A STUDY ON CONSTRUCTING DATA PATHS IN NETWORK PROCESSOR-BASED ROUTERS

Sreedhar Pasumarthi
Senior lecturer, I/c HOD IN CCN
Sir CRR Polytechnic College
sreedhar36@gmail.com

Kiranmayi Akkabattula
Lecturer in computer engineering
Sir CRR Polytechnic College
kiranmayi1301@gmail.com

Tejaswini kothapalli
Lecturer in computer engineering
Sir CRR Polytechnic College
tejakothapalli@gmail.com

Abstract

There is developing revenue in network processor advances fit for handling parcels at line rates. In the next generation of high-speed router and switch architectures, network processors are likely to replace the Application Specific Integrated Circuits (ASICs) that are currently utilized in routers. NetBind, a high-performance, adaptable, and scalable binding tool for dynamically constructing data paths in network processor-based routers, is the subject of our design, implementation, and evaluation in this paper. NetBind's methodology strikes a balance between the need to process and forward packets at line speeds and the programmability of the network. One of the most important features of next-generation Internet architectures is the functionality of routers for custom packet processing. As an abstraction for describing, composing, and deploying end-to-end connections with custom communication features, network services have been proposed. We present a novel hardware architecture for high-performance processing of such network services in the data path.

Keywords: *Network processor, Binding, Service Creation, Programmable Networks.*

Introduction

Recently, there has been a growing interest in network processor technologies that can support software-based implementations of the critical path while processing packets at high speeds. We believe that introducing programmability into routers based on network processors is a significant area of research that has not yet been fully explored. The difficulty arises from the fact that network processor-based routers must simultaneously support modular and extensible data paths and forward minimum size packets at line rates. Typically, the higher the line rate supported by a network processor-based router the smaller the set of instructions that can be executed in the critical path. This poses significant challenges when introducing programmability into the critical path. Data path modularity and extensibility requires the dynamic binding between independently developed packet processing components.

We believe that traditional techniques for realizing dynamic binding cannot be applied to network processors because these techniques introduce considerable overhead in terms of additional instructions in the critical path. In this paper, we present a methodology and binding tool for constructing programmable data paths in network processor-based routers without introducing overhead in the critical path. We have developed NetBind, a binding tool that can create packet processing pipelines through the dynamic binding of small pieces of machine language code. In NetBind, data path components export "symbols", which are used during the binding process. A NetBind "binder" modifies the machine language code of components at runtime. As a result,

components can be combined into a single piece of code without any problems. While the current implementation of the tool is focused on the Intel IXP1200 network processor, the design of NetBind is guided by a set of general principles that make our method applicable to a class of network processors.

Future work will focus on porting the NetBind tool to other network processors. We have used NetBind to create data paths for virtual routers sharing the resources of the same IXP1200 network processor. Specifically, we have created an IPv4 data path and used this data path to evaluate NetBind's performance. In the most recent release of the IXP1200 Software Development Kit (IXA SDK 2.0), Intel has provided the MicroACE system for modularizing packet processing functions. In this paper, we evaluate NetBind and MicroACE and identify potential pros and cons of each scheme.

Network Processors

A common practice when designing and building high performance routers is to implement the fast path using Application Specific Integrated Circuits (ASICs) in order to avoid the performance cost of software implementations. ASICs are usually developed and tested using Field Programmable Gate Arrays, which are arrays of reconfigurable logic. Network processors represent an alternative approach to ASICs and FPGAs, where multiple processing units, offer dedicated computational support for parallel packet processing. Processing units often have their own on-chip instruction and data stores. In some network processor architectures, processing units are multithreaded. Hardware threads usually have a separate program counter and manage a separate set of state variables. However, each thread shares an arithmetic logic unit and register space. Network processors do not only employ parallelism in the execution of the packet processing code, rather, they also support common networking functions realized in hardware, Network processors typically consume less power than FPGAs and are more programmable than ASICs.

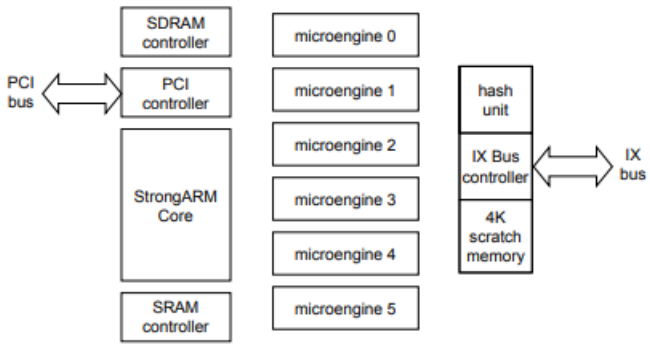


Figure 1: Internal Architecture of IXP1200

The IXP1200 Network Processor

Our study on the construction of modular data paths is focused on the Intel IXP1200 network processor. The IXP1200 incorporates seven RISC CPUs, a proprietary bus (IX bus) controller, a PCI controller, control units of accessing off-chip SRAM and SDRAM memory chips, and an on-chip scratch memory. The internal architecture of the IXP1200 is illustrated in Figure 1. In what follows, we provide an overview of the IXP1200 architecture. The information presented here is essential for understanding the design and implementation of NetBind.

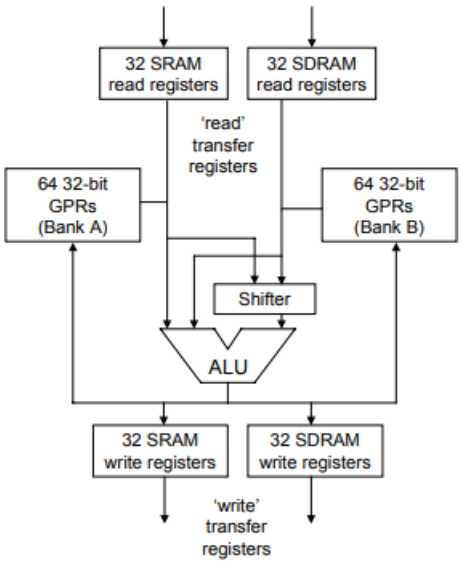


Figure 2: Microengine Registers

Each microengine incorporates 256 32-bit registers. Among these registers, 128 registers are General Purpose Registers (GPRs) and 128 are memory transfer registers. The register space of each microengine is shown in Figure 2. Registers are used in the following manner. GPRs are divided into two banks (i.e., banks A and B, shown in Figure 2) of 64 registers each. Each GPR can be addressed in a "context relative" or "absolute" addressing mode. By context relative mode, we mean that the address of a register is meaningful to a particular context only. Each context can access one fourth of the GPR space in the context relative addressing mode. By absolute mode, we mean that the address of a register is meaningful to all contexts.

Dynamic Binding Issues

There are many different techniques for introducing new services into software-based routers. At one end of the spectrum, the code that implements a new service can be written in a high level, platform-independent programming language (e.g., Java) and compiled at runtime producing optimized code for some specific network hardware. In contrast, data paths can be composed from packet processing components. Components can be developed independently from each other, creating associations at run time. Although algorithmic components must be developed and tested in advance, this dynamic binding approach speeds up the development and installation of new services. The design of a dynamic binding system for network processor-based routers is the topic of our subsequent discussion. The most significant problems that arise when creating a binding system for network processor-based routers can be summed up as:

- Headroom limitations;
- Register space and state management;
- Choice of the binding method;
- Data path isolation and admission control;
- Processor handoffs;
- Instruction store limitations;
- Complexity of the binding algorithm.

Headroom Limitations

Line rate forwarding of minimum size packets (64 bytes) is an important design requirement for routers. Routers that can advance least size bundles at line rates are commonly more strong against refusal of-administration assaults, for instance. Line rate sending doesn't mean zero lining. Instead, it indicates that the routers' output links can be fully utilized when forward minimum size packets. A necessary condition for achieving line rate forwarding is that the amount of time dedicated to the processing of a minimum size packet does not exceed the packet's transmission or reception times, assuming a single queuing stage. If multiple queuing stages exist in the data path, then the processing time associated with each stage

should not exceed the packet's transmission or reception times. Given that the line speeds at which network processor-based routers operate are high, (e.g., in the order of hundreds of Mbps or Gbps), the amount of instructions that can be executed in the critical path is typically small, ranging between some tens to hundreds of instructions. The amount of instructions that can be executed in the critical path without violating the line rate forwarding requirement is often called headroom.

Register Space and State Management

The exchange of information between data path components typically incurs some communication cost. The performance of a modular data path depends on the manner in which the components of the data path exchange parameters between each other. Data transfer through registers is faster and more efficient than memory operations. Therefore, a well-designed binding tool should manage the register space of a network processor system such that the local and global state information is exchanged between components as efficiently as possible. The number of parameters which are used by a component determines the number of registers a component needs to access. If this number is smaller than the number of registers allocated to a component, the entire parameter set can be stored in registers for fast access. The placement of some component state in memory impacts the data path's performance. Although modern network processors support a large number of registers, register sets are still small in comparison to the amount of data that needs to be managed by components. Transfer through memory is necessary when components are executed by a separate set of hardware contexts and processing units. A typical case is when queuing components store packets into memory, and a scheduler accesses the queues to select the next packet for transmission into the network.

Choice of the Binding Method

Apart from the manner in which parameters are exchanged between components, the choice of the binding technique significantly impacts the performance of a binding algorithm. There are three methods that can be used for combining components into modular data paths. The first method is to insert a small code stub that implements a dispatch thread of control into the data path code. The dispatch thread of control directs the program flow from one component to another based on some global binding state and on the parameters returned by each module. This method is costly. The minimum amount of state that needs to be checked before the execution of a new component is an identifier to the next module and a packet buffer handle exchanged between components. Checking this amount of state requires at least four compute cycles. If a data path is split between six components, then the total amount of overhead introduced in the critical path is twenty-four compute cycles which is a significant part of the network processor headroom in many different network processors. For example, in IXP network processors that target the OC-48 and OC-192 line rates, the headroom is equal to 57 compute cycles. In this case the binding overhead accounts for 42% of the headroom. The dispatch loop approach is more appropriate for static rather than dynamic binding and can impact the performance of the data path because of the overhead associated with the insertion of a dispatch code stub. An enhancement

on the first method for dynamic binding adds a small vector table into memory. The vector table contains the instruction store addresses where each component is located.

Data Path Isolation and Admission Control

To forward packets without disruption, data paths sharing the resources of the same network processor hardware need to be isolated. In addition, an admission control process needs to ensure that the resource requirements of data paths are met. Resource assignments can be controlled by a system-wide entity. Resources in network processors include bandwidth, hardware contexts, processing headroom, on-chip memory, register space, and instruction store space. One way to support isolation between data paths is to assign each data path to a separate processing unit, or a set of hardware contexts, and to make sure that each data path does not execute code that exceeds the network processor headroom. Determining the execution time of programs given some specific input is typically an intractable problem. However, it has been shown that packet processing components can have predictable execution times One solution to the problem is to allow code modules to carry their developer's estimation of the worst case execution time in their file headers. The time reported in each file's header should be trusted, and code modules should be authenticated using well-known cryptographic techniques. To determine the worst case execution time for components, developers can use reasonable upper bounds for the time it takes to complete packet processing operations.

Processor Handoffs

Sometimes the footprints of data paths can be too large so that they cannot be placed in the same instruction store. In this case, the execution of the data path code has to be split across two, or more processing units. Another case arises when multiple data paths are supported in the same processor. In this case, the available instruction store space of processing units may be limited. As a result, the components of a new data path may need to be distributed across multiple instruction stores. Third, the execution of a data path may be split across multiple processing units when "software pipelining" [21] is employed for increasing the packet rate that can be achieved in a network processor architecture. Software pipelining is a technique that divides the functionality of a data path into several stages. Pipeline stages can potentially run in different processing units. Multiple stages can be executed at the same time forwarding the packets of different data flows. We call the transfer of execution from one processing unit to another (that takes place when a packet is being processed), "processor handoff". Processor handoffs impact the performance of data paths and need to be taken into account by the binding system. The performance of processor handoffs depends on the type of memory used for communication between processing units. A dynamic binding system should try to minimize the probability of having high latency processor handoffs in the critical path.

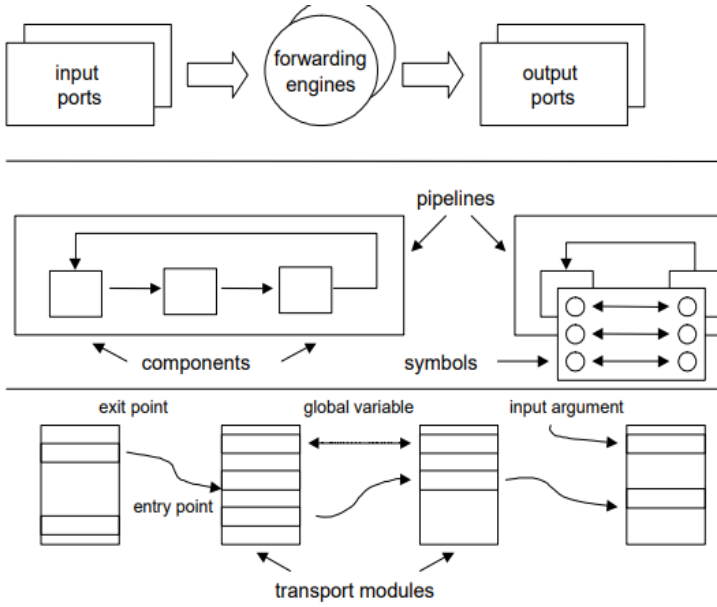
Instruction Store Limitations

Instruction store limitations represent a constraint on the number of data paths or processing functions that can be simultaneously executed in the same network processor. A solution to this problem would be to have the binding

system fetch code from off-chip memory units into instruction stores on an on-demand basis. This solution, however, can significantly impact the performance of the critical path because of the overhead associated with accessing memory units.

Complexity of the Binding Algorithm

The last consideration for designing a dynamic binding system is the complexity of the binding algorithm. In many cases, the complexity of a binding algorithm affects the time scales over which the binding algorithm can be applied. A complex binding algorithm needs time to execute, and is typically, not suitable for applications that require fast data path composition. Keeping the binding algorithm simple while producing high performance data paths is an important design requirement for a good binding system. Applications that require real-time binding include classification, forwarding and traffic management. The performance of such data path algorithms depends on the properties of classification databases, routing tables and packet scheduling configurations. These properties typically change at run time calling for advanced service creation environments for programming the data path efficiently. A dynamic binding system should ideally support a wide range of packet processing applications ranging from the creation of virtual networks over slow time scales to the fast creation of customized data paths, after disasters occur. Disasters typically result in rapid changes on the input traffic characteristics and topologies of communication networks. To accommodate increased traffic demands or to reroute traffic to alternate links once some part of the communication infrastructure is physically damaged, communication networks need to be highly adaptive, calling for new techniques and software methodologies for rapid service creation.



NETBIND SYSTEM

NetBind is a binding tool we have developed that offers dynamic binding support for the IXP1200 network processor and consists of a set of libraries that can modify IXP1200 instructions, create processing pipelines or perform higher-level operations (e.g., data path admission control). Components are written in machine language code called "microcode" and grouped into processing pipelines that execute in the IXP1200 microengines. As part of the Genesis Project, we use the NetBind tool to dynamically create virtual routers for spawning networks. These virtual routers share the resources of the same IXP1200 chip. In what follows, we present the design and implementation of NetBind and discuss support for virtual routers.

Data Path Specification Hierarchy

Before creating data paths, NetBind captures their structure and building blocks in a set of executable profiling scripts. NetBind uses multiple specification levels to capture the building blocks of packet forwarding services and their interaction. NetBind uses multiple specification levels in order to offer the programmer the flexibility to select the amount of information that can be present in data path profiling scripts. Some profiling scripts are generic, and thus applicable to any hardware architecture. Some other profiling scripts can be specific to network processor architectures, potentially describing timing and concurrency information associated with components. A third group of scripts can be specific to a particular chip such as the IXP1200 network processor. Figure III illustrates the different ways data paths are profiled in NetBind. First, a virtual router specification can be applied to any hardware architecture. The virtual router specification is generic and can be applied to many different types of programmable routers. The purpose of the virtual router specification is to capture the composition of a router in terms of its constituent building blocks and their interaction, without specifying the method which is used for component binding. Programmable routers can be constructed using a variety of programming techniques ranging from higher level programming languages (e.g., Java) to hardware plugins based of FPGA technologies. The virtual router specification can be used for service creation in a heterogeneous infrastructure of programmable routers of many different types. The virtual router specification describes a virtual router as a set of input ports, output ports and forwarding engines. The components that comprise ports and engines are listed, but no additional information is provided regarding the contexts that execute the components and the way components create associations with each other. There is no information about timing and concurrency in this specification.

Experimental Environment

Hardware Environment

To evaluate NetBind, we have set up an experimental environment consisting of three "Bridal Veil" development boards, interconnecting desktop and notebook computers in our lab. Bridal Veil [17] is an IXP1200 network processor development board running at 200 MHz and using 256 Mbytes of SDRAM off-chip memory. The Bridal Veil board is a PCI card that can be connected to a PC running Linux. The host processor of the PC and the IXP1200 unit can communicate over the PCI bus. The PC serves as a file server and boot server for the Bridal Veil system. PCI bus network drivers are provided for the PC and for the embedded ARM version of Linux running on the StrongARM Core processor. In our experimental environment, each Bridal Veil card is plugged into a different PC. In each card, the IXP1200 chip is connected to four fast Ethernet ports that can receive or transmit packets at 100 Mbps.

Software Environment

Initially NetBind was developed for an IXP1200 Ethernet evaluation board running at 166 MHz. To evaluate NetBind we ported the NetBind code into the Bridal Veil system where we could also execute MicroACE. MicroACE is a systems architecture provided by Intel for composing modular data paths and network services in network processors and on IXP1200 in particular. The

MicroACE adopts a static binding approach to the development of modular data paths based on the insertion of a "dispatch loop" code stub in the critical path. The dispatch loop is provided by the programmer and directs the program flow through the components of a programmable processing pipeline. MicroACE is a complex system that can be used for programming the microengines and the Strong ARM Core as well. Describing MicroACE in detail is beyond the scope of this paper. The information provided in this section is for the purpose of understanding our evaluation and the potential advantages or disadvantages of using MicroACE and NetBind.

MicroACE overview

MicroACE is an extension of the ACE [17] framework that can execute in the microengines of IXP1200. In MicroACE, an application defines the flow of packets among software components called "ACEs" by binding packet destinations called "targets" to other ACEs in a processing pipeline. A series of concatenated ACEs form a processing pipeline. A MicroACE consists of a "microblock" and a "core component". A microblock is a microcode component running in the microengines of IXP1200. One or more microblocks can be combined into a single microengine image called "microblock group". A core component runs on the StrongARM Core processor. Each core component is the control plane counterpart of a microblock running in the microengines. The core component handles exception, control and management packets.

Binding in MicroACE

The binding between microblocks in the ACE framework is static and takes place offline. The targets of a microblock are fixed and cannot be changed at run time. For each microengine, a microcode "dispatch loop" is provided by the programmer, which initializes a microcode group and a pipeline graph. The size of a microblock group is limited by the size of the instruction store where the microcode group is placed. Before a microblock is executed, some global binding state needs to be examined. The global binding state consists of two variables. A "dl_next_block" variable holds an integer identifying the next block to be executed, whereas a "dl_buffer_handle" variable stores a pointer to a packet buffer exchanged between components. Specialized "source" and "sink" microblocks can send or receive packets to/from the network.

Dynamic Binding Analysis

MicroACE and NetBind have some similarities in their design and realization but also differences. MicroACE offers much higher programming flexibility allowing the programmer to construct processing pipelines in the StrongARM Core and the microengines as well. NetBind on the other hand offers a set of libraries for the microengines only. MicroACE supports static linking allowing programmers to use registers according to their own preferences. MicroACE does not impose any constraints on the number and purpose of registers used by components. NetBind on the other hand supports dynamic binding between components that can take place at run time. To support dynamic binding NetBind imposes a number of constraints on the purpose and number of registers used by components as discussed in Section III-A. The main difference between NetBind and MicroACE in terms of performance comes from the choice of the binding technique.

MicroACE follows a dispatch loop approach where some global binding state needs to be checked before each component is executed. In NetBind there is no explicit maintenance of global binding state. Components are associated with each other at run time through the modification of their microcode. The modification of microcode at run time, which is fundamental in our approach, poses a number of security challenges, which our research has not addressed as yet. Another problem with realizing dynamic binding in the IXP1200 network processor is that microengines need to temporarily terminate their execution when data paths are created or modified.

Quantitative Analysis

We evaluated the performance of the IPv4 data path discussed in Section III-C when no binding is performed (i.e., the data path is monolithic) and when the data path is created using NetBind and MicroACE. We also implemented a simple binding tool based on the third alternative technique of the vector table discussed in Section II. We used this tool in our experiments in order to contrast the three binding techniques, presented in Section II, against each other in a quantitative manner. To analyze and compare the performance of different data paths, we executed these data paths on Intel's "transactor" simulation environment and on the IXP1200 Bridal Veil cards. Table 1 illustrates the additional instructions that are inserted in the data path for each binding technique:

Table 1: Binding Instructions in the Data Path

binding technique	per component binding instructions			
	register operation	**conditional branch**	**unconditional branch**	**memory (scratch) transfer**
NetBind	**1**	**1**		
dispatch loop	**2**	**1**	**2**	
vector table			**1**	**1**

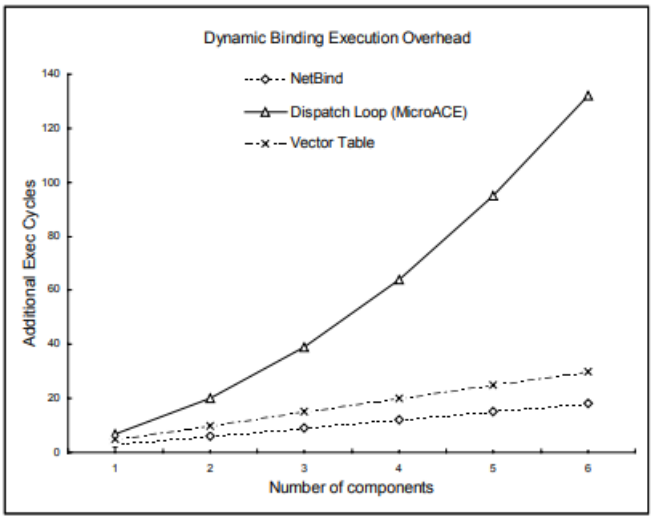


Figure: Dynamic Binding Overhead

We wrote a "worst case" but also generic dispatch loop for the MicroACE data path. The dispatch loop we wrote includes a set of comparisons that determine the next module to be executed on the basis of the value of the "dl_next-block" global variable. The loop is repeated for each component. We have added 5 instructions for each component in the dispatch loop: (i) an "alu" instruction to set the "dl_next_block" variable to an appropriate value (ii) an unconditional branch to jump to the beginning of the dispatch loop (iii) an "alu" instruction to check the value of the "dl_next_block" variable (iv) a

conditional branch and an unconditional branch to jump to the appropriate next module in the processing pipeline.

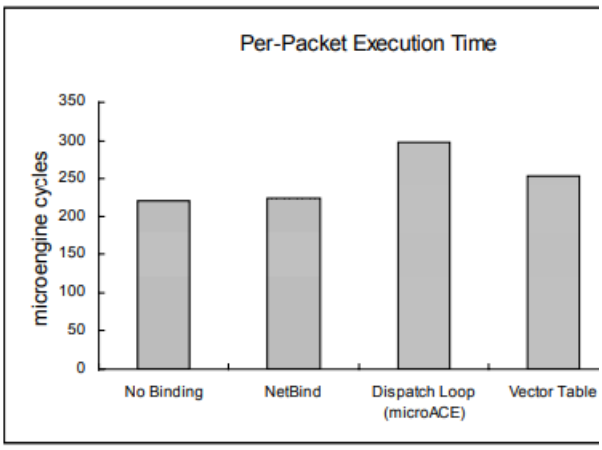


Figure: Per-Packet Execution Time

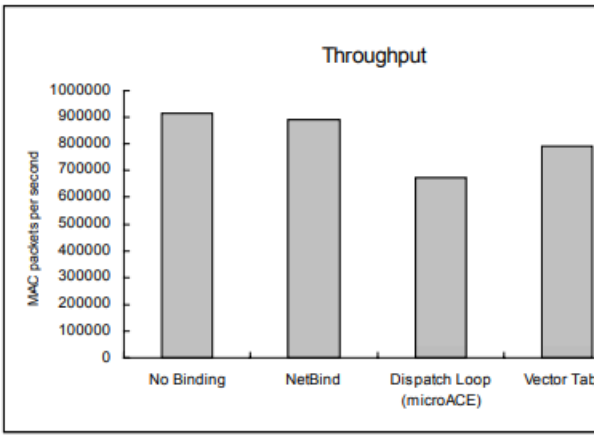


Figure: Packet Processing Throughput

The vector table binding technique works as follows: At the initialization stage, we retrieve the entry point locations using "load_addr" instructions. The vector is then save into the 4K scratch memory of IXP1200. For each component, we insert a scratch "read" instruction to retrieve the entry point from the vector table and a "rtn" instruction to jump into the entry point of the next module. The performance of vector table scheme is heavily dependent on the speed of memory access. If the vector for the next component in the pipeline can be retrieved in advance, the overhead of binding can be reduced drastically to 5 cycles per binding. The advantage of having a multithreaded network processor is that memory access latencies can be hidden if the processor switches context when performing time consuming memory transfer operations. Figure 9 shows additional execution cycles for each binding technique. From the figure, we observe that the dispatch loop binding technique introduces the largest overhead as expected, while the NetBind code morphing technique and the vector table technique demonstrate lesser overhead. NetBind demonstrates the best performance in terms of binding overhead adding only 18 execution cycles when connecting 6 modules to construct the IPv4 data path. Figures 10 and 11 show per-packet execution times and packet processing throughput for the three binding techniques and a monolithic data path.

Conclusion

In this paper, we have presented the design, implementation and evaluation of NetBind, a binding tool that can dynamically construct data paths in the IXP 1200 network processor. The methodology that underpins NetBind balances the flexibility of network programmability against the need to process and forward packets at line speeds. We have compared the performance of NetBind against Intel's MicroACE system and evaluated these two approaches in terms of their binding overhead. We proposed a binding technique that is optimized for network processor-based architectures, minimizing the binding overhead in the critical path, and, allowing network processors to forward minimum size packets at line rates. NetBind aims to balance the flexibility of network programmability against the need for high performance.

References

1. *Kounavis, M. E., Campbell A. T., Chou, S., Modoux F., Vicente, J., and Zhang H., (2001) "The Genesis Kernel: A Programming System for Spawning Network Architectures", IEEE Journal on Selected Areas in Communication, Vol. 19, No 3, pg. 511-526.*
2. *Campbell, A. T., Gormez, J., Kim, S., Valko, A., Wan, C., Turanyi, Z.,(2000) "Design Implementation, and Evaluation of Cellular IP", IEEE Personal Communications, vol. 7 No. 4, pg. 42-49.*
3. *Pier Luigi Ventre et al (2019) Segment Routing: a Comprehensive Survey of Research Activities, Standardization Efforts and Implementation Results, Submitted To Ieee Communications Surveys & Tutorials, arXiv:1904.03471v3 [cs.NI] 11.*
4. *Khalid Abualsaud et al (2008) Impact of MD5 authentication on routing traffic for the case of: EIGRP, RIPv2 and OSPF, Journal of Computer Science ISSN 1608-4217, 4(9), DOI:10.3844/jcssp.2008.721.728.*
5. *Siddesh Vishesh (2017) Simulation of Wireless BAN using Network Simulation Tool, International Journal of Advanced Research in Computer and Communication Engineering, ISSN: 2278-1021, Vol. 6, Issue 1, DOI 10.17148/IJARCCE.2017.6181.*
6. *Kailasam Selvaraj et al (2019) Design and Development of Virtual Local Area Network and Securing Network using Network Address Translation, International Journal of Recent Technology and Engineering, ISSN: 2277-3878, Volume-8 Issue-4S2.*
7. *Sanam. Nagendram et al (2019) Performance Evaluation of Wide Area Network using Cisco Packet Tracer, International Journal of Advanced Trends in Computer Science and Engineering, ISSN 2278-3091, Volume 8, No.6, https://doi.org/10.30534/ijatcse/2019/3886 2019.*
8. *Ashish Kumar (2017) Implementation of a Company Network Scenario Module by using Cisco Packet Tracer Simulation Software, Advances in Computer Science and Information Technology, SSN: 2393-9915; Volume 4, Issue 5, pp. 285-291, http://www.krishisanskriti.org/Publication. html.*
9. *Michel Bakni et al (2018) An Approach to Evaluate Network Simulators: An Experience with Packet Tracer, Revista Venezolana de Computación, ISSN: 2244-7040, Vol. 5, No. 1, pp. 29-36, http://www.svc.net.ve/revecom.*