A Hybrid Framework for Network Processor System Analysis

Patrick Crowley & Jean-Loup Baer
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350
{pcrowley, baer}@cs.washington.edu

Abstract

This paper introduces a modeling framework for network processing systems. The framework is composed of independent application, system and traffic models which describe router functionality, system resources/organization and packet traffic, respectively. The framework uses the Click Modular router to describe functionality. Click modules are mapped onto an object-based description of the system hardware and are profiled to determine maximum packet flow through the system and aggregate resource utilization for a given traffic model. This paper presents several modeling examples of uniprocessor and multiprocessor systems executing IPv4 routing and IPSec VPN encryption/decryption. Model-based performance estimates are compared to the measured performance of the real systems being modeled; the estimates are found to be accurate within 10%. The framework emphasizes ease of use, and permits a quick system analysis of existing or novel applications.

1. Introduction

Network processors provide flexible support for communications workloads while achieving high performance. Designing a network processor can involve the design and optimization of many component devices and subsystems, including: (multi)processors, memory systems, hardware assists, interconnects and I/O systems. Performance modeling can be an important early step in the design process. Models can be used to find feasible solutions which are then used to guide the design of detailed simulators or prototypes.

Analytical modeling is seldom applied in computer architecture research and development since the statistical assumptions required for such modeling poorly match the problem requirements seen in traditional computer architecture. Instead, software-based timing simulators are built and employed. Networking, however, is a better fit for analytical and queue-based models; their use in state-of-the-art research is routine and unquestioned. Network processor system design happens at the intersection of these two fields.

This observation, that analytical models are widely applicable in networking and less so in computer architecture, is a prime motivating factor in the design of the modeling framework presented here. The framework seeks an efficient balance between modeling and simulation: cycle-accurate simulation is used only when modeling would yield unsupportable inaccuracies (e.g., in a processor core), and modeling is employed everywhere else (e.g., traffic at a network interface). With this hybrid approach, accurate performance estimates are obtained with relatively little development or simulation time. This paper includes several modeling examples in which estimates are compared to the performance of actual systems and found to be accurate within 10%. Efficiency is important for such absolute performance estimates, but is more so for trend analysis, in which many experiments are conducted over a range of experimental factors.

The performance of a network processor-based system depends on: the available hardware resources and their organization, the software controlling those resources (the application) and the network traffic applied to the system. The framework models each of these aspects independently, thereby allowing experimentation with each. Thus, the system, application and traffic models may be changed, one at a time, in order to observe performance under new conditions. This provides the user of the framework a great amount of flexibility, quite unreachable via detailed system simulation or prototyping alone.

Section 2 presents a detailed discussion of the framework's structure, components and algorithms. Sections 3- 5 demonstrate the flexibility and accuracy of the framework with several modeling examples. Conclusions and future work are outlined in Section 6.

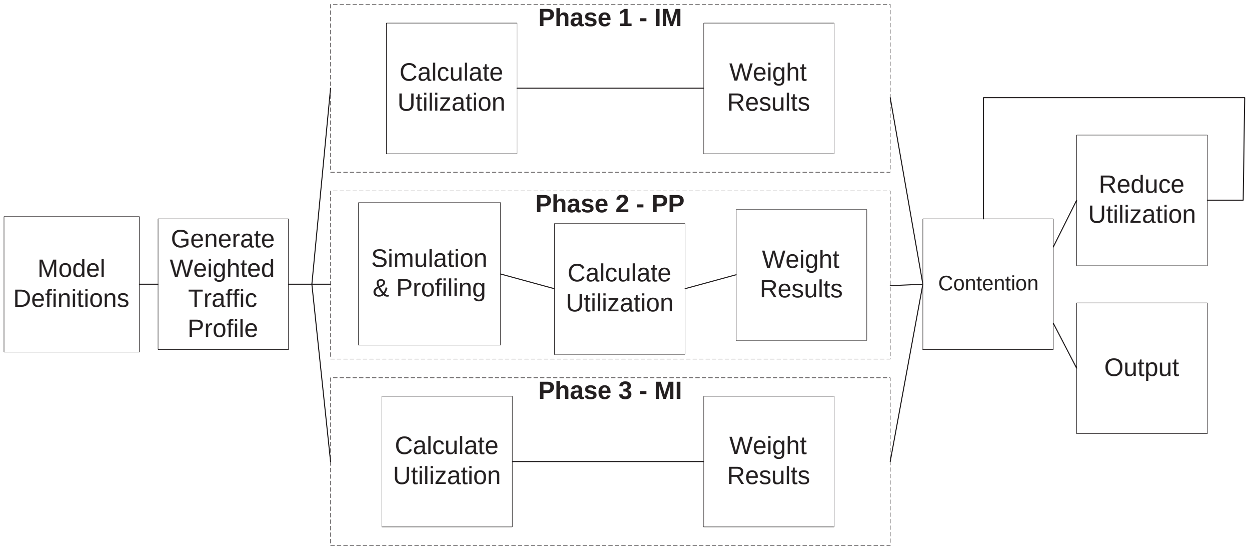

Figure 1. Framework Flowchart.

2. Framework Description

Cycle-accurate, software-based simulation and analytical modeling occupy opposite endpoints on the spectrum of computer systems modeling; the former exhibits relatively good accuracy, along with significant development and simulation time, while the latter provides lesser accuracy in exchange for rapid results. The modeling framework described in this paper is a combination of these two approaches and therefore falls somewhere between these extremes. Its goal, i.e., to strike an efficient balance between development and simulation times and accuracy, is similar to that taken in the design of a high-level statistical simulator for general-purpose computing [11]. By focusing on network processing systems, however, there is an even greater opportunity for efficiency since the workloads have nice properties (e.g., programs are short and protocols are well-defined) and network traffic permits accurate statistical description.

Figure 1 depicts the overall flow of events in the framework. Users of the framework first define the three component models, briefly introduced below.

- Application - A modular, executable specification of router functionality described in terms of program elements and the packet flow between them. Examples of program elements for an Ethernet IPv4 router include: IP Fragmentation, IP address lookup, and packet classification. Click modular router configurations [9] serve as application model descriptions.
- System - A description of the processing devices, network interfaces, memories, and interconnects in the system being modeled. For instance, one or more processors could be connected to SDRAM via a high-

speed memory bus. All elements of the application model are mapped to system components.

- Traffic - A description of the type of traffic offered to the system. Traffic characteristics influence 1) which paths (sequence of elements) are followed in the application model and 2) the resources used within each element along the path.

The general approach is to approximate application characteristics statistically, based on instruction profiling and program simulation, and to use those approximations to form system resource usage and contention estimates.

In addition to these models, the user provides a *mapping* of the application elements onto the system; this mapping describes (explicitly) where each element gets executed in the system and (implicitly) how the packets flow between devices when they move from one program element to another.

As indicated in Figure 1, framework operation is divided into three phases. The first and third deal with packet arrival (interface-to-memory, denoted IM) and packet departure (memory-to-interface, denoted MI), respectively. The second phase is concerned with packet processing (denoted PP). The output of each phase is a parameterized description of resource utilization. The user's definition of the application, system and traffic models describes the path to and from interfaces and main memory. This path information, along with traffic attributes such as packet sizes and inter-arrival times, helps determine the utilization of the shared channels between interfaces and memory. Phase 2 is different in that the traffic attributes are used to determine the frequency with which paths through the application will be executed. This frequency information is used to drive cycle-accurate simulation of the processing elements in the system, the output of which drives traffic on shared resources (e.g., loads and stores from/to main memory). The three phases operate in parallel, and therefore can compete for shared resources. The contention block from Figure 1 checks for oversubscribed resources and iteratively solves the contention by slowing down those devices causing contention.

2.1. The Click Modular Router

This section describes the Click modular router. A good introduction to Click is found in [9]; Kohler's thesis [8] describes the system in greater detail. Click is fully functioning software built to run on real systems (primarily x86 Linux systems). When run in the Linux kernel, Click handles all networking tasks; Linux networking code is avoided completely. This is of great benefit; Click can forward packets around 5 times faster than Linux on the same system due to careful management of I/O devices [9].

Click describes router functionality with a directed graph of modules called *elements*; such a graph is referred to as a Click *configuration*. A connection between two elements indicates packet flow. Upon startup, Click is given a configuration which describes router functionality. An example click configuration implementing an Ethernet-based IPv4 router is shown in Figure 2. (We often refer to configurations as routers, even if they do more than route packets.) The Click distribution comes with a catalog of elements – sufficient to construct IP routers, firewalls, quality-of-service (QoS) routers, network address-translation (NAT) routers and IPSec VPNs. Click users can also build their own elements using the Click C++ API. Configurations are described in text files with a simple, intuitive configuration language.

We ported Click to the Alpha ISA [4] in order to profile and simulate the code with various processor parameters using the SimpleScalar toolkit [1]. Within the framework, Click configurations function as application specifications and implementations. Using Click fulfills our goal to use real-world software rather than benchmark suites whenever possible; previous experience in evaluating network processor architectures [3] has shown the importance of considering complete system workloads.

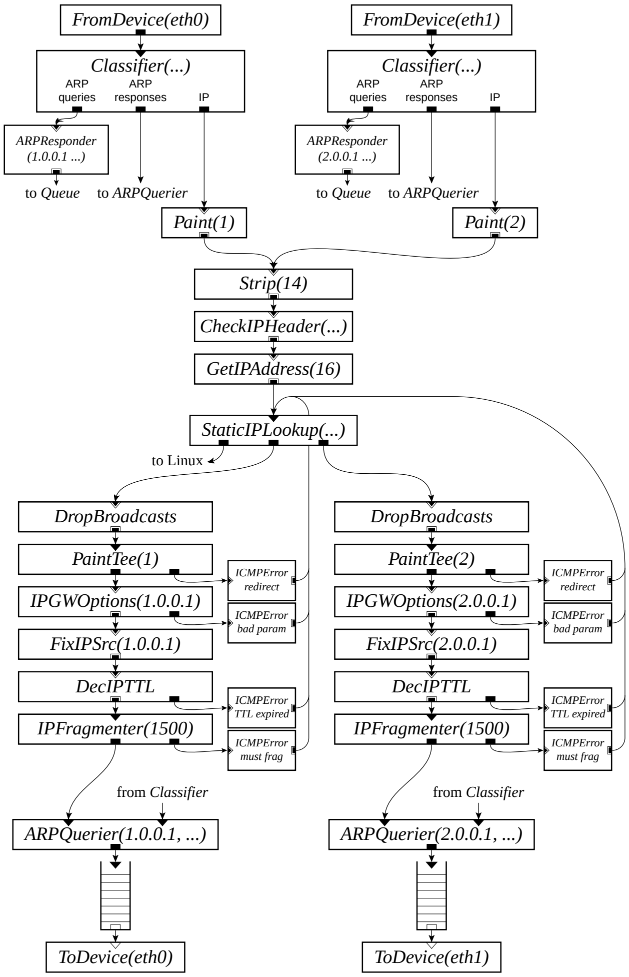

Figure 2. Sample Ethernet IP router click configuration. (From [9].)

In Click, the mechanism used to pass packets between elements depends on the type of output and input ports involved. Ports can use either *push* or *pull processing* to deliver packets. With push processing, the source element is the active agent and passes the packet to the receiver. In pull processing, the receiver is active and requests a packet from an element on its input port. Packet flow through a Click configuration has two phases corresponding to these two types of processing. When a *FromDevice* element is scheduled, it pulls a packet off the inbound packet queue for a specified network device and initiates push processing; push processing generally ends with the packet being placed in a *Queue* element near a *ToDevice* element; this can be seen by examining paths through Figure 2. When a *ToDevice* element gets scheduled, it initiates pull processing, generally by dequeueing a packet from a *Queue*. As a general rule, push processing paths are more time consuming that pull processing paths.

Click maintains a worklist of schedulable elements and schedules them in round-robin fashion. *FromDevice*, *PollDevice* (the polling version of *FromDevice* used and described later in this paper) and *ToDevice* are the only schedulable elements. On a uniprocessor-based system, there is no need for synchronization between elements or for shared resources like free-buffer lists, since there is only one packet active at a time and packets are never pre-empted within an element. However, Click also includes support for shared-memory multiprocessors [2] and thread-safe versions of elements with potentially shared state. Synchronized access and contention for shared resources will be of concern for shared-memory multiprocessor-based (SMP) systems, as will be seen later.

2.2. Application Model

The application is modeled by profiling the Click configuration and simulating its execution on the target processor over the packet input described in the traffic model. This yields information such as: instruction counts and type information, cache behavior, branch behavior, amount of instruction-level parallelism (ILP) and their relative percentage contribution to cycles per instruction (CPI). The specific information kept in the model depends on what aspects of the system are of interest.

The framework's intent is to have the application be simulated over the target traffic, on the target processor(s). Note that execution time so determined is the major part of the overall packet forwarding time, but it will have to be adjusted later to account for contention for shared resources and the cost of any synchronization overhead.

The examples presented in this paper illustrate only a portion of the flexibility of application analysis provided by the framework. Click configurations, and individual elements, can be profiled in great detail. The studies found in later sections only use the number of instructions, CPI achieved and number of system bus transactions (i.e., second level cache L2 misses) to estimate performance.

2.3. System Model

The system model consists of devices (e.g., processor cores and memories) and channels (e.g., memory and I/O buses) that connect devices. Each device component has internal operations (e.g., an add instruction), which are completely specified by the device, and external operations (e.g, a load that misses in caches), whose implementations are dependent on target devices (e.g., SDRAM) and the channels that connect them.

System resources and organization are described with two basic objects: devices and channels. Device and channel objects, once instantiated, are attached to one another to describe the system's organization. Once connected, the system can begin to deduce the cost of the external operations for each device; this is only a portion of the cost, however, since certain aspects of external operation cost such as locality and contention depend on the application and traffic models. A sample system model that will be used in this paper is shown in Figure 3 with four device types (processors, memories, interfaces and bridges) and one channel type (bus). Other device and channel types are certainly possible within the framework.

Many network processor designs include specialized instructions that might not be present in the simulated processor. To explore the impact of a specialized instruction or hardware assist, the user can do the following: add a new op code and instruction class to the profiling tool, define a macro (e.g., compiler intrinsic) that uses the new instruction class, modify at least one Click element to use the macro rather than C/C++ statements, and update the system device to include the new instruction. Adding a new instruction to the profiler involves adding the new instruction class and op code declarations, as well as adding a C function that implements the new instruction..

2.4. Traffic Model

The traffic model is a statistical description of packet stream characteristics. These characteristics include: packet size distribution, IP packet type distribution (e.g., UDP or TCP), inter-arrival time distribution, distribution of source addresses, and distribution of destination addresses. This description can be specified directly by the user or measured from a trace of packets.

2.5. Application-System Mapping

As mentioned previously, a complete model description also requires a mapping of Click elements onto system devices. Each element must be assigned to a processing device (usually a processor) for execution and one or more memory devices for data and packet storage. For example, if the system is a PC-based router, then all computation elements in the application are assigned to the host processor for execution and all packets and data are assigned to system RAM.

Given this mapping, the framework can find a packet's path through the system as it flows through the Click configuration. The mapping indicates where each element stores its packets. Thus the packet must flow through the system, across channels from memory to memory, according to the elements visited along the packet's path through the Click configuration. Accurately modeling the performance of packet movements is difficult because, in general, it is no different from modeling a distributed shared memory system. The framework currently can model systems with a small number of shared memories with some accuracy as shown in the multiprocessor examples presented later. Extensions to the framework for distributed memory modeling is under development.

While Click describes packet flow, it does not describe how packets are delivered to Click in the first place; the packet delivery mechanism is defined by the operating system (OS) and device drivers. Since OS code is not included in the application model, a manual analysis is required to model the packet arrival mechanism. The mapping must also indicate whether packet arrival is implemented with interrupts, polling, a hybrid interrupt/polling scheme [10], or special hardware supporting a scheme like active messages [5]. The Click system itself, when running in kernel mode on Linux, uses polling to examine DMA descriptors (a data structure shared by the CPU and the device for moving packets.) This issue is of paramount importance in real systems, and is worthy of the attention it has received in the research literature. All examples presented in this paper will use polling. Other examples using interrupts can be found in an upcoming technical report.

2.6. Metrics & Performance Estimates

The primary goal of the performance analysis is to find the forwarding rate, forwarding latency and resource utilization for a given application, system and traffic description. Intuitively, this means finding the time and resources needed to move the packet through the system, from input interface to output interface, as shown in the three phases: IM, PP and MI of Figure 1.

The principal metrics – latency, bandwidth and resource utilization – for these three phases directly determine system performance. Global analysis for the three phases is required since they generally rely on shared resources. For example, suppose the input and output interfaces both reside on the same I/O bus. Then, inbound and outbound packets will be forced to share the I/O bus's total bandwidth. Moreover, while each phase may have a different maximum throughput, the effective throughput of each is limited by that phase's arrival rate. A phase's arrival rate is determined by the throughput of the preceding phase since they act as stages in a pipeline: phase 3 is fed by phase 2, phase 2 by phase 1, and phase 1 by the traffic model.

2.7. Framework Operation

Overall forwarding rate (i.e., system throughput) will be limited to the lowest forwarding rate among the phases. Once the lowest of the three maximum forwarding rates is found, it can be used as the arrival rate for all phases to determine whether any resources are over-subscribed at that rate; if no shared resource is over-subscribed at this rate, then that rate is the maximum loss-free forwarding rate (MLFFR). System forwarding latency will be the sum of these individual latencies. Finally, resource utilization will be the aggregate utilization of all phases, since they will generally operate in pipelined fashion.

The steps to determine MLFFR, as well as the latency and resource utilization seen at that rate, are summarized as follows:

1. Find the latency and throughput of each phase assuming no contention for shared resources.
2. Use the lowest resulting throughput as the arrival rate at each phase, and find the offered load on all shared resources.
3. If no shared resources are over-subscribed, then the MLFFR has been found. Otherwise, the current rate is too high and must be adjusted down to account for contention. The adjustment is made by iteratively charging arbitration cycles to thosee devices using the congested channel until it is no longer over-subscribed.

The framework assumes that only shared channels can be over-subscribed; this is a reasonable assumption so long as the channels leading to a device saturate before the device itself does. For the buses presented here, over-subscription is said to occur when utilization reaches 80%. If no shared resources in the system are over-subscribed (that is, if offered load is less than capacity), then the system is contention-free. Any over-subscribed resources represent contention and changes must be made to estimates of latency and throughput for any phases using those resources. To resolve contention, slowdown is applied equally to contributors. If capacity represents x% of offered load, then all users of the congested resource must reduce their offered load to x% of its initial value. Offered load is reduced by iteratively increasing the number of arbitration cycles needed for each user to master the bus until offered load on the channel drops below the saturation point. Initially, the model includes no arbitration cycles; they are only added to a device's bus usage latency once saturation is reached.

2.8. Implementation

The framework's object-based system description and modeling logic are implemented in the Python programming language [12]. A sample set of model declarations using the Python object syntax is shown in Figure 4. Python is also used as a glue language to permit the modeling logic to interact with Click and Simplescalar. As mentioned previously, Click provides the router specification and implementation, and Simplescalar is used to profile and simulate Click's execution on a target processor.

In general, the framework operates quickly, on the order of minutes when considering simple traffic models. However, processor simulation can be time consuming for experiments involving great amounts of traffic (i.e., from a trace) or complex traffic models. Note that while an hour can be spent simulating system operation over a trace of tens of millions of packets, there are two reasons why this is acceptable: 1) most analyses can be conducted with far fewer packets (so this situation is unlikely to be useful), and 2) this is simulation time, not development time, and can, when needed, be tolerated.

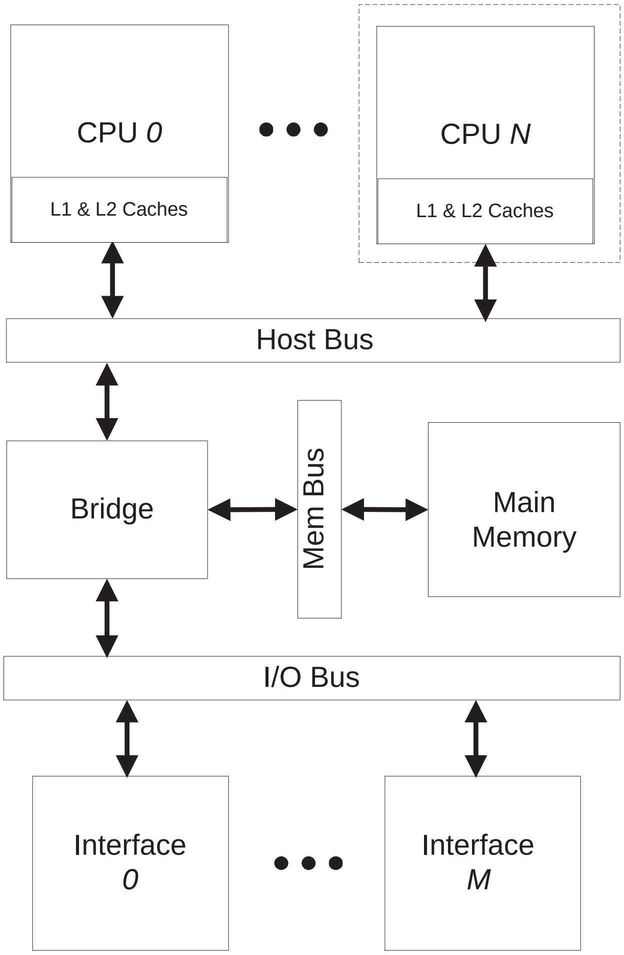

Figure 3. System organization for a PC-based router with N processors and M network interfaces.

System	Processor							Memory		Buses		Chipset
	Num	Name	Clk	I.W.	Caches (L1s/L2)		Thds	Lat	Width	I/O	Mem	
Uni-PC	1	PIII	700	4	16/4/32	256/4/32	1	60	64	PCI/32/33/2-8/No	GTL/64/100/4-4/Yes	440GX
SMP-PC	1-4	PIII	500	4	16/4/32	512/2/32	1	50	64	PCI/64/33/1-4/No	GTL/64/100/4-4/Yes	440NX

Table 1. Parameters for the base systems used in this paper. Clock units are MHz. Issue width is denoted I.W.. Cache descriptions are Size(KB)/associativity/Line Size(bytes). All caches are LRU. Memory units are ns and bits, for access latency and width, resp. Bus descriptions are Name/width(bits)/speed(MHz)/read-write burst size/separate address and data buses.

3. System Modeling Examples

This section will now illustrate, by way of example, how the framework can be used to model network processing systems. The systems modeled below were chosen because they are the systems used by the Click designers to evaluate Click performance. Hence, the performance estimates yielded by the framework can be compared to the observed performance of the actual system.

3.1. Uniprocessor PC

The first example models the PC-based system used in [9] to measure the performance of the Click IP router configuration from Figure 2. The details of the system being modeled can be seen in Table 1 in row “Uni-PC”. The main features are: a 700MHz Pentium III Xeon processor, and a 32-bit wide 33MHz PCI I/O bus. There are three channels in this system: the host, memory and PCI buses. Of these, only the memory and PCI buses are shared.

The organization of the system model used here is shown in Figure 3 with only CPU 0 being present. The system contains 8 interfaces in total: four are used as packet sources and four as destinations. The code declaring models in this experiment is shown in Figure 4. The traffic used in the Click study was very simple. The packet stream consisted of 64 byte UDP packets generated by each of the sources; each source generated packets with uniformly chosen destinations (from among the 4 destination interfaces). Finally, since there is only one processing device, there are no mapping decisions to be made. The packet delivery mechanism, as mentioned above, is polling.

Polling, with Click, involves: checking the DMA descriptor for notice of a new packet, moving the new packet onto an incoming packet queue for processing, moving a packet from the outgoing packet queue into the DMA buffer, and updating the DMA descriptor to indicate the buffer modifications. Both the DMA descriptor and the DMA buffers are effectively non-cached (since they are accessed by both the processor and the I/O devices), thus each of these steps involve L2 cache misses, for a total of 5 L2 misses per polling event.

Figure 5 shows the forwarding rate of the router as a function of the input rate. The framework estimates

```
# Declare model objects
app = Application('npf_iprouter.click')
traffic = Traffic([[64], [1.0]],
                  'Uniform')
sys = System()

# Create channels
pci = bus('PCI',32, 33, 2, 8, 0, 0)
membus = bus('Membus',64, 100, 4, 4,
             0, 1)
hostbus = bus('Hostbus',64, 100, 4, 4,
              0, 1)

# Create & attach devices
brdge = bridge('Bridge',
               [pci, membus, hostbus])
cpu = proc('CPU', 700, 4, '16/4/32/l',
           '16/4/32/l','256/4/32/l', 1)
hostbus.attach(cpu)
ram = mem('Mem', 60, 64)
membus.attach(ram)
# 8 such interfaces
int0 = interface('Eth0', 'source')
pci.attach(int0)

# Add channels and bridge to system
sys.addbuses([pci, membus, hostbus])
sys.addbridges([brdge])
```

Figure 4. Framework code declaring traffic, application and system models for the uniprocessor example.

an MLFFR of 322,000 packets per second. This is within 3% of the value of 333,000 reported in [9]. At this rate, system throughput is limited by the packet processing (PP) phase. Table 2 reports the per-phase details. Note that neither bus is over-subscribed at the MLFFR. With the PP phase, the Click configuration executes 2110 instructions at a CPI of 0.8467 for a total of 1787 cycles. With a cycle time 1.43ns (i.e., 700MHz), those 2554ns, plus the 550ns spent on the 5 polling L2 misses per packet, become the bottleneck to system throughput.

In addition to the base system, the experiment presented here includes a number of speculative systems for which real performance numbers are not available. These speculative systems include faster processors and faster PCI buses. By increasing the CPU clock rate to 1GHz, the MLFFR increases to 428,000 packets per second.

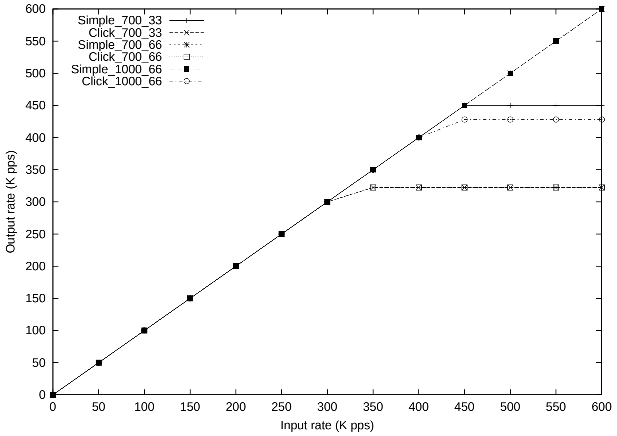

Figure 5. Forwarding rate vs. input rate for the uniprocessor-based router with 64-byte packets. Curves marked 'Click' reflect modeling of full IP router performance; 'Simple' indicates an empty configuration and reports maximum system forwarding rate. The two numbers at each label report CPU clock rate and PCI bus speed, both in MHz.

Figure 5 also shows the performance for a simple configuration that performs no packet processing. This configuration is limited by PCI bandwidth. The PCI bus saturates (achieves 80% utilization) at 450,000 packets per second. This number also agrees well with the measured rate in [9] of 452,000 packets per second.

	Latency	Max Rate	Utilization @ Min Rate	
	(ns)	(Kpps)	Membus	PCI
Phase 1 (IM)	574	1442	0.04	0.18
Phase 2 (PP)	**3102**	**322**	0.03	0.00
Phase 3 (MI)	1212	825	0.07	0.39
Result	4888	322	0.14	0.58

Table 2. Per-phase details for Click IP routing on a uniprocessor. The result entry for Max Rate is a minimum of the column; other rate entries are sums.

While the thrust of this paper is to introduce the modeling framework, it is interesting to briefly discuss the bottleneck of this PC-based system. No matter how fast the processor, the forwarding rate will remain at 450K packets per second due to PCI saturation. This PCI saturation is due to Click's packet delivery mechanism. PCI could, in theory, deliver more throughput if the source of the data was to always initiate the burst. With Click, the network interface masters all transactions. This is desirable for packet delivery into memory; the interface is the source of the data, so the packet can be bursted into memory. For packet retrieval from memory, on the other hand, the interface is not the source and thus must use un-bursted memory read transactions. This fact is reflected in the model's PCI bus characterization; PCI reads have a burst size of 2, while writes have a burst size of 4.

3.2. SMP PC

Click's operation on SMP-based systems was described and measured in [2]. The system organization is similar to that of the single processor system; Figure 3 is still an accurate depiction. The only significant resource differences are: additional processors on the host bus, slower clock rates on those processors (500 MHz vs. 700 MHz), a wider PCI bus (64b vs. 32b), and fewer interfaces (4 vs. 8). System details can be found in Table 1.

Another difference of note is found in the packet handling mechanism. The network interfaces used in the uniprocessor system allowed the Click designers to implement polling and to make all host-device interaction indirect through DMA buffers. The interfaces on the systems used in the SMP study, however, do not have this flexibility. In fact, on every packet send, the host must write a pair of device registers on the interface; this requires an additional PCI transaction for every packet. In the uniprocessor case, the packet processing phase (PP - phase 2) involved no PCI bus traffic (the host just moved the packet into a DMA buffer and updated a descriptor); this is not the case any longer for SMP-based system.

Aside from system resources and organization, there are several other important differences between SMP and uniprocessor Click operation that need to be discussed. These include: scheduling elements on CPUs, synchronizing access to shared data (such as mutable elements and the free buffer list) and cache misses due to shared data (such as *Queue* elements).

The four-interface version of Figure 2 has eight schedulable elements; a *PollDevice* and *ToDevice* element for each interface. Each of these must be assigned to a CPU. The Click designers considered both dynamic and static assignments and found static scheduling of elements to CPUs to be more effective than their own adaptive approach, for the following reason. Any sharing or synchronization between CPUs will involve accesses to memory. With a static assignment of elements to CPUs, each CPU is given a private worklist of elements to schedule. The adaptive approach could not account for cache misses in scheduling decisions and therefore was capable of poor locality. Additionally, each CPU will maintain its own list of packet buffers; only when a CPU is overloaded will it incur cache misses to allocate a buffer from another CPU. Static scheduling is modeled in all experiments in this paper.

The experiments in this section will consider two and four CPU SMPs. In the two CPU system, each CPU will be assigned two *PollDevice* and two *ToDevice* elements. To increase locality, each CPU is assigned *PollDevice* and *ToDevice* elements for different devices, since incoming packets rarely depart on the same interface on which they arrived. Likewise in the four CPU system, each CPU will host one *PollDevice* element and one *ToDevice* element, each element associated with a different interface.

Some elements, in particular *Queue*s, have shared state that must be protected via synchronized access. It is

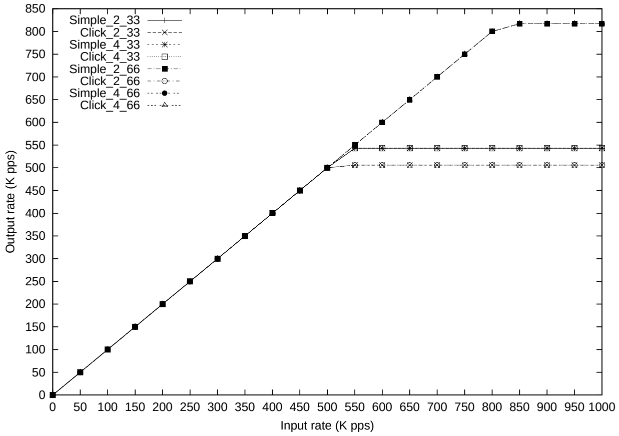

Figure 6. Forwarding rate vs. input rate for the SMP-based router with 64-byte packets. The two numbers at each label report number of CPUs and PCI bus speed in MHz.

often the case that paths from multiple interfaces, and therefore multiple CPUs, lead to the same output *Queue*. As a result, enqueues must be synchronized. However, the dequeue operations on a given *Queue* are often, although not always, initiated by a single CPU, since *Queue* elements often feed CPU-specific *ToDevice* elements. Click recognizes when this common case holds and disables *Queue* dequeue synchronization.

Packets move between CPUs, and consequently between interfaces, via *Queue* elements. For example, suppose a packet arrives at CPU A destined for interface N, which is assigned to CPU B. CPU A will initiate push processing, which culminates in an enqueue onto a *Queue* element. When the *ToDevice(N)* element gets scheduled on CPU B, it will initiate pull processing by dequeuing a packet from element *Queue* and send the packet on its way. Since the *Queue* element is used by both CPUs (and any other CPUs in the system), either of these accesses might have resulted in an L2 cache miss. Such cache misses will increase with the number of CPUs in the system. Note that in the uniprocessor case, there are no L2 misses of this sort.

Synchronization and L2 misses due to shared data are important because they represent the cost of having multiple CPUs. These costs tend to increase as CPUs are added to the system. The benefit, of course, is that multiple packets (ideally N, when there are N CPUs) can be processed in parallel. For the situation modeled here, the cost is not very high. This is because, when processing a packet, the time spent outside of synchronization regions is much greater than the time spent within synchronized regions. Pushing packets onto *Queue* elements is the only significant synchronization event; enqueues, even when they involve L2 cache misses, are at least an order of magnitude faster than the rest of the IP routing processing path. So CPUs do not, for a moderate number of CPUs, backup at *Queue* elements. Furthermore, while the expectation of an L2 cache miss per enqueue goes up as CPUs are added to the system (more CPUs implies more remote users), the maximum is 1 L2 cache miss per enqueue. Again, this represents only a fraction of the total time needed to process the packet. Thus, increasing the number of CPUs is a design win.

The results of the IP routing experiments on SMP systems of 2 and 4 CPUs are shown in Figure 6. The experiment presented here includes the systems used in [2] as well as speculative systems for which no data has been published. The results obtained by the model are within 10% of the original Click study's results. The 2 CPU SMP model estimates the MLFFR at 506K packets per second, as compared to the observed rate of 492K packets per second. The 4 CPU SMP model yields an MLFFR of 543K packets per second, as compared to a measured value of 500K packets per second.

2 CPU	Latency	Max Rate	Utilization @ Min Rate	
	(ns)	(Kpps)	Membus	PCI
Phase 1 (IM)	382	2612	0.05	0.19
Phase 2 (PP)	**3952**	**506**	0.06	0.03
Phase 3 (MI)	970	1031	0.10	0.49
Result	5304	506	0.21	0.71

4 CPU	Latency	Max Rate	Utilization @ Min Rate	
	(ns)	(Kpps)	Membus	PCI
Phase 1 (IM)	382	2612	0.10	0.34
Phase 2 (PP)	**3952**	**1012**	0.11	0.12
Phase 3 (MI)	970	1031	0.20	0.98
Result	5304	1012	0.41	1.44

Table 3. Per-phase details for Click IP routing on an SMP (2 and 4 CPU). The Max Rate result is a column minimum, not a sum. Note that the 4 CPU case has an over-subscribed PCI bus at the minimum phase forwarding rate.

The per-phase details prior to contention analysis are shown in Table 3. The 2 CPU case involves no over-subscribed channels, therefore the MLFFR is equal to the minimum phase forwarding rate. However, this is not the case for the 4 CPU scenario since its PCI bus is over-subscribed at the minimum phase forwarding rate. The MLFFR, then, is the minimum forwarding rate beyond which the PCI bus becomes saturated. As described in Section 2.7, the framework finds the MLFFR by solving for the number of bus arbitration cycles needed per packet in order to keep the PCI bus from saturating. The resulting rate is 543K packets per second, rather than the greater than 1M packets per second reported prior to factoring in contention.

While accurate to within 10%, the SMP models are not as accurate as the uniprocessor models. This is because the SMP system is more complicated, and the model leaves out many details. For instance, the contention resolution method is only a coarse approximation. Also, the Pentium III processor modeled here uses a specific coherency protocol not captured by the model. Cache misses due to sharing are modeled, but the overhead due to coherency traffic is not.

4. Application Modeling Examples

This section demonstrates how the framework can be used to explore application performance. The intent is to show that the framework can help determine the speed at which a given system can support a target application.

4.1. IPSec VPN Decryption

In this experiment, the Click IP router configuration is extended to include IPSec VPN [7] encryption and decryption. The VPN decryption and encryption blocks are shown in Figure 7 and Figure 8, respectively. The figures indicate how simple it is to add significant functionality to an existing router; adding the same functionality to a Linux or FreeBSD router would be, by no means, trivial.

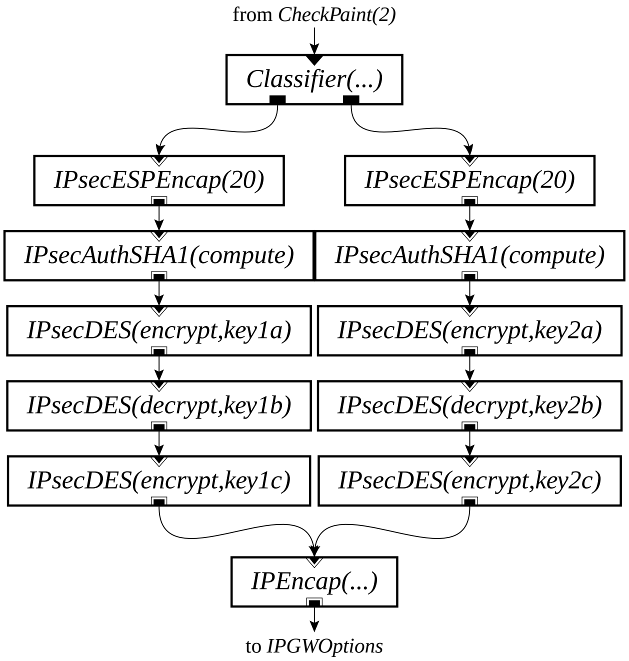

Figure 7. Click VPN decryption configuration. (From [2].)

Note that in these experiments, only the packet processing phase is changed; packet receive and transmit will have the same characteristics for these systems as before. Simulation of the new Click configuration on the target processors shows that the processing of each packet requires 28325 instructions, over 13 times as many instructions compared to the baseline IP router model. Instruction processing, rather than packet traffic on the PCI bus, dominates performance in this case.

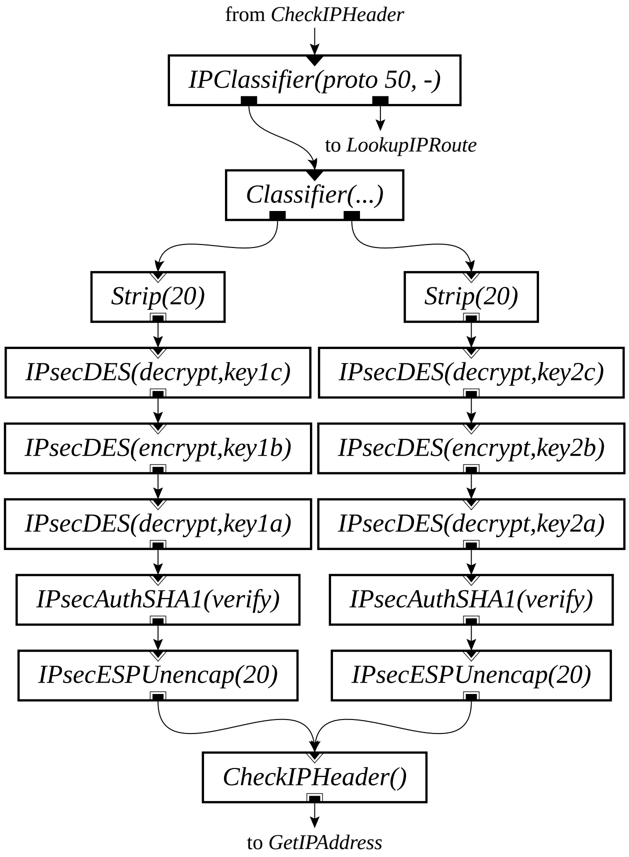

Figure 8. Click VPN encryption configuration. (From [2].)

Hence, as the results indicate in Figure 9, the SMP system experiences linear speed up in the number of CPUs. Once again, the model's performance estimates match the observed values closely. Estimated MLFFRs for the uniprocessor, 2 CPU SMP and 4 CPU SMP are 28K, 46K and 92K, respectively. Observed values [2] for the same systems are: 24K, 47K, 89K – all well within 10% of the model's estimate. The one speculative system included in the experiment, a 1GHz uniprocessor, sees a performance increase in direct proportion to the increase in clock rate; an expected result on such a compute bound workload. Forwarding rate, it is noted, remains well below the system limit.

4.2. IP Lookup

The routing examples presented in Sections 3.1 & 3.2 used a very small routing table to handle IP lookups. This was natural since the experiment only involved four hosts. In this section, we consider the effect of increasing route table size upon routing performance. We will use the system from Sections 3.1 & 3.2 as a router between four networks (carrying traffic from many hosts), rather than four hosts. The primary effect is that the route table cannot

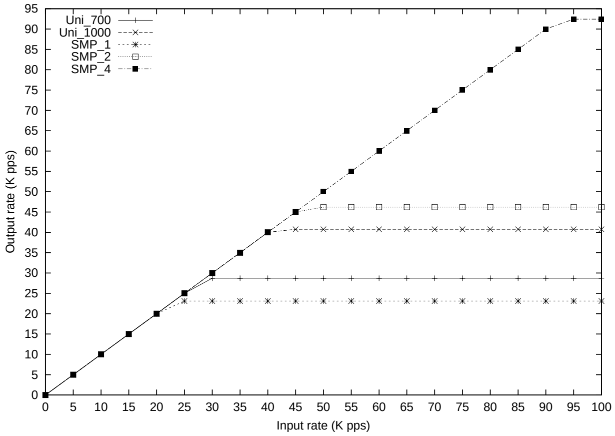

Figure 9. Forwarding rate vs. input rate for the VPN router with 64-byte packets.

remain in the processor caches, thus each IP lookup will invoke L2 cache misses.

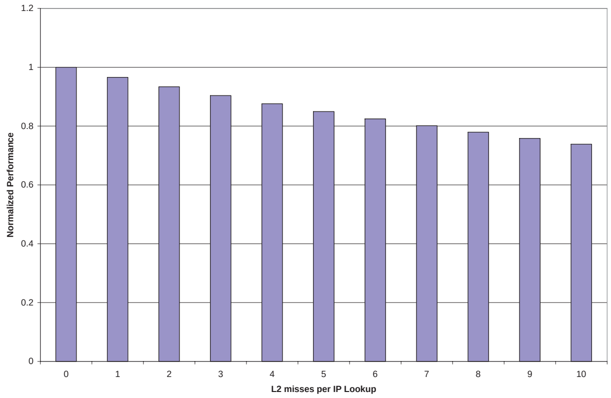

Figure 10. Normalized Performance vs. Memory References per IP Lookup.

Large route tables require good lookup algorithms. The primary metric by which IP lookup algorithms are judged is the number of main memory accesses required per lookup [13]. The number of memory references depends on the data structure and algorithm being used, the size of the lookup table, the size of processor caches, and the locality in the address stream. In these experiments, we will not consider the implementation details of any particular lookup algorithms, but rather observe the effect upon performance as the number of main memory references per IP lookup increases. The results from these experiments, shown in Figure 10, report that forwarding rate decreases linearly as the number of main memory references per IP lookup increases from 1 to 10.

Note that these results did not require that we implement an algorithm that achieves the desired number of memory accesses per lookup. Rather, annotations were made in the experiment scripts indicating the cost of each IP lookup. Such an approach allows a designer to first determine how many references can be tolerated, as shown here, and then choose a lookup algorithm that provides that performance (if one exists).

5. Traffic Modeling Examples

All of the previous experiments have used a simple traffic model: a stream of 64B packets at varying uniformly-distributed arrival rates. Changing the traffic model is a simple matter of specifying the distribution of packet sizes, distribution of packet arrival times and the distribution of packet type (TCP, UDP, errors, etc.). Application profiling and system resource usage are both dependent on packet size. Thus, both steps must be carried out for relevant packet sizes and then weighted according to the distribution (and influence) of those sizes.

By using minimum sized packets, these experiments tend to emphasize per packet overhead. In the 700MHz uniprocessor IP routing experiment, for instance, increasing packet size from 64B to 1000B reduces the MLFFR from 322K packets per second to 53K packets per second. While MLFFR decreases as packet size increases, the bit throughput of the system actually increases from 165 Mbps to 424 Mbps. This is, in fact, the expected result for two reasons: 1) the original system was limited by packet processing time (phase 2), and 2) that processing time is more or less independent of packet size (since only packet headers are involved).

Additionally, we can consider a range of packet sizes, such as a packet size distribution of 43% at 64 bytes, 32% at 500 bytes, 15% at 1000 bytes and 10% at 250 bytes. At this mix of packet size, the IP routing configuration on the uniprocessor system forwards 216K packets per second.

6. Conclusions

This paper has presented a hybrid framework for network processing system analysis. It employs a combination of analytical modeling and cycle-accurate simulation to provide accurate, efficient results. The framework is intended to enable quick analysis of systems that implement complete and detailed router functionality. To illustrate the modeling approach, a number of system, application and traffic models were presented. The framework yielded performance estimates accurate to within 10%, as compared to the measured performance of the systems being modeled.

This modeling framework is still in development. The models presented in this paper have served a dual purpose: 1) they helped to introduce Click as Click was meant to be used and 2) they helped to informally validate the modeling approach embodied in the framework. However, there are several other types of network processing systems that have organizations quite different from a PC. These systems tend to use a greater variety of devices, including switches and special-purpose hardware assists, as well as employ distributed memory and hardware queues and buffers. In future work, we plan to model these systems and devise ways to map Click configurations onto more heterogeneous systems such as the Intel IXP1200 [6].

The framework, including complete source and the experiment scripts used to generate the results presented in this paper, is available for download on the web at `http://www.cs.washington.edu/homes/pcrowley`.

7. Acknowledgements

This work was supported in part by NSF grant MIP-9700970.

References

[1] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, 1997.

[2] B. Chen and R. Morris. Flexible control of parallelism in a multiprocessor pc router. In *Proceedings of the 2001 USENIX Annual Technical Conference (USENIX '01)*, pages 333–346, Boston, Massachusetts, June 2001.

[3] P. Crowley, M. E. Fiuczynski, J.-L. Baer, and B. N. Bershad. Characterizing processor architectures for programmable network interfaces. In *Proceedings of the 2000 International Conference on Supercomputing*, May 2000.

[4] R. L. S. (editor). *Alpha Architecture Reference Manual*. Digital Press, 1992.

[5] T. Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrating communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, May 1992.

[6] Intel Corp. Intel IXP1200 Network Processor Datasheet. `http://developer.intel.com`, 2001.

[7] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Internet Engineering Task Force, RFC 2401, `ftp://ftp.ietf.org/rfc/rfc2401.txt`, November 1998.

[8] E. Kohler. *The Click modular router*. PhD thesis, Massachusetts Institute of Technology, November 2000.

[9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[10] J. C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, 1997.

[11] M. Oskin, F. T. Chong, and M. K. Farrens. HLS: combining statistical and symbolic simulation to guide microprocessor designs. In *ISCA*, pages 71–82, 2000.

[12] Python. The Python programming language. on the web, 2002. `http://www.python.org`.

[13] V. Srinivasan and G. Varghese. Fast address lookups using controlled prefix expansion. *ACM TOCS*, 17(1):1–40, Feb. 1999.