

# Design of an OC-192 Flow Monitoring Chip

John Huber  
March 19,2000  
CSE 222 Final Project

## 1. Introduction

Being able to monitoring network traffic passing through a router in the internet today is a very challenging, yet important task. The number of independent flows, which we will define as a source-destination pair, traversing the internet is growing very rapidly. In recent studies, the number of separate flows in a one hour period monitored across large ASs has increased into the millions, and will mostly likely continue to grow. The speed of network links is also rapidly increasing, reducing the amount of time available to do memory lookups and making it unfeasible to use network processor/DRAM based techniques to monitor the flows. When line speeds reach OC-192 and higher, due to these timing limitation, there will need to be dedicated hardware using SRAM to maintain the flow counters. Since SRAM is quite expensive, it will be impossible to track the exact number of bytes sent from each of the million flows passing through a given router. Instead, the goal of this chip is to identify those flows taking up the largest percentages of the bandwidth and do accounting on only these.

Tracking the bandwidth usage of just the largest flows will allow several important network administrative tasks to be carried out. These include usage-based pricing (if the usage is greater than a certain threshold), queue management based upon penalizing over-subscribed users, and detection of denial of service attacks.

The design of this chip is based upon the architecture described in the paper, "Scalable Measurement: Finding some Elephants in a Swarm of Ants" by Christian Estan and George Varghese. In Section 2, I will describe the general architectural implementation. Section 3 will discuss the features and low level design details. Section 4 will discuss a simulation which was run to verify the functional correctness and Section 5 will finish with some ideas for future work.

## 2. Architecture

As stated above, the goal of the architecture is to filter out the flows that are using the largest share of the bandwidth and keep detailed flow statistics for these, while using only a minimal amount of memory on the vast number of smaller flows. The architecture that was developed to meet these goals can be divided into two components, 'cascaded' hash stages and a flow container. The hash stages act to filter out only the largest flows which have an arrival rate greater than some threshold rate, which are then stored in the flow container where the detailed accounting is performed.

Each time a new packet arrives, the flow identifier for the packet is extracted from the packet header. This can be a combination of any header fields which uniquely identify a packet down to the granularity which flow accounting will take place. The flow identifier is used as an index into a series of hash stages. Each entry in these hash stages stores the arrival rate for all the flows which map to that entry. A unique hash key is used to hash a flow into each stage. In this way, the entries in each stage should represent the rate for a unique, independent combination of flows. The rate update which is performed when a new packet arrives can be done in a couple of



### 3. Design Details

In this section, I will begin by giving a brief overview of the high level features used in the design, then I will discuss the lower level blocks used for the implementation of these, and finally I will give some physical estimates for the chip

#### 3.1 Features

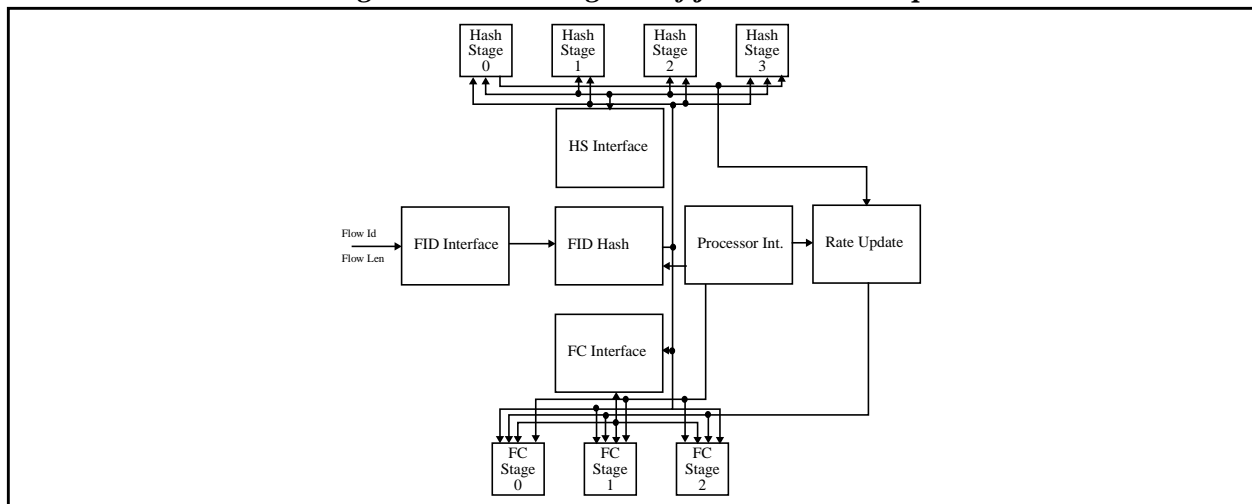
The architecture which was implemented for this chip consisted of 4 hash stages each of which was 4K in depth. The updates to the hash tables was performed in the parallel cascade fashion(i.e. for each packet which arrived, updates were done for every hash stage). The flow container was implemented as a 3 level hash, the first stage was 2K in depth, followed by a 1K stage, followed by a final stage containing 512 entries. When a flow identifier is to be written into the flow container, the first stage is checked to see if the entry for the flow is in use. If it is available it will store the flow in this stage. If not, it will move on to the second and third stage checking to see if there is a free entry in one of these. If both of these entries are also occupied, the flow will not be stored in the flow container and the ‘dropped flow id’ counter will be incremented. By implementing the flow container in this way it has been shown that the number of rejected flows is decreased compared to using the same amount of memory in a single stage. The accounting fields which are maintained for each flow stored in the containers include time of first packet arrival, time of last packet arrival, number of packets which have arrived from the flow, number of bytes which have arrived from the flow, and the arrival rate of the flow.

An external processor interface has been implemented in the chip which is used for configuration and receiving status information from the flow monitoring chip. This interface allows the processor to set the rate threshold at which flow will be stored in the flow container, configure the hash keys which are used for the hash stages and the flow container stages, and to clean the flow containers when a flow has reduced its sending rate to a point it is no longer of interest. The processor can use the interface to read out the contents of any of the flow containers, as well as to determine the number of flow ids which were dropped due to insufficient space in the container.

#### 3.2 Block-level Details

Figure 2 shows the block diagram for the chip. I will go through each block giving a functional description of the block and how it fits into the overall chip function.

**Figure 2: Block diagram of flow monitor chip**



The FID Interface is the primary external interface. Each time a packet arrives, the flow id, packet length and a valid will be sent to this interface. It will accept a new flow id every 32 nano-seconds, along with a packet lengths in the range of 40-1500 bytes. This means that the chip can monitor traffic at an aggregate rate of 10 Gb/s. The chip has a cycle time of 8ns which gives it 4 cycles to process each flow id. When a valid flow id is received at this interface, it is forwarded to the FID Hash block and the packet length is sent to the HS interface and FC interface.

The FID Hash block is responsible for determining the addresses which will be accessed in the hash stages and flow container stages. For each stage it calculates an address using a Carter-Wegman H3 hashing function. The hash keys which are used are provided by the processor block and are user programmable. While the storage for these key takes up a considerable number of registers, it alleviates the need for random number generation on chip, considerably reducing the logic complexity. After the address has been calculated it is sent directly to the memory for the read operation and again later for the write update.

The HS interface block is responsible for the providing the data and control signals for all rate updates to the Hash Stages. Each hash stage is implemented using a dual port 4Kx33 SRAM. The rate is a 32-bit quantity which represent the arrival rate of the packets in bytes per second using fixed point notation with the binary point located after the third bit. This allows the representation of rates up to 20Gb/s, much greater than the maximum bandwidth. The most significant bit is a 'lock' bit which will be set when the data is read and cleared when the write back completes.

Each time a new packet arrives, the HS interface will set the lock bit and simultaneously do a read from each of the 4 stages. The flow container stages are also being read at the same time and the flow id/valid data from these is forwarded to this block. The HS interface will then compare the flow container data to see if the flow id is already stored in the flow container and if it is, will not update the hash stages. If it does not find the flow id in the flow container, it will update the rate with the length of the current packet \* 5(see the rate update section for further details) and write the new rate and unlock bit back to all 4 stages.

The FC interface block is responsible for deciding when flows need to be written into the flow container and for updating the contents of all the flow container fields (see figure 3 for a description of a flow container entry). When a packet arrives, the FC interface set the lock bit in the three flow container stages and simultaneously reads the data from them. When the data is received, the block compares the flow id to each entry to see if it is already stored in the flow container. If the flow is found in one of the container stages, the number of bytes, number of packets, time of last packet, and rate will all be updated. The rate data contained in each of the 4 hash stages is also forwarded to this block. If the flow is not present in the container, but all of the hash stages entries have exceeded the threshold rate, it will try to create a new entry in the flow container. If a non-valid entry can be found in any of the stages, it will updated all of the flow container fields. At the same time, the flow container address is sent to external pins, so a processor monitoring these pins can keep a table of the addresses at which each flow id is stored. If an entry should be created, but all of the flow stages have their valid bits set, it will report this to the processor block which keeps a running count of these drops.

**Figure 3: Flow container entry**

217	216	184	152	128	96	95	63
lock	first packet	last packet	#packets	#bytes	valid	rate	flow id

In order to simplify the rate update process and make it as efficient as possible for hardware implementation a slightly different approach was taken than described in the architecture section. When a packet arrives, the rate is increased by 5\*current packet length. In order to convert this to a rate in bytes/second, the rate update block goes through each cell every 26.7 ms and multiplies the contents by 0.875. This is derived from the fact that we want to multiply each entry by  $e^{(-t/K)}$  where K is 200ms and the granularity can only be on 3 binary places. Over counting the length by a factor of 5 is needed to give the correct rate in bytes/second over a one second time period. The disadvantage of using this technique is that the rate stored in the stages will fluctuate around the correct rate, but the SRAM savings and simplicity it brings to the calculation seem to more than make up for this.

Every 26.7 ms or 3,337,500 cycles, The rate update block starts the update process. It sequentially steps through each hash stage and flow container location reading the contents, calculating the new rate and writing it back. These reads and writes are done through the second port of the memory so that it does not effect the normal operation of the chip. If data is trying to be updated and the lock bit is set, it will send a 'rate update' signal to the FC or HS interface who will then have the responsibility for doing the update since they are in the process of modifying the location.

The final block is the processor interface. This block interfaces with the external processor through read address/read data buses and write address/write data buses. The address space is show in Appendix A. When a write need to be done, the processor will supply the address and data of the register it wished to write and strobe the write enable pin. When a read is done the processor supplies the read address and read enable strobe and the processor block will return the read data with a valid data strobe. There are two 'special' functions which are provided by this block. The first is to read a flow container. This is done by the processor writing to address 452 specifying the address it wants to read and the contents of that location will be returned as read data. As an example if one wanted to read the rate stored in the contents of flow container stage 1 address 5, a value of 805h would be written to address 452. The second function provided is flow container cleaning. When a flow container address is written to register 453, the processor block will invalidate that address in the flow container thereby allowing it to be reused by another flow. The one restriction on both of these functions is that they must not be used while a rate update is in progress to the flow containers. There is an external pin which informs the process when this rate update is in progress (only 0.4% of the time) so that resource conflicts do not occur.

### **3.3 Size and Power estimate**

From a synthesis of the design, the core logic consisted of approximately 450,000 transistors. Assuming a very conservative area utilization due to the routing of the wide data buses, the core logic would occupy no more than a 2mmx2mm on a .18 micron process. The memory estimate was done based a data from Virage for the area of their .18 micron TSMC memory technology. The size of the hash stage memory would be approximately 8.4mm<sup>2</sup> and the flow container would occupy an estimated 21mm<sup>2</sup>. Assuming 20% overhead on top of this, the total die size would be approximately 5.5x5.5mm. The total power for the chip would come in under 1 watt based on the core size estimate and the power data from the Virage memory. Both of these metrics come in well at the low end of today's current IC designs.

## 4. Simulation Results

In order to verify the design was functionally correct, an OC48 internet trace was run on the flow monitor chip at 4x the rate. Due to the slow nature of simulating Verilog, I was only able to run the simulation for approximately 80 ms on 1500 flow ids. This was enough time for the rate update mechanism to be tested to see that it was doing the updates, but did not allow enough time for the rates to settle to a level which accurately reflected the actual arrival rates of the packets. At the end of the simulation, the flow container memories were dumped to a file and figure 4 summarizes the top 6 flow id rates as calculate by the chip versus the actual rates. As can be seen, there is an exact correlation between which flow ids were taking up the most bandwidth, but due to the short simulation time, the rates in the flow containers are not very accurate

*Figure 4: simulation results comparing measured rates versus actual*

Flow Monitor Chip Flow ID	Rate	Actual Flow ID	Rate
0.200.0.1.0.1.0.189	129375	0.24.0.1.0.1.0.23	620750
0.95.0.1.0.1.0.87	129375	0.14.0.1.0.1.0.10	452200
0.103.0.1.0.1.0.93	129375	0.95.0.1.0.1.0.87	450000
0.14.0.1.0.1.0.10	101080	0.103.0.1.0.1.0.93	450000
0.24.0.1.0.1.0.23	94050	0.9.0.5.0.1.0.22	337500
0.9.0.5.0.1.0.22	91875	0.200.0.1.0.1.0.189	337500

## 5. Future Work

There are several areas in which future research could be pursued for this design. First, the simulation performed on the chip were insufficient to prove the accuracy of the rate estimates and determine its performance under actual OC192 traffic. A software model of the chip should be constructed in order to quantify the behavior of the chip on much larger traces. This would also allow the percentage of dropped flows to be measured to ensure that the size of the hash stages and flow container is sufficient.

There are a several short-comings of the current chip implementation which could be improved. The first is the remote management of the flow container. Currently, the processor can only invalidate entries in the flow container. To most effectively utilize the memory, when a flow in an early stage is invalidated and later stages contain valid flows these should be moved to the earliest available stages. Another short-coming is the way in which dropped flows are reported. There is no way to differentiate between packets from the same flow being dropped from the case where many different flows are dropped. If many different flows are consistently being dropped, it is a good sign that the memory is full, but a large burst of packet all being dropped from a single flow may not indicate this at all.

Finally, the current implementation of the hashing mechanism is not very efficient on space. Approximately 1/3 of the core of the design is the registers used to store the hash keys. A different technique of hashing using fewer registers could decrease the size considerably.

## Appendix A

This table defines the address space for the processor accessible read/write registers.

Address(es)	Width	Description
0-63	12	Hash Keys for hash stage 0
64-127	12	Hash Keys for hash stage 1
128-191	12	Hash Keys for hash stage 2
192-255	12	Hash Keys for hash stage 3
256-319	11	Hash Keys for flow container stage 0
320-383	10	Hash Keys for flow container stage 1
384-448	9	Hash Keys for flow container stage 2
449-450	32	Rate Threshold (when rate > threshold flow will be put in flow container)
451	Init	A write of 1 to this register will write 0s to all memory locations
452	16	<p>FC Read Register                      By writing an address to this register, the uP can read the contents of the Flow Containers</p> <p style="margin-left: 40px;">Bits 0-10: Address                      Bit 11-12: Flow Container: 00=FC0, 01=FC1, 10=FC2                      Bit 13-15: Field</p> <p style="margin-left: 80px;">000 = Rate,                      001 = Time first packet arrived from Flow,                      010 = Time Last Packet arrived from flow,                      011 = # packet received from flow,                      100 = # bytes received from flow</p>
453	13	<p>FC Clean Register                      When the uP writes an address to this register, the valid bit in the corresponding flow container will be cleared.                      This should only be written when ru_fc_ip is deasserted.</p> <p style="margin-left: 40px;">Bits 0-10: Address                      Bits 11-12: FC - 00=FC0, 01=FC1, 10=FC2</p>
500	16	Number of dropped flow ids (this will be reset when read).