 and E1000-B6.


Test Bed 1

Test Bed 2

SDRAM
CPU AMD Opteron 248
HT
Memory Control
HT
HT
HT
CPU AMD Opteron 248
Memory Control
HT
SDRAM
HT
AMD-8131 HyperTransport PCI-X Tunnel
HT
PCI-X Bridge A
PCI-X Bridge B
PCI-X 64/133, E1000 NIC, E1000-1
PCI-X 64/66, E1000 NIC, E1000-2
BroadCom BMC 5703 GbE, Bcom1
BroadCom BMC 5703 GbE, Bcom2
LSI53C1020 U320 SCSI
HT
AMD-8111 HyperTransport I/O Hub
SDRAM
CPU AMD Opteron 254
HT
Memory Control
HT
HT
HT
HT
CPU AMD Opteron 254
Memory Control
HT
SDRAM
HT
AMD-8131 HyperTransport PCI-X Tunnel
HT
PCI-X Bridge A
PCI-X 64/133 E1000 NIC E1000-B5
PCI-X 64/133 E1000 NIC E1000-B6
HT
AMD-8131 HyperTransport PCI-X Tunnel
HT
PCI-X Bridge A
PCI-X Bridge B
PCI-X 64/100, E1000 NIC, E1000-B1
PCI-X 64/100, E1000 NIC, E1000-B2
PCI-X 64/100, E1000 NIC, E1000-B3
PCI-X 64/100, E1000 NIC, E1000-B4
HT
AMD-8111 HyperTransport I/O Hub

Figure 3.3: Partial architecture block diagram of our test-beds

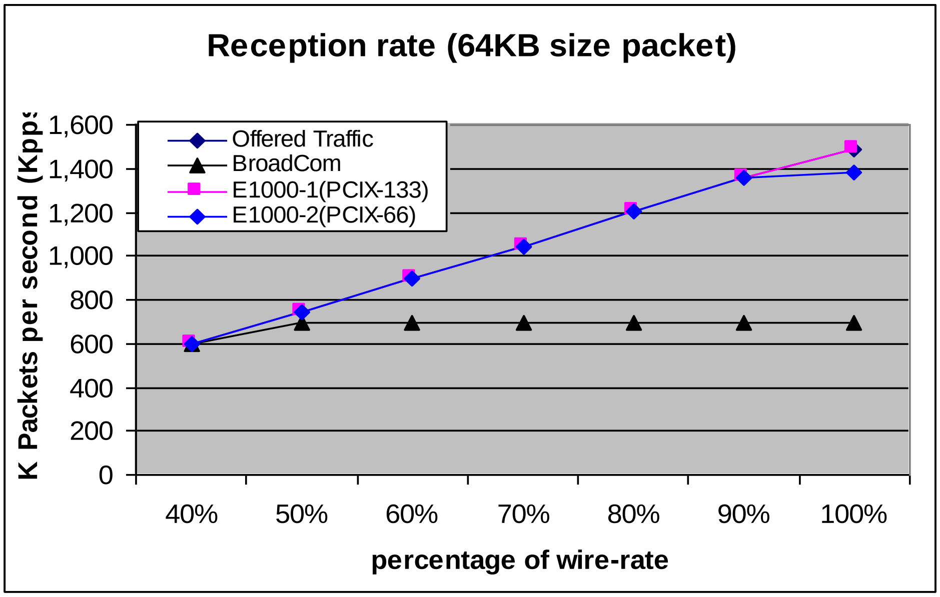


Figure 3.4: Reception rates of individual ports for 64B packet

During our evaluation, we run several sets of tests on these two test-beds to compare the behaviors of different NICs, evaluate the impact of different PCI-X buses, and study the CPU capability. The detailed evaluation and analysis are as follows.

3.4.2 NIC

In this section, we first compare the performance (peak reception, transmission, and forwarding rate) of different NICs in different buses under the receive-only, transmit-only, and concurrent receive and transmit test cases, and then we study the correlation between transmission and reception capabilities of each type of NIC. All these tests are done on test-bed 1.

During the receive-only test, packets is fed to the port under test, and a Click thread is created to receive the packets. We define the peak reception rate as the maximum rate of packets received by the Click thread given the wire rate traffic. During the transmit-only test, we run a Click thread to generate packets of given sizes at wire rates and send them out from the port under test. We define the peak transmission rate as the rate of packets received on the traffic analyzer. During concurrent receive and transmit test, packets are sent to the port under test, and a Click thread is created to receive the packets and send them back to the traffic analyzer via the same port on which they were received. We define the peak forwarding rate as the maximum rate of packets received at the traffic analyzer given different loads.

Figure 3.4 gives the behavior of each NIC with an increasing load of 64B packets. Figure 3.5 compares the peak reception rates of individual ports for packets of different sizes. The data shows that, the E1000 NIC on a PCI-X 133MHz bus can receive all the offered packets (The offered traffic

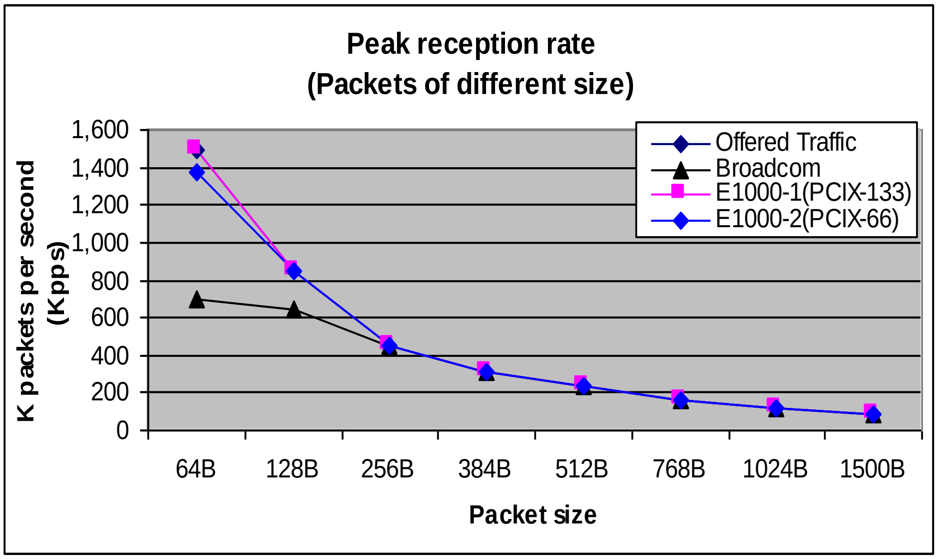


Figure 3.5: Reception rates of individual ports

curve is overlapped by the E1000-1 reception rate curve in Figure 3.4 and Figure 3.5), and the BroadCom NIC performs poorly compared to the E1000 NIC in the case of receive-only operation for small packets. For the minimum-size packets, the receive rate of the BroadCom NIC is limited to around 700K packets per second, which is much less than the E1000 NIC installed on the PCI-X 66MHz slot. We also find that the frequency of PCI-X bus affects the reception rate of a NIC.

The peak transmission rates for packets of different sizes are given in Figure 3.6. In the case of transmission only, the BroadCom NIC performs best for the minimum-size packets even compared to the E1000 NIC on the PCI-X 133MHz bus, transmitting 64B packets at 965 Kpps. As in the reception only case, the frequency of PCI-X bus affects the transmission rate of a NIC, but neither the E1000 NIC nor the BroadCom NIC can send the minimum-size Ethernet packets at the wire rate.

Figure 3.7 shows the peak forwarding rates of individual ports for packets of different sizes. From this data we conclude that, for small packets, the BroadCom NIC performs less efficiently than the E1000 NIC on the PCI-X 133MHz bus but does much better than the E1000 NIC on the PCI-X 66MHz bus.

In order to evaluate the correlations of reception and transmission capabilities of individual NICs, we ran a set of tests on three ports of test-bed 1 given the minimum-size packets. The ports are the BroadCom on PCI-X 66MHz bus, the E1000 NICs on PCI-X 66MHz bus, and the E1000 NIC on PCI-X 133MHz bus. During these tests, we make the NIC under test keep transmitting at its highest rate and, at the same time, increase the offered traffic rate on the same port. Figure 3.8 gives the trend of reception and transmission rate in this situation. For the BroadCom NIC, as the offered rate increases, the transmission rate decreases by 200Kpps while the reception rate increases

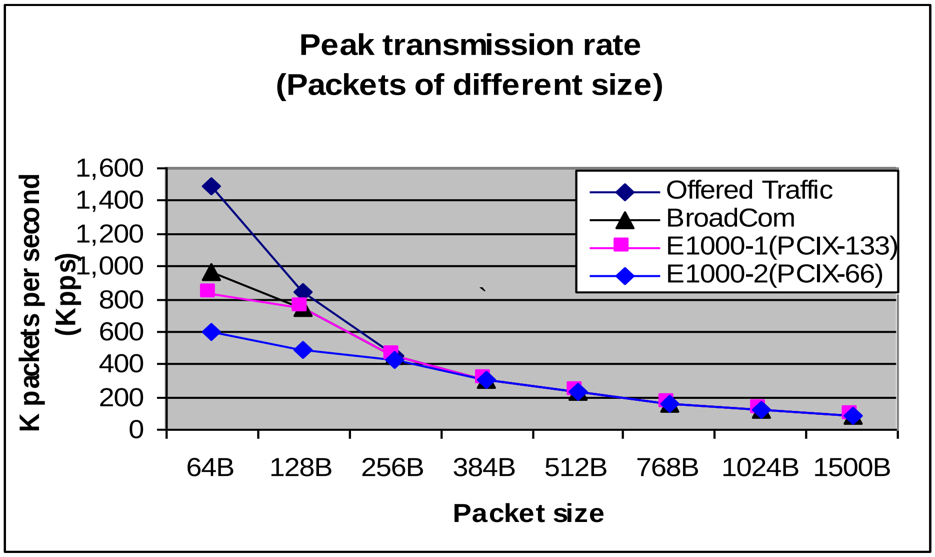


Figure 3.6: Transmission rates of individual ports

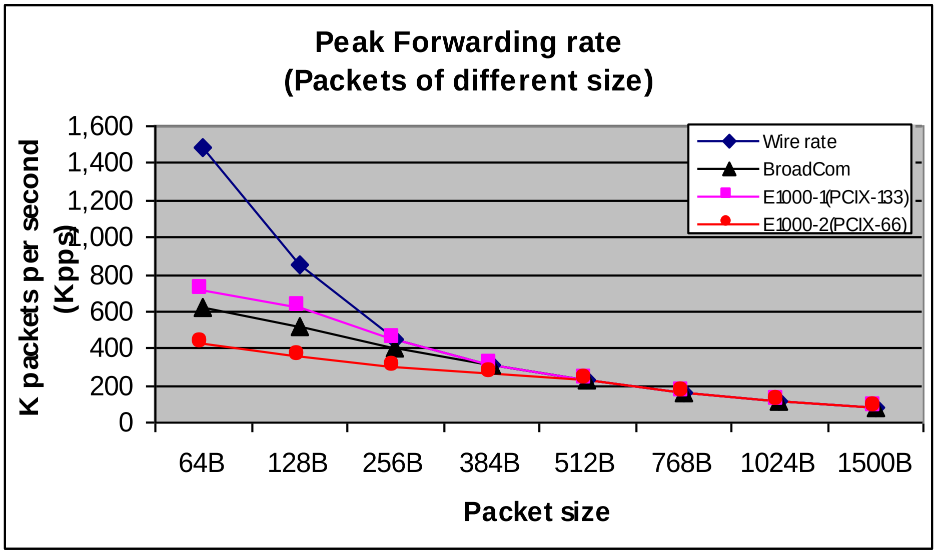


Figure 3.7: Peak forwarding rates of individual ports

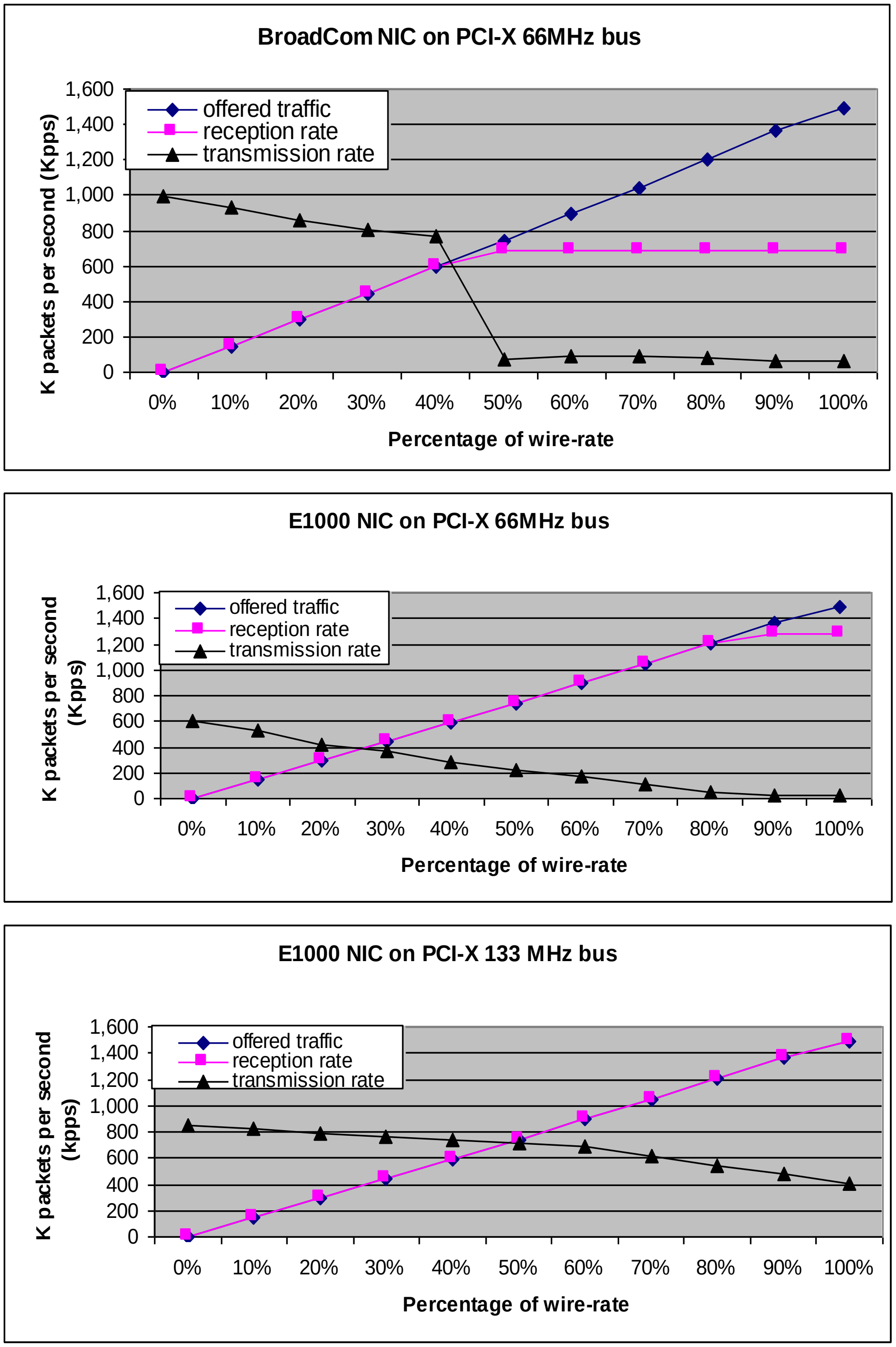


Figure 3.8: Correlations of transmission and reception capabilities of individual ports

by 600Kpps in the range from 0% to 40% of wire rate. At around 50% of wire rate, the reception rate stops increasing, and the transmission rate decreases dramatically from about 800Kpps to less than 100Kpps. Above 50% of wire rate, the transmission rate stabilizes at low rates as the reception rate stabilizes around 700Kpps. While for E1000 NIC, as the offered rate increases, the reception rate increases gradually, and the transmission rate decreases accordingly. As shown in Figure 3.4 and Figure 3.5, the E1000 NIC on the PCI-X 133MHz bus can receive all the offered packets. We suspect that the different correlation behaviors of BroadCom and E1000 NICs are partly caused by their different on-card buffer architectures and on-card memory access schemes.

3.4.3 PCI BUS

As exposed in the NIC capability tests, the frequency of PCI bus affects the performance of individual NICs. In the following tests, we investigate the correlation of behaviors of ports on the same PCI bus and the potential limitation of PCI bus on the aggregate throughput of ports on the same bus.

To study the correlation of behaviors of ports on the same bus, we did tests on three sets of ports: two BroadCom and E1000 port on the PCI-X 66MHz bus of test-bed 1, two E1000 ports on the PCI-X 100MHz bus of test-bed 2, and two E1000 ports on the PCI-X 133MHz bus of test-bed 2. During these tests, we make one port keep transmitting minimum packets at its highest rate and, at the same time, increase the offered traffic rate on the other port of the same bus. We find that as the reception rate of one port increases, the transmission rate of the other port on the same bus decreases. Figure 3.9 illustrates this correlation. The E1000 ports on the PCI-X 100MHz and PCI-X 133MHz bus can receive all the offered packets, resulting in the overlapping of the offered rate and reception rate curves.

Figure 3.10 shows the peak aggregate forwarding rates of ports on the same bus with one, two, or four ports on the same bus involved concurrently in the test. From this figure, we can see that the real forwarding rates are far less than the theoretical bandwidths of the buses. With the number of ports involved in the test increases, the peak aggregate forwarding rates increases but not scaled linearly. On the PCI-X 100 MHz bus, the aggregate forwarding rate of 3 and 4 ports are close to each other, resulting in overlapped curves. Both utilizations of these two buses are around 70% even given the maximum packets.

The results of these two sets of tests and the behaviors of E1000 ports on different PCI buses illustrated in Figure 3.5 , Figure 3.6, and Figure 3.7 indicate that the frequency of PCI bus directly affects the performance of NIC, and it is a potential bottleneck in a PC-based router.

3.4.4 CPU

The goal of this test is to determine the potential impact of processors on the forwarding rate of the PC-based router. In our test, we tested the aggregate forwarding rate for packets of different sizes on the two E1000 ports on the same PCI-X 133MHz, four E1000 ports on the same PCI-X 100MHz

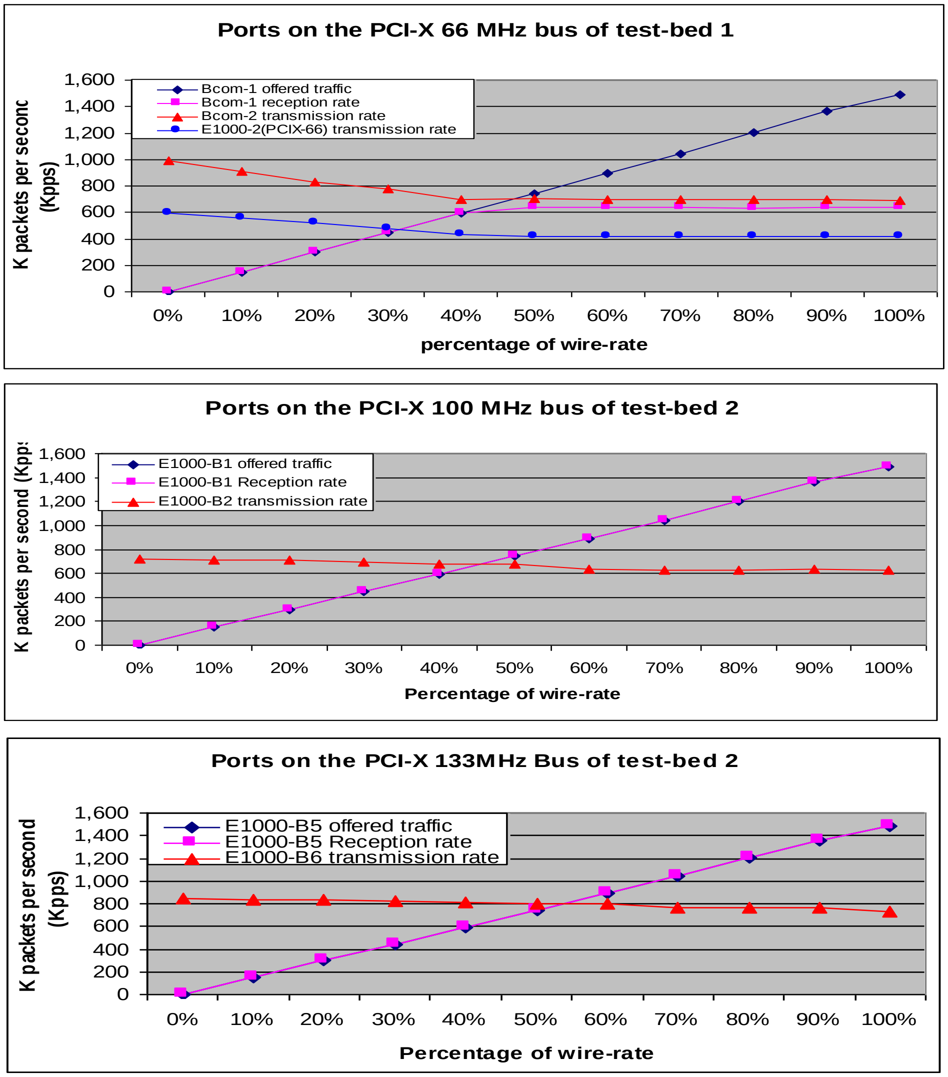


Figure 3.9: Correlations of behaviors of ports on the same bus

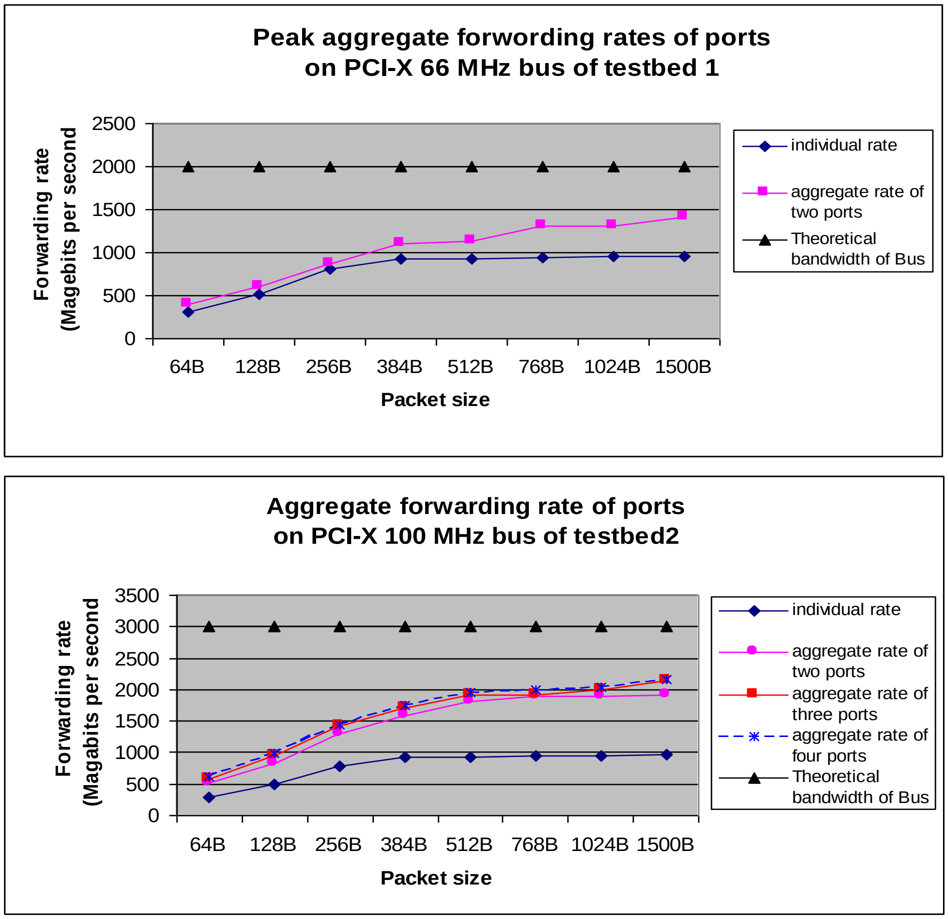


Figure 3.10: Peak aggregate forwarding rates of ports on the same bus

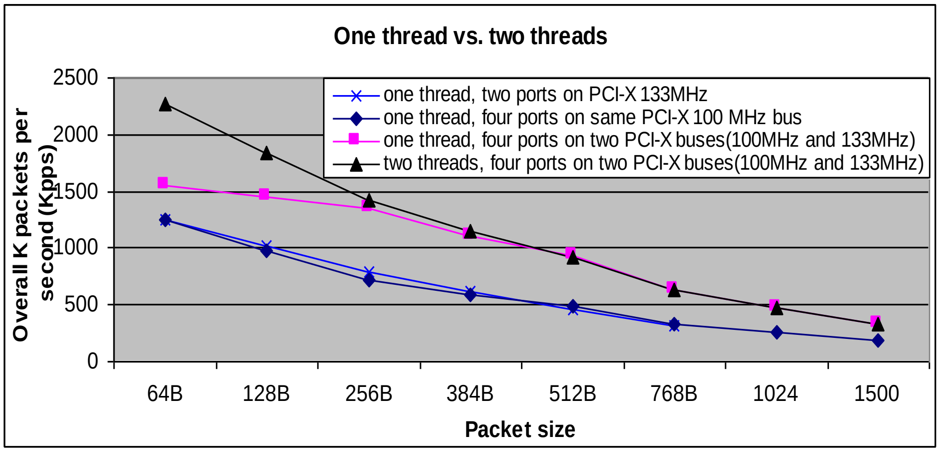


Figure 3.11: Comparison of aggregate forwarding rate between one and two threads

bus, as well as four E1000 ports on different PCI-X buses (two on PCI-X 100MHz bus and two on PCI-X 133MHz bus). For the later case, both one and two Click threads are involved in the test. All these tests are done on the test-bed 2.

Figure 3.11 presents the overall forwarding rates of one thread and two-thread Click running on the above sets of ports. We find that one-thread Click can saturate two ports on the PCI-X 133MHz bus or four ports on the PCI-X 100 MHz bus for packets of different sizes, and it can forward as many as 1.5M minimum size packets per second with ports distributed on different buses. For small packets, the two-thread Click can forward more packets than one-thread Click. Due to the bus limitations, we cannot evaluate the maximum forwarding rate of two-thread Click, but it should be much higher than we achieved as depicted with the black triangle line. The number we get more or less exposes the impact of processor capabilities.

We note that, in our test, no other processing functions are involved in the critical forwarding path in addition to the basic forwarding functions. With more processing functions involved, CPU may become the bottleneck.

3.5 Conclusions

With the availability of high performance router software based on PC hardware and fast development of commodity PC industry, the PC-based software router attracts more and more scientific researchers and business users because of its flexibility and low cost. However, careful selection of PC architectures and components is helpful to maximize its performance/cost ratio.

In this paper, we evaluated and analyzed the potential hardware bottlenecks of a PC-based router, by exploring the reception, transmission and forwarding rates of different configurations of Click modular router. We found that, by applying the polling extension of network driver and buffer recycling techniques, the CPU is more than able to process the minimum-size Ethernet packets received and transmitted at one Gigabit network port as well as multiple Gigabit network ports on the same PCI-X bus, reaching around 1.5M packets per second. However, the Gigabit NICs cannot send the minimum-size Ethernet packets at a full speed. In addition, for both the minimum-size and maximum-size Ethernet packets, there is a potential PCI bus bottleneck in the forwarding path. The utilization of PCI bus is limited to around 70% of its theoretical bandwidth. The E1000 and BroadCom card behave differently in the reception and transmission capabilities. There is a nonlinear correlation between the reception and transmission capabilities of an individual port, as well as ports on the same bus.

As part of our future work, we will extend our work to investigate the Intel architecture and compare its performance with that of AMD architecture.

Bibliography

[1] Andrea Bianco, Robert Birke, Davide Bolognesi, Jorge M. Finochietto, Giulio Galantet, Marco Mellia, Prashant M.L.N.P.P, and Fabio Neri. Click vs. Linux: two efficient open-source IP network stacks for software routers. In *Workshop on High Performance Switching and Routing*, May 2005.

[2] Andrea Bianco, Jorge M. Finochietto, Giulio Gelante, Marco Mellia, and Fabio Neri. Open-source PC-based software routers: A viable approach to high-performance packet switching. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[3] Raffaele Bolla and Roberto Bruschi. RFC 2544 performance evaluation for a Linux based open router. In *Workshop on High Performance Switching and Routing*, June 2006.

[4] Benjie Chen and Robert Morris. Flexible control of parallelism in a multiprocessor PC router. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 333–346, Berkeley, CA, USA, 2001. USENIX Association.

[5] Advanced Micro Devices Corp. AMD-8131 hypertransport PCI-X tunnel data sheet. Data sheet, August 2004.

[6] Intel Corp. Small packet traffic performance optimization for 8255x and 8254x Ethernet controllers application note (AP-453). Application Note, September 2003.

[7] Intel Corp. Intel E7520 memory controller hub (MCH) datasheet. Data sheet, February 2005.

[8] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernhard Plattner. Router plugins: a software architecture for next generation routers. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 229–240, New York, NY, USA, 1998. ACM.

[9] Ehud Finkelstein and Shlomo Weiss. A PCI bus simulation framework and some simulation results on PCI standard 2.1 latency limitations. *Journal of Systems Architecture : the EUROMICRO Journa*, 47(9):807–819, 2002.

[10] Calarco Giorgio, Raffaelli Carla, Schembra Giovanni, and Tusa Giovanni. Comparative analysis of SMP click scheduling techniques. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[11] Mark Handley, Orion Hodson, and Eddie Kohler. XORP: an open platform for network research. *SIGCOMM Computer Communnication Review*, 33(1):53–57, 2003.

[12] Mellanox Technologies Inc. Understading PCI Bus, PCI-Express and InfiniBand architecture. White paper, 2006.

[13] Kunihiro Ishiguro. Zebra routing software. Available online. http://www.zebra.org.

[14] Scott Karlin and Larry Peterson. VERA: an extensible router architecture. *Computer Networks.*, 38(3):277–293, 2002.

[15] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[16] Sun Microsystems. SunFire V20Z server architecture. White paper, April 2004.

[17] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 9–9, Berkeley, CA, USA, 1996. USENIX Association.

[18] Prashant Pradhan and Tzi-cker Chiueh. Implementation and evaluation of a QoS-capable cluster-based IP router. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society Press, 2002.

[19] Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov. Beyond softnet. In *ALS '01: Proceedings of the 5th annual Linux Showcase & Conference*, pages 18–18, Berkeley, CA, USA, 2001. USENIX Association.

[20] Qinghua Ye and Mike H. MacGregor. Cluster-based IP router: Implementation and evaluation. In *IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006.

Chapter 4

Adaptive Scheduling to Maximize NIC Throughput Despite Correlated Transmission and Reception Capabilities

4.1 Introduction

With the fast pace of improvements in the performance of commodity off-the-shelf (COTS) components, COTS routers based on commodity hardware and open source operating systems are gaining more and more interest from both scientific and small business users because of their low cost, flexibility, and extensibility. They provide opportunities to implement new router operations and modify or extend router functions to meet specific research and business requirements. Significant research has been directed toward COTS routers, including the development of the Click Modular Router [10], Router Plugins [5], and Extensible Router [9] for the data plane; and XORP [7] and Zebra [8] for the control plane. All this research manages to increase the flexibility and manageability and improve overall forwarding performance by reducing software overheads. Several new distributed router architectures [13] [14] have also been introduced to break the hardware limitations of COTS routers by utilizing clusters interconnected by high-speed networks. COTS routers can take advantage of the rapid pace of component improvements and are more flexible than standard commercial products.

The maturity of software routers based on COTS hardware provides us an opportunity to explore the performance of COTS routers. Much work has been done to compare different software routers or different techniques used in a software router. A. Bianco et al [1] [2] evaluated and compared the reception, transmission, and forwarding rates of a Linux based software router and Click Modular router. R. Bolla [3] completed RFC 2544 performance evaluation including throughput, latency,

[0] This chapter is based on the paper ”Adaptive Scheduling to Maximize NIC Throughput Despite Correlated Transmission and Reception Capabilities” published on ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, California, USA, November 6-7, 2008

back-to-back, and loss rate on a dual-processor Linux 2.5.75 standard kernel, a single-processor Linux 2.5.75 optimized kernel, and the Click Modular Router on a Linux 2.4.21 Kernel. G. Calarco [6] did comparative analysis of different thread scheduling techniques on SMP Click.

In a recent paper [15] we reported our experimental analysis of the potential bottlenecks in a COTS router. We found that there is a nonlinear correlation between the transmission and reception capabilities of an individual network interface card (NIC) as well as multiple NICs on the same bus. As the reception rate of a NIC increases the transmission capability of that NIC and the transmission capabilities of other NICs on the same bus decrease nonlinearly. Due to this correlation, the forwarding performance of a COTS router can deteriorate dramatically under overload.

In order to balance the transmission and reception capabilities of a single NIC, or NICs on the same bus, and to increase the forwarding rate under overload, we propose an adaptive scheduling mechanism. This mechanism adjusts the scheduling strides of transmission and reception elements dynamically based on the status of queues and the relative ratio of transmission and reception rates. As our experiments show, the proposed mechanism greatly enhances the transmission capabilities of NICs when they are overloaded at the cost of very little additional overhead.

This paper is organized as follows. In section 4.2, we describe the Click Modular Router, an extended version of which we used in our experiments. Then we briefly describe the nonlinear correlation of transmission and reception capabilities of a NIC or NICs on the same bus in section 4.3. Section 4.4 details our proposed adaptive scheduling scheme based on the queue status and the relative ratio of transmission and reception rates in the system. The experimental evaluation of our proposed scheduling mechanism is presented in section 4.5. Finally, we conclude our work in section 4.6.

4.2 Click modular router

Click [10] is a modular software architecture for building flexible and configurable routers based upon general-purpose processors. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions such as packet classification, queuing, scheduling, and interfacing with network devices. A router configuration is modelled as a directed graph with elements at the vertices: packets flow along the edges of the graph. As a result, Click is very easy to extend and to reconfigure at runtime. Extending Click to support new services simply requires adding elements that implement these functions at the right place in the flow-based context configuration. These elements can be scheduled without modifying the core scheduling.

Two important features of the Click architecture are ”pull” processing and flow-based router contexts. ”Pull” processing decouples the source and sink of a packet forwarding path by allowing the sink to decide when to pull data from the source. ”Pull” processing can recursively propagate towards the source, if needed. Flow-based router contexts allow a module to perform a closure oper-

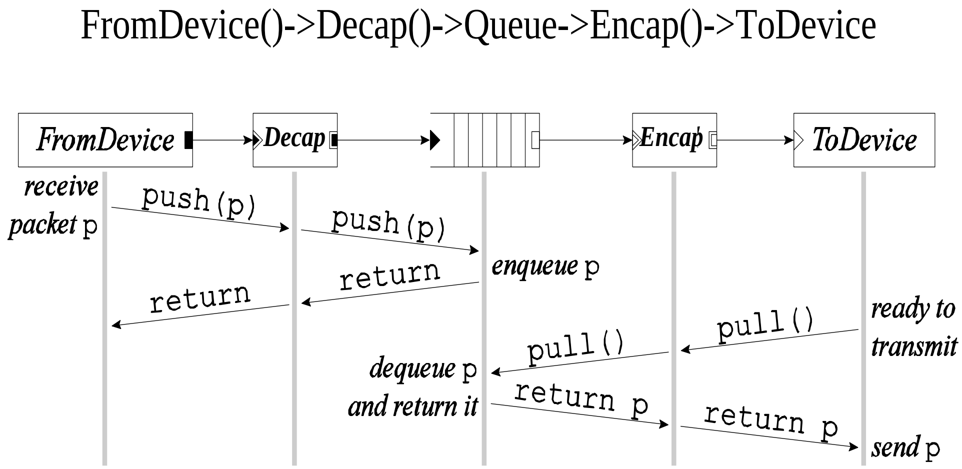


Figure 4.1: A simple Click configuration

ation over the modules in the computation graph that are reachable from it and to extract important information about them.

Figure 4.1 shows a simple Click router configuration as a diagram of function calls that are made as a packet moves through the router [10]. In this figure, the elements with solid triangle inputs and solid rectangle outputs work in ”push” mode, while the elements with hollow triangle inputs and hollow rectangle outputs work in ”pull” mode. The Queue element is the only element that can store packets. It has a ”push” mode input and a ”pull” mode output.

One of the fundamental operations of packet processing is I/O. A widely held misconception is that polling is always less efficient than interrupt-based I/O because of the processor time spent polling. However, a pure interrupt-based approach can cause receive live-lock problems at high packet arrival rates, in which case the system spend all of its time processing interrupts [12]. In order to solve the receive live-lock problem, the latest Linux kernels disable interrupt generation on the NIC in the interrupt handler and raise a soft interrupt to poll for packets in the NIC’s reception ring. After all these packets are processed, the kernel re-enables the hardware interrupt on the NIC. In this way, the hardware interrupt rate is reduced significantly, and the CPU will not be consumed entirely in handling interrupts.

The use of polling solves the live-lock problem, therefore it increases the maximum forwarding rate. However, even the remaining relatively infrequent interrupts are simply too expensive, and the use of soft interrupts still causes significant context switch overhead. For these reasons, Click introduced a pure polling support in the device handling elements called PollDevice and ToDevice and modified the driver for the Intel E1000 NIC to support this pure polling mechanism [10]. When the pure polling is activated, PollDevice polls its device’s receive DMA queue for newly arrived packets; if any are found, it pushes them through the configuration path. ToDevice examines its device's transmit DMA queue for empty slots and fills them by pulling packets from its input. These two elements also refill the receive DMA list with empty buffers and remove transmitted buffers from the transmit DMA list.

The use of pure polling also helps to implement buffer recycling. In the standard implementation of the Linux network stack, buffer management is performed by the operating system's general-purpose memory management algorithm, which requires expensive CPU operations. This memory allocation / deallocation overhead is reduced in Click by buffer recycling. Click stores unused packet buffers on a recycling list to speed up subsequent packet buffer allocations and falls back on the general-purpose memory allocator only when the recycling list is empty.

SMP Click [4] extends Click to run on a symmetric multiprocessor and provides both scheduling flexibility and high performance. Although SMP Click can exploit the power of SMP machines, as the number of CPUs increases the cost of synchronization and cache misses impact scalability significantly [4].

By applying the techniques mentioned above, Click outperforms other software routers in all previous studies [1] [2] [3] [10]. In our work, we extended Click to operate in a cluster interconnected with an InfiniBand switch [14]. This formed the basis for the router software we used to explore the potential correlations between transmission and reception capabilities. We implemented and evaluated our proposed adaptive scheduling scheme on this prototype.

4.3 The nonlinear correlation of transmission and reception capacities of a NIC or multiple NICs on the same bus

Previously [15] we analyzed the potential hardware bottlenecks that may exist in a COTS router, and we found that there is a nonlinear correlation between the transmission and reception capabilities of an individual port as well as multiple ports on the same bus. As the reception rate of a NIC increases, the transmission capability of that NIC and the transmission capabilities of other NICs on the same bus decrease nonlinearly. In this section, we first introduce the setup of our test-bed, and then we briefly describe the nonlinear correlation we have discovered.

4.3.1 Test setup

Our tests were performed on an extended version of the Click modular router installed on a high-end server with AMD HyperTransport architecture. The server is equipped with two 2.8GHz AMD Opteron 252 processors and four on-board Intel E1000 ports. Figure 4.2 gives a partial architectural block diagram of our test-bed [11].

In this system, four on-board E1000 ports are connected to PCI-X bridge ”A” on one of the AMD-8131 HyperTransport Tunnels, which operates at 100MHz.

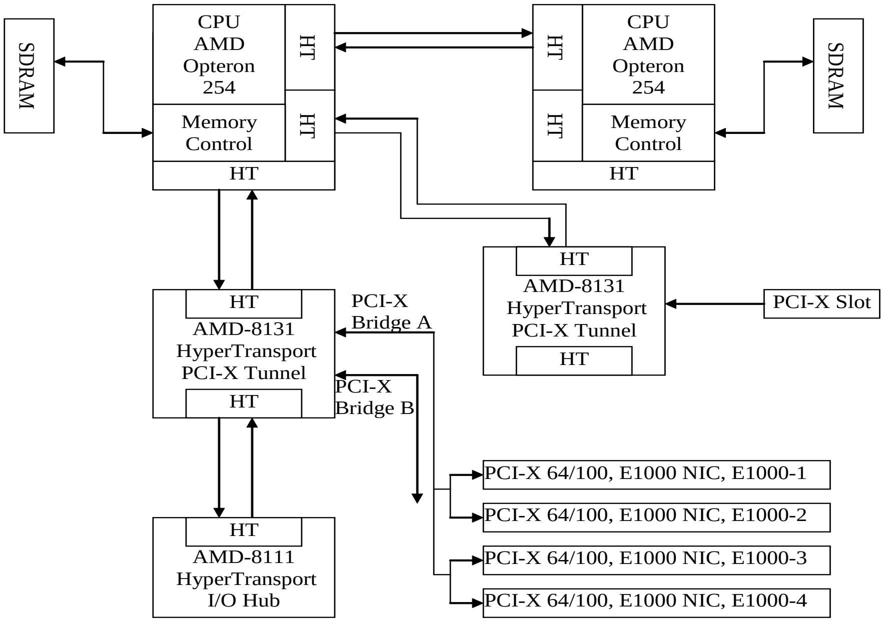


Figure 4.2: Partial architecture block diagram of our test-bed

During our tests, a traffic analyzer is used to generate traffic to the router under test. Since small packets stress the router more than large packets at the same aggregate bit rate because more packets have to be handled per second, we use 64B packets in our tests. The detailed results and analysis follow.

4.3.2 A nonlinear correlation of transmission and reception capacities of a NIC

In order to evaluate the correlation of reception and transmission capabilities of an individual NIC, we make the NIC under test keep transmitting at its highest rate and, at the same time, increase the offered traffic rate on the same port. Figure 4.3 gives the trend of reception and transmission rate in this situation. As the offered rate increases, the reception rate increases gradually, and the transmission rate decreases accordingly.

4.3.3 A nonlinear correlation of transmission and reception of NICs on the same bus

As should be expected and as noted previously [15], the frequency of the PCI bus affects the performance of individual NICs. In the following tests, we investigate the correlation of the behaviors of ports on the same PCI bus by two sets of tests on the two E1000 ports.

In the first test, we evaluated the correlation between the transmission capability of one port and the reception capability of the other. During this test, we make one port transmit at its highest rate

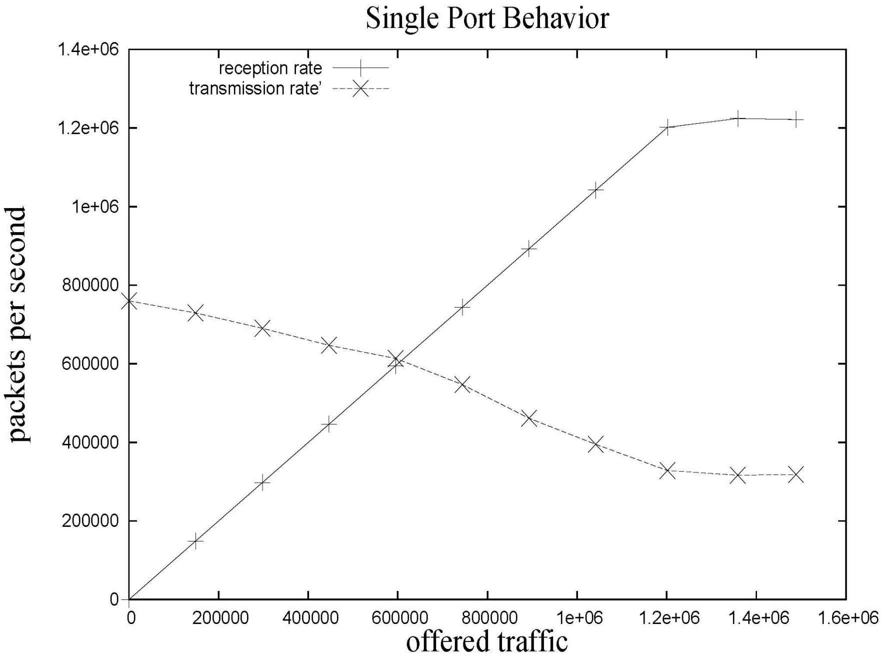


Figure 4.3: A correlation between transmission and reception capabilities of an individual port

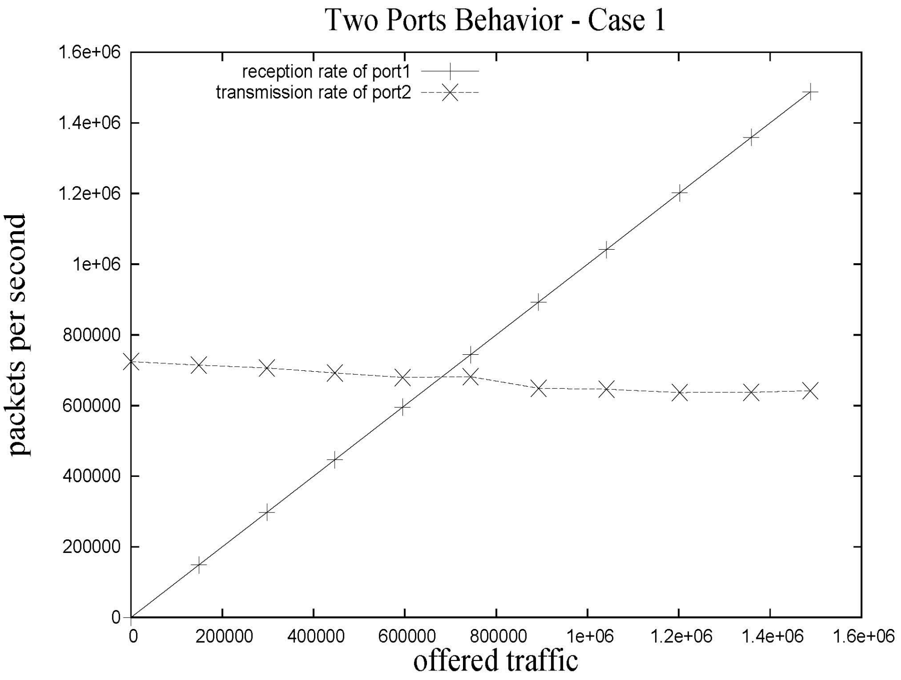


Figure 4.4: A correlation between the transmission of one port and the reception of the other port on the same bus

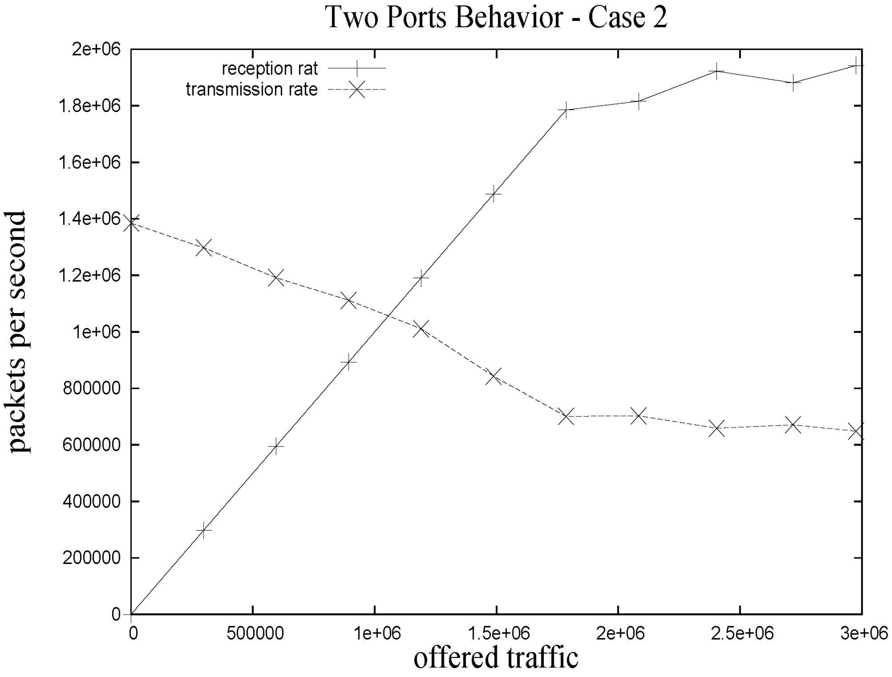


Figure 4.5: A correlation between transmission of both ports on the same bus and the reception of these two ports

and, at the same time, increase the offered traffic to the other port on the same bus. We find that as the reception rate of one port increases, the transmission rate of the other port on the same bus decreases, although slowly. Figure 4.4 illustrates this correlation. This correlation is much slighter compared to the correlation between transmission and reception of a single port described in section 4.3.2.

In the second test, we studied the correlation between transmission rates of both ports on the same bus and the reception rates of these two ports. Different from our previous test, two Click threads are running. During the test, we make both ports keep transmitting at their highest rate and, at the same time, increase the offered traffic rate on both ports. We find that as the offered rate increases, the reception rate increases, and the transmission rate decreases. Figure 4.5 illustrates this correlation.

4.3.4 Controllability of NIC behavior

Noticing the correlation between transmission and reception capabilities, we investigated the controllability of the correlated transmission and reception capabilities. We applied a rate control scheme to the receive side of the NICs and then tested their transmission rates. We ran two sets of tests: one with a single port with the maximum reception rate set to 600Kpps, and the other with

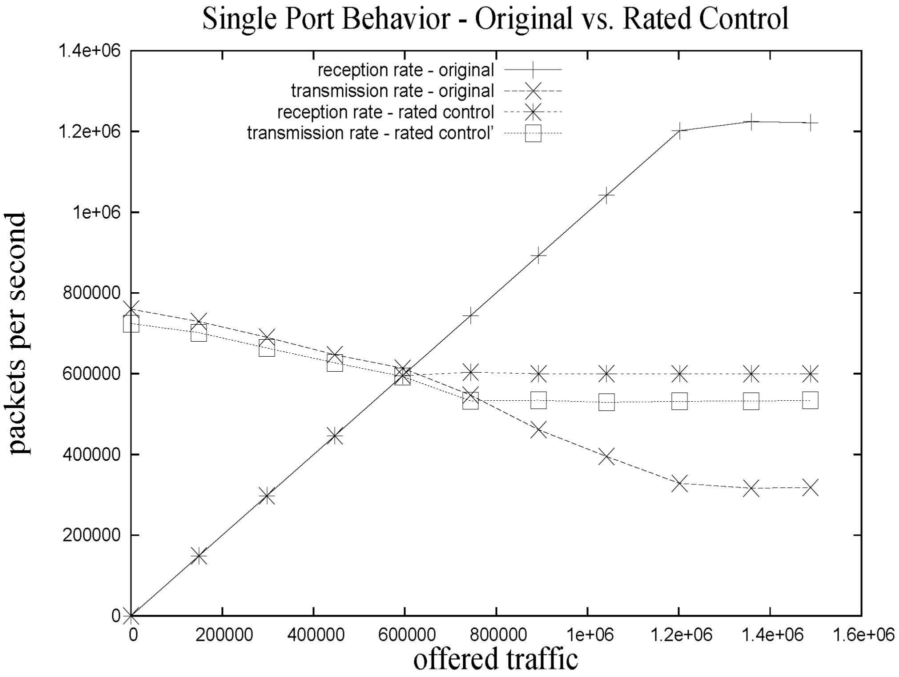


Figure 4.6: Controllability of single port behavior

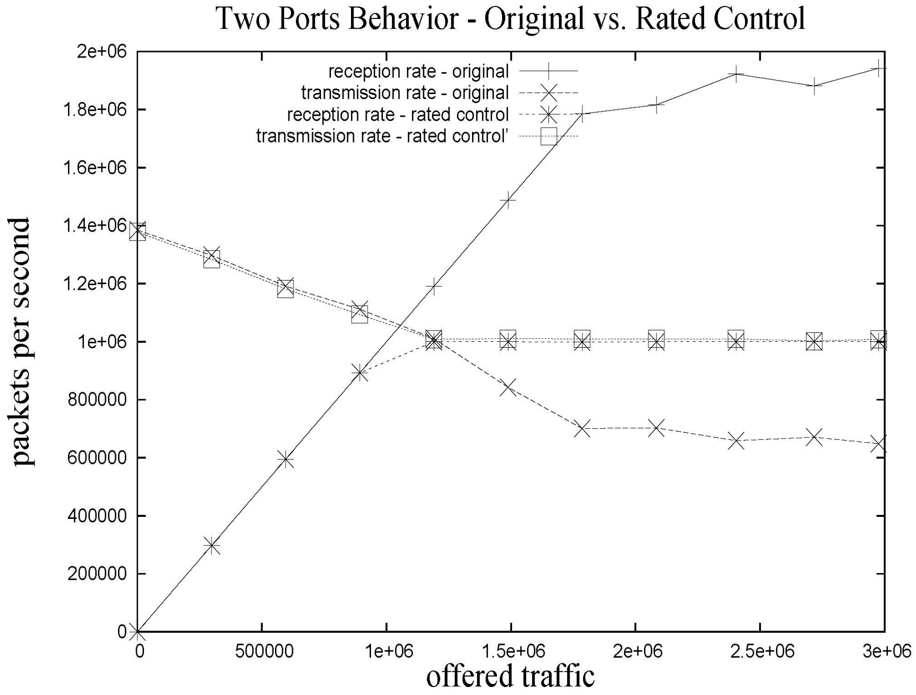


Figure 4.7: Controllability of two ports behavior on the same bus

both ports on the same bus and the maximum reception rate set to 1000Kpps. Figure 4.6 and Figure 4.7 show the NICs' transmission behaviors. With the reception rate controlled to be below a certain rate, the transmission rate under overload can be increased significantly compared to the case of the uncontrolled reception.

We should mention that, due to different NIC architectures and different memory access schemes adopted in different NICs, this controllability may not always be the same. For example, BroadCom BMC 5703 NIC behaves differently than the Intel E1000 NIC. In this paper, we evaluate our scheduling schemes on the E1000 NICs, but the scheme can be applied to other NICs with different control methods.

4.4 Adaptive scheduling for correlated transmission and reception capabilities of NICs

Due to the correlation between the transmission and reception capabilities of an individual NIC or NICs on the same bus, an adaptive scheduling of transmission and reception elements is necessary in the COTS router. If the scheduling weight is updated based on the number of packets in the queue each time a transmission or reception element processed, as Click usually does, the scheduling scheme will favor the reception elements over the transmission elements due to this correlation, resulting in a more unbalanced scheduling and more packets being dropped on the upstream queues of the transmission elements under a heavy load. As a consequence, the forwarding rate deteriorates.

Given this correlation and the controllability of NIC behaviors, to increase the forwarding rate when NICs are overloaded, we propose an adaptive scheduling mechanism that takes into account the correlations between Click elements. This mechanism adjusts the scheduling weights of transmission and reception elements dynamically based on the status of queues and the relative ratio of transmission and reception rates. As shown in our experiments, the proposed mechanism incurs little overhead and greatly enhances the transmission capabilities of NICs when they are overloaded.

In this section, we first describe the usual scheduling scheme used in the Click modular router, and then detail our adaptive scheduling scheme.

4.4.1 Scheduling framework

In the current implementation of the Click modular router [10], each Click thread schedules work by maintaining a task list of schedulable elements. These schedulable elements are those that would like special accesses to CPU times, including the device elements interacting with device drivers such as PollDevice, ToDevice, PullToPush elements, and tasks to handle timer events. These schedulable elements initiate an arbitrary sequence of push and pull requests. However, most elements like EthernetEncap and Queue are not schedulable; they are never placed on the task lists but are implicitly scheduled when their push or pull methods are called by schedulable elements.

The schedulable elements on each task list are scheduled by means of the flexible and lightweight stride-scheduling algorithm [10]. The stride changes according to the number of packets in the queue each time an element returns. If there are packets returned when the element was scheduled, the stride is decreased in proportion to the number of packets returned. If there is no packet returned, then the stride is increased. This scheme works well if there is no correlation between elements. Considering the unbalanced correlation of transmission and reception capabilities of a NIC or NICs on the same bus, this algorithm will make any imbalance worse.

4.4.2 Adaptive scheduling scheme

The application of rate control scheme on the reception rate of NICs can increase the transmission capability of NICs, however, the rate control scheme is expensive in terms of processor overhead. Also, the rate control cannot be set static in the multi-port and multi-bus system where it may limit the performance of uncorrelated ports.

Based on the above observation and Click element scheduling algorithm, we propose and implement an adaptive scheduling mechanism that takes into account the correlations of elements and the system status information. According to the status of queues and the relative ratio of transmission and reception rates, our mechanism dynamically adjusts the scheduling weights of transmission and reception elements at intervals.

In our adaptive scheduling scheme, a schedulable element containing the correlation information is created. The correlation information includes a description of the correlation between transmission and reception elements, their corresponding queues, and thresholds of queue and rate ratio. This schedulable element is scheduled at intervals (according to configured scheduling stride) to monitor the status of queues. If the exponentially weighted moving average of queue length exceeds the preset queue threshold, then the corresponding transmission element is said to be in a congested state. In the congested state, the statuses of the transmission and reception elements are checked to determine whether the congestion is caused by over-scheduling correlated reception elements. By over-scheduling we mean that the reception elements are scheduled too frequently which makes the transmission capability of correlated NICs decrease as a consequence. As a rule of thumb, packets that must be dropped should be dropped as early as possible. Incoming packets should be dropped rather than packets already in the system because packets in the system have already consumed system resources. Considering the correlation of transmission and reception elements of NICs, fewer CPU cycles should be assigned to the corresponding reception elements if there are packets waiting for transmission.

We use the relative ratios of reception and transmission rates as the indicators of the correlation status between transmission and reception elements. Depending on the ratio, three actions are taken. If the ratio is less than MinRatio, we consider the congestion is not caused by the correlation and no scheduling stride is adjusted. If the ratio is larger than MinRatio but less than MaxRatio, the scheduling stride of the reception element is multiplicatively increased according to the current ratio so that the element is scheduled less frequently. If the ratio is larger than MaxRatio, we consider the congestion is definitely caused by the correlation, and the reception element is unscheduled for some time.

According to the Click flexible and lightweight stride-scheduling algorithm, the scheduling strides of reception elements decrease additively after each scheduling as long as there are some packets received during this schedule. In this way, in our adaptive scheduling scheme, the scheduling frequency of reception elements is controlled in an additive increase, multiplicative decrease (AIMD) manner.

4.4.3 Implementation

Our adaptive scheduling scheme is implemented in Click. As mentioned before, a configurable and schedulable element called NICBalance is created. An example of the element configuration is: NICBalance(16, T0 CORS P0 P1 QUEUES Q1 Q2 QTHRESHOLD 100 QSTABILITY 3 RTHRESHOLD 1.3 1.6,). 16 is the scheduling ticket of this element, which is used to calculate the scheduling stride. The larger the scheduling ticket, the smaller the scheduling stride, so that this element is scheduled more frequently. T0 is a transmission element. CORS describes the correlated reception elements of T0 - P0 and P1. The Q1 and Q2 are the upstream queues of T0. The threshold of the queue length is 100. When the exponential moving average queue length is larger than 100, then the queue is considered congested. QSTABILITY describes the factor used to calculate the exponential moving average queue length. The values of 1.3 and 1.6 are MinRatio and MaxRatio thresholds of relative ratio between the reception and transmission rates of correlated elements.

To resume the scheduling of unscheduled elements, the Click main scheduling routine is modified to process unscheduled elements.

4.5 Evaluation

To evaluate the effectiveness of our adaptive scheduling mechanism, we ran two sets of tests on single port behaviors and two sets of tests on two ports behaviors by running the router with and then without our adaptive scheduling mechanism. The test-bed is same as described in section 4.3. Different from the tests in section 4.3, we used a simple Click router configuration on the server under test. In each set of tests, the reception rate, transmission rate, and average queue length are recorded.

In the first set of tests, 64B packets were offered to the port under test. A Click thread was running to receive packets and then send the packets back out the same port after passing through a routing table lookup. The offered rate was increased from 35% of wire rate until full wire rate in steps of 5% of wire rate. Figure 4.8 gives the results. In the original Click router, the reception rate increased as the offered load increases until 80% of wire rate at around 1200kpps. At the same time,

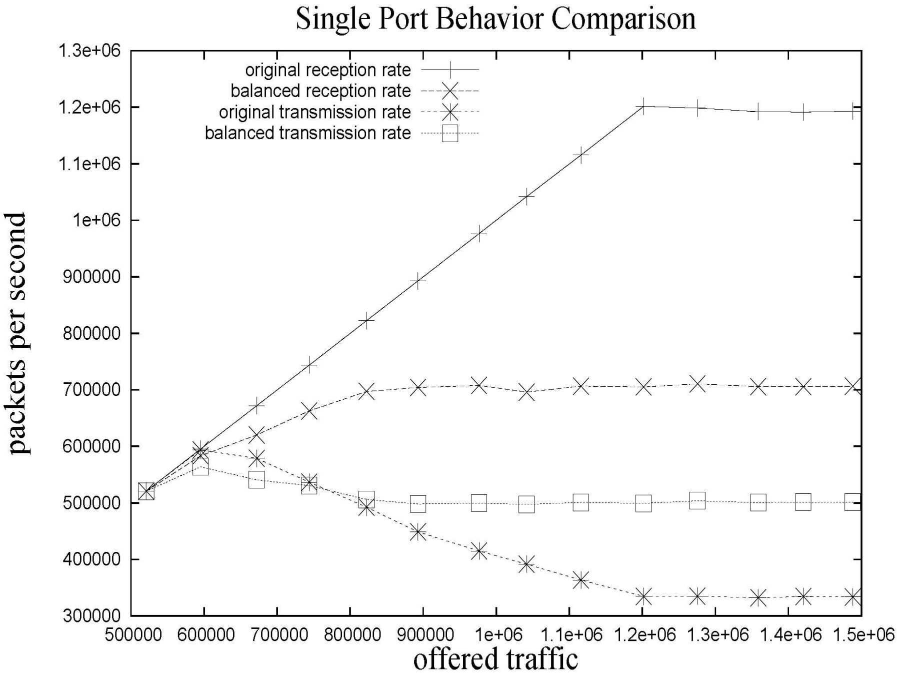


Figure 4.8: Single port behavior comparison

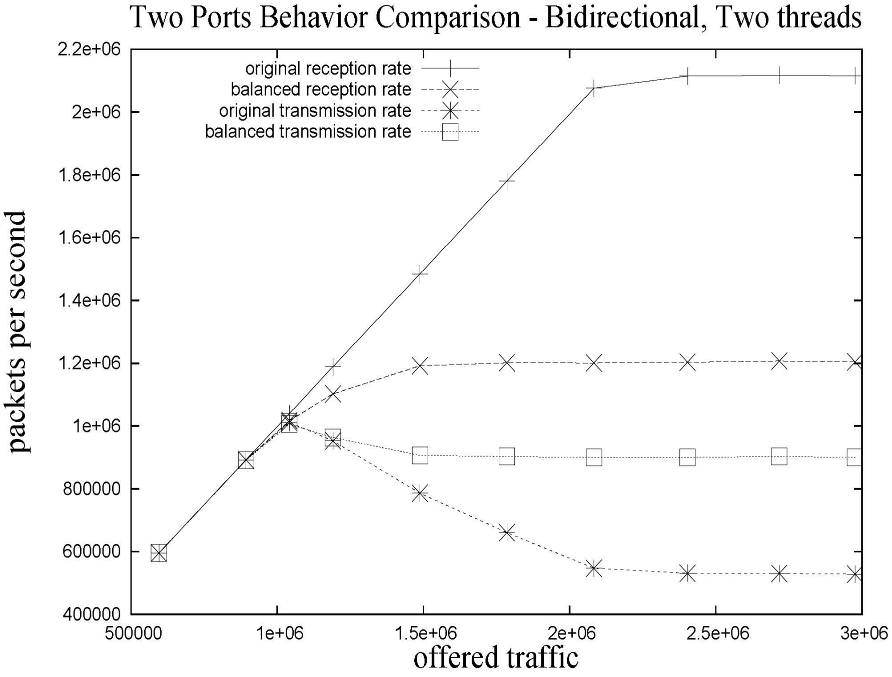


Figure 4.9: Two ports behavior comparison

the transmission rate decreases from 600kpps to 333kpps. In the balanced Click router, the reception rate stops increasing at around 700kpps, but the transmission rate stabilizes at around 500kpps. In most overloaded cases, our scheme can improve the forwarding rate by up to 50%.

In the second set of tests, 64B packets were offered to two ports on the same PCI bus. Two Click threads were running on the router and each thread received packets from one port and sent them back out the other port after passing through routing table lookup. The offered rate was increased from 20% of wire rate to full wire rate in steps of 10% of wire rate. Figure 4.9 gives the results. In the original Click router, the reception rate increases as the offered load increases until 70% of wire rate at around 2100kpps. At the same time the transmission rate decreases from 1000kpps to 500kpps. In the balanced Click router, the reception rate stops increasing at around 1200kpps, but the transmission rate stabilizes at around 900kpps. In most cases, our scheme can improve the forwarding rate by up to 80%.

Figure 4.10 gives the average queue length comparison between the original Click router and the balanced Click router in the two sets of tests mentioned above. When the router is overloaded, the queues in the original Click router are always full. The balanced Click router with our adaptive scheduling mechanism maintains much shorter queues.

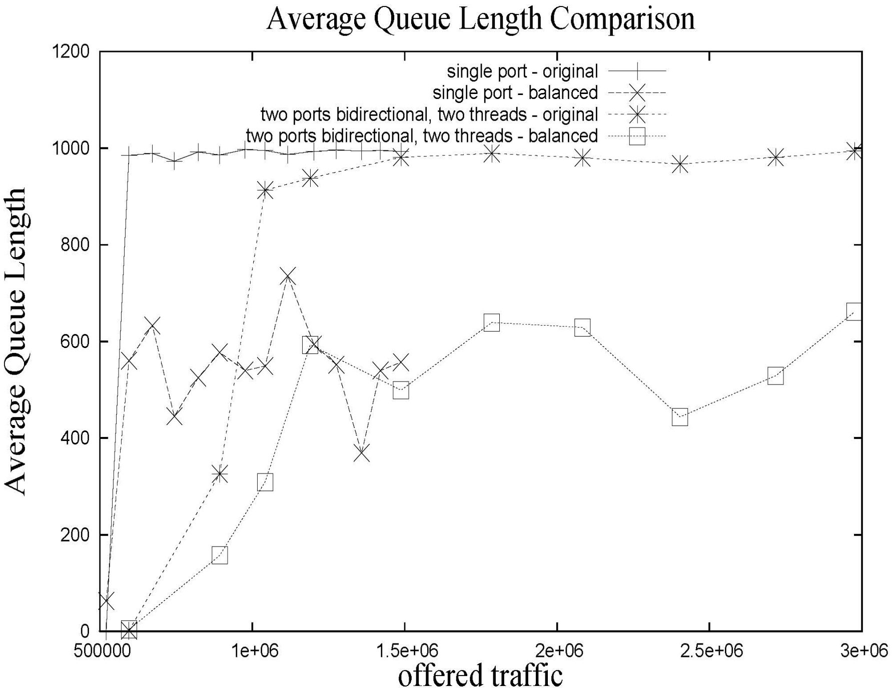


Figure 4.10: Average queue length comparison

4.6 Conclusions

With the availability of high performance router software based on COTS hardware and the rapid development of commodity components the COTS router attracts more and more scientific researchers and business users because of its flexibility and low cost. The architecture and behavior of COTS systems must be understood in detail to maximize the performance/cost ratio of a COTS router.

In this paper, we presented the evaluation and analysis of the effects of the correlation between transmission and reception capabilities of an individual NIC as well as multiple NICs on the same bus. We found that, as the reception rate of a NIC increases, the transmission capability of that NIC or other NICs on the same bus decreases nonlinearly. To manage the adverse effects of this correlation, we propose an adaptive scheduling mechanism based on system state information to balance the transmission and reception rates under overload. Our experimental results show that our proposed adaptive scheduling mechanism is quite effective in increasing the forwarding rate of the COTS router under overload. By applying our proposed adaptive scheduling scheme in an extended version of the Click modular router, the forwarding rate of a single port can be increased by 50% and the forwarding rate of two ports on the same bus can be increased by 80% while the average queue length can be decreased significantly. This also decreases the delay through the router.

4.7 Appendix - Scheduling weight adjust algorithm

The following gives the scheduling weight adjust algorithm as described in section 4.4.2:

Scheduling weight adjust algorithm

Q_{avg}: Average queue length

Q_{ema}: Exponential moving average queue length

R_{send}: Transmission rate

R_{recv}: Reception rate

$Ratio$: Reception/Transmission rate ratio

$Ticket$: Scheduling ticket of an element

```
FOR each correlation DO
REPEAT
    Get the average queue length of queues in the correlations $Q_{avg}$
    $Q_{ema} += Q_{avg} >> QSTABILITY$
    IF ( $Q_{ema} > QTHRESHOLD$)
        Get the transmission rate of transmission element in this correlation $R_{send}$
        FOR each correlated reception element DO
        REPEAT
            Get the reception rate of the reception element $R_{recv}$
            $Ratio = R_{recv}/R_{send}$
            IF ( $Ratio > MaxRatio$ )
                Remove the reception element from task list
            ELSE IF ( $Ratio > MinRatio$ )
                Get the scheduling ticket of the reception element $Ticket$
                $Ticket = Ticket >> 10 * (Ratio - MinRatio)/(MaxRatio - MinRatio)$
                Set the scheduling ticket of the reception element as $Ticket$
            END IF
        END FOR
    END IF
END FOR
```

Figure 4.11: Scheduling weight adjust algorithm

Bibliography

[1] Andrea Bianco, Robert Birke, Davide Bolognesi, Jorge M. Finochietto, Giulio Galantet, Marco Mellia, Prashant M.L.N.P.P, and Fabio Neri. Click vs. Linux: two efficient open-source IP network stacks for software routers. In *Workshop on High Performance Switching and Routing*, May 2005.

[2] Andrea Bianco, Jorge M. Finochietto, Giulio Gelante, Marco Mellia, and Fabio Neri. Open-source PC-based software routers: A viable approach to high-performance packet switching. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[3] Raffaele Bolla and Roberto Bruschi. RFC 2544 performance evaluation for a Linux based open router. In *Workshop on High Performance Switching and Routing*, June 2006.

[4] Benjie Chen and Robert Morris. Flexible control of parallelism in a multiprocessor PC router. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 333–346, Berkeley, CA, USA, 2001. USENIX Association.

[5] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernhard Plattner. Router plugins: a software architecture for next generation routers. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 229–240, New York, NY, USA, 1998. ACM.

[6] Calarco Giorgio, Raffaelli Carla, Schembra Giovanni, and Tusa Giovanni. Comparative analysis of SMP click scheduling techniques. In *Third International Workshop on Quality of Service in Multiservice IP Networks*, February 2004.

[7] Mark Handley, Orion Hodson, and Eddie Kohler. XORP: an open platform for network research. *SIGCOMM Computer Communnication Review*, 33(1):53–57, 2003.

[8] Kunihiro Ishiguro. Zebra routing software. Available online. http://www.zebra.org.

[9] Scott Karlin and Larry Peterson. VERA: an extensible router architecture. *Computer Networks.*, 38(3):277–293, 2002.

[10] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[11] Sun Microsystems. SunFire V20Z server architecture. White paper, April 2004.

[12] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 9–9, Berkeley, CA, USA, 1996. USENIX Association.

[13] Prashant Pradhan and Tzi-cker Chiueh. Implementation and evaluation of a QoS-capable cluster-based IP router. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society Press, 2002.

[14] Qinghua Ye and Mike H. MacGregor. Cluster-based IP router: Implementation and evaluation. In *IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006.

[15] Qinghua Ye and Mike H. MacGregor. Hardware bottleneck evaluation and analysis of a software PC-based router. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systerms*, pages 480–487, Edinburgh, UK, June 2008.

Chapter 5

Internal Congestion Control in a Cluster-based Router

5.1 Introduction

Software routers based on commodity hardware and open source operating systems are gaining more and more interest from both scientific researchers and business users. They have advantages in terms of price-performance and customizability compared to closed proprietary systems. For small and medium enterprises the price-performance advantage may be attractive. In a research setting, an open source router provides an opportunity to explore new algorithms and modify or extend router functions. However, in both contexts scalability is necessarily a concern.

In order to address scalability we have proposed and evaluated an extensible and scalable cluster-based router framework [8]. Using this framework we have developed a prototype router based on a cluster of commodity processors interconnected by an InfiniBand network. Our cluster-based router performs very well and can be scaled as needed to deliver higher packet forwarding rates.

As in any distributed system, congestion control is a critical design issue. Congestion can affect both the forwarding rate and the latency of packets travelling through the router. If packets are sent too quickly from multiple ingress nodes to a single egress node, the egress node will become overloaded and its forwarding capability will be reduced. At this point, packets will be delayed and perhaps even dropped due to the queue overflow at the egress. The congestion control scheme proposed here prevents the ingress nodes from overwhelming the egress nodes. It also reduces the waste of internal network bandwidth and CPU cycles by dropping packets as early as possible, so as to maximize the forwarding rate under overload.

We propose the use of Backward Explicit Congestion Notification within the router, from egress to ingress, combined with a novel queue scheduling method. We have implemented and evaluated this scheme in our experimental prototype. Our measurements show an increase in forwarding rate of nearly 33% with a concomitant reduction in traffic in the internal network of up to 59%.

[0] This chapter is based on the paper ”Internal Congestion Control in a Cluster-based Router” published on IEEE International Conference on High Performance Switching and Routing, Shanghai, China, May 14th-17th, 2008

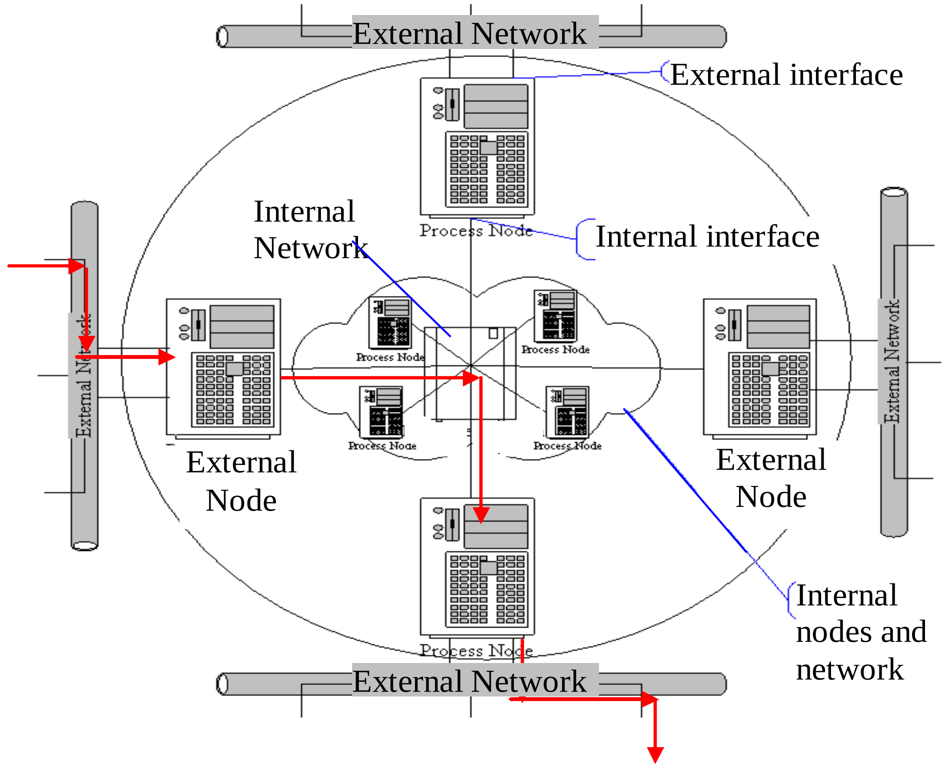


Figure 5.1: Cluster-based router architecture

The rest of the paper is organized as follows. In section 5.2, we briefly introduce the architecture of our cluster-based router. In section 5.3, we describe how and where congestion occurs in our router. The internal congestion control scheme is explained in section 5.4 and the evaluation of this scheme is reported in section 5.5. We report conclusions in section 5.6.

5.2 Cluster-based router architecture

A cluster-based router [8] is composed of two main parts: the processing nodes and the interconnection network. The processing nodes are commodity server blades each equipped with one or more interface cards terminating the links to the outside world. The server blades also carry interfaces for terminating the links to the internal interconnection network. Our prototype uses 1 Gbps Ethernet links to the outside world. The internal interconnection network is a high-throughput/low-latency InfiniBand fabric.

Figure 5.1 shows the architecture of our prototype. In this system we can divide the processing nodes into two categories: external processing nodes and internal processing nodes. The external nodes connect to the attached outside networks, and they are responsible for receiving and forwarding packets from/to other attached outside networks. The external nodes are dedicated to packet forwarding in the critical data path. The internal processing nodes are dedicated to control and management tasks such as routing table updates, network management and other non-time-critical

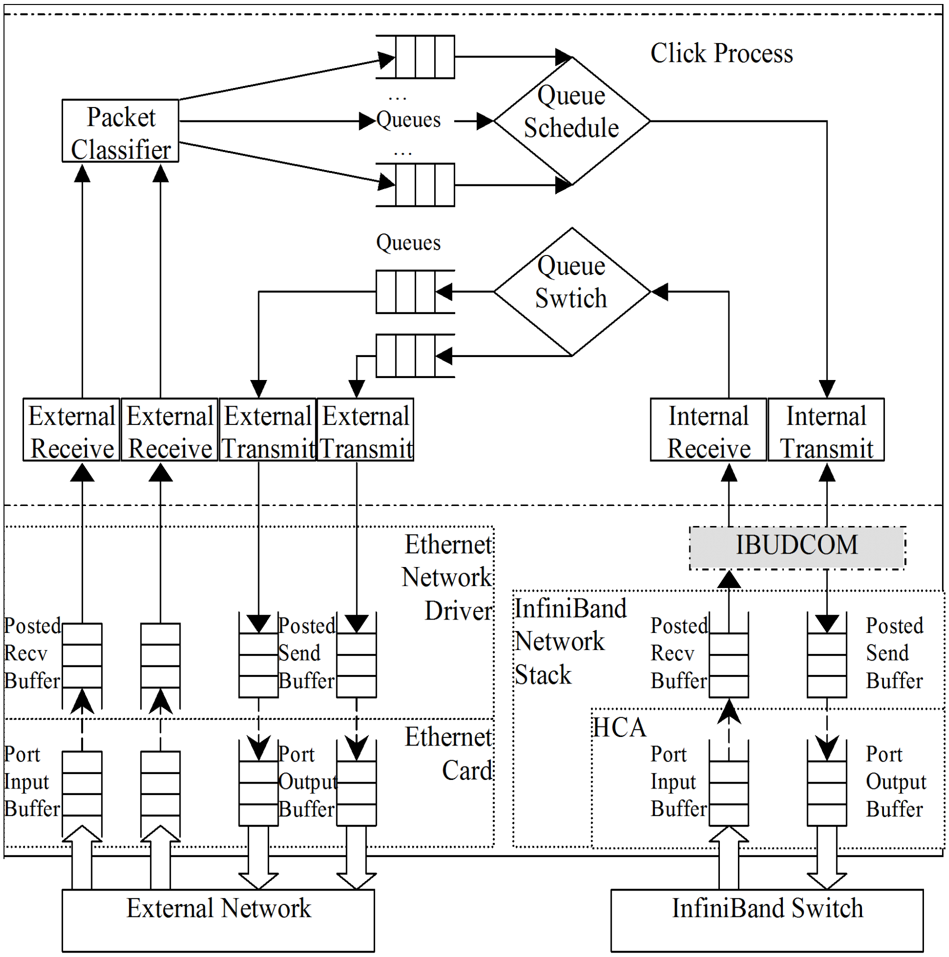


Figure 5.2: Packet forwarding path

processing such as authentication and accounting.

All nodes - both internal and external - are connected to the internal interconnection network. Packets must cross the internal interconnection network for forwarding unless they are destined to a network connected to the receiving external node, most likely on a different physical interface, although the nodes may also forward traffic between multiple virtual interfaces on the same physical interface. A packet crosses the interconnection network only once if there is no additional processing needed in the internal processing nodes.

Figure 5.2 details the forwarding path in each external node. In the following we will discuss ”ingress” and ”egress” nodes but it should be noted that these terms are relative to a given flow, and the connotation needs to be kept separate from that of ”external” and ”internal” nodes. There are no dedicated ingress or egress nodes as such - a particular external node may function simultaneously as an ingress node for some flows, and an egress node for others. The IP packet is received by the Ethernet driver at the ingress node where it will be captured by the external receiving element in a Click kernel thread [1]. The packet then follows the processing path in the ingress node according to the defined Click configuration. At the end of this processing path, the internal interface passes the packet to the IBUDCOM layer and the packet traverses the internal network to the egress node. The packet is captured from the IBUDCOM layer by the internal interface at the egress node, switched to a specific outbound queue, and finally sent to the external network by the external interface through the Ethernet driver.

In this process the overhead of receiving and transmitting a packet account for a large part of a packet's processing time. It is also the case that small packets of less than 64 bytes account for a large proportion of Internet traffic [7]. Therefore, we pack multiple Ethernet packets into one InfiniBand packet before traversing the internal network. (InfiniBand packets have a maximum size of 4KB.) This amortizes the internal overheads such as PCI arbitration and DMA operations over several packets, thus reducing the average processing cost of each packet and increasing maximum throughput.

5.3 Congestion mechanisms

In any network, when the total demand for a resource is greater than the available capacity in any time interval, the resource is said to be congested for that time interval. The resources required in a router include internal link bandwidth, switch buffer capacity, processor cycles, etc. [5].

Congestion control is a critical issue in the design of our cluster-based router. Congestion can affect both the forwarding rate of the router and the latency experienced by packets travelling through the router. The internal communication between processing nodes in our cluster-based router is based on the InfiniBand Unreliable Datagram service. This determines the potential congestion mechanisms on the path between the internal interface on the ingress node and the internal interface on the egress node.

Firstly, there is no packet dropping at the InfiniBand link layer. The internal physical interfaces are called host connection adapters (HCAs). They connect each node to the internal InfiniBand switch. InfiniBand uses a link layer flow control scheme [3] to prevent an HCA port from transmitting packets when the downstream port lacks sufficient buffer space to receive them. Specifically, InfiniBand manages the HCA on-card virtual link buffer rather than the host buffer. As a result no packets will be lost on the InfiniBand link layer and there can be no congestion between the HCA on the ingress node and the HCA on the egress node. It is important to recall that these HCAs face the internal network - they are not the external Ethernet interfaces.

Although the InfiniBand architecture guarantees no congestion between HCAs and thus across the internal switch, congestion can still arise inside the ingress node upstream of its HCA and inside the egress node downstream of its HCA.

If the ingress node transmits packets at a rate that the InfiniBand link layer can service but is faster than the egress node can process, the egress node will eventually run out of posted buffers. At that point, packets will be discarded by the HCA on the egress node because it cannot place them in a posted buffer for processing. Thus in order to avoid buffer overrun on the egress node, a congestion control scheme is required at or above the InfiniBand link layer.

Congestion can also arise in the queues on both the ingress and egress nodes. At an ingress node, incoming flows from multiple Ethernet interfaces must be queued for the single InfiniBand interface to the internal switch. As a result, the input queues to the internal transmitting element on the ingress node may overflow. At an egress node, due to the difference between the transmission capabilities of the external Ethernet interface and the reception capabilities of the InfiniBand HCA on that node, packets from multiple ingress nodes may be queued in the egress node or on the external Ethernet interface.

A variety of mechanisms might be considered to deal with congestion in these various locations. We start by considering the following: if the ingress nodes can be signalled to reduce their transmission rates to specific overloaded egress nodes then the drop rate of packets on those egress nodes could be reduced. This improves overall router throughput because it avoids wasting internal network bandwidth on dropped packets and reduces the proportion of cycles in both ingress and egress nodes which are spent processing packets that will be dropped due to congestion.

5.4 Internal congestion control

Based on the mechanisms by which congestion arises in our prototype, we propose an internal congestion control scheme based on Backward Explicit Congestion Notification (BECN) combined with a novel queue scheduling weight control. In this section, we first set out the guidelines used in the design of our congestion control and then survey several main types of congestion control strategies. Following that, our specific congestion control scheme is described.

5.4.1 Guidelines

Many different schemes have been proposed for congestion control because the method developed for one network may not work within a different architecture [4]. In our cluster-based router, the internal congestion control scheme must meet the following requirements.

The scheme must have low overhead. The instrumentation for the scheme must not consume much CPU to measure congestion, to generate control messages, or to implement the control. Significant CPU overheads will only worsen the effects of congestion. For the same reason, the congestion control must not consume much internal network bandwidth. If this requirement is not met congestion collapse will occur at relatively low throughput.

The scheme must be fair. Fairness means that the congested resources should be allocated to each source/flow in a fair manner. The scheme should not starve any source or flow, and it should reduce the overloaded traffic according to its fair share instead of its absolute bandwidth.

The scheme must be responsive. The congestion control scheme must match demand dynamically to the available capacity. Once the router becomes congested, the control must reduce congestion quickly. Also, the reduced demand should be able to recover its share quickly.

The scheme must be efficient in its use of resources. It should maximize overall router throughput and minimize latency on the forwarding path.

5.4.2 Types of congestion control

Numerous strategies have been developed for congestion control. Existing schemes can be classified as credit-based, window-based, and rate-based according to the method used to control the rate at which sources are allowed to inject traffic. All three methods can be defined in terms of a (virtual) link with a source at one end and a sink at the other.

Credit-based: the sink specifies the number of packets it is currently willing to accept from the source. This number is called the source's credit. The source decreases its credit for each packet sent and increases it upon notification from the sink. The main drawback of this scheme is its scalability in the face of increasing numbers of sources and sinks. Each source must keep track of the credit information for each sink and each sink must signal to each source separately. In our case, each external node may have to implement many sources and sinks simultaneously.

Window-based: the sink specifies the number of packets that a source can currently send. This number is called the window size. The source reduces the window size for each packet sent and also in response to a congestion feedback signal from the sink. The source may periodically test the link by increasing the window size.

Rate-based: the sink specifies a maximum rate to the source, in terms of packets per second or bits per second. The source adjusts its rate in response to network conditions.

Alternatively, according to the method of notification used between sinks and sources, congestion control schemes can be categorized as Forward Explicit Congestion Notification (FECN), Backward Explicit Congestion Notification (BECN) [5], and timeout-based [4].

FECN: when congestion is detected along the path from a source to a sink, a Congestion Indication bit is set in all packets passing the congested point. When a sink receives packets with the Congestion Indication bit set, it generates congestion notification packets in the reverse direction back to the source alerting the fact that congestion has been detected somewhere along the path. Sources which are notified of congestion reduce their rates accordingly. The main disadvantage of FECN is its slow effect due to the round-trip delay between source and sink.

BECN: congestion notification packets are sent directly from the point at which congestion is detected back to the source. Sources respond to BECN packets by reducing their rates. BECN leads to quicker reaction by reducing the delay between detection of congestion and notification of the source.

Timeout-based: timeout-based congestion control uses the idea that packet loss is a good indicator of congestion. Load is reduced as the result of a timeout and then increased slowly as long as there is no timeout. However, the use of timeouts is inefficient in high bandwidth networks [6].

5.4.3 Observations

As described earlier there are two potential levels at which congestion may occur in our system: the Click packet processing layer and the IBUDCOM transport layer. For the former, queues on both ingress and egress nodes must be monitored. Congestion on the egress queues can easily be measured by queue occupancy, while the detection of congestion in the IBUDCOM transport layer is much more complex. Fortunately, we noticed and also validated experimentally that congestion in the transport layer is always preceded by congestion in the egress queues if the quotas of the internal receiving element and the external transmitting elements on the egress node are configured appropriately. ”Quota” means the maximum number of packets an element can process in each round. In this scheme, the egress queues will already be overloaded before any packets are dropped by the transport layer due to overrun of the InfiniBand receive buffer at the egress. The reason is that the InfiniBand receive buffer on the egress node is overrun only when the internal receiving element at the egress node processes the received packets more slowly than the rate at which they arrive. Given this observation, we need to monitor only the status of the egress queues to detect potential congestion.

5.4.4 BECN notification and queue scheduling

Our control scheme is composed of congestion detection, congestion notification, queue schedule weight reduction, and queue schedule weight restoration as shown in Figure 5.3.

Congestion detection is performed by the Queue switch element on the egress node. When a packet is switched to a queue, the queue switch element checks the queue status. If the queue status meets the congestion criterion then the queue is said to be in a congested state. The congestion criterion can be queue length, incoming rate, etc. The criterion can be judged in terms of absolute value such as the number of packets in the queue or windowed over a time interval as when calculating incoming rate. We have tested a variety of measurements and have found monitoring queue length over a specified time interval to be most reliable.

Once congestion has been detected by the Queue switch element, it generates BECN packets and sends them back to the appropriate source(s).

When it receives a BECN packet, the queue scheduling element in the ingress node reduces the scheduling weight for the corresponding queue. As shown in Figure 5.3, the queue scheduling element on the ingress node schedules each queue according to its scheduling quantum - the accumulated scheduling weights. During each scheduling round, the queue’s quantum is increased by its scheduling weight, and the queue is scheduled only when its scheduling quantum reaches the scheduling threshold. After the queue is scheduled, the scheduling quantum is decreased by the amount of scheduling threshold. In this way, the scheduling weight of a queue determines how often the queue is scheduled relative to other queues on the same ingress node. As a result, the reduction of scheduling weight reduces the rate at which packets are removed from the queue for transmission,

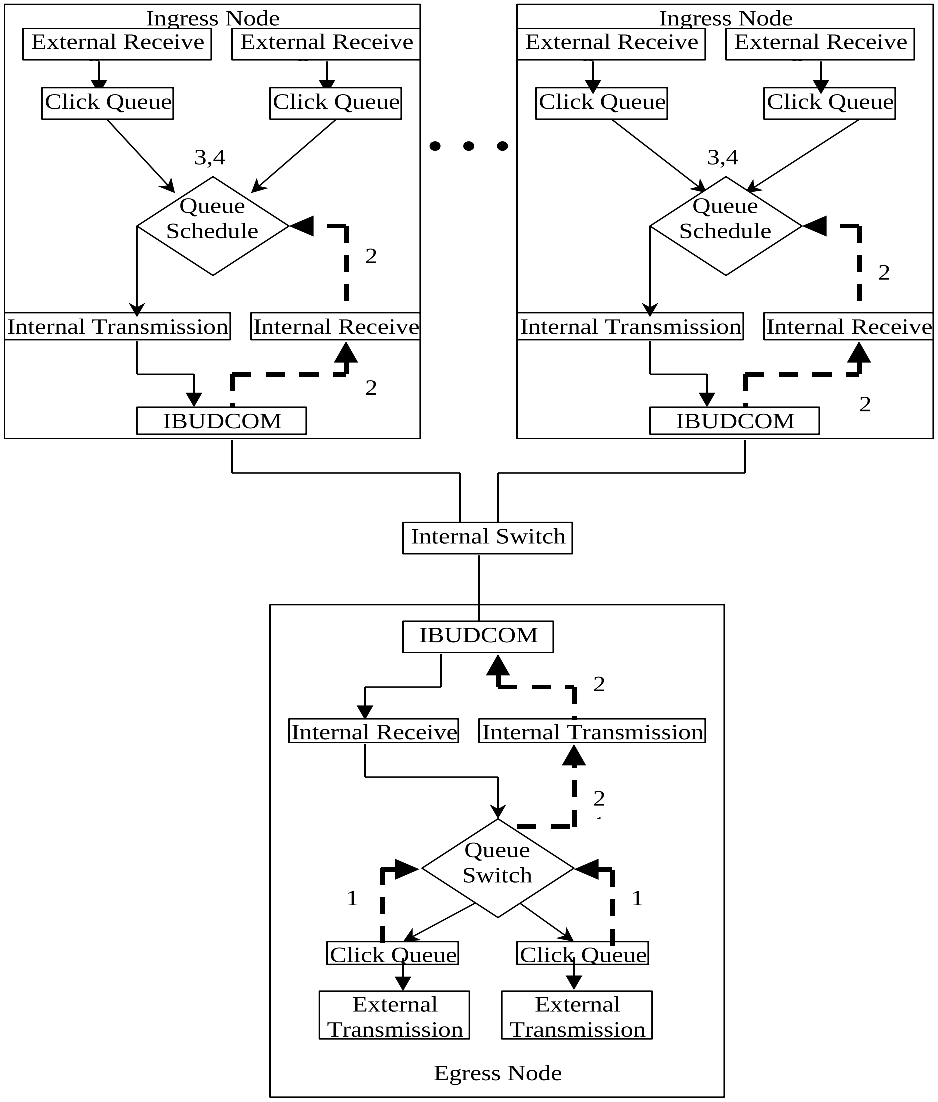


Figure 5.3: Cluster-based router architecture

but in an indirect manner which requires much less overhead than directly controlling rate.

The queue scheduling weight is restored based on a timer. This timer runs over the same interval used by the congestion detection in the egress node. If no BECN packet was received in the previous interval, then the scheduling weight is incremented.

5.4.5 Implementation

We implemented the BECN notification and queue scheduling weight control with two new Click elements: IBBECNPack and IBBECNUnpack. The IBBECNPack element is inherited from our original IBPack element with the addition of the BECN packet processing and queue schedule weight reduction and increase. The IBBECNUnpack element is inherited from the original IBUnpack element and in addition, it implements monitoring queues, generation BECN packets, and sorting incoming BECN packets from the other incoming Infiniband packets. Figure 5.4 shows the details of how the BECN processing is incorporated into the original Click configuration [8]. The generated and incoming BECN packets are pushed into the BECN queue. IBBECNPack removes them for processing.

5.4.6 Parameter settings

To make the congestion control scheme responsive and efficient, it is important to set the congestion detection time interval and the queue threshold wisely. The time interval should be set according to the round trip time between the ingress node and the egress node. It should allow the effect of previous control actions to be reflected in the next detection interval. If the time interval is set too small, then the effects of a control action in one or more previous intervals may not have time to be seen. If the time interval is set too large, then the control is not responsive enough. The queue threshold should be set to minimize the dropping of incoming packets before control actions have time to take effect. Thus, the values chosen for these two parameters are somewhat interdependent.

Another, less obvious set of parameters arises when choosing how and when sources should be sent congestion notifications. If this is not done correctly it is possible for multiple sources to become synchronized, and then the aggregate internal forwarding rate will begin to oscillate. This is called global synchronization [2]. In this situation multiple ingress nodes simultaneously reduce their transmission rates and later restore their transmission rates simultaneously at the same pace. We can produce global synchronization experimentally in our system by having an egress node generate BECN packets to all its ingress sources as soon as one queue becomes congested. We have found experimentally that global synchronization and the subsequent oscillation of the forwarding rate can be prevented by generating BECN packets to just a subset of the ingress sources of a congested egress node. To guarantee fairness, in each round we notify just a subset of the ingress source nodes which are sending traffic to a congested egress. The subset is chosen randomly from all the sources for the congested queue according to their share of the total load offered to the queue.

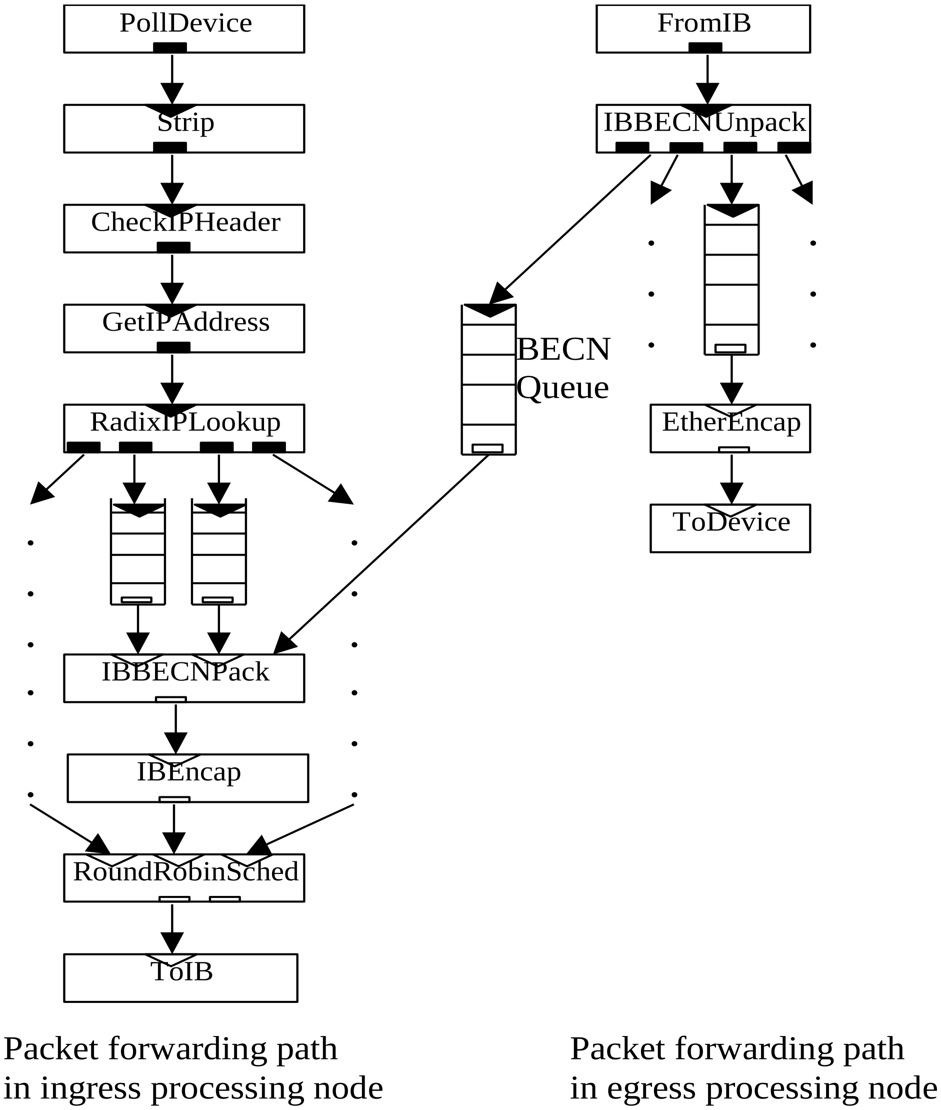


Figure 5.4: Node configuration

Queue scheduling weights are modified dynamically using ”additive increase, multiplicative decrease” (AIMD). The amount used to increase the queue scheduling weight during the ”additive increase” phase affects both the accuracy and the responsiveness of the control. If this value is too small, then the source restores the transmission rate too slowly and throughput suffers. If this value is too large, then the congestion control scheme simply alternates between the increase and decrease phases and is ineffective. Multiplicative decrease punishes overloaded sources more than sources which are not overloaded.

5.5 Evaluation

In this section, we report the forwarding rates, drop rates, queue lengths and fairness under overload of the proposed control scheme for our cluster-based router.

5.5.1 Experimental setup

Our router is composed of a cluster of four SunFire X4100 nodes interconnected by a Mellanox InfiniBand switch. Each SunFire node has two AMD Opteron 254 processors operating at 2.8GHz, 4 GB of registered DDR-400 SDRAM, two Intel E1000 Ethernet ports connected to a 100MHz PCI-X bus, and one Voltaire InfiniBand 4x PCI-X HCA installed in a 133MHz 64-bit PCI-X slot. A Mellanox MTS2400 24-port Modular InfiniBand Switch System that can support 10 Gbps switching rate at each port is connected to the HCA on each node. Packets are generated by a commercial traffic analyzer and sent to the router external Ethernet ports for forwarding. Packets that are successfully forwarded through the router return to the traffic analyzer via the router Ethernet ports.

In order to evaluate the effectiveness of our proposed congestion control scheme we compare the behavior of the cluster-based router first with, and then without, internal congestion control. The traffic analyzer is configured to offer Ethernet traffic to three of the nodes with the destination being a subnet reached via the fourth node. We increase the offered load until the router was heavily overloaded - the offered load is over 100% more than the maximum load the egress node can forward. As well as acting as a traffic source and sink, the network analyzer is used to monitor the rate at which Ethernet packets are sent to each ingress node and the rate at which the egress node return the packets to the traffic analyzer. Small packets stress the router more than large packets at the same aggregate bit rate because more packets have to be handled per second. As a result we use 64B packets in our tests.

5.5.2 Measurements

Forwarding rate is one of the most important performance metrics of a router. In our cluster-based router, if too much traffic is destined to an egress node, the egress node will become overloaded. Beyond this point of ”congestion collapse” the forwarding capability of the egress node will actually

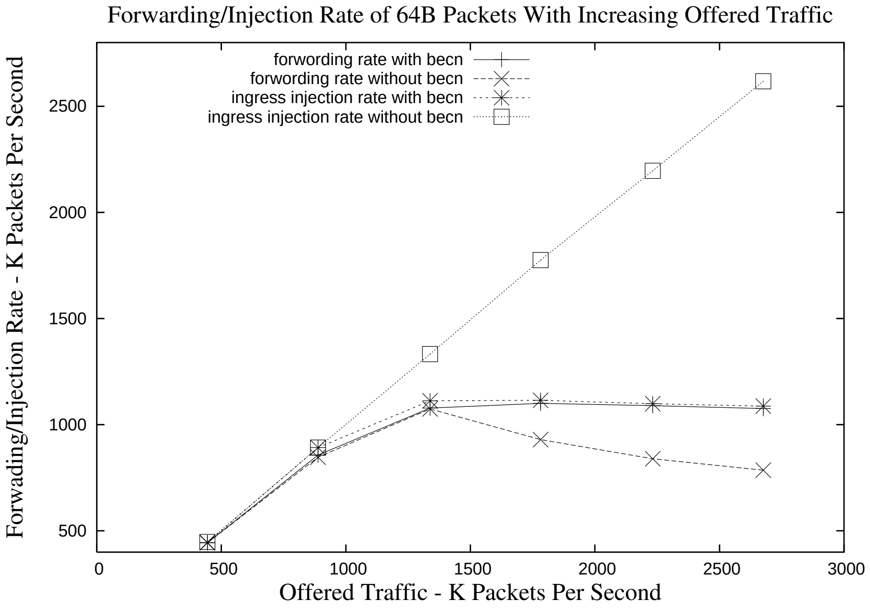


Figure 5.5: Forwarding/injection rates

decrease as traffic increases further. With the addition of our congestion control, the egress node should not be pushed into congestion collapse.

The dropping rate and injection rate at the ingress and egress are two more metrics which can be used to determine the effectiveness of the internal congestion control scheme. One goal of our scheme is, when it is necessary to drop packets, they should be dropped as early as possible. This will help avoid occupying the internal network with packets that are later dropped at an egress. Therefore, our goal is to increase the dropping rate at ingress nodes and decrease it as much or more at the egress. The injection rate is the rate at which packets are injected into the internal network. Dropping packets at the ingress should decrease the injection rate; our goal is that overall router throughput will increase at the same time because the internal network and egress node are doing more useful work.

Fairness is an important goal of a congestion control scheme. Fairness means that the congestion control scheme should allocate available resources to sources/flows in a fair manner. We measure fairness by recording the injection rates at each ingress node.

Queue length is another metric that shows how serious the congestion is. With the application of our internal congestion control scheme, we expect that the queue length on the egress node will be reduced under overload while the queue length on the ingress node will be increased.

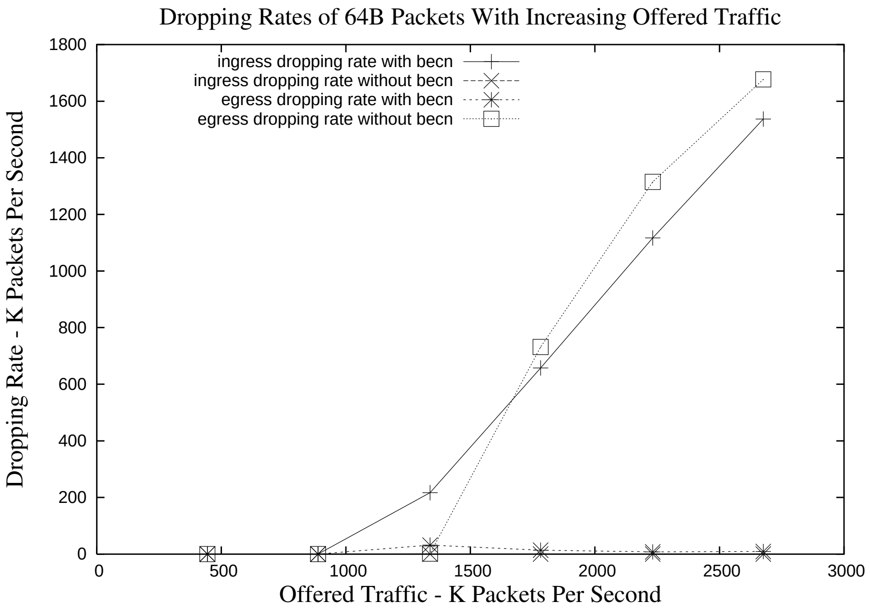


Figure 5.6: Dropping rates

5.5.3 Results

Figure 5.5 gives the packet forwarding and injection rates of the cluster-based router with and without the congestion control scheme. Figure 5.6 gives the packet dropping rate on the queues of the Click processing path at both the ingress and egress nodes, without including packet drops in the IBUDCOM transport layer. In these tests the offered traffic to all three ingress nodes is the same. The offered traffic presented on the x-axis is the sum of offered traffic to all three ingress nodes. From these figures we conclude that our congestion control scheme prevents congestion collapse and reduces the injection rates of packets into the internal network when the router is overloaded. In the most heavily overloaded case the forwarding rate increases by 33% while the injection rate is reduced by 59%.

We note that when the offered traffic reaches 1300 kpps a higher dropping rate is reported at the egress node when using our internal congestion control. This is caused by the fact that more packets are dropped on the InfiniBand IBUDCOM layer in the case without our congestion control and thus they never enter the Click processing path at the egress node.

Figure 5.7 and Figure 5.8 give the packet injection rates of each ingress node as a function of offered loads. The values on the x-axis are the load offered to each ingress node. Without our congestion control, almost all the incoming packets at each ingress node are injected into the internal network. With the congestion control there is a significant decrease in the injection rate, as desired.

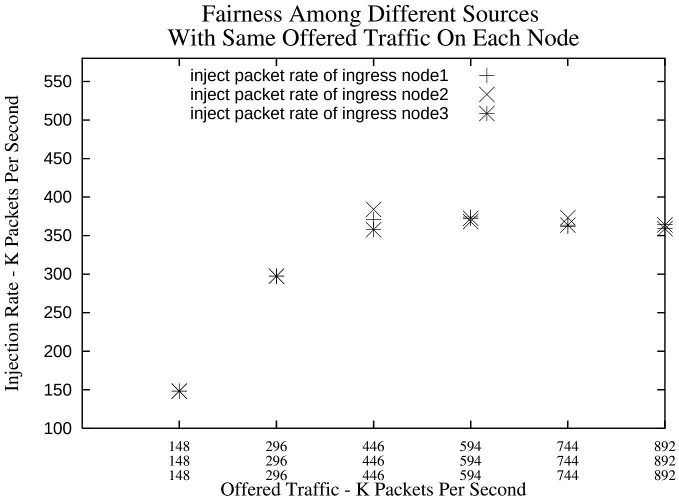


Figure 5.7: Fairness with same traffic

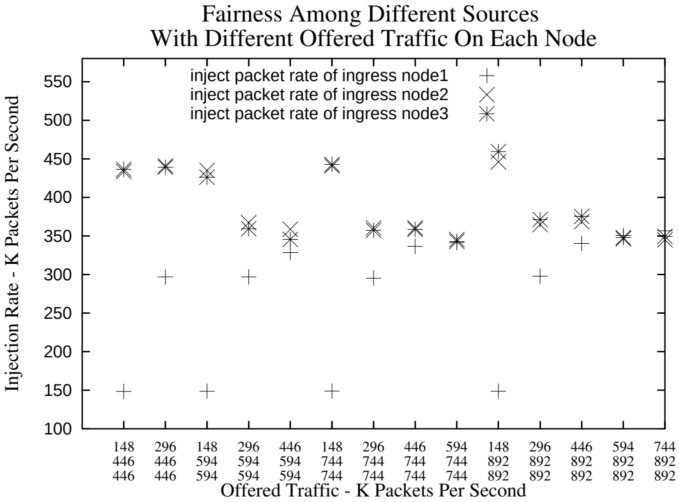


Figure 5.8: Fairness with different traffic

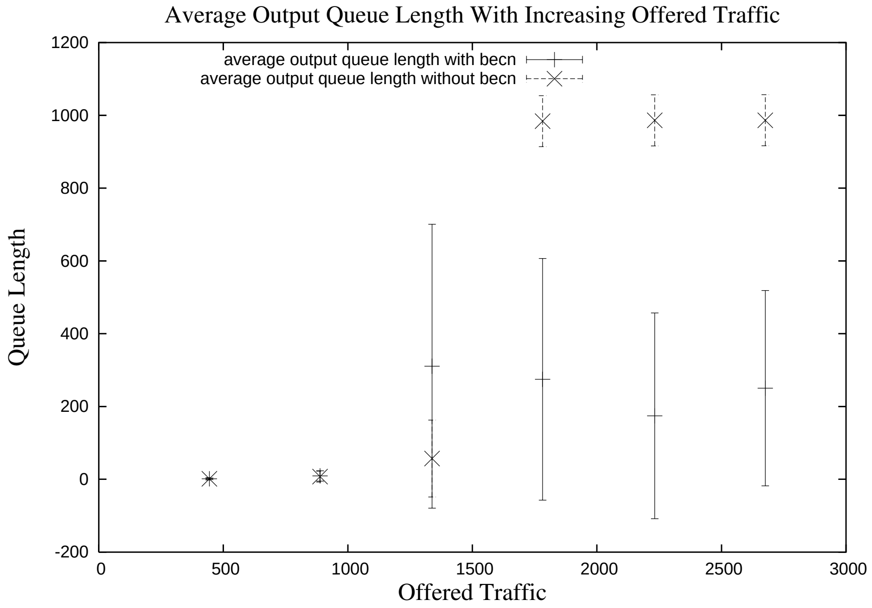


Figure 5.9: Output queue length

Our proposed scheme is also fair: if all sources are offered the same load, then they get an equal share of the internal network. If one source is offered less load than the other two, the more heavily loaded sources are punished more severely than the more lightly loaded source, and all sources still get an equal share until the more lightly loaded source is 100% serviced.

Figure 5.9 presents the average queue length and variation of queue length on the egress node, while Figure 5.10 gives the average queue length and variation of queue length on the ingress nodes. It shows that our proposed internal congestion control scheme is effective in reducing the queue size on the egress node. We should mention that, the large queue length in the ingress node with our scheme can be reduced easily by queue management schemes but at the cost of overhead on the ingress nodes.

5.6 Conclusions

Congestion control is a universal problem in distributed systems. The specific characteristics of the forwarding path through our cluster-based router make the detailed features of the congestion problem unique. In particular, the link-layer flow control implemented as part of the InfiniBand standard pushes congestion outward from the internal network to the dependent ingress and egress nodes.

In this paper, we propose a congestion control scheme which uses BECN notification internal to

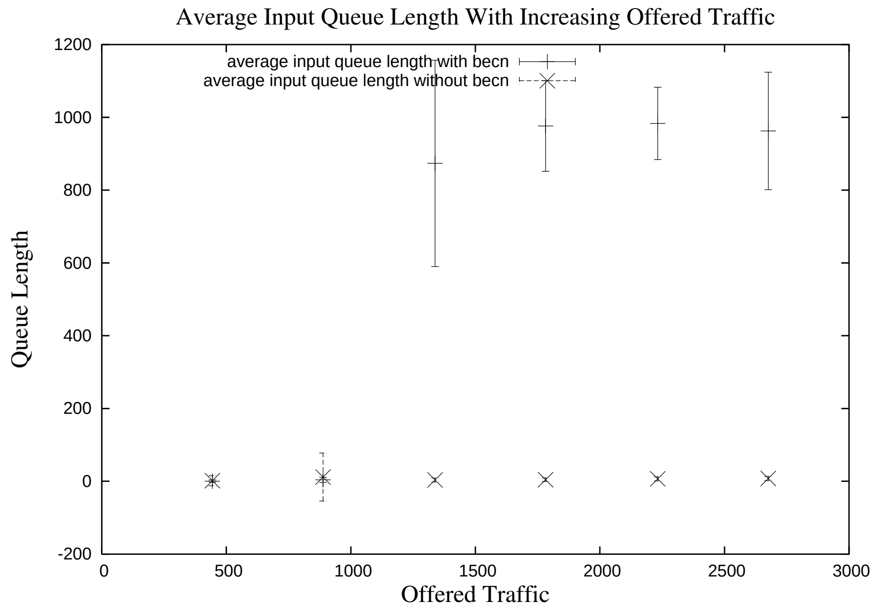


Figure 5.10: Input queue length

the router along with queue scheduling. The queue scheduling mechanism requires less overhead than other forms of congestion control. We arrived at this conclusion through the implementation and testing but did not present the detailed results here due to lack of space. BECN is used to push congestion back upstream to the ingress nodes. Global synchronization is prevented by sending BECN packets to only a subset of the ingress nodes. Fairness is ensured by selecting source nodes randomly according to the sources' share of the incoming traffic.

The results presented here show that our proposed congestion control scheme is effective, efficient and fair. It is effective in preventing offered traffic from causing congestion collapse. It achieves efficiency by reducing the waste of internal network bandwidth and CPU cycles by dropping overloaded packets early, and allocates throughput fairly. With this congestion control scheme in place, the forwarding rate of our router increased by up to 33% under overload while the traffic in the internal network was reduced by up to 59%.

5.7 Appendix - Experimental setup

The processing nodes' architecture is described in figure 5.11.

The connections between cluster-based router and traffic analyzer are described in figure 5.12. Packets are generated to three of the processing nodes (two ports each). The destinations of the packets are configured to be the two ports on the traffic analyzer which are connected to the ports on

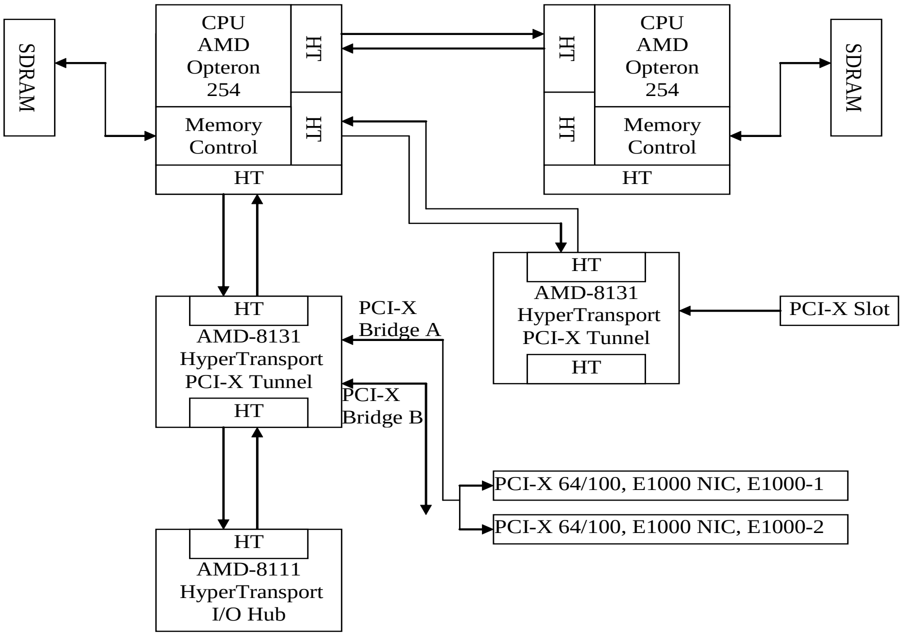


Figure 5.11: Processing nodes' architecture block diagram

the fourth processing node (egress node).

5.8 Appendix - Algorithms

The figures 5.13 and 5.14 give the algorithms implemented in IBBECNPack and IBBECNUnpack elements as described in section 5.4.4 and section 5.4.5:

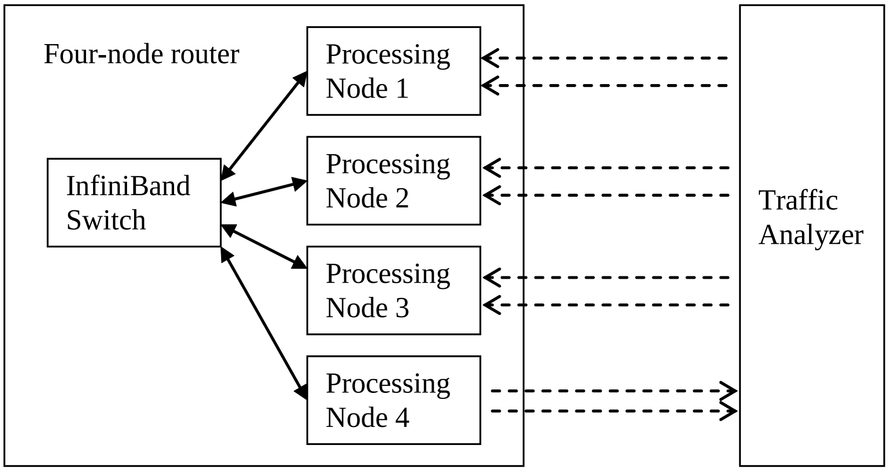


Figure 5.12: Connections between router and traffic analyzer

IBBECNPack pull (Input: None, Output: A InfiniBand Packet)
$T_{current}$: Current time
T_{last}: Last control time
$T_{interval}$: Control interval
$Weight[i]$: The ith upstream queue's scheduling weight
$Incrweight$: The increasing weight when there is no congestion notification received
$MAXWEIGHT$: Maximum queue's scheduling weight
$BECNpacket$: A BECN packet to be sent or received
$Dest$: Destination address field in a BECN packet
$Queueid$: Queue id field in a BECN packet
$Local$: Local address

```
IF ($T_{current} - T_{last} > T_{interval}$)
    FOR each input queue of this IBBECNpack element DO
    REPEAT
        $Weight[i] += Incrweight$
        IF ($Weight[i] > MAXWEIGHT$)
            $Weight[i] = MAXWEIGHT$
        END IF
    END FOR
END IF
WHILE The BECN queue is not empty DO
REPEAT
    Get a BECN packet $BECNpacket$
    Get the destination address $Dest$ and queue id $Queueid$
    IF $Dest == Local$
        Return the BECN packet $BECNpacket$
    ELSE
        $Weight[Queueid] = Weight[Queueid]/2$
    END IF
END WHILE
Get enough Ethernet packets from upstream queues according to the corresponding queues' scheduling weights
Pack the Ethernet packets into one InfiniBand packet and return the packet
```

Figure 5.13: IBBECNPack pull

IBBECNUnpack push (Input: A InfiniBand Packet, Output: None)

$IBpacket$: The InfiniBand packet which is being processed

$T_{current}$: Current time

T_{last}: Last control time

$T_{interval}$: Control interval

q: A queue length

$Q_{threshold}$: The congestion queue length threshold

$Lastsrc$: The source address of last Ethernet packet

$Packet$: A Ethernet packet

i: A Ethernet packet's index

```
Get a InfiniBand packet $IBpacket$
IF ($T_{current} - T_{last} > T_{interval}$)
    FOR each output queue of this IBBECNUnpack element DO
    REPEAT
        Get this queue's queue length $q$
        IF ($q > Q_{threshold}$)
            Create a BECN packet destined the source address of last packet $Lastsrc$ and push it to the BECN queue
        END IF
    END FOR
END IF
IF ($IBpacket$ is a BECN packet)
    Push $IBpacket$ to a downstream BECN queue according to the source address
ELSE
    Unpack the packet $IBpacket$ to multiple Ethernet packets
    FOR each Ethernet packet $Packet$ and it's index $i$ DO
    REPEAT
        Set $Lastsrc$ to $Packet$'s source address
        Push $Packet$ to the downstream $i$th queue
    END FOR
END IF
```

Figure 5.14: IBBECNUnpack push

Bibliography

[1] Benjie Chen and Robert Morris. Flexible control of parallelism in a multiprocessor PC router. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 333–346, Berkeley, CA, USA, 2001. USENIX Association.

[2] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[3] IBTA. *InfiniBand Architecture Specification*. InfiniBand Trade Association.

[4] Raj Jain. Congestion control in computer networks: Issues and trends. *IEEE Network Magazine*, 4:24–30, 1990.

[5] Anup Kumar, Arpana Maniar, and Adel S. Elmaghraby. A fair backward explicit congestion control scheme for ATM network. In *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, pages 452–457, Washington, DC, USA, 1999. IEEE Computer Society.

[6] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 2002.

[7] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.

[8] Qinghua Ye and Mike H. MacGregor. Cluster-based IP router: Implementation and evaluation. In *IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006.

Chapter 6

Stability of Optimal Internal Congestion Control in a Cluster-based Router

6.1 Introduction

Software routers based on commodity hardware and open source operating systems are gaining more and more interest from both scientific researchers and business users. They have advantages in terms of price-performance and customizability compared to closed proprietary systems. For small and medium enterprises the price-performance advantage may be attractive. In a research setting, an open source router provides an opportunity to explore new algorithms and modify or extend router functions. However, in both contexts scalability and congestion control are of concern.

In order to address scalability we have proposed and evaluated an extensible and scalable cluster-based router framework[16]. Using this framework we have developed a prototype router based on a cluster of commodity processors interconnected by an InfiniBand network. Our cluster-based router performs very well and can be scaled as needed to deliver higher packet forwarding rates.

As in any distributed system, congestion control is a critical design issue. Congestion can affect both the forwarding rate and the latency of packets travelling through the router. If packets are sent too quickly from multiple ingress nodes to a single egress node, the egress node will become overloaded and its forwarding capability will be reduced. At this point, packets will be delayed and perhaps even dropped due to the queue overflow at the egress. A good congestion control scheme can prevent the ingress nodes from overwhelming the egress nodes, which can also reduce the waste of internal network bandwidth and CPU cycles by dropping packets as early as possible, so as to maximize the forwarding rate under overload[17] [18].

In the design of congestion controllers, stability is one of the important criteria: the system should be able to avoid oscillations in the steady state. If the system is not stable, the average

[0] This chapter is based on the paper ”Stability of Optimal Internal Congestion Control in a Cluster-based Router” submitted to Elsevier ”Performance Evaluation”

forwarding rate of the cluster-based router is reduced or the packet forwarding delay is affected due to the variance of the queue length in the router. Parameter settings also play an important role in the stability of a congestion control scheme[5] [6].

A significant amount of work has been done to analyze the stability of different congestion control schemes[3] [11] [14]. Lyapunov's stability theorem gives a sufficient condition to check for the stability of nonlinear systems[15]. It is used widely [4] [11] [14] to analyze the stability of optimal utility-based Internet congestion control schemes through approximating a discrete system by a continuous model. However, we find that the stability conclusions reached through Lyapunov analysis of a continuous model meant to approximate an underlying discrete system are not always correct.

Other than the additive increase, multiplicative decrease Scheme used in [18], there are many congestion control schemes that can be applied in our cluster-based router [9] [13]. We have developed a utility-based congestion control scheme that is provably optimal, and combined it with backward explicit congestion notifications within the router, from egress to ingress. We prove that the continuous model of the proposed optimal scheme is globally, asymptotically stable with any values of the control parameters. We then show that the stability region of the discrete system is different from that of the continuous model. We verify this inconsistency between the continuous model and the underlying discrete system by simulating the scheme with ns-3 and plotting the stability region. We also present examples of the convergence behavior of the simulated congestion control scheme as the values of the control parameters are changed. Finally we present the results of applying the scheme on our cluster-based router prototype. The convergence and stability of the prototype are consistent with our ns-3 simulations.

The rest of the paper is organized as follows. In section 6.2, the optimal utility-based internal congestion control scheme is introduced, and then its stability analysis is presented in section 6.3. We simulate our scheme in terms of stable conditions and convergence in section 6.4. Evaluation in the prototype system are given in section 6.5. Finally, we conclude our work in section 6.6.

6.2 Optimal utility-based control

Optimal utility-based congestion control is an optimization approach to congestion control problems where the objective is to maximize the aggregate source utility. In this approach, the network links and traffic sources are viewed as a distributed system that acts to solve the optimization problem given the constraints of network link capacities. The traffic sources adjust their transmission rates in order to maximize their own benefit, which is the utility minus bandwidth cost, while the network links adjust bandwidth prices to coordinate the sources' decisions on the evolution of their transmission rates[12].

Based on the controlled objects, optimal utility-based congestion control schemes can be divided into three classes - primal algorithms, dual algorithms and primal-dual algorithms[11]. In primal algorithms, users adapt their source rates dynamically based on route prices, and links select a static law to determine the link prices directly from the arrival rates at the links[8]. An example of a primal algorithm is TCP flow control[7]. In dual algorithms, on the other hand, links adapt the link prices dynamically based on the link loads, and users select a static law to determine the source rates directly from route prices and source parameters. REM[1] and AVQ (Adaptive Virtual Queue) [10] are good examples of dual algorithms. In primal-dual algorithms, both link prices and source rates are slowly adjusted so that they asymptotically converge to the optimal solution[11]. In today's Internet, the combination of TCP flow control algorithm and some active queue management algorithms like REM[1], RED[6] and AVQ[10] are examples of primal-dual algorithms.

In this section, we first present our model of the internal network in the cluster-based router, and then develop the discrete congestion control scheme.

6.2.1 Internal congestion control as an optimization problem

Different from previous work, instead of the links in a network, we focus on the state of the egress node in our cluster-based router. The cluster can be modelled as a network with only one egress node as follows.

Consider a network with unidirectional links. There is a finite forwarding capacity C associated with the egress. The egress is shared by a set S of sources, where source $s \in S$ is characterized by a utility function $U_s(x_s)$ that is concave increasing in its transmission rate x_s to the egress.

The goal of congestion control is to calculate the source rates that maximize the sum of the utilities $\sum_{s\in S} U_s(x_s)$ over x_s subject to the capacity constraints. The model can be written as:

$$P : \sum_{s\in S} U_s(x_s) \tag{6.1}$$

subject to

$$\sum_{s\in S} x_s \leq C \tag{6.2}$$

(6.2) means that the aggregate source rate at the egress should not exceed its outgoing transmission capacity. Since the utility function $U(x)$ is strictly concave and continuous, an optimal solution exists. Solving this problem centrally would require the knowledge of all utility functions, and also the knowledge of all the source rates and aggregate rate at the egress. This makes it inflexible for changes in the numbers and/or types of sessions, so a decentralized approach is preferable.

6.2.2 Decentralized approach

From (6.1), the objective function is separable in x_s. However, the source rates are coupled by the constraint (6.2). Fortunately, the dual theory of optimization [2] leads us to a distributed and decentralized solution which results in the coordination of all sources implicitly[12].

As we described in the previous subsection, the utility function $U(x)$ is a differentiable concave function, so we can define the Lagrangian function of the original primal problem as:

$$\begin{aligned} L(x,p) &= \sum_{s\in S} U_s(x_s) - p(\sum_{s\in S} x_s - C) \\ &= \sum_{s\in S} U_s(x_s) - \sum_{s\in S} x_s * p + p * C \end{aligned} \tag{6.3}$$

The objective function of the dual problem is:

$$\begin{aligned} D(p) &= \max_{x_s} L(x,p) \\ &= \sum_{s\in S} \max(U_s(x_s) - x_s * p) + p * C \end{aligned} \tag{6.4}$$

and the dual problem is

$$D : \min_{p\geq 0} D(p) \tag{6.5}$$

In this dual problem, if p is interpreted as the price per unit bandwidth at the egress, then the first term of the objective function $D(p)$ is decomposed into S separable subproblems which can be solved at each source s with some information from the egress. Let

$$B_s(p) = \max(U_s(x_s) - x_s * p) \tag{6.6}$$

Hence, $x_s * p$ represents the bandwidth cost to source s when it transmits at rate x_s, and $B_s(p)$ represents the maximum benefit source s can achieve at the given price. Based on the congestion information p from the egress, each source s can solve the subproblem $B_s(p)$ without coordinating with other sources. Let $x_s(p)$ be the local optimizer to $B_s(p)$ at source s. Then from (6.4), $x_s(p)$ is the local maximizer of the Lagrangian function (6.3).

In general, $x_s(p)$ may not be the primal optimal. However, according to the Karush-Kuhn-Tucker theorem, if there is a minimizer p^* which is the optimizer of the dual problem (6.5), and $x^* = x_s(p^*), s \in S$ is the maximizer of (6.4), then (x^*, p^*) constitutes a saddle point for the function $L(x,p)$, and x^* solves the primal optimization problem P.

If we can solve the dual problem (6.5) locally at the egress to obtain p^*, then the original maximization problem P can be solved in a distributed manner using only the decentralized information available locally at each ingress. In this case, p^* serves as a coordination signal that aligns individual optimality of (6.6) with social optimality of (6.1).

Based on the above theory, the congestion control problem can be generalized to the tasks of finding distributed algorithms that can make sources adapt transmission rates with respect to the egress price and make the egress adapt prices with respect to loads. The optimal solution to the distributed congestion control problem satisfies:

$$\left\{ \begin{array}{l} \frac{\partial D(p)}{\partial x_s} = \frac{\partial U_s(x_s)}{\partial x_s} = U_s'(x_s) - p = 0 \\ \frac{\partial D(p)}{\partial p} = \sum_{s\in S}(-x_s) + C = 0 \end{array} \right.$$

These two equations can be solved in a distributed manner at the sources and the egress. To reduce the overhead of transferring the link price, we only send the price from the egress to the sources at the beginning of each control interval, which results in a discrete-time control model.

Applying the gradient projection method where the source rate and egress price are adjusted according to the gradients (above two equations), we get a primal-dual algorithm. Here we use a proportional fairness utility function $U(x) = W * \log x$ [15], where W is the proportional weight of sources.

$$\{ \begin{array}{rl} x_s(k+1) & = [x_s(k) + K * x_s(k) * (U_s'(x_s(k)) - p(k))]^+_{x_s[k]} \\ & = [x_s(k) + K * (W - x_s(k) * p(k))]^+_{x_s[k]} \\ p(k+1) & = [p(k) + (\sum_{s \in S} x_s(k) - C)/R]^+_{p(k)} \end{array} \tag{6.7}$$

Here

$$[g(x)]^+_y = \{ \begin{array}{ll} g(x), & \text{y > 0} \\ max(g(x), 0), & \text{y = 0} \end{array}$$

and K and 1/R are step sizes.

6.3 Stability analysis

Lyapunov's theorem[15] gives a sufficient condition to check for the stability of nonlinear systems.

Consider a continuously differentiable function $V(x)$ such that

$$V(x) > 0, \forall x \neq 0$$

and $V(0) = 0$.

1. If $\dot{V}(x) \leq 0 \ \forall x$, then the equilibrium point is stable. Here. $\dot{V}$ denotes the time derivative of the function V - we are interested in the way this function varies along the state-space trajectory of the system.

2. In addition, if $\dot{V}(x) < 0, \forall x \neq 0$, then the equilibrium point is asymptotically stable.

3. In addition to the above two conditions, if V is radially unbounded, i.e., $V(x) \to \infty$, when $\|x\| \to \infty$, then the equilibrium point is globally asymptotically stable.

Lyapunov's theorem can be used to prove stability of nonlinear systems. However, the design of a Lyapunov function is not an easy task. Also, Lyapunov's theorem only provides sufficient conditions for stability. Even if we cannot find a Lyapunov function that satisfies the above theorem, we cannot conclude that the system is unstable.

For discrete systems, i.e. $x(k+1) = f(x(k))$, then we can use $V(x(k))$ and $\Delta V(x(k))$ instead of $V(x)$ and $\dot{V}(x)$. As long as $V(x(k))$ and $\Delta V(x(k))$ satisfy the conditions of Lyapunov's theorem, then the same conclusions regarding stability stand.

In [11] Srikant simulates a discrete-time system with a continuous model and designs a Lyapunov function. He then presents a proof that the primal-dual algorithm is stable as long as $U(x)$ is strictly concave on x and $k(x) > 0$ for any $x > 0$ is a non-decreasing continuous function.

Similarly, in our optimal utility-based internal congestion control, if we use the continuous model

$$\{ \begin{array}{l} \dot{x_s} = [K * (W - x_s * p)]_x^+ \\ \dot{p} = [(\sum_{s \in S} x_s - C)/R]_p^+ \end{array}$$

and set

$$V(x,p) = \sum_{s \in S} \int_{x_s^*}^{x_s} \frac{u - x_s^*}{K * u} du + \int_{p^*}^{p} R * (v - p^*) dv$$

then our optimal utility-based internal congestion control scheme is globally, asymptotically stable since $W * log(x)$ is a strictly concave function on x and $K * x > 0$ for any $x > 0$ is a non-decreasing continuous function.

Proof.

$$\begin{aligned} \dot{V} &= \sum_{s \in S} \frac{x_s - x_s^*}{K * x_s} * \dot{x_s} + R * (p - p^*) * \dot{p} \\ &\leq \sum_{s \in S} \frac{x_s - x_s^*}{K * x_s} * K * (W - x_s * p) \\ &+ R * (p - p^*) * (\sum_{s \in S} x_s - C)/R \\ &= \sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * p) + (p - p^*) * (\sum_{s \in S} x_s - C) \\ &= \sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * (p - p^* + p^*) \\ &+ (p - p^*) * (\sum_{s \in S} x_s - x_s^* + x_s^* - C) \\ &= \sum_{s \in S} (-x_s + x_s^*) * (p - p^*) + (p - p^*) * \sum_{s \in S} (x_s - x_s^*) \\ &+ \sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * p^*) + (p - p^*) * (\sum_{s \in S} x_s^* - C) \\ &= \sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * p^*) + (p - p^*) * (\sum_{s \in S} x_s^* - C) \\ &\leq 0 \end{aligned}$$

Since $\sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * p^*) = 0$ when $x_s = x_s^*$, and $\frac{W}{x_s}$ decreases as x_s increases, $\sum_{s \in S} \frac{x_s - x_s^*}{x_s} * (W - x_s * p^*) \leq 0$. On the other hand, $\sum_{s \in S} x_s^* \leq C$ and $p^* = 0$ if $\sum_{s \in S} x_s^* < C$, so $(p - p^*) * (\sum_{s \in S} x_s^* - C) \leq 0$. □

However, we can show that the underlying discrete-time system is not globally stable. We demonstrate this by simulating the discrete model numerically and plotting the phase space of stable points in terms of K and R in Figure 6.1 (We set $W = 1$). In this simulation, we set the maximum bandwidth C to be 764500 packets per second, which is the measured maximum transmission rate of the Intel E1000 Ethernet network port on PCI-X64/100 bus for 64B packets [17]. We also set the allowed error ratio to 0.001 which means that the system is said to be stable if the source rates converge to the optimal allocated bandwidth with error ratio less than 0.1%. For the discrete system, the

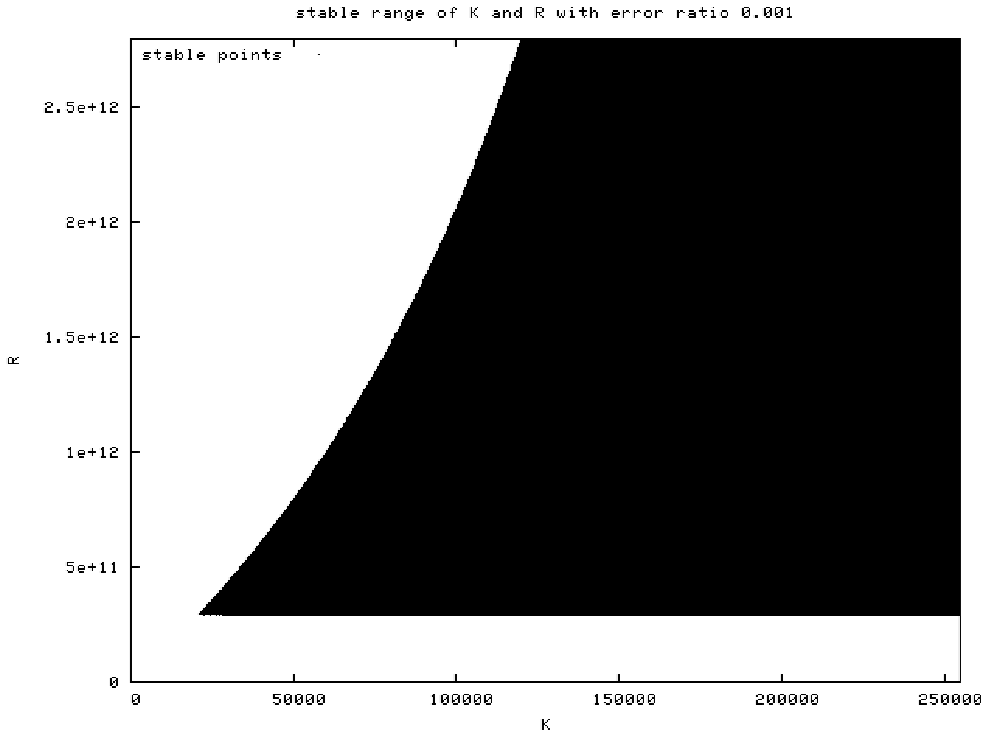


Figure 6.1: Stability region of discrete system

stable region starts at around $K = 25000$ and $R = 3 * 10^{11}$. Note that this plot is for the parameter ranges $K < 255000$ and $R < 2.8 * 10^{12}$. Whereas the continuous system is stable for any K and R, the discrete system is only stable for values of K and R in the black region of the phase plot.

6.4 Simulation with ns-3

To study the behavior of our congestion control algorithm, we simulate it using the ns-3 network simulator. This simulation is based on our extended Ipv4L3Protocol class which includes a packet classifier, queue management and scheduling, and backward explicit congestion notification (BECN)[19] (Note: For more detail information, please refer to section 7.6).

6.4.1 Simulation setup

Four general nodes installed with TCP/IP network stack are created and connected to the cluster-based router processing nodes (configured with one external network port and one internal port, and installed with the extended Ipv4L3Protocol instance), with one general ns-3 node connecting to one processing node. On three of them, we install ns-3 on/off traffic generator applications to generate traffic flow with specified traffic pattern, and the other node (connecting to the egress processing node) is installed with ns-3 traffic sink application.

The patterns of the traffic flows generated on the three traffic source nodes are configured with

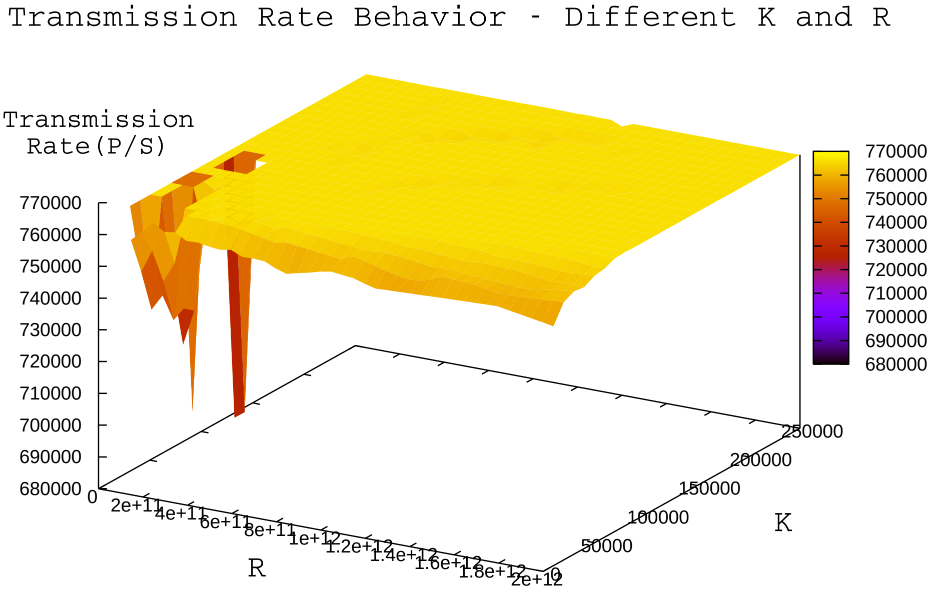


Figure 6.2: Transmission rate on the egress node

the traffic sink node as the destination. In this setup, three processing nodes (ingress nodes) of the cluster-based router receive packets from external networks and then forward the packets to the other processing node (egress node) which forwards the packets to the traffic sink node in the external network.

Since small packets stress the router more than large packets at the same aggregate bit rate because more packets have to be handled per second, we use 64B packets in our simulation. Previously [17], we measured the maximum transmission rate of the Intel E1000 Ethernet network port on PCI-X64/100 bus as 764500 packets/second for 64B packets. Thus, we set the capacity of the transmission rate on the egress node to this value.

6.4.2 Simulation results and analysis

We ran a set of simulations with varying values of K and R, which are used to calculate the utility and the congestion price. On the egress node, we measured the average transmission rate of the external network interface and the average length of the external queue which is used to buffer the packets that will be transmitted from the external network interface. In our simulation, the ingress nodes are all offered the same amount of traffic unless otherwise noted.

Stability phase space

As shown by the plot in Figure 6.1, the proposed algorithm is stable only in some specific region, although the continuous model can be proved to be stable for any values of K and R. By "stable"

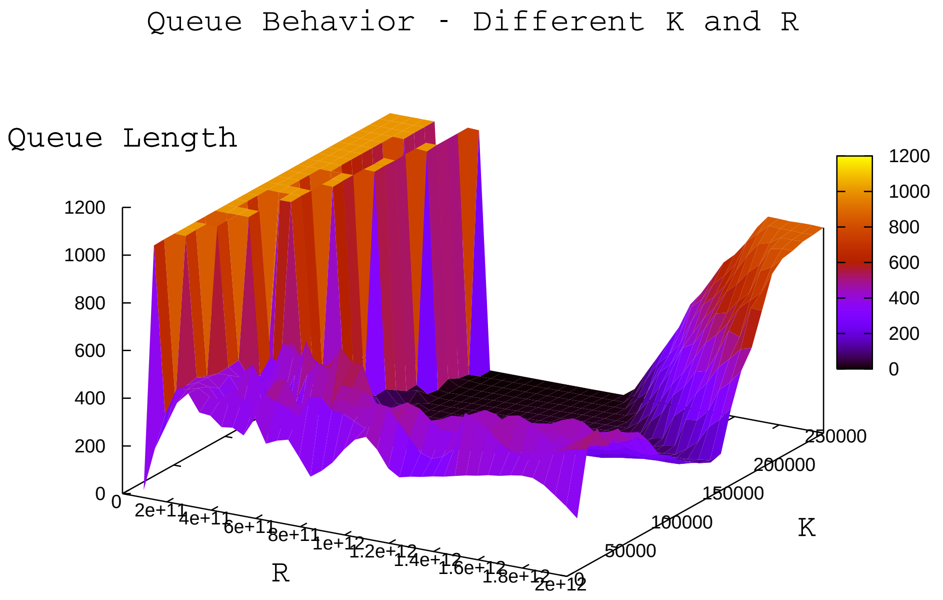


Figure 6.3: Average output queue length

we mean the transmission rate on the egress node and the sum of ingress transmission rates are close to each other and are also close to the egress transmission capacity, and that the transmission output queue on the egress node is relatively short. Due to the discrete accuracy on the control of the transmission rates on the ingress nodes and the price calculation on the egress node, a small variance of the transmission rates and the queue length should be acceptable.

Figure 6.2 shows average transmission rates on the egress node given different values for K and R after 500 iterations. For most values of K and R, the transmission rate is close to the transmission capacity of the egress node.

Figure 6.3 shows the average output queue length on the egress node. From this figure, we see that, for $K > 50000$ and $R > 3 * 10^{11}$, the output queue size remains small, which means that the reception rate on the egress node is not exceeding the transmission capacity. Combined with the fact that the transmission rate is close to the transmission capacity (Figure 6.2), we can conclude that the reception rate and the transmission rate converge to the transmission capacity and the control scheme is stable while K and R are in this range.

Convergence behavior

Figure 6.2 and Figure 6.3 give the range of K and R over which the proposed internal congestion control is likely to be convergent. Now we illustrate the convergence behavior of our internal congestion control with several different values for K and R. First, with $K = 100000$ and $R = 5 * 10^{11}$, where the average output queue length is small, we plot the behavior of the internal congestion con-

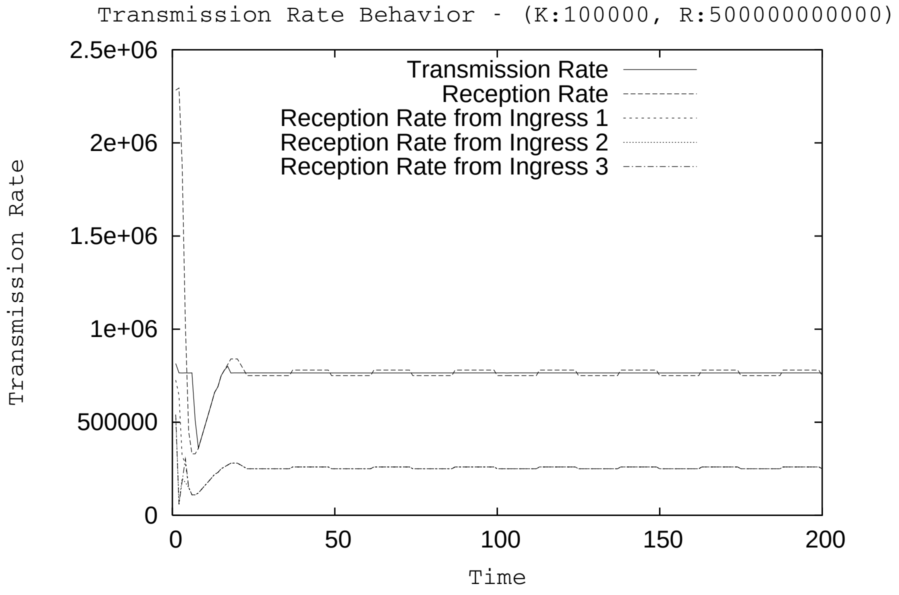


Figure 6.4: Transmission rate convergence

trol in Figure 6.4 when the three sources offer traffic at the same rates. We collect the reception and transmission rates of the egress node, and the reception rates from each ingress node at an interval which equals the control interval(1ms). The plot shows that after 20 control intervals, the rates start to converge and the reception and transmission rates of the egress node stabilize around the transmission capacity of the egress.

Given different values for R with the same value for K ($K = 100000$), the congestion control scheme behaves differently. With smaller R, the response time becomes shorter and the control step becomes bigger, which may result in the system not converging, or worse yet becoming unstable. Figure 6.5 is a plot of one unstable case with $R = 1.5 * 10^{11}$. With larger R, the response time becomes longer, the control step becomes smaller, and the system reacts more slowly. Figure 6.6 gives the behavior of our control scheme at $R = 10^{12}$.

In the other dimension, given different values for K at the same value for R, the congestion control scheme again behaves differently. With smaller K, the response time is longer while the response time is shorter with bigger K. Figure 6.7 and Figure 6.8 illustrate the behavior of our congestion control scheme with $K = 20000$ and $K = 150000$ when $R = 5 * 10^{11}$. Figure 6.8 shows a shorter convergence time compared to all other cases.

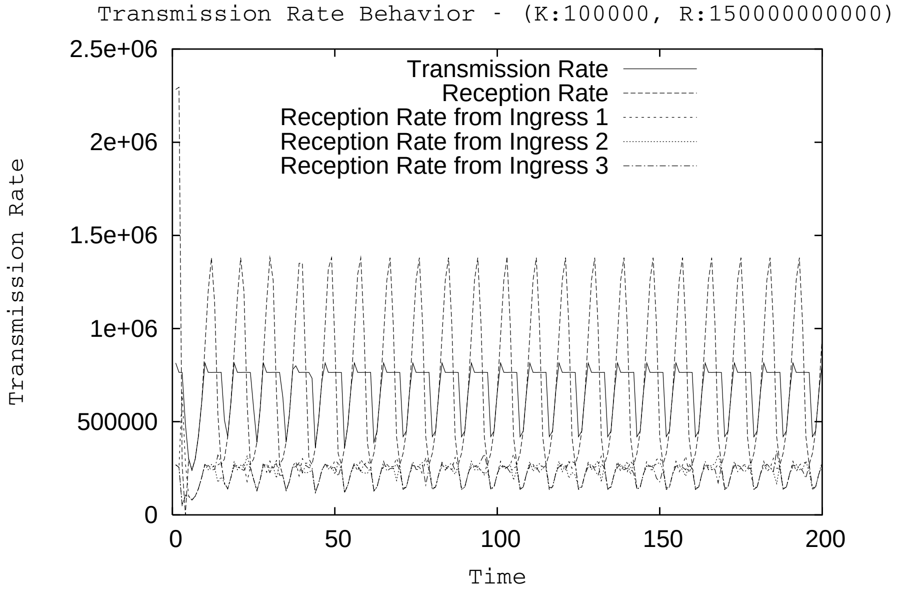


Figure 6.5: Transmission rate convergence

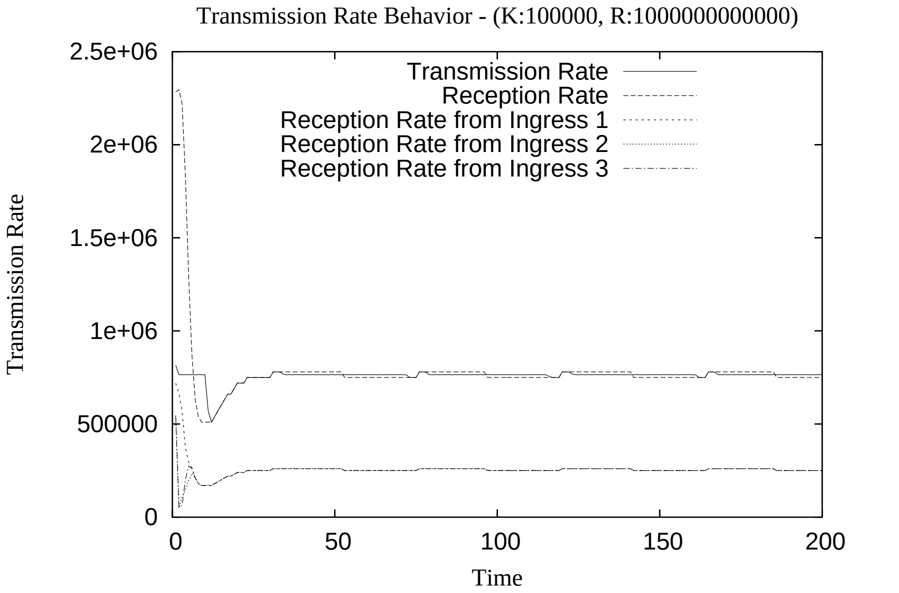


Figure 6.6: Transmission rate convergence

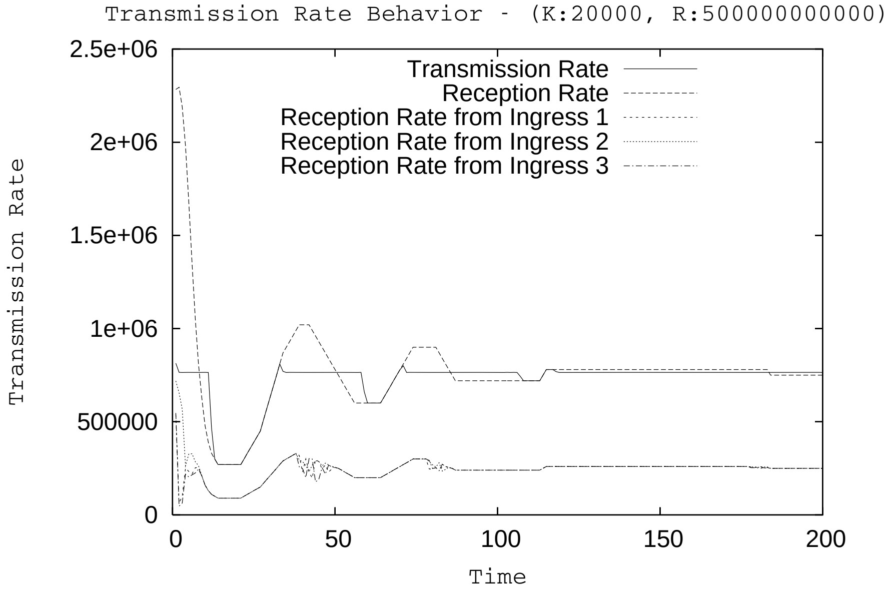


Figure 6.7: Transmission rate convergence

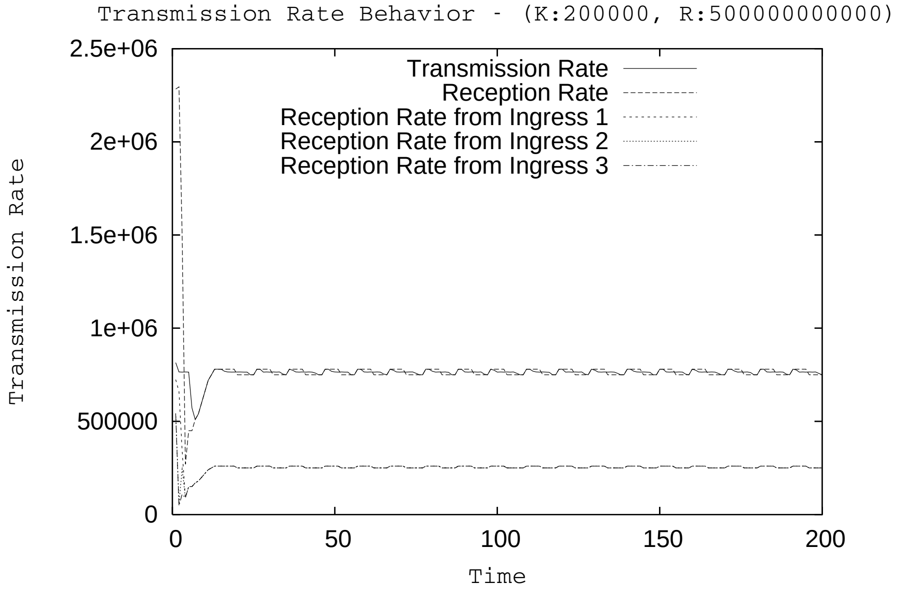


Figure 6.8: Transmission rate convergence

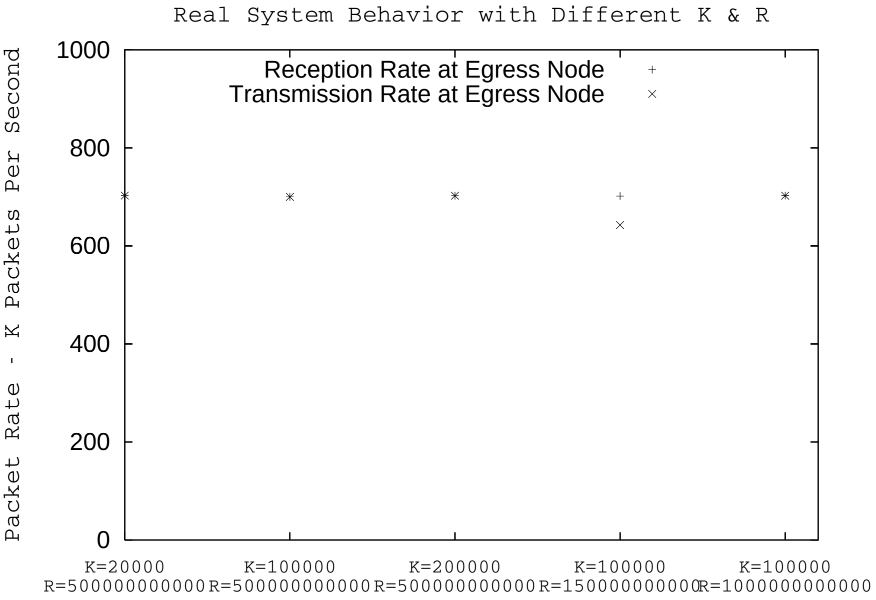


Figure 6.9: Real system behavior with different K&R

6.5 Evaluation in the real system

In this section, we present experimental results for our congestion control running in the lab on our cluster-based router prototype with the same sets of control parameter values as in the previous section.

6.5.1 Experiment

Our prototype router is composed of a cluster of four SunFire X4100 nodes interconnected by a Mellanox InfiniBand MTS2400 switch which can support 10Gbps switching rate at each port. A commercial traffic analyzer is used to generate and receive traffic. We offer traffic to make the egress port heavily overloaded - the offered load is four times the maximum load the egress port can forward. As in the simulation we use 64B packets in our tests. Different from the simulation, due to the correlation between different devices connected to the PCI bus [17], the setpoint for the egress port transmission rate is 700K packets/second. The setpoint must be less than the transmission capacity of the port (760K packets/second) measured when there is no other traffic on the system due to the effects of bus contention and live-lock on the interface card itself.

6.5.2 Measurements

The way our congestion control scheme behaves with different values for K and R is presented in Figure 6.9. For most settings the transmission rate is close to the transmission setpoint of 700K packets/second, while in the case corresponding to that shown in Figure 6.5, the transmission rate is somewhat less than the setpoint. The large fluctuation of the injection rate in this case causes overflow of the egress queue, which results in packet drops and low average transmission rate on the egress node. For all other cases, because the fluctuation of the injection rate is small and the transmission capacity of the egress port used by the controller is less than the actual transmission capacity, the fluctuation is absorbed by the difference between the set and actual transmission capacity, which results in fewer dropped packets.

These experimental results from the real system show that either the ns-3 simulations or the calculations from our discrete model can be used to predict the stability and performance of our optimal congestion control. The predictions from the stability analysis associated with the continuous model, however, have been shown unreliable.

6.6 Conclusions

Stability is an important criterion in the design of congestion control schemes. In this paper, we introduce an optimal utility-based congestion control scheme and use a continuous-time model to prove that it is globally asymptotically stable. However the results from a discrete-time numeric simulation and from a discrete-event simulation using ns-3 point to a much different conclusion. These two discrete-time methods establish that the congestion control scheme has a specific and limited stability region. Experimental results from our cluster-based prototype confirm the correctness of the results from the discrete-time methods.

Stability analyses for discrete systems based on continuous methods have been published widely. Our results show that these analyses need to be questioned. Our results also show that the proposed congestion control scheme is effective with control parameter values in the stable region. The discrete stability analysis provides guidance on choosing specific values for the control parameters, which is often problematic in actual applications. As shown in ns-3 simulation (Figure 6.5) and experiment results, choosing control parameter values in the unstable region makes the optimal congestion control scheme inefficient, where the ingress and egress transmission rates fluctuate, and the egress transmission capacity is underutilized.

6.7 Appendix - Experimental setup

The processing nodes' architecture is described in figure 5.11.

For detailed description of experimental setup, please refer to section 7.7. The connections between cluster-based router and traffic analyzer are described in figure 6.10. The traffic analyzer is

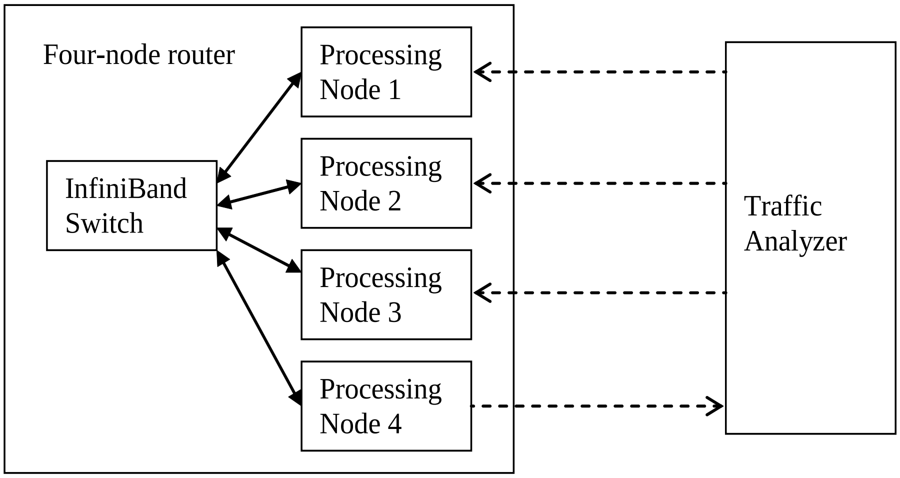


Figure 6.10: Connections between router and traffic analyzer

configured to offer Ethernet traffic to three of the nodes (one port each) with the destination being a subnet reached via the fourth node.

6.8 Appendix - Algorithms

The figures 6.11 and 6.12 give the algorithms implemented in IBBECNPack and IBBECNUnpack elements as described in section 6.2:

Optimal IBBECNPack pull (Input: None, Output: A InfiniBand Packet)
$BECNpacket$: A BECN packet to be sent or received
$Dest$: Destination address field in a BECN packet
$Queueid$: Queue id field in a BECN packet
$Local$: Local address
$MaxRate[i]$: The ith queue's maximum rate
$CurrentRate[i]$: The ith queue's current rate
K: Rate adjusting step size
W: Proportional weight
p: Congestion price

```
WHILE The BECN queue is not empty DO
REPEAT
    Get a BECN packet $BECNpacket$
    Get the destination address $Dest$, queue id $Queueid$, and congestion price $p$
    IF $Dest = Local$
        Return the BECN packet $BECNpacket$
    ELSE
        $MaxRate[Queueid] = CurrentRate[Queueid] + K * (W - CurrentRate[Queueid] * p$
        IF ($MaxRate[Queueid] < 0$)
            $MaxRate[Queueid] = 0$
        END IF
    END IF
END WHILE
Get enough Ethernet packets from upstream queues according to the corresponding queues' $MaxRate$
Pack the Ethernet packets into one InfiniBand packet and return the packet
```

Figure 6.11: Optimal IBBECNPack pull

Optimal IBBECNUnpack push (Input: A InfiniBand Packet, Output: None)

$IBpacket$: The InfiniBand packet which is being processed

$T_{current}$: Current time

T_{last}: Last control time

$T_{interval}$: Control interval

C: Maximum transmission rate of a queue

R_{recv}: Incoming rate to a queue

p: Congestion price associated with a queue

R: Price adjusting step size

$Packet$: A Ethernet Packet

i: A Ethernet packet's index

```
Get a InfiniBand packet $IBpacket$
IF ($T_{current} - T_{last} > T_{interval}$)
    FOR each output queue of this IBBECNUnpack element DO
    REPEAT
        Get this queue's maximum transmission rate $C$
        Get this queue's incoming rate $R_{recv}$
        $p+ = (R_{recv} - C)/R$
        IF ($p < 0$)
            $p = 0$
        END IF
        Create a broadcast BECN packet with price $p$ and push it to the BECN queue
    END FOR
END IF
IF ($IBpacket$ is a BECN packet)
    Push $IBpacket$ to a downstream BECN queue according to the source address
ELSE
    Unpack the packet $IBpacket$ to multiple Ethernet packets
    FOR each Ethernet packet $Packet$ and it's index $i$ DO
    REPEAT
        Push $Packet$ to the downstream $i$th queue
    END FOR
END IF
```

Figure 6.12: Optimal IBBECNUnpack push

Bibliography

[1] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: active queue management. *IEEE Network*, 15:48–53, 2001.

[2] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[3] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[4] Xingzhe Fan, Murat Arcak, and John T. Wen. L_p stability and delay robustness of network flow control. In *Proceedings of the IEEE Conference on Decision and Control, Maui*, pages 3683–3688, 2003.

[5] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. In *IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 3, pages 1435–1444, March 2000.

[6] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[7] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988. ACM.

[8] Frank P. Kelly, Aman Kumar Maulloo, and David Tan. Rate control in communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[9] Anup Kumar, Arpana Maniar, and Adel S. Elmaghraby. A fair backward explicit congestion control scheme for ATM network. In *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, pages 452–457, Washington, DC, USA, 1999. IEEE Computer Society.

[10] Srisankar S. Kunniyur and R. Srikant. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Transactions on Networking*, 12:286–299, 2004.

[11] Shao Liu, Tamer Basar, and R. Srikant. Controlling the Internet: a survey and some new results. In *Proceedings of the 43nd IEEE Conference on Decision and Control*, pages 3048–3057, Maui, Hawaii USA, December 2003.

[12] Steven H. Low and David E. Lapsley. Optimization flow control - I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999.

[13] Peter Newman. Traffic management for ATM local area networks. *IEEE Communications Magazine*, 8:44–50, 1994.

[14] Fernando Paganini. A global stability result in network flow control. *Systems & Control Letters*, 46:165–172, 2002.

[15] R. Srikant. *The Mathematics of Internet Congestion Control*. Systems & Control: Foundations & Applications. Birkhauser, 2003.

[16] Qinghua Ye and Mike H. MacGregor. Cluster-based IP router: Implementation and evaluation. In *IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006.

[17] Qinghua Ye and Mike H. MacGregor. Hardware bottleneck evaluation and analysis of a software PC-based router. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systerms*, pages 480–487, Edinburgh, UK, June 2008.

[18] Qinghua Ye and Mike H. MacGregor. Internal congestion control in the cluser-based router. In *Syposium on High Performance Switching and Routing*, pages 13–20, Shanghai, China, May 2008.

[19] Qinghua Ye and Mike H. MacGregor. Optimal utility-based internal congestion control in a cluster-based router. Technical Report TR09-12, Department of Computing Science, University of Alberta, Edmonton, Alberta, 2009.

Chapter 7

Performance Analysis and Evaluation of an Optimal Internal Congestion Control Scheme in a Cluster-based Router

7.1 Introduction

Software routers based on commodity hardware and open source operating systems are gaining more and more interest from both scientific researchers and business users. They have advantages in terms of price-performance and customizability compared to closed proprietary systems. For small and medium enterprises the price-performance advantage may be attractive. In a research setting, an open source router provides an opportunity to explore new algorithms and modify or extend router functions. However, in both contexts scalability and congestion control are of concern.

In order to address scalability we have proposed and evaluated an extensible and scalable cluster-based router framework[20]. Using this framework we have developed a prototype router based on a cluster of commodity processors interconnected by an InfiniBand network. Our cluster-based router performs very well and can be scaled as needed to deliver higher packet forwarding rates.

As in any distributed system, congestion control is a critical design issue. Congestion can affect both the forwarding rate and the latency of packets travelling through the router. If packets are sent too quickly from multiple ingress nodes to an egress node, the egress node will become overloaded and its forwarding capability will be reduced. At this point, packets will be delayed and perhaps even dropped due to the queue overflow at the egress. A good congestion control scheme can prevent the ingress nodes from overwhelming the egress nodes, which can also reduce the waste of internal network bandwidth and CPU cycles by dropping packets as early as possible, so as to maximize the forwarding rate under overload[21].

[0] This chapter is based on the paper ”Performance Analysis and Evaluation of an Optimal Internal Congestion Control Scheme in a Cluster-based Router” submitted to Elsevier ”Computer Networks”

A significant amount of work has been done in the area of network congestion control. TCP [6] adopts an adaptive AIMD (Additive Increase, Multiplicative Decrease) window flow control algorithm depending on the detection of packet loss due to the overflow of buffers in the network. AQM (Active Queue Management) schemes like REM [1] and RED [4] avoid congestion and maintain low queue lengths on the router by dropping packets earlier while allowing occasional bursts of packets in the queues. Based on RED, ECN (Explicit Congestion Notification)[16] allows end-to-end notifications of network congestion without dropping packets by marking any packet that RED drops. To address the instability of TCP in high-bandwidth optical networks, XCP [7] generalizes the Explicit Congestion Notification proposal (ECN) to include the degree of congestion at the bottleneck instead of the congestion indication, and it introduces the concept of decoupling the utilization control from the fairness control in the calculating of the congestion notification. In ATM networks, Newman[14] proposed a backward explicit congestion notification AIMD scheme for ATM local networks, and Kumar[10] developed a fair backward explicit congestion control scheme which monitors the buffer utilization in the network and generates control cells which give the recommended rates the sources should use whenever buffer overflows.

Many high performance routers use crossbars as internal switching fabrics [5] [18], which segment received variable length packets into fixed length cells for transmission through the crossbar and reassemble at the output. This simplifies the centralized scheduler in the crossbar and allows synchronous operations. However, this requires the scheduler to make decisions at every cell transmission interval. It becomes infeasible with increasing numbers of links and increasing link rates in a large router. To break this limitation, with the addition of buffers to each of the crosspoint in the crossbar, Pappu[15] proposed a distributed queueing algorithm. His algorithm regulates the rates at which traffic is forwarded through the interconnection network from inputs to outputs using asynchronous coarse-grained scheduling instead of iteratively scheduling the transmission of individual packets. In this algorithm, the port processors periodically exchange information about the status of their VOQs (Virtual Output Queues). This information is then used to rate control the VOQs, with the objective of moving packets to the output side of the router as expeditiously as possible while avoiding congestion within the switch fabric. The scheme allows the forwarding of variable length packets, which eliminates the need for segmentation and reassembly and is not subject to bandwidth fragmentation [18].

Despite the numerous studies of congestion control, the optimality of such schemes under various measures of utility has not been established. High-speed routers are characterized using a variety of performance measures (maximum forwarding rate and minimum latency, among others). Thus, we require a flexible congestion control that can be crafted to the demands of the specific product or network situation. Of course, we should not have to sacrifice performance for flexibility. In the following we present an optimal congestion control scheme based on the general framework of ”utility”. The behavior of this method can be tuned to the requirements at hand by altering the mathematical form adopted for the utility function. As such, this method represents a very wide range of specific controllers, and the performance results presented in the following are exemplar of a specific instance of the general method. In this paper, we show results for a proportionally-fair utility function. However some framework can accommodate other utility forms of utility function, and lead to other forms of fairness.

In this paper, we compare our optimal scheme to an AIMD scheme. Both methods use BECN (Backward Explicit Congestion Notification) to manage the congestion internal to our cluster-based router. In the optimal utility-based scheme, BECN messages including the congestion price for each egress queue are sent from the egress to the ingresses at every control interval, and the ingress nodes adjust the injection rates of flows to the egress queues based on the congestion prices received. In the AIMD scheme, BECN messages are sent from the egress to the ingresses only when congestion is detected (the queue length exceeds the congestion threshold) on the egress queues. In this situation, the ingress nodes decrease the injection rates of flows to the congested egress queue by half if a congestion notification is received.

We have simulated both the optimal and AIMD schemes with ns-3 and evaluated them experimentally in our cluster-based router prototype. Our results show that, both schemes can be fair to different incoming flows and improve the overall forwarding rate of the router by reducing the injection rate of packets to the internal network when the router is under a heavy load. However, in the AIMD scheme, the injection rates of flows, the forwarding rate of the egress port and the egress queue length fluctuate markedly, while the optimal utility-based scheme provides a stable control with small fluctuation. This results in a higher average forwarding rate, lower forwarding delay, and smaller queue length and variance.

The rest of the paper is organized as follows. In section 7.2, we briefly introduce the architecture of our cluster-based router. Section 7.3 describes the internal congestion problem in the cluster-based router and our design considerations. The optimal utility-based internal congestion control scheme is explained in section 7.4 and the additive increase, multiplicative decrease scheme is presented in section 7.5. Section 7.6 gives the ns-3 simulations of both schemes. We report the experimental evaluation in section 7.7. We conclude our work in section 7.8.

7.2 Cluster-based router

A cluster-based router [20] is composed of two main parts: the processing nodes and the interconnection network. In our prototype (See Figure 7.1), the processing nodes are commodity server blades each equipped with one or more interface cards terminating the links to the outside world. The server blades also carry interfaces for terminating the links to the internal interconnection network. Our prototype uses 1 Gbps Ethernet links to the outside world. The internal interconnection network is a high-throughput/low-latency InfiniBand fabric.

We can divide the processing nodes into two categories: external processing nodes and internal

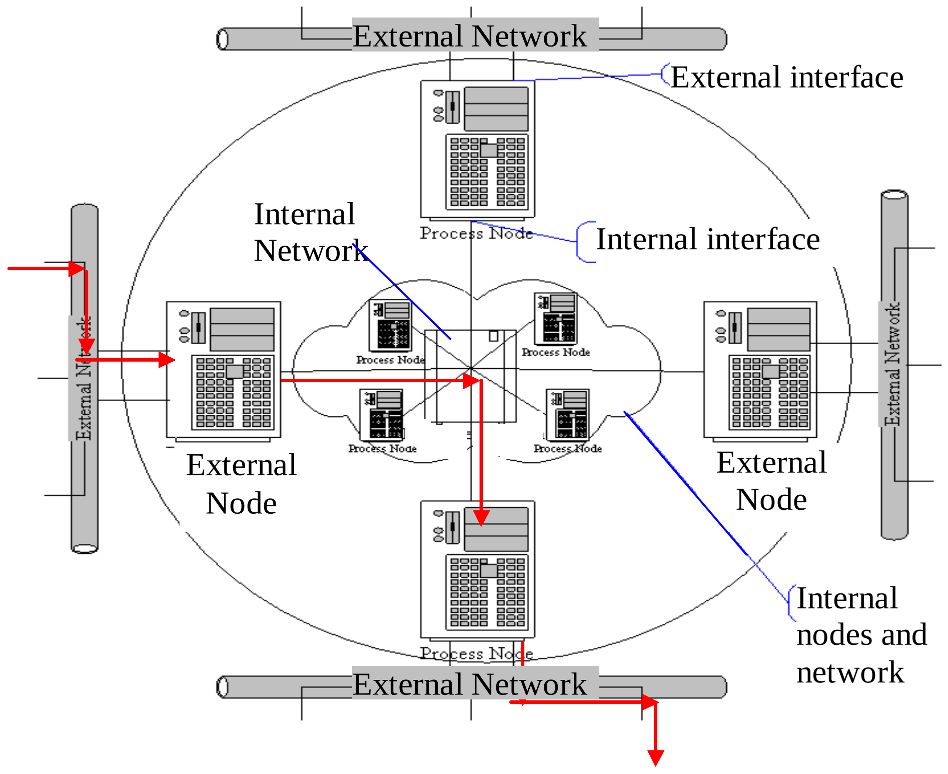


Figure 7.1: Cluster-based router architecture

processing nodes. The external nodes connect to the attached outside networks, and they are responsible for receiving and forwarding packets from/to the attached outside networks. The external nodes are dedicated to packet forwarding in the critical data path. The internal processing nodes are dedicated to control and management tasks such as routing table updates, network management and other non-time-critical processings such as authentication and accounting.

All nodes - both internal and external - are connected to the internal interconnection network. Packets must cross the internal interconnection network for forwarding unless they are destined to a network connected to the receiving external node, most likely on a different physical interface, although the nodes may also forward traffic between multiple virtual interfaces on the same physical interface. A packet crosses the interconnection network only once if there is no additional processing needed in the internal processing nodes. Figure 7.2 gives the forwarding path in each processing node. In the following we will discuss ”ingress” and ”egress” nodes but it should be noted that these terms are relative to a given flow, and the connotation needs to be kept separate from that of ”external” and ”internal” nodes. There are no dedicated ingress or egress nodes as such - a particular processing node may function simultaneously as an ingress node for some flows, and an egress node for others.

An IP packet is received by the Ethernet driver at the ingress node where it will be captured by the external receiving element in a Click kernel thread [9]. The packet then follows the processing

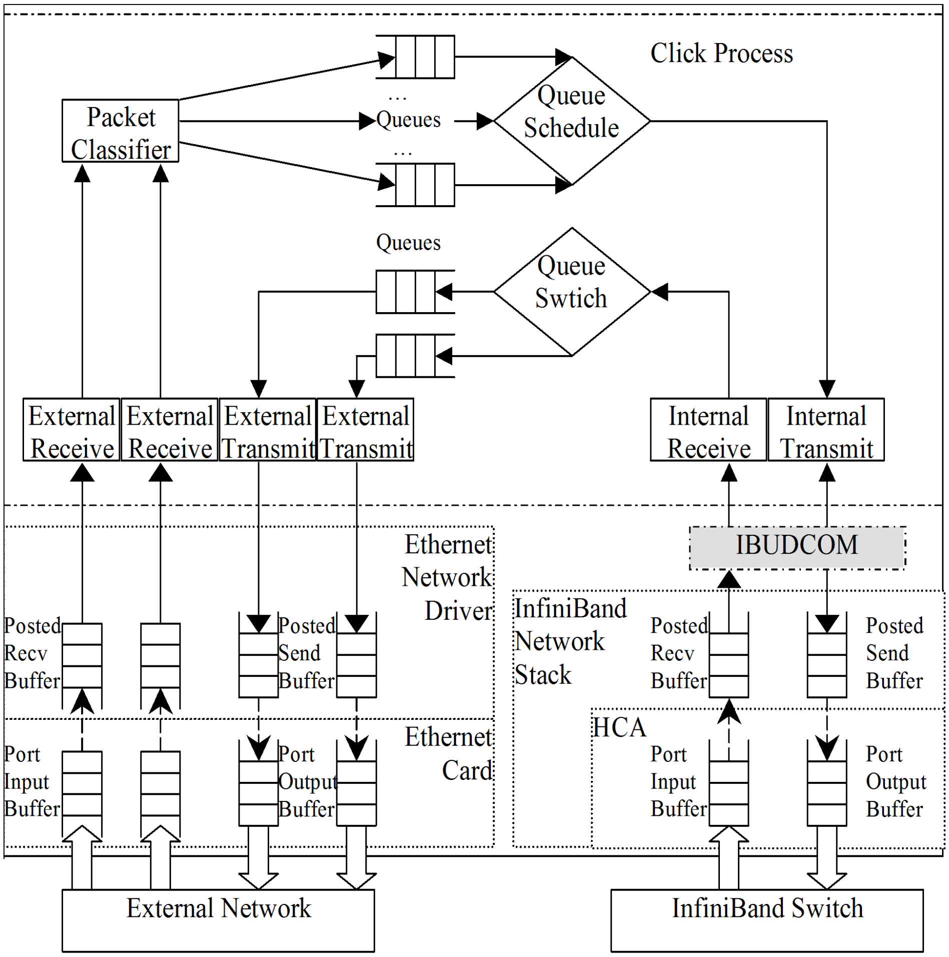


Figure 7.2: IP forwarding path

path in the ingress node according to the defined Click configuration. At the end of this processing path, the internal interface passes the packet to the IBUDCOM (InfiniBand Unreliable Datagram COMmunication) layer and the packet traverses the internal network to the egress node. The packet is captured from the IBUDCOM layer by the internal interface at the egress node, switched to a specific outbound queue, and finally sent to the external network by the external interface through the Ethernet driver.

7.3 Internal congestion problem and design considerations

Congestion control is a critical issue in the design of our cluster-based router. As described in [22], the internal communication between processing nodes in our cluster-based router determines the potential congestion mechanisms on the path from ingress ports to the egress ports.

There are two potential levels at which congestion may occur in our system: the queues in the Click packet processing layer and the posted buffers in the InfiniBand network stack posted by the IBUDCOM transport layer. In the first case, congestion can be detected by monitoring the queues on both ingress and egress nodes. The detection of congestion in the IBUDCOM transport layer is much more complex. Fortunately, we observed experimentally that any congestion in the transport layer is always preceded by congestion in the egress queues if the scheduling weights are set properly. We can choose the scheduling weights so that, before any packets are dropped by the transport layer due to the overrun of the egress InfiniBand posted receive buffer, the egress queues are already overloaded. This is possible because the receive buffer on the egress node is overrun only when the egress node processes received packets more slowly than the rate at which the packets arrive. If we give higher weights to the receiving of internal packets than to the transmission of external packets, then the receiving of internal packets will be scheduled more often than the transmission of external packets. As a result, if the receive buffer is overrun the egress queue must be congested. Given this observation, we need to monitor only the status of the queues to detect potential congestion.

A variety of mechanisms might be considered here. We start by considering the following: if the ingress nodes can be signalled to reduce their transmission rates to any overloaded egress queues then the drop rate of packets on those egress queues will be reduced. This improves the overall router throughput because it avoids wasting the internal network bandwidth on dropped packets and reduces the proportion of cycles in both ingress and egress nodes which are spent processing packets that will be dropped due to congestion.

Similar to previous work, an internal congestion control scheme that achieves above requirements is composed of the following three steps: congestion detection, congestion notification and congestion reaction.

When the arrival rate of packets on a queue exceeds the transmission capacity of the queue, congestion occurs. Monitoring the difference between the arrival rate and transmission capacity of a queue seems a natural way to detect congestion. However, due to the variance of transmission capacity of a queue (multiple queues may share an egress port and the transmission capacity of a queue or a port may be affected by the system status due to the correlation of system components [21]), the difference is very difficult to monitor and is not a reliable congestion metric. Given the burstiness of Internet traffic, we believe that queue length variance is a better congestion metric.

A congestion notification could include either or both of the congestion status of egress queues on an egress node, or the recommended transmission rates for the ingress nodes. There are a variety of choices for when to send a congestion notification from an egress to an ingress. Notification can be sent to ingress nodes whenever congestion occurs, an indication whether an egress is congested could be sent periodically, or an indication could be sent every time a packet is received at the egress queue. Sending notifications periodically will help reduce the transmission overhead of congestion notifications. Due to the variance of transmission capacities of egress queues and the overhead of monitoring packet rates and computing recommended transmission rates, we include only the congestion status of egress queues in the congestion notification and leave calculation of transmission rates to the ingress nodes.

In reacting to the congestion notification, the ingress nodes adjust their maximum transmission rates to the egress queues according to the egress queue congestion status in the congestion notification and the current transmission rates. The token bucket algorithm is used to control the transmission rates on the ingress nodes.

In the following sections, we propose and describe how congestion notifications are used by our optimal utility-based internal congestion control scheme, and then introduce the AIMD scheme which is used for comparison.

7.4 Optimal utility-based control

Optimal utility-based congestion control is an optimization approach to congestion control problems where the objective is to maximize the aggregate source utility. In this approach, the network links and traffic sources are viewed as a distributed system that acts to solve the optimization problem given the constraints of network link capacities. The traffic sources adjust their transmission rates in order to maximize their own benefit, which is the utility minus bandwidth cost, while the network links adjust bandwidth prices to coordinate the sources' decisions on the evolution of their transmission rates[13].

Based on the controlled objects, optimal utility-based congestion control schemes can be divided into three classes - primal algorithms, dual algorithms and primal-dual algorithms[12]. In primal algorithms, users adapt their source rates dynamically based on route prices, and links select a static law to determine the link prices directly from the arrival rates at the links[8]. An example of a primal algorithm is TCP flow control[6]. In dual algorithms, on the other hand, links adapt the link prices dynamically based on the link loads, and users select a static law to determine the source rates directly from route prices and source parameters. REM[1] and AVQ[11] are good examples of dual algorithms. In primal-dual algorithms, both link prices and source rates are slowly adjusted so that they asymptotically converge to the optimal solution[12]. In today's Internet, the combination of TCP flow control algorithm and some active queue management algorithms like REM[1], RED[4] and AVQ[11] are examples of primal-dual algorithms.

In this section, we first present our model of the internal network in the cluster-based router, and then develop the discrete congestion control scheme.

7.4.1 Internal congestion control as an optimization problem

Different from previous work, instead of the links in a network, we focus on the state of the egress node in our cluster-based router. The cluster can be modelled as a network with only one egress node as follows.

Consider a network with unidirectional links. There is a finite forwarding capacity C associated with the egress. The egress is shared by a set S of sources, where source $s \in S$ is characterized by a utility function $U_s(x_s)$ that is concave increasing in its transmission rate x_s to the egress.

The goal of congestion control is to calculate the source rates that maximize the sum of the utilities $\sum_{s\in S} U_s(x_s)$ over x_s subject to the capacity constraints. The model can be written as:

$$P : \sum_{s\in S} U_s(x_s) \tag{7.1}$$

subject to

$$\sum_{s\in S} x_s \leq C \tag{7.2}$$

(7.2) means that the aggregate source rate at the egress should not exceed its outgoing transmission capacity. Since the utility function $U(x)$ is strictly concave and continuous, an optimal solution exists. Solving this problem centrally would require the knowledge of all utility functions, and also the knowledge of all the source rates and aggregate rate at the egress. This makes it inflexible for changes in the numbers and/or types of sessions, so a decentralized approach is preferable.

7.4.2 Decentralized approach

From (7.1), the objective function is separable in x_s. However, the source rates are coupled by the constraint (7.2). Fortunately, the dual theory of optimization [2] leads us to a distributed and decentralized solution which results in the coordination of all sources implicitly[13].

As we described in the previous subsection, the utility function $U(x)$ is a differentiable concave function, so we can define the Lagrangian function of the original primal problem as:

$$\begin{aligned} L(x,p) &= \sum_{s\in S} U_s(x_s) - p(\sum_{s\in S} x_s - C) \\ &= \sum_{s\in S} U_s(x_s) - \sum_{s\in S} x_s * p + p * C \end{aligned} \tag{7.3}$$

The objective function of the dual problem is:

$$\begin{aligned} D(p) &= \max_{x_s} L(x,p) \\ &= \sum_{s\in S} \max(U_s(x_s) - x_s * p) + p * C \end{aligned} \tag{7.4}$$

and the dual problem is

$$D : \min_{p\geq 0} D(p) \tag{7.5}$$

In this dual problem, if p is interpreted as the price per unit bandwidth at the egress, then the first term of the objective function $D(p)$ is decomposed into S separable subproblems which can be solved at each source s with some information from the egress. Let

$$B_s(p) = \max(U_s(x_s) - x_s * p) \tag{7.6}$$

Hence, $x_s * p$ represents the bandwidth cost to source s when it transmits at rate x_s, and $B_s(p)$ represents the maximum benefit source s can achieve at the given price. Based on the congestion information p from the egress, each source s can solve the subproblem $B_s(p)$ without coordinating with other sources. Let $x_s(p)$ be the local optimizer to $B_s(p)$ at source s. Then from (7.4), $x_s(p)$ is the local maximizer of the Lagrangian function (7.3).

In general, $x_s(p)$ may not be the primal optimal. However, according to the Karush-Kuhn-Tucker theorem, if there is a minimizer p^* which is the optimizer of the dual problem (7.5), and $x^* = x_s(p^*), s \in S$ is the maximizer of (7.4), then (x^*, p^*) constitutes a saddle point for the function $L(x,p)$, and x^* solves the primal optimization problem P.

If we can solve the dual problem (7.5) locally at the egress to obtain p^*, then the original maximization problem P can be solved in a distributed manner using only the decentralized information available locally at each ingress. In this case, p^* serves as a coordination signal that aligns individual optimality of (7.6) with social optimality of (7.1).

Based on the above theory, the congestion control problem can be generalized to the tasks of finding distributed algorithms that can make sources adapt transmission rates with respect to the egress price and make the egress adapt prices with respect to loads. The optimal solution to the distributed congestion control problem satisfies:

$$\left\{ \begin{array}{l} \frac{\partial D(p)}{\partial x_s} = \frac{\partial U_s(x_s)}{\partial x_s} = U_s'(x_s) - p = 0 \\ \frac{\partial D(p)}{\partial p} = \sum_{s\in S}(-x_s) + C = 0 \end{array} \right.$$

These two equations can be solved in a distributed manner at the sources and the egress. To reduce the overhead of transferring the link price, we only send the price from the egress to the sources at the beginning of each control interval, which results in a discrete-time control model.

Applying the gradient projection method where the source rate and egress price are adjusted according to the gradients (above two equations), we get a primal-dual algorithm. Here we use a proportional fairness utility function $U(x) = W * \log x$ [17], where W is the proportional weight of sources.

$$\{\begin{array}{l} x_s(k+1) = [x_s(k) + K * x_s(k) * (U_s'(x_s(k)) - p(k))]^+_{x_s[k]} \\ \qquad\qquad = [x_s(k) + K * (W - x_s(k) * p(k))]^+_{x_s[k]} \\ p(k+1) = [p(k) + (\sum_{s \in S} x_s(k) - C)/R]^+_{p(k)} \end{array} \tag{7.7}$$

Here

$$[g(x)]^+_y = \{ \begin{array}{ll} g(x), & \text{y} > 0 \\ max(g(x), 0), & \text{y = 0} \end{array}$$

and K and 1/R are step sizes.

7.4.3 Queue status as an indicator of congestion

Equation (7.7) gives the calculation of p based on the difference of aggregate transmission rate from all sources and the transmission capacity of the egress. However, in the real system, the correlation of components in commodity hardware[21] (more than one port may share the same bus) and sharing of a single egress port by multiple egress queues make the transmission capacity of the egress in the model (corresponding transmission capacity of specific port or queue) vary for different situations or times.

Fortunately, without knowing the transmission capacity of the egress, we can get the difference of aggregate transmission rate from all sources (reception rate on the egress) and the transmission capacity of the egress from the variance of the egress queue, i.e. $\sum_{s \in S} x_s(k) - C = \frac{q(t2) - q(t1)}{t2 - t1}$ if $0 < q(t) < q_{max}$. $q(t)$ is the length of the egress queue at time t, while q_{max} is the maximum capacity of the egress queue. So we get:

$$\{\begin{array}{l} x_s(k+1) = [x_s(k) + K * (W - x_s(k) * p(k))]^+_{x_s[k]} \\ p(k+1) = [p(k) + (delta(q))/R]^+_{p(k)} \end{array} \tag{7.8}$$

However, with this algorithm, the system may be stable at large queue length, which means that the aggregate transmission rate from sources stabilizes around transmission capacity of the egress, but the queue stabilizes around a large queue length. This is undesirable because large queues will lead to large packet delays. To reduce the stable queue length we introduce queue length into the price calculation.

$$\{\begin{array}{l} x_s(k+1) = [x_s(k) + K * (W - x_s(k) * p(k))]^+_{x_s[k]} \\ p(k+1) = [p(k) + (delta(q) + f(q))/R]^+_{p(k)} \end{array} \tag{7.9}$$

Here, $f(q)$ is a function of queue length indicating the effect of queue length that should be taken account into the price calculation. Let $f(q) = (q - q_o) * u$, where q_o is the objective of an egress queue length that we would like to achieve, and u is the degree that the queue length would be taken account into the price calculation. As an optimal congestion control scheme, we should make q_o as small as possible. However, if q_o is set too small, the queue length will stay around 0, which makes the $delta(q)$ not reflect the actual difference between reception and transmission rates of the egress queue.

In [23], we analyzed the stability of optimal congestion control expressed by equation (7.7). For the maximum bandwidth $C = 764500$ packets per second, which is the measured maximum transmission rate of the Intel E1000 Ethernet network port on PCI-X64/100 bus for 64B packets [21], setting $K = 100000$ and $R = 5 * 10^{11}$ makes equation (7.7) a stable optimal congestion control scheme which is efficient and convergent. Starting from $K = 100000$ and $R = 5 * 10^{11}$, our experiments show that $q_o = 100$ and $u = 0.01$ make (7.9) a good congestion control scheme. In the following simulations and experiments, we use these parameters.

7.5 AIMD congestion control

The additive increase, multiplicative decrease (AIMD) algorithm is a feedback control algorithm used in many congestion control/avoidance schemes like TCP congestion avoidance [6]. AIMD combines linear growth of the congestion rate/window with an exponential reduction when congestion takes place. It probes for available bandwidth in the network by increasing the transmission rate or window size on the source until congestion occurs.

7.5.1 Network model

Consider a network with unidirectional links. There is a finite forwarding capacity C associated with the egress. The egress is shared by a set S of sources. Let $x_s(k)$ be the rate at which source s is injecting packets into the network, $I(k)$ be the feedback that the egress provides to the sources indicating whether the total arrival rate at the egress is greater than the forwarding capacity of the egress. The AIMD congestion control scheme can be described by:

$$\{ \begin{array}{l} x_s(k+1) = x_s(k) + M * I(k) * x_s(k) + W * (1 - I(k)) \\ I(k+1) = \{ \begin{array}{ll} 0, & \sum_s x_s(k) < C \\ 1, & otherwise \end{array} \end{array} \tag{7.10}$$

Where W is the additive constant indicating how much the source rate should increase if there is no congestion, and M is the multiplicative constant representing the degree that the source rate should decrease when congestion happens.

In [3], Chiu and Jain analyzed a wide class of increase/decrease algorithms and proved that a simple AIMD congestion control algorithm ($W > 0$ and $M < 1$) satisfies the sufficient conditions for convergence to an efficient and fair state regardless of the starting state of the network. However, they didn't investigate the impact of M and W on the performance metrics. In [19], Yang and Lam investigated in detail on the relationship between parameters W and M and various performance metrics for the TCP algorithm, particularly the sending rate as a steady state metric, and responsiveness, aggressiveness and rate fluctuations as transient metrics.

7.5.2 Queue status as an indicator of congestion

For the AIMD algorithm described by equation (7.10), we set the congestion indicator when the injection rates from sources exceed the transmission capacity of the egress. However, in the real system, the correlation of commodity components [21] (more than one port may share the same bus) and the sharing of a single egress port by multiple egress queues make the transmission capacity of the egress in the model (corresponding transmission capacity of a specific port or queue) vary for different situations or times. Therefore, we use queue status as the indicator of congestion as in many active queue management congestion control schemes.

The AIMD algorithm based on queue status can be described as:

$$\{ \begin{array}{l} x_s(k+1) = x_s(k) + M * I(k) * x_s(k) + W * (1 - I(k)) \\ I(k+1) = \{ \begin{array}{ll} 0, & q(k) > Q \\ 1, & otherwise \end{array} \end{array} \tag{7.11}$$

Where Q is the threshold indicating congestion.

7.6 Simulation with ns-3

To study and compare the optimal and AIMD congestion control algorithms, we simulated them using the ns-3 network simulator. ns-3 is a discrete-event network simulator that is targeted primarily for research and education use. The network components in ns-3 can be easily extended.

ns-3 implements most key network components, from network devices to socket APIs and applications. However, it limits itself to the standard network stack or Internet protocol suites. To simulate novel network stacks, extra work is necessary to extend existing network components.

In terms of router components, ns-3 implements the standard IP forwarding path in the Ipv4L3Protocol class. To simulate the internal congestion control schemes in the cluster-based router, we extended the Ipv4L3Protocol class to include the packet classifier, queue management and scheduling, and backward explicit congestion notification (BECN) functionalities. With the extended Ipv4L3Protocol instances installed in ns-3 nodes, we can simulate the behaviors of processing nodes in the cluster-based router and study the effects of different internal congestion control schemes in the cluster-based router.

In this section, we first introduce the IP forwarding path implemented in our simulation, then we describe the simulation setup, and finally we present and analyze the simulation results.

7.6.1 IP forwarding path

In ns-3, the IP forwarding path is implemented in a class called Ipv4L3Protocol. To make use of an Ipv4L3Protocol object, network devices should be attached to a node configured with an Ipv4L3Protocol instance. The Ipv4L3Protocol class provides the interface to add network interfaces with given network devices. The Ipv4L3Protocol instance registers the Receive callback method with the network devices. In this way, any IP packet received by the network devices will be forwarded to the Ipv4L3Protocol instance by the Receive callback method. Once the Ipv4L3Protocol instance gets the packet, the packet will go through the IP forwarding path as implemented.

The cluster-based router is composed of several routing nodes called processing nodes which can be ingress or egress nodes in terms of an individual flow. However, a network interface can only be an external interface or an internal interface. By external interface we mean a network interface that connects to the external network, while by internal interface we mean a network interface that connects to the internal network which connects all processing nodes together. Therefore, we extended the Ipv4L3Protocol class implemented in ns-3 to distinguish these two different kinds of network interfaces. A method called SetInternalDevice sets a network device to be internal. Otherwise, the network device is considered to be external by default. Once a network device is marked as internal, the Ipv4L3Protocol instance will register with it to receive BECN type packets in addition to the general IP packets.

Figure 7.3 describes the IP forwarding path in our simulation. Once a packet is received in the network device, the Receive method is called by the network device. According to the network device from which the packet is received, we can classify the packet to be an external traffic packet or an internal traffic packet.

If the packet is an external packet, after being looked up in the routing table with its destination IP, the packet will be pushed to a classifier if the next hop of the packet is another processing node in the cluster-based router. There it is switched to different queues according to the preset classification rules. Packets in these queues will be scheduled to be sent from the internal network interface. The scheduler controls the pace of putting packets into the output queue of the internal network device. If the output queue is available, the scheduler schedules a packet from upstream queues and puts the packet into the output queue. Whenever the internal device is not busy, a packet will be dequeued from the output queue for transmission. A packet being dequeued for transmission from the output queue will not be dropped by the network device since the packet is forwarded to the internal network device only when the internal network device is not busy. Without the output queue in the original implementation of Ipv4L3Protocol and without checking the status of the network device, a packet forwarded to the network device may be dropped due to the network device being busy.

If the packet is an internal packet which is received by the internal network device, or an external packet that will be transmitted out through the same processing node, the packet is switched to the output queue of the external network device where it will be transmitted out to the external network. A global monitoring process is scheduled periodically to check the status of the output queues of external network devices and send BECN packets to the other processing nodes if necessary.

If a BECN packet (only from internal network device) is received by a processing node, the parameters in the scheduler are adjusted, which updates the rate of packets being scheduled from each queue.

7.6.2 Simulation

Having all the functional components of a processing node in the cluster-based router ready, we can set up the simulator to simulate the internal congestion control schemes in the cluster-based router.

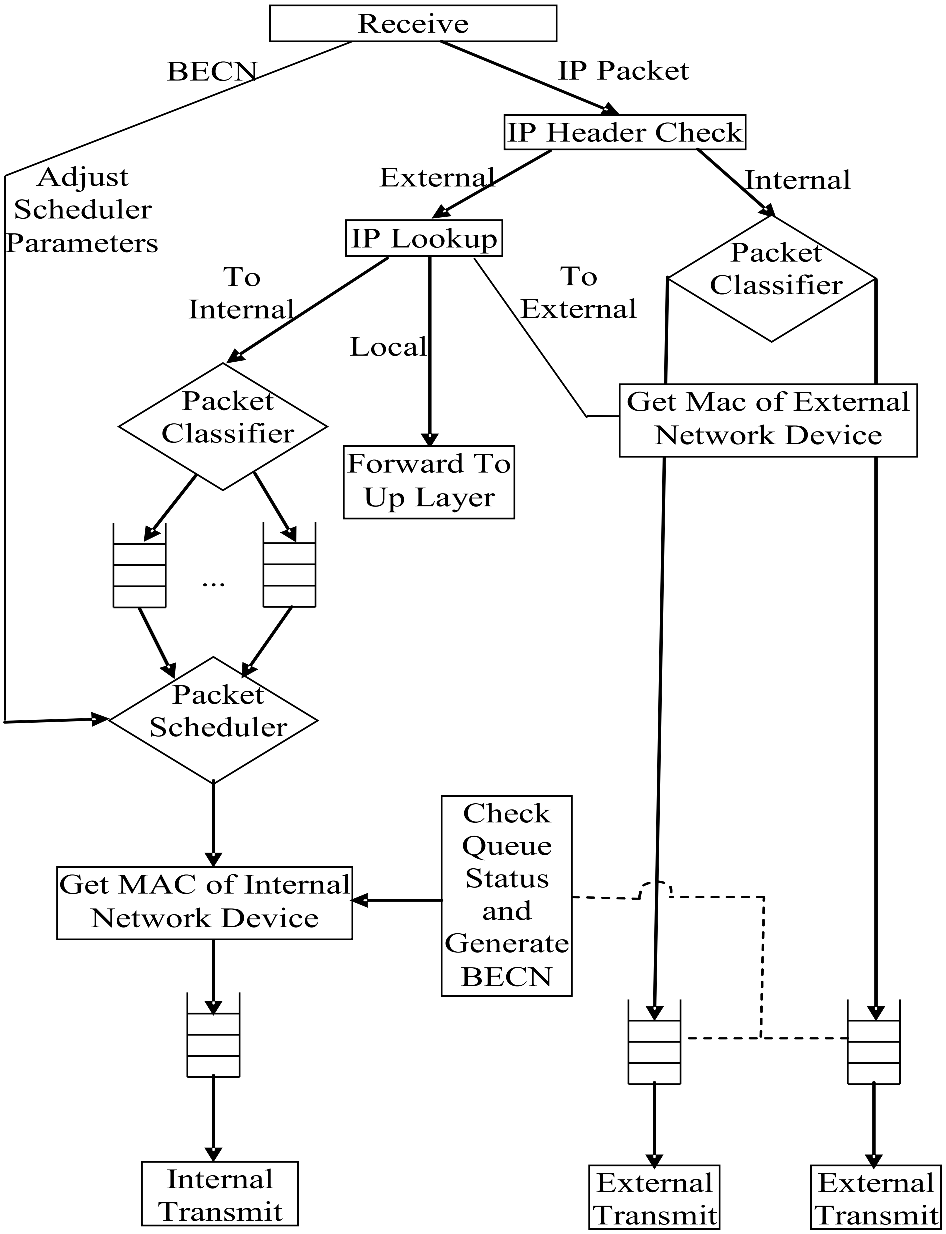


Figure 7.3: IP forwarding path in simulation

Cluster-based Router

First of all, we set up the cluster-based router. We extend the Ipv4L3Protocol class in ns-3 to include all of the basic functional components of processing nodes in the cluster-based router. We configure the nodes installed with the extended Ipv4L3Protocol instance to make them behave like processing nodes. Then we set up a CSMA (Carrier Sense Multiple Access) link with 10G data rate and 1ns delay to connect all the processing nodes together to simulate the InfiniBand internal network. The network interfaces of this CSMA link on the processing nodes are configured to be internal interfaces.

Simulating the InfiniBand internal network of our real prototype with a CSMA link is an approximate, but very useful choice. Firstly, due to the capacity difference between the external network and internal network, the internal network will never be a bottleneck in our simulation. Secondly, in the real system, there are statistical factors which can not be captured by simulating a deterministic switch fabric, and we have not as yet characterized the Infiniband fabric. The collision detection and exponential backoff algorithms implemented in CSMA bring a statistical flavor to the behavior of the fabric that makes the simulation more realistic.

The processing nodes also connect to the external network via Ethernet interfaces. We configure one external interface for each processing node.

After setting up the network interfaces and devices, we configure the extended components in the Ipv4L3protocol instance: we initialize the queues for each class of traffic, configure the classification rules, set the global time interval to schedule packets from output queues of network devices for transmission, set the congestion control interval, set the congestion control scheme and corresponding control parameters, and set the nodes to be processing nodes.

Simulation environment

Now we have a cluster-based router ready for processing packets. To study its behavior, traffic sources and destinations are needed. The goal of this simulation is to study the behavior of congestion control, so we configure three traffic sources and one traffic destination.

Four general nodes installed with TCP/IP network stack are created and connected to the cluster-based router processing nodes, with one general ns-3 node connecting to one processing node. On three of them, we install ns-3 on/off traffic generator applications to generate traffic flow with specified traffic pattern, and the other node is installed with ns-3 traffic sink application.

The patterns of the traffic flows generated on the three traffic source nodes are configured with the traffic sink node as the destination. In this setup, three ingress processing nodes of the cluster-based router receive packets from external networks and forward the packets to the other egress processing node which forwards the packets to the traffic sink node in the external network. By configuring the traffic rates and patterns on the traffic sources, we are able to simulate different congestion states and study the behaviors of different congestion control schemes.

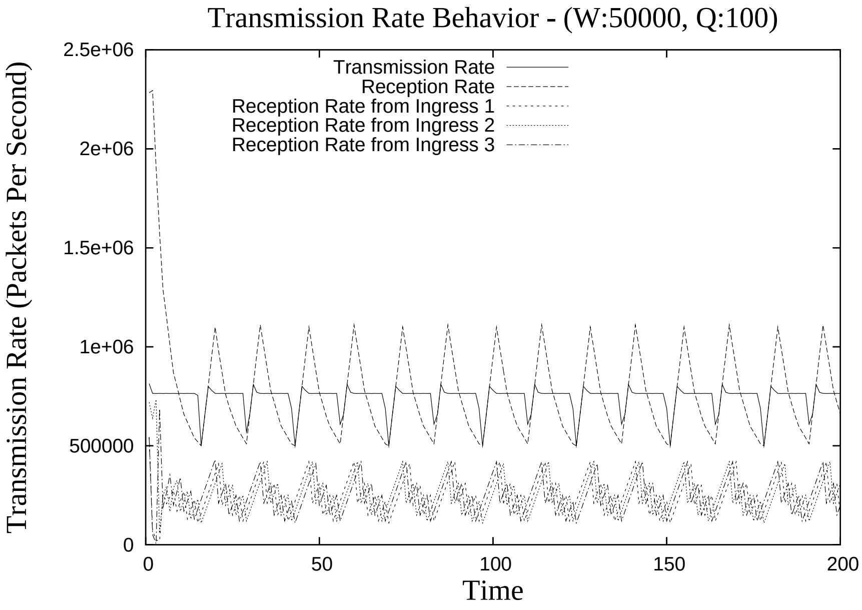


Figure 7.4: AIMD scheme transmission rate behavior

Since small packets stress the router more than large packets at the same aggregate bit rate because more packets have to be handled per second, we use 64B packets in our simulation.

Simulation data collection

To study different congestion control schemes including the throughput and convergence of the system, detailed system state information is required to facilitate the analysis process. In the implementation of the processing nodes, we added many status logging plugins to collect the status of the system during simulation. These include queue lengths, receiving and transmitting rates of network devices, and source distributions of received and transmitted packets.

7.6.3 Simulation results and analysis

In our simulation, we simulate the AIMD congestion control scheme with $W = 50000$, $M = 0.5$ and $Q = 100$ and the optimal congestion control scheme with $K = 100000$, $R = 5 * 10^{11}$ and $f(q) = (q - 100)/100$. We measure the average transmission rate of the egress network interface and the average length of the egress queue which is used to buffer the packets that will be transmitted from the external network interface.

Figure 7.4 and Figure 7.5 give the transmission behaviors of AIMD and optimal congestion control schemes when the ingress nodes are all offered the same amount of traffic. We collect the reception and transmission rates of the egress node, and the reception rates on the egress node from

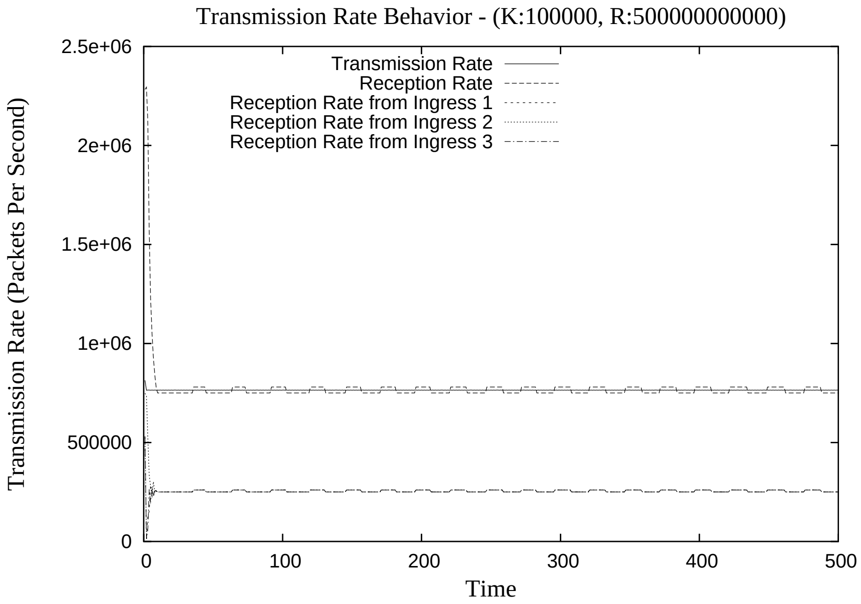


Figure 7.5: Optimal scheme transmission rate behavior

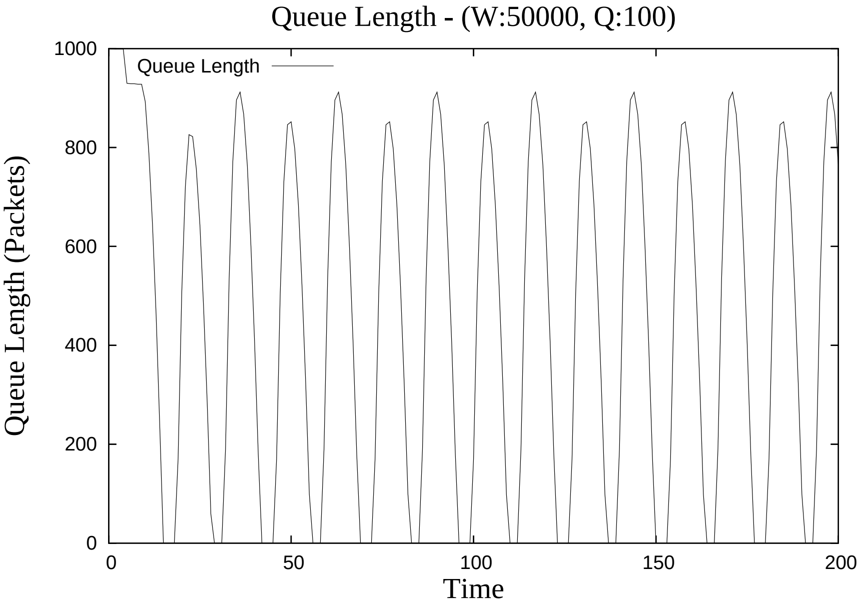


Figure 7.6: AIMD scheme queue behavior

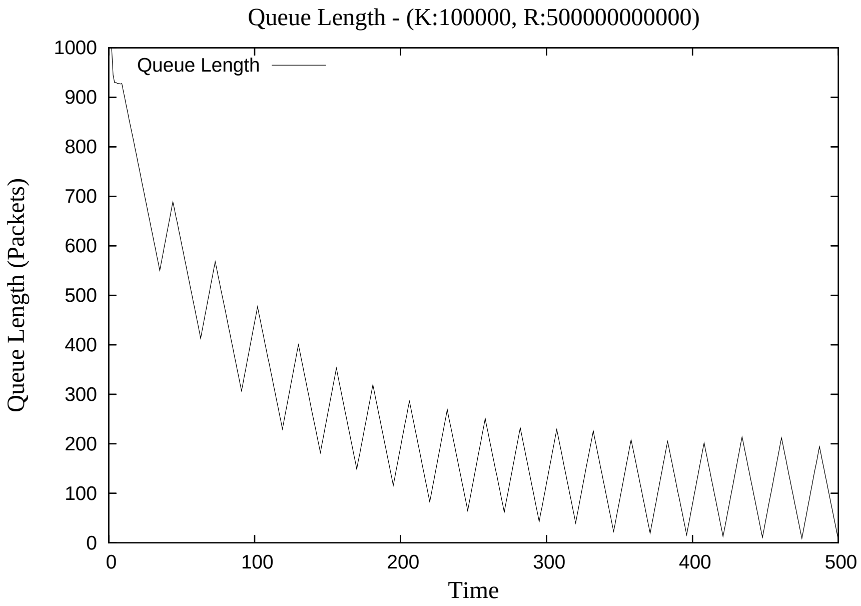


Figure 7.7: Optimal scheme queue behavior

each ingress node at an interval which equals the control interval(1ms). Figure 7.6 and Figure 7.7 present the egress queue length of AIMD and optimal congestion control schemes.

For the AIMD scheme, due to the multiple points at which injection rate is decreased, the injection rates of the incoming flows fluctuate dramatically. This results in a fluctuation of reception rate on the egress node, and makes the egress queue jump from empty to full. As a consequence, there are jitters on the egress transmission rate due to the overflow and emptying of the egress queue.

For the optimal scheme, the injection rates of flows, reception rate on the egress node, and the transmission rate of the egress port respond quickly, and the total injection rates on the ingress nodes, reception and forwarding rates of the egress node stabilize around the optimal transmission rate with a small variance. The egress queue length gradually decreases to around 100, which is equal to the queue objective in the $f(q)$ used in the control.

For both AIMD and optimal schemes, the above figures show that flows from different ingress nodes are serviced at the same pace when they are offered at the same traffic rates. They are fair to all flows in this case. Given differing traffic rates, we find that both congestion control schemes are also fair to all flows. Figure 7.8 and Figure 7.9 present the behaviors of AIMD and optimal congestion control schemes when source 1 is given 500Mbits/second traffic, source 2 is given 50Mbits/second traffic, and source 3 is given 5Mbits/second traffic. We can see that source 3 is 100% injected into the internal network since it requests less than 1/3 of bandwidth, source 2 is also 100% injected into the internal network since it requests less than half of the remaining bandwidth, and source 3 is not

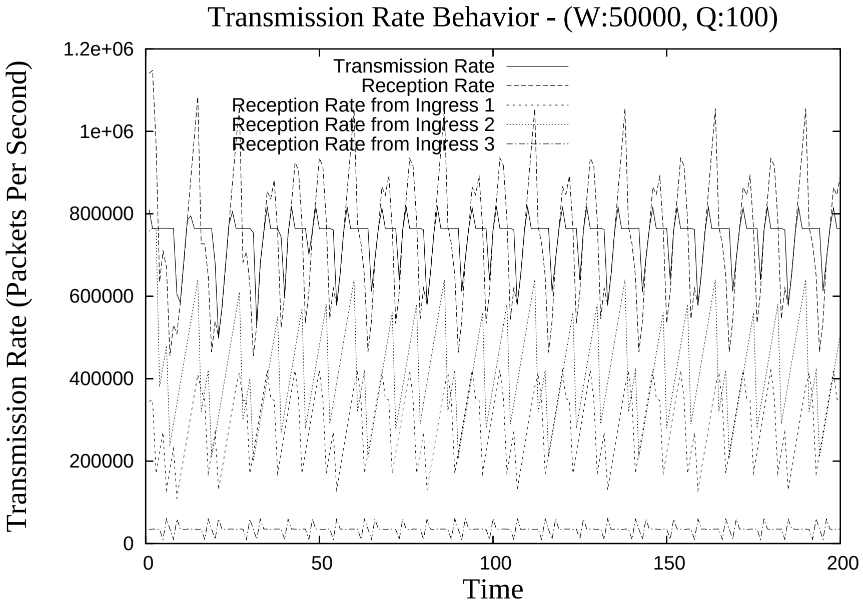


Figure 7.8: Fairness - AIMD scheme transmission rate behavior

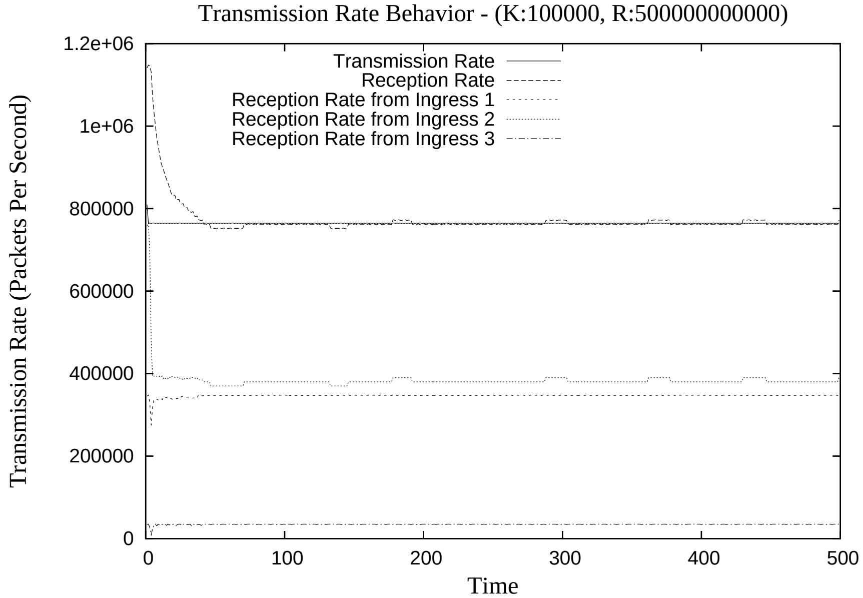


Figure 7.9: Fairness - Optimal scheme transmission rate behavior

100% injected into the internal network since it requests more than the remaining bandwidth the system can provide. Note that because of the fluctuations of the injection rates from ingress nodes in the AIMD scheme, the egress queue may overflow, so the packets from source 1 and 2 may not be 100% serviced due to the potential dropping of packets on the egress queue. In contrast, in the optimal control scheme, the injection rates are stable and there is almost no packet dropped on the egress queue, so the packets from source 1 and 2 are 100% serviced and transmitted from egress node.

Overall, with the given control parameter values, both AIMD and optimal schemes are fair to different incoming flows and effective to reduce the injection rates to the internal network when the router is under a heavy load. However, in the AIMD scheme, the injection rates of flows, forwarding rate of the router and the queue length of output queue fluctuate significantly. The optimal utility-based scheme provides a stable control with small fluctuation, which results in a higher average forwarding rate and small queue length and variance.

7.7 Evaluation in the real system

In this section, we present the results of evaluating our AIMD and optimal congestion control schemes on our cluster-based router prototype. We implemented the schemes as described in Figure 7.3 and in section 7.6.1.

7.7.1 Experiment

Our prototype router is composed of a cluster of four SunFire X4100 nodes interconnected by a Mellanox InfiniBand switch. Each SunFire node has two AMD Opteron 254 processors operating at 2.8GHz, 4 GB of registered DDR-400 SDRAM, two Intel E1000 Ethernet ports connected to a 100MHz PCI-X bus, and one Voltaire InfiniBand 4x PCI-X HCA installed in a 133MHz 64-bit PCI-X slot. A Mellanox MTS2400 24-port Modular InfiniBand Switch System that can support 10 Gbps switching rate at each port is connected to the HCA on each node. Packets are generated by a commercial traffic analyzer and sent to the router external Ethernet ports for forwarding. Packets that are successfully forwarded through the router return to the traffic analyzer via the router Ethernet ports.

In order to evaluate the effectiveness of our proposed optimal congestion control scheme and the AIMD scheme, we compare the behavior of the cluster-based router first with, and then without, internal congestion control. The traffic analyzer (with four ports) is configured to offer Ethernet traffic to three of the nodes (one port each) with the destination being a subnet reached via the fourth node. We increase the offered load until the egress port is heavily overloaded - the offered load is over three times more than the maximum load the egress port can forward. As well as acting as a traffic source and sink, the network analyzer is used to monitor the rates at which Ethernet

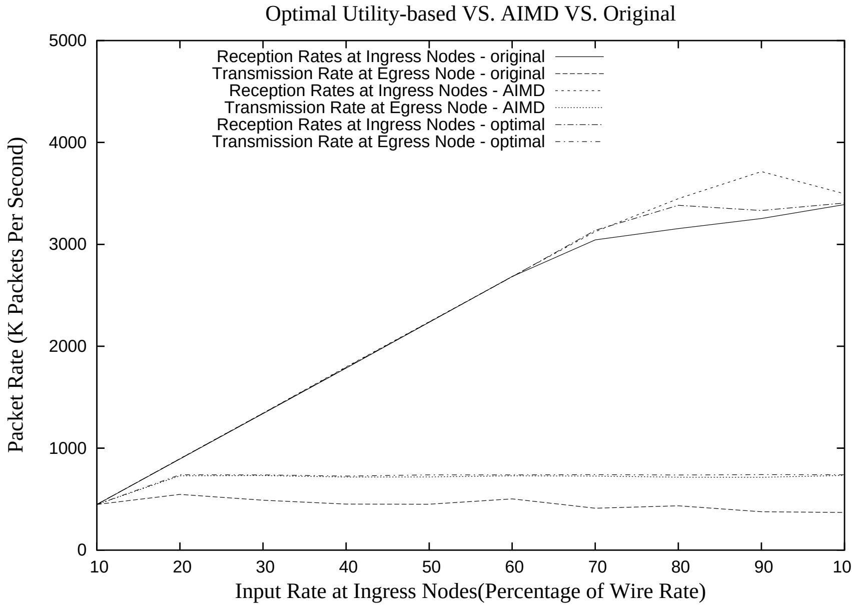


Figure 7.10: Transmission rate comparison

packets are sent to each ingress port and the rate at which the egress port returns packets to the traffic analyzer. As in the simulation we use 64B packets in our tests.

7.7.2 Measurements

We first compare the transmission rates on the egress node with and without the congestion control schemes applied when all the ingress nodes are fed at the same rate. As plotted in Figure 7.10, in the original situation, with increasing offered traffic on the ingress nodes, the ingress nodes inject more traffic into the internal network, which causes the transmission rate on the egress node to decrease after the offered load exceeds the point of ”congestion collapse”. With congestion control schemes, the ingress nodes only inject the packets which can be processed on the egress node to avoid overwhelming the internal network, the internal network port and the external port on the egress node. This results in a stable transmission rate on the egress node. The optimal congestion control scheme behaves somewhat better than the AIMD scheme. In this figure, the injection rate lines overlap with the egress transmission rate lines for the cases with congestion control schemes applied. The injection rate line overlaps the reception rate line on the ingress nodes for the original case without any congestion control scheme applied.

Figure 7.11 presents the average queue length and standard deviation of egress queue lengths with and without congestion control schemes. It shows that both optimal and AIMD schemes are effective in reducing the congestion on the egress node, which helps to keep the output queue smaller.

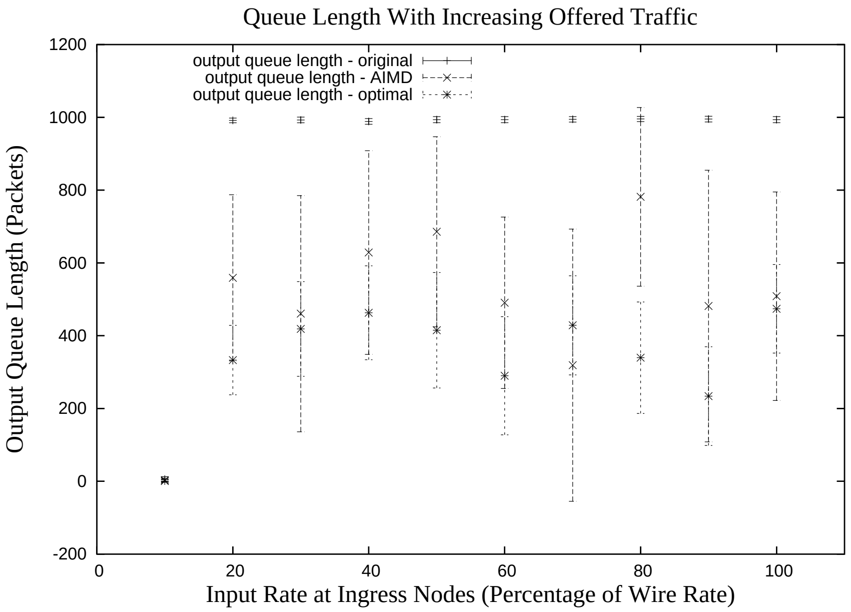


Figure 7.11: Queue variance comparison

Compared to the AIMD scheme, optimal congestion control maintains smaller average egress queue length and standard deviation of queue length on the egress node.

We also plot the fairness between ingress nodes when the ingress nodes are offered different traffic loads in the case that the congestion control schemes are in place. Table 7.1 gives the packet reception and injection rates at each ingress node as a function of offered loads. In both cases with the AIMD and optimal congestion control schemes applied, if all ingress nodes are offered the same load, then they inject packets into the internal network at the same rate and get the same share of transmission capacity on the egress node. If one ingress node is offered less load than the other two, the more heavily loaded nodes are punished more than the lighter loaded node and all overloaded

Table 7.1: Fairness comparison (kpps)

Congestion Scheme	Offered Rate At Ingresses	Reception Rate At Ingresses	Injection Rate At Ingresses	Transmission Rate At Egress
Optimal	299-298-74	299-298-74	299-298-74	671
AIMD	300-298-75	300-299-75	300-299-75	674
Optimal	299-299-149	299-299-149	293-293-149	734
AIMD	299-299-149	299-299-149	281-283-149	734
Optimal	300-299-224	300-299-224	257-257-224	737
AIMD	299-299-224	299-299-224	256-255-216	723
Optimal	299-299-299	299-299-299	247-247-247	739
AIMD	300-299-299	300-299-299	240-241-256	728

ingress nodes still get an equal share until the more lightly loaded node is 100% serviced.

7.8 Conclusions

Congestion control is a universal problem in distributed systems. The specific characteristics of the forwarding path through our cluster-based router make internal congestion a potential problem.

In this paper, we propose an optimal utility-based congestion control scheme and compare it to an AIMD scheme. Both methods use BECN notification. We simulated both methods in ns-3 to compare their behaviors. Both are fair to different flows and efficient to reduce the injection rates of traffic to the internal network when the router is under a heavy load. The application of these two schemes on our cluster-based router shows that both schemes are effective in preventing offered traffic from causing congestion collapse. They achieve efficiency by reducing the waste of internal network bandwidth and CPU cycles by dropping overloaded packets early, and they allocate throughput fairly among different flows. The results are consistent with our ns-3 simulations.

However, in the AIMD scheme, the injection rates of flows, forwarding rate of the router and the queue length of output queue fluctuate significantly, while the optimal utility-based scheme provides a stable control with small fluctuation. This results in a higher average forwarding rate, lower forwarding delay and small queue length and variance.

Many internal or local network congestion control schemes have been proposed. However, Kumar's recommended transmission rate notification for ATM networks [10] requires explicit source rates on the link and the link utilization, which are expensive to monitor. In Pappu's distributed queueing algorithm [15], a port processor controls the Virtual Output Queues's transmission rates based on all the other VOQs' status from other port processors, which makes the computing overhead expensive. Similar to Newman's AIMD scheme [14], our optimal scheme has little computing overhead requiring little information from the others, but it has a better control than AIMD. Moreover, the optimal scheme is based on the mathematical definition of a utility function. The specific form of this function can be changed to suit a wide range of performance requirements.

7.9 Appendix - Experimental setup

The processing nodes' architecture is described in figure 5.11.

The connections between cluster-based router and traffic analyzer are described in figure 6.10.

7.10 Appendix - Optimal congestion control algorithm

The figures 7.12 and 7.13 give the optimal congestion control algorithms implemented in IBBECNPack and IBBECNUnpack elements as described in section 7.4. Compared to algorithms presented in chapter 6, the transmission rate control algorithms implemented in IBBECNPack elements are same, but the price calculation algorithms implemented in IBBECNUnpack elements are different:

Optimal IBBECNPack pull (Input: None, Output: A InfiniBand Packet)
$BECNpacket$: A BECN packet to be sent or received
$Dest$: Destination address field in a BECN packet
$Queueid$: Queue id field in a BECN packet
$Local$: Local address
$MaxRate[i]$: The ith queue's maximum rate
$CurrentRate[i]$: The ith queue's current rate
K: Rate adjusting step size
W: Proportional weight
p: Congestion price

```
    WHILE The BECN queue is not empty DO
    REPEAT
        Get a BECN packet $BECNpacket$
        Get the destination address $Dest$, queue id $Queueid$, and congestion price $p$
        IF $Dest = Local$
            Return the BECN packet $BECNpacket$
        ELSE
            $MaxRate[Queueid] = CurrentRate[Queueid] + K * (W - CurrentRate[Queueid] * p$
            IF ($MaxRate[Queueid] < 0$)
                $MaxRate[Queueid] = 0$
            END IF
        END IF
    END WHILE
    Get enough Ethernet packets from upstream queues according to the corresponding queues' $MaxRate$
    Pack the Ethernet packets into one InfiniBand packet and return the InfiniBand packet
```

Figure 7.12: Optimal IBBECNPack pull

Optimal IBBECNUnpack push (Input: A InfiniBand Packet, Output: None)

$IBpacket$: The InfiniBand packet which is being processed
$T_{current}$: Current time
T_{last}: Last control time
$T_{interval}$: Control interval
p: Congestion price associated with a queue
R: Price adjusting step size
$Packet$: A Ethernet Packet
i: A Ethernet packet's index
q: A queue length
q_{old}: Previous queue length
q_o: The objective queue length
u: The degree that queue length would be taken account into price calculation

```
Get an InfiniBand packet $IBpacket$
**IF** ($T_{current} - T_{last} > T_{interval}$)
    **FOR** each output queue of this IBBECNUnpack element **DO**
    **REPEAT**
        Get this queue's queue length $q$
        $p+ = (q - q_{old} + (q - q_o) * u)/R$
        **IF** ($p < 0$)
            $p = 0$
        **END IF**
        Create a broadcast BECN packet with price $p$ and push it to the BECN queue
    **END FOR**
**END IF**
**IF** ($IBpacket$ is a BECN packet)
    Push $IBpacket$ to a downstream BECN queue according to the source address of the BECN packet
**ELSE**
    Unpack the packet $IBpacket$ to multiple Ethernet packets
    **FOR** each Ethernet packet $Packet$ and it's index $i$ **DO**
    **REPEAT**
        Push $Packet$ to the downstream $i$th queue
    **END FOR**
**END IF**
```

Figure 7.13: Optimal IBBECNUnpack push

AIMD IBBECNPack pull (Input: None, Output: A InfiniBand Packet)

$T_{current}$: Current time

T_{last}: Last control time

$T_{interval}$: Control interval

W: The increasing rate when there is no congestion notification received

$BECNpacket$: A BECN packet to be sent or received

$Dest$: Destination address field in a BECN packet

$Queueid$: Queue id field in a BECN packet

$Local$: Local address

$MaxRate[i]$: The ith queue's maximum rate

$CurrentRate[i]$: The ith queue's current rate

```
IF ($T_{current} - T_{last} > T_{interval}$)
    FOR each input queue of this IBBECNpack element DO
    REPEAT
        $MaxRate[i] += W$
    END FOR
END IF
WHILE The BECN queue is not empty DO
REPEAT
    Get a BECN packet $BECNpacket$
    Get the destination address $Dest$, queue id $Queueid$, and congestion price $p$
    IF $Dest = Local$
        Return the BECN packet $BECNpacket$
    ELSE
        $MaxRate[Queueid] = CurrentRate[Queueid]/2$
    END IF
END WHILE
Get enough Ethernet packets from upstream queues according to the corresponding queues' $MaxRate$
Pack the Ethernet packets into one InfiniBand packet and return the InfiniBand packet
```

Figure 7.14: AIMD IBBECNPack pull

7.11 Appendix - AIMD congestion control algorithm

The figures 7.14 and 7.15 give the AIMD congestion control algorithm implemented in IBBECNPack and IBBECNUnpack elements as described in section 7.5:

AIMD IBBECNUnpack push (Input: A InfiniBand Packet, Output: None)
$IBpacket$: The InfiniBand packet which is being processed
$T_{current}$: Current time
T_{last}: Last control time
$T_{interval}$: Control interval
q: A queue length
$Q_{threshold}$: The congestion queue length threshold
$Packet$: A Ethernet packet
i: A Ethernet packet's index

```
Get an InfiniBand packet $IBpacket$
IF ($T_{current} - T_{last} > T_{interval}$)
    FOR each output queue of this IBBECNUnpack element DO
    REPEAT
        Get this queue's queue length $q$
        IF ($q > Q_{threshold}$)
            Create a broadcast BECN packet and push it to the BECN queue
        END IF
    END FOR
END IF
IF ($IBpacket$ is a BECN packet)
    Push $IBpacket$ to a downstream BECN queue according to the source address of the BECN packet
ELSE
    Unpack the packet $IBpacket$ to multiple Ethernet packets
    FOR each Ethernet packet $Packet$ and it's index $i$ DO
    REPEAT
        Push $Packet$ to the downstream $i$th queue
    END FOR
END IF
```

Figure 7.15: AIMD IBBECNUnpack push

Bibliography

[1] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: active queue management. *IEEE Network*, 15:48–53, 2001.

[2] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[3] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[4] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[5] Pankaj Gupta. Scheduling in input queued switches: A survey. Technical report, Department of Computer Science, Stanford University, 1996.

[6] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988. ACM.

[7] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[8] Frank P. Kelly, Aman Kumar Maulloo, and David Tan. Rate control in communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[9] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[10] Anup Kumar, Arpana Maniar, and Adel S. Elmaghraby. A fair backward explicit congestion control scheme for ATM network. In *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, pages 452–457, Washington, DC, USA, 1999. IEEE Computer Society.

[11] Srisankar S. Kunniyur and R. Srikant. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Transactions on Networking*, 12:286–299, 2004.

[12] Shao Liu, Tamer Basar, and R. Srikant. Controlling the Internet: a survey and some new results. In *Proceedings of the 43nd IEEE Conference on Decision and Control*, pages 3048–3057, Maui, Hawaii USA, December 2003.

[13] Steven H. Low and David E. Lapsley. Optimization flow control - I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999.

[14] Peter Newman. Traffic management for ATM local area networks. *IEEE Communications Magazine*, 8:44–50, 1994.

[15] Prashanth Pappu, Jyoti Parwatikar, Jonathan Turner, and Ken Wong. Distributed queueing in scalable high performance routers. In *Proceedings of IEEE Infocom*, 2003.

[16] K. K. Ramakrishnan, Sally Floyd, and David Black. The addition of explicit congestion notification (ECN) to IP, 2001.

[17] R. Srikant. *The Mathematics of Internet Congestion Control*. Systems & Control: Foundations & Applications. Birkhauser, 2003.

[18] Jonathan S. Turner. Strong performance guarantees for asynchronous buffered crossbar scheduler. *IEEE/ACM Trans. Netw.*, 17(4):1017–1028, 2009.

[19] Yang. R. Yang and Simon S. Lam. General aimd congestion control. In *ICNP '00: Proceedings of the 2000 International Conference on Network Protocols*, page 187, Washington, DC, USA, 2000. IEEE Computer Society.

[20] Qinghua Ye and Mike H. MacGregor. Cluster-based IP router: Implementation and evaluation. In *IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006.

[21] Qinghua Ye and Mike H. MacGregor. Hardware bottleneck evaluation and analysis of a software PC-based router. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systerms*, pages 480–487, Edinburgh, UK, June 2008.

[22] Qinghua Ye and Mike H. MacGregor. Internal congestion control in the cluser-based router. In *Syposium on High Performance Switching and Routing*, pages 13–20, Shanghai, China, May 2008.

[23] Qinghua Ye and Mike H. MacGregor. Optimal utility-based internal congestion control in a cluster-based router. Technical Report TR09-12, Department of Computing Science, University of Alberta, Edmonton, Alberta, 2009.

Chapter 8

Conclusions

It is challenging to design a router architecture which supports both extensibility of packet processing services and scalability of packet forwarding capacity. Most commercial routers support a set of closed, predefined packet processing functions without offering extensibility. Software routers can be extended to support new packet processing functions, but their forwarding capacities are limited by underlying hardware.

To address the performance scalability of software routers and the function extensibility of commercial routers, we proposed a cluster-based router framework by deploying the software router on a cluster of commodity processors interconnected by a high-speed and low-latency network. Through evaluation, we demonstrated that this cluster-based router architecture can be scaled linearly to support higher packet forwarding rates.

As with other software routers, significant considerations have been given to reduce operating system overheads and mitigate hardware limitations. We implemented packet polling method in the Linux network driver to replace the software interrupt scheme, which reduces the context switch overhead in the kernel and facilitates packet buffer recycles to save CPU operations spent on buffer allocation/deallocation. To improve the performance scalability of the cluster-based router and reduce the internal latency of packets forwarded by the router, we developed a highly efficient communication layer called IBUDCOM (InfiniBand Unreliable Datagram COMmunication) above the InfiniBand network stack. We removed the overhead of software interrupts in the InfiniBand network stack and disabled the notification feature of InfiniBand Queue Pair, which allows the IBUDCOM to provide a polling interface to above router components. Also, IBUDCOM enables the packing of multiple Ethernet packets into one large InfiniBand packet, which reduces the average processing cost of IP packets.

We evaluated and analyzed the potential hardware bottlenecks of a PC-based router. We found that, by applying the polling extension of network driver and buffer recycling techniques, the CPU is more than able to process the minimum-size Ethernet packets received and transmitted at one Gigabit network port as well as multiple Gigabit network ports on the same PCI-X bus. Also, we observed that there is a nonlinear correlation between the reception and transmission capabilities of an individual port as well as ports on the same bus. All these helped us understand the potential bottlenecks in the cluster-based router.

Because of the nonlinear correlation between the reception and transmission capabilities of an individual port as well as ports on the same bus, the performance of a PC router or the processing nodes in the cluster-based router can be affected if the transmission and reception elements are scheduled without taking into account of this correlation. We proposed an adaptive scheduling mechanism based on system state information, which manages this adverse effect and enhances the transmission capabilities of NICs when they are overloaded at the cost of very little additional overhead.

We analyzed the potential internal congestion problem in the cluster-based router. To manage the internal congestion, we proposed two backward explicit congestion notification schemes - a novel additive increase, multiplicative decrease queue scheduling method, and an optimal utility-based congestion control scheme. In the BECN queue scheduling scheme, BECN messages are sent from the egress to the ingresses only when the congestion is detected (the queue length exceeds the congestion threshold) on the egress node, and the ingress nodes decrease the injection rates of flows to the congested egress queue by half if BECN messages are received, or increase the injection rates by a certain value. In the optimal utility-based scheme, BECN messages including the congestion price for each egress queue are sent from the egress to the ingresses at every control interval, and the ingress nodes adjust the injection rates of flows to the egress queues with the congestion prices received. Ns3 simulations and the applications of these two schemes to our cluster-based router showed that, when the system is under overload, both schemes prevent the ingress nodes from overwhelming the egress nodes by dropping packets early on the ingress nodes, which reduces the waste of internal network bandwidth and CPU cycles spent on the dropped packets, so as to maximize the forwarding rate. Compared to AIMD scheme, the optimal scheme provides a stable control with small fluctuations. This results in a higher average forwarding rate, reduced delay, and small queue length and variance.

With the application of Lyapunov's theorem, we also proved the stability of the corresponding continuous model of the optimal utility-based congestion control scheme. We observed that the stability condition of the discrete model is different from the continuous model by plotting the stability region of control parameters via numeric simulation. This observation was validated by our simulation of the scheme with ns-3 and experimental testing in our cluster-based router prototype. Both the simulation and experiment showed that the stability region predicted by the discrete model is reliable.

With this thesis, we can conclude that a relatively powerful, extensible, scalable router can be built from commodity components economically. The cluster-based router inherits extensibility from Click software router, and scalability from clustering.

8.1 Future directions

We have investigated the feasibility of the cluster-based router prototype and addressed several key issues in the cluster-based router. To increase the usability and scalability of this cluster-based router, there are numerous possibilities that can be investigated as future work:

1. Communication optimization: in our prototype, we implemented the IBUDCOM layer for Click elements to interact with lower InfiniBand stack. By using packing and polling instead of event notification, we alleviated the overhead of software IRQs and improved the overall throughput especially for small packets. However, the interaction with the InfiniBand devices is still much more expensive than that with other network devices due to the overhead of the InfiniBand stack and hardware interrupt handling scheme.

   To improve the efficiency of the communication software in our cluster-based router, the IBUDCOM layer can be optimized by incorporating the InfiniBand RDMA (Remote Direct Memory Access) operations into the IBUDCOM layer and utilizing multiple queue pairs. In addition to the channel communication semantics (send/receive), the InfiniBand also provides memory communication semantics such as RDMA write and RDMA read. Compared to the send/receive channel semantics, RDMA operations have the advantages that they are one-sided operations and do not involve the remote process in the communication, so they have less software overhead at the other side.

2. Support of high rate uplinks: with the evolution of Internet, edge routers in large enterprises or ISPs are expected to have 10Gb or higher uplinks to the core Internet WAN. To make the cluster-based router applicable in these environments, it is important for it to support high rate uplinks. This imposes a challenge for the processing nodes in terms of CPU and bus capacities.

3. Impact of new hardware: commodity hardware is evolving quickly all the time. With the increasing performance of PCI-Express, HyperTransport, and InfiniBand connection fabric, and the popularization of multi-core CPU technology, the performance bottleneck can change in PC servers with different architectures. The multi-core CPU also places a new challenge on the scheduling of router tasks such as the localization of computing.

4. Control Plane: this thesis addresses several issues of data plane (or packet forwarding path) of the cluster-based router, which focuses on the increasing of forwarding performance and decreasing of forwarding latency. To increase the usability of the cluster-based router, a lot work on the control plane should be done. This includes the computation and updating of routing tables, processing of routing protocols, management and deployment of router components.