# Beyond Softnet


Jamal Hadi Salim
*Znyx Networks*
Robert Olsson
*Uppsala University/Swedish University of Agricultural Sciences*
Alexey Kuznetsov
*Swsoft/INR*



## Abstract

The current 2.4 Linux network stack is based on a solid multi−processor−threaded implementation, known as *softnet*, which has been validated over the last 2 years.

Linux is also used extensively in Internet packet forwarding services such as firewalls and routers etc. Linux has a very modular packet processing framework based on the (ingress and egress) Traffic Control Framework as well as the Netfilter IP firewalling architecture.

While the Linux framework for "plugging in" packet services is very flexible, and very much proven in the real world, for Linux to be considered a strong Network Operating System, robustness under all conditions (including severe overload) is a key requirement that must be met.

Our work is to further improve Linux to have the following attributes:

- Robustness at any input rate and any number of input devices.
- Controlled and low Latency.
- cure packet reordering that is inherent with SMP support.
- Provide fairness in greedy network when supporting many interfaces under overload

We discuss the problems, solutions and provide experimental results in our attempts to deal with these issues. While the focus is on using a PC as a router, the solutions provided are applicable to Linux in use in all aspects as a network device (such as a server).


## 1. Introduction

Linux is widely deployed for forwarding IPV4 packets on commodity off−the−shelf PCs in setups such as home gateways and enterprise networks, as well as in fairly large setups[**routeref**]. Linux is also beginning to be deployed in commercial ASIC based Layer 2 and 3 switches[**znyx**] in mission critical carrier class setups. This is attributed to Linux's stability and openness and a dynamic array of features that make it more viable than most commercial offerings.

In this paper, we only talk about Linux's IP capability. In passing we would like to mention that Linux boasts of a lot of other forwarding capabilities, such as frame relay, ATM, Layer 2 ethernet bridging, and perhaps the only SMP multi−threaded DECnet implementation in existence.

### 1.1. Netfilter

Netfilter[**netfilterref**] is an IP packet munging framework that draws its roots in firewalling. It provides several fixed locations in the IP code (as well as DECnet), known as hooks, where packet mungers can be inserted. For example, the IPV4 code has 5 hook locations. Mungers can steal packets to be later re−injected; they can also modify, drop or simply account for something within the packet before letting it continue. Several mungers may treat the packet in sequence before it is re−inserted into the stack.

Netfilter is very similar to the architecture defined in ip filter[**ip filter**], as well as router plugins [**plugin**].

## 1.2. IP forwarding capabilities

Linux provides a very feature rich set for IPV4 forwarding that is in full conformance to RFC 1812 [**rfc1812ref**]. Linux can do IPv4 routing based not only on destination IP address, rather also on source IP address, TOS, and incoming interface.

The ability to deploy up to 255 IP routing tables in conjunction with classification on anything on the header for forwarding rule decisions, makes the policy routing machinery powerful. The Linux policy routing is flexible as the ones provided by commercial vendors [**ciscoref**].

## 1.3. Traffic Control Framework

The Traffic Control Framework, introduced in Linux 2.1 days, is an abstraction that allows packet classification, policing, munging, queuing disciplines, and hierarchical scheduling algorithms to be implemented[**werner**]. The Framework covers both the ingress and egress of a router.

Current implementation covers many classifier algorithms, general QoS(Diffserv[**diffref**] and RSVP); congestion control and queuing algorithms, such as RED, Stochastic Fair Queuing, and just basic FIFO; scheduling algorithms such as a DRR−based CBQ implementation, Token Bucket and a strict priority scheduler.

The Traffic control Framework is very similar to ALTQ[**altqref**] (which was recently integrated into FREEBSD) but more flexible.

## 1.4. Netlink

Netlink is a powerful wire protocol that is used as a messaging system to control and provide asynchronous event notification amongst the different networking modules in Linux.

Netlink messages can go from a service controller in user space (such as a routing daemon) towards the forwarding path in the kernel (e.g. a command to add a route after a route computation). Likewise, a service in the forwarding path can send a multicast message to multiple listeners to inform them of asynchronous events (e.g. when a new link is provisioned).

Although netlink started as a mechanism to emulate BSD route sockets it has evolved into something a lot more powerful in the form of a service messaging system.

## 1.5. Linux Forwarding path

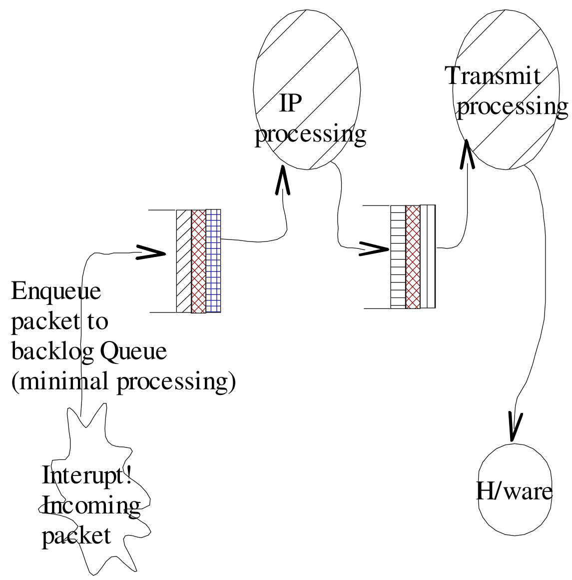


**Figure 1. Linux IP Forwarding path**

Figure 1 shows the classical setup used by many operating systems, including Linux (NT, Solaris and the BSDs fit here as well).

An incoming packet generates an interrupt. The device driver does very minimal work and enqueues the packet on the backlog queue. Later on, a kernel thread is scheduled to pull packets off the backlog queue and call the IP protocol handler (in the case of IP). If the packet is to be forwarded, it is enqueued on the egress queue of the appropriate interface. At some later point, a system transmit thread is scheduled. This thread pulls packets off the egress queue of a scheduled interface and, if approved by scheduling, sends them to be transmitted by the device.

## 1.6. Motivation to scale

In March 1999, Microsoft recruited Mindcraft(www.mindcraft.com) to carry out a comparison between NT and Linux. The test systems included a 4 way SMP and quad network interface setup. The results[**mindcraftref**] revealed, what was already known in the developer community, that Linux networking did not scale in an environment as one used in the testbed.

## 1.7. Linux SMP support

Linux SMP support was introduced in the 2.0 days by Alan Cox. It constituted a giant lock that serialized access to the system.

By kernel 2.2, most subsystems except for the network stack were threaded. If you were to look at figure 1, only one packet at a time could enter the system, despite the number of processors or interfaces attached. In the Mindcraft test, Linux would have done a lot better by having a single huge interface (such as Gigabit ethernet).

## 1.8. Enter softnet and the 2.4 kernel

By August of 1999, after 2.2 was deemed stable, patches started appearing to address the Mindcraft scalability problem. By September of 1999, the patch had found a name−−*softnet* and massive upgrade of the drivers in the Linux kernel was initiated in December of 1999. In February of 2000, the softnet changes were merged into the development kernel, 2.3.43.

One of the most important changes introduced by softnet was the concept of a backlog queue per processor. This, in addition to a round robin interrupt scheduling in Linux, meant that the Linux network stack as of 2.3.43 can concurrently process as many packets as there are processors.

In 2.3.58, IRQ affinity was introduced. This technique means that in an SMP machine, one can dedicate a set of processors to do network processing, by attaching interfaces to the set, while maintaining other processors for other types of workloads.

# 2. Old Problems

Under heavy network load, Linux reaches system congestion collapse.

While investigating issues with how fast Linux 2.3.99 can forward on a Pentium−II based PC [**FFref**] revealed that Linux reaches congestion collapse with input of about 60Kpps. We will refer to this study as FF. At the collapse point, 60K packets per second were coming into the system but none were going out. That study revealed also that the Maximum Loss Free Forwarding Rate (MLFFR) was around 27Kpps, with the CPU being used 100% to process networking and therefore starving any user space processing. In addition, the study showed that any greedy interface, one which receives at a very fast rate, would starve out the others due to the shared backlog queue.

With a feature known as *hardware flow control (HFC)*, supported by some drivers (the 21x4x and the 8390 based hardware), the collapse level is improved but not totally healed. HFC, which has been in the Linux kernel since the 2.1 days, a driver has all its interrupts disabled as the backlog queue is totally filled. The hardware is allowed to interrupt again once the backlog queue is emptied.

FF improves on the HFC solution. A moving window average estimator that samples the backlog queue to provide early system congestion warnings is introduced to improve on the abrupt shutdown that is provided by HFC. The growth rate of the backlog queue is used as a congestion watermark is and passed back to the driver, as a return code, everytime it attempts to queue a packet. The driver then adjusts its sending rate to the stack based on the congestion level fed back to it.

The tulip driver (DEC 21143 chip) is modified to use the feedback congestion levels to adjust its mitigation[1] parameters. As congestion worsens, the hardware is instructed to generate fewer interrupts (and send less packets per unit time up the stack). When the feedback indicates less system load, more interrupts are allowed. This see−saw adjustment will eventually adjust to a steady state of the long term packet rate.

If a driver does not adjust according to the congestion level, the backlog queue is overflown. At that point, hardware flow control kicks in as a last resort.

With this combination, the MLFFR was brought up to 80Kpps and remained constant[2].

The described core changes are in the 2.4 kernel. Only the tulip driver currently supports these changes with only two adjustment levels instead of the dynamic levels. The feature is still, unfortunately, not being used as widely as was hoped.

1 Some literature refers to this technique as interupt batching. Interupts are allowed either after a grace period is exceeded or when a certain number of packets are received in the hardware. The 21143 can wait a maximum of 16 packets.

2Note that these numbers have improved dramatically since 2.3.99−pre8 after some memory management issues were resolved.

## 2.1. Congestion collapse

The main contributor to congestion collapse is *interrupt livelock*. Interrupt livelock was first coined and documented in [**JMKKR96**]. Given the interrupt rate coming in, the IP processing thread (in figure 1) never gets a chance to remove any packets off the system. Essentially, there are so many interrupts coming into the system such that no useful work is done. Packets go all the way to be queued, but are dropped because the backlog queue is full. The result is that all system resources are being abused extensively but no useful work is accomplished. It takes a substantial amount of resources in the form of CPU computation, memory allocation, and the PCI bus bandwidth to discover that a packet needs to be dropped. Note that CPU cache locality also gets heavily trashed by these kind of activities[3].

## 2.2. Analysis of mitigation solution

The HFC and FF solutions both introduce early dropping. When interrupts are delayed or shut down, the DMA ring on the 21143 is filled. Subsequent packets when the ring is full get dropped without generating any interrupts. The result is that no system resources are used when the system is not ready to process packets. To the end to end observer, as the system becomes congested, the latency increases and some packets are lost.

There are still problems with the FF and HFC solution:

1. It is very hardware specific. It depends on the ability of the system to have some mechanism for slowing down, as well as shutting off, interrupts. As far as the authors know, the only 10/100 Mbps ethernet chip capable of this is the 21143[4]. Because this is not a general solution, it becomes difficult to widely deploy.

2. As the number of NICs is increased, a threshold point is reached where regardless of how far away you mitigate, the overall system effect would still result in a collapse. So with an increase number of NICs that are capable of mitigating, the collapse is delayed, but not totally avoidable.

## 2.2. Greedy interfaces

FF attempted several techniques to solve the fairness issue. None was deemed sufficient as a general solution without introducing extra complexities such as adding a lot of overhead in maintaining per device state knowledge.

One solution that was found to improve fairness was to emulate RED[**redref**]. A scheme known as *Random Lie* was introduced. Random Lie observed the average queue length, as measured by the estimator. When a threshold is exceed, the feedback sent to the driver is made to provide information that there was a slightly higher congestion than was really being experienced by the system. This feedback forces the interface to slow down its sending rate. The assumption is that the interface that is sending at a higher rate is most likely going to be hit by the lie. This theory was also proven by experimental data collection.

Under the condition that the system has many interfaces, Random Lie relatively improves the fairness value. A problem was found in the case of a single interface which attempts to send at the system MLFFR. Random Lie slows down the single interface case.

At the moment, the random lie code is in 2.4 but commented out because it does not solve the general case.

## 3. New Problems: Packet re−ordering

Parallelization, as introduced by softnet, is always challenging when done without early demuxing and/or stateful classification. For example, [**BPS99**] points out that the MAE−East exchange point DEC Gigaswitch FDDI scales by adding more parallel ports to increase perceived bandwidth. The Gigaswitch parallelization setup results in head−of−line blocking, which does not guarantee that packets arriving at the switch will leave in the same sequence that they arrived. As far back as 1998, up to 90% of connections had re−ordering problems.

Reordering provides false information to end to end TCP connections (which make 95% of the internet load) that there is network congestion. TCP flows back off and the result is underutilization of the network.

3 Under normal conditions, the standard Donald Becker Linux network driver ammortizes cache locality by sending a batch of packets up the stack. The number of packets is configurable via a parameter that is passed to the driver known as *max_interrupt_work*.

4 All the gigabit ethernet interfaces that have Linux drivers are known to be capable of mitigating, however the current deployed base is mostly 10/100Mbps.

Softnet does, in fact, introduce packet re−ordering. Think of two packets that are going to a client socket arriving back−to−back in a two processor machine. The interrupt scheduler assigns CPU0 to pick that packet, which then sits on the CPU0 backlog queue. The second packet heading towards the same client socket comes in and the interrupt scheduler gives the processing to CPU1. There is no guarantee which of the two CPUs IP processing threads will execute first. It depends on many factors, such as CPU load. If CPU1's thread executes first, it would mean that the second packet will arrive at the TCP and socket level before the first (which came in via CPU0).

Linux has in fact alleviated this problem by implementing RFC 2883[**rfc2883ref**]. As far as Linux TCP is concerned, as long as the remote end is implementing SACK, this problem is highly alleviated, but not totally removed.

One can totally remove this problem by statically attaching interfaces to single CPUs via IRQ Affinity.

## 4. NAPI

Given the issues with the FF solution and the need to be more generic, a new design was put in place. For the lack of a better name we call it NAPI (New API).

### 4.1. NAPI design goals

The authors laid out a set of goals that needed to be met.

1. Maintain the parallellization and scaling benefits of softnet
2. Remove packet re−ordering in SMP.
3. Reduce interrupts on overload to allow the system to peak to a flat curve at MLFFR.
4. Drop Early on overload.
5. Remove or reduce the unfairness issues
6. Balance between latency and throughput.
7. Should not be dependent on any hardware specifics

### 4.2. NAPI SOLUTION

The general overview of NAPI is described in figure 2. It constitutes a mixture of interrupts and polling mechanisms. While polling is useful under heavy load, it does end up introducing more latency under light load as well as abusing the CPU by polling devices that have no packets to offer. On the other hand, interrupts improve latency under low load, but make the system vulnerable to livelock as the interrupt load exceeds the MLFFR.

NAPI offers a middle ground inspired by [**JMKKR96**]. Interfaces are allowed to interrupt on the arrival of the first packet in a batch. They then register to the system that they have work. The interfaces subsequently turn off any interrupts that might be caused by receiving an incoming packet or by running out of receive buffers in a DMA ring. Any arriving packets after the DMA ring is filled will be dropped without disturbing the system. This approach meets design requirement 4.

At some later point, a softirq is activated to poll all devices that registered to offer packets. All interfaces are given an opportunity to send up to a (configurable) number of packets known as quota. Once this quota is exceeded, the device is returned to the end of the work queue, if it still has packets to offer, else the device is taken off the work list and allowed to interrupt again. The quota concept allows for cache amortization and is inherited from the *max_interupt_work* concept in the case of a standard Donald Becker Linux network driver. The quota also enforces fairness which meets design requirement 5.

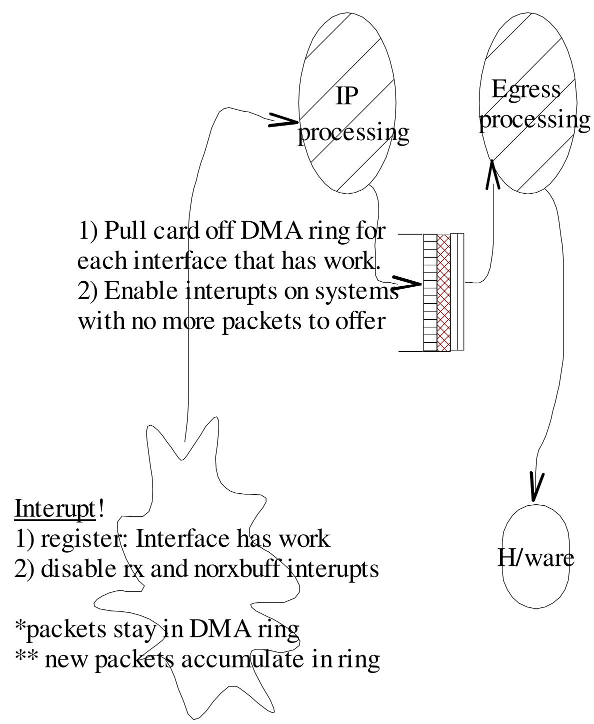


**Figure 2. NAPI data path**

Under heavy load, the system takes its time to poll devices registered. In this sense, requirement 3 is met, in that the MLFFR is dependent on the system capacity and that interrupts are allowed as fast as the system can process them. The only hardware requirement is that an interface is able to own DMA hardware. This ability is a given these days and therefore requirement 7 is met. However, in order to accommodate devices not capable of DMA, the old interface is still available for drivers. A new API is added to the driver interface. Note: this also allows for gradual migration of drivers[5].

As is observed from figure 2, the backlog queue has disappeared. Instead packets are left on the hardware's DMA ring. This enforces serialization of the packet to the system and meets requirement 2. of the design goals.

Requirement 6 is met because NAPI switches between interrupt and poll mode. Under low load before the MLFFR is reached, the system converges towards a system that is interrupt driven (like in the current kernels), so the packets/interrupt ratio is much lower and latency is reduced in the system. In the case of a system under heavy load, the packets/interrupt ratio is higher and the latency is increased.

Requirement 1 is verified experimentally as discussed in the next section.

## 5. Experimental setup

Figure 3 shows the experimental setup used. An IXIA hardware traffic generator[**IXIAref**] is sending from one of its ports to the PC on eth0. Linux routes the packet out eth1 back to the IXIA.

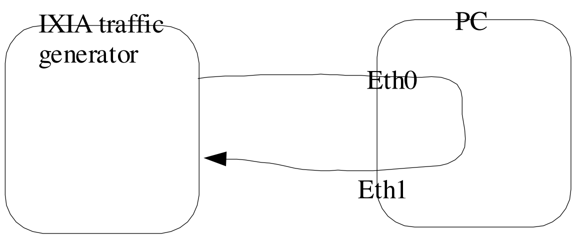


**Figure 3. Experimental setup**

The IXIA is capable of measuring throughput, latency and packet reordering.

The PC is a dual PII 350 Mhz ASUs motherboard with 128M of RAM. The NIC is a Znyx 4–port 32–bit 33Mhz PCI board. Three different variants of kernel 2.4.7 were used: Plain (regular tulip driver), FF (driver responded to feedback), and NAPI (with the changes described above).

The system was not under any load other than the described traffic.

When trying to emulate a uniprocessor setup, interfaces eth0 and eth1 were both tied to one of the processors using IRQ affinity.

Tests were run to test throughput, latency and packet reordering for both SMP and uniprocessor setups.

The initial test was to attempt to validate design requirement 1(refer to section 4.1) . The easiest way to verify this is to run the system under a variety of loads and check that each of the processors had received an even distribution of the packet load[6]. This proved to be the case.

Finally, we run several tests (one for each of the kernels):

1. A single CPU test with the IXIA recording both the throughput and latency results

2. A repeat of the above with SMP. In both this and the above test, each trial set constitutes sending 4 Million packets in total.

3. An SMP test for packet reordering. In all the tests 17203 packet were sent in each.

## 5.1. Throughput results

Figure 4 below shows the results with a single CPU being used. The regular Linux 2.4 without any

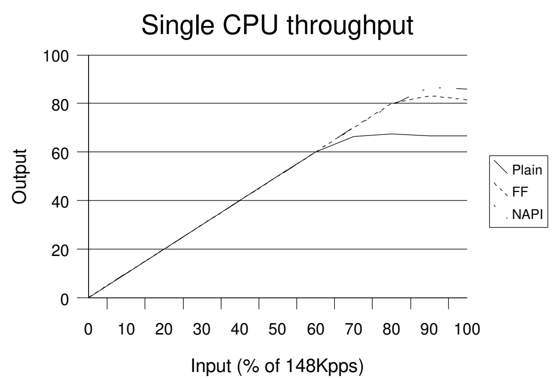


**Figure 4. Single CPU throughput results**

5 Is thus the first time backward compatibility has been done in Linux? ;–>

6 /proc/net/softnet_stat records the appropriate stats.

spicing appears to perform rather well reaching a MLFFR at 65% input. The experiments did not measure the available CPU on the system during the test period. However, a file listing of a directory took about 3−4 times longer with the plain setup than it did with either the FF or the NAPI version. The NAPI version does better in the single CPU case than FF.

The SMP results in figure 5 indicate the same behavior as in the case of the single CPU.

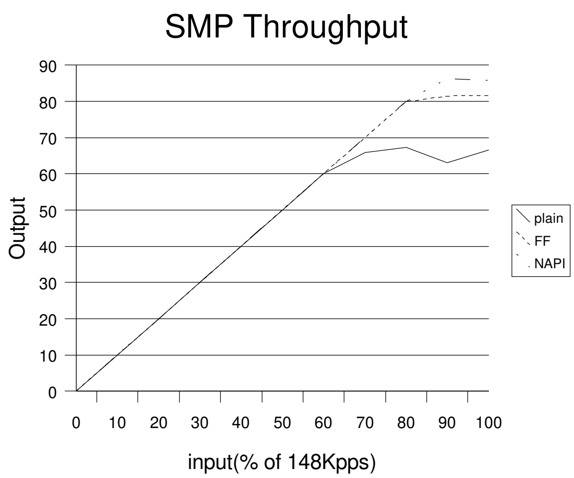


**Figure 5. SMP throughput results**

## 5.2. Latency results

The latency results shown in figure 6 were extracted from the same tests that are shown in the throughput graph of figure 4.

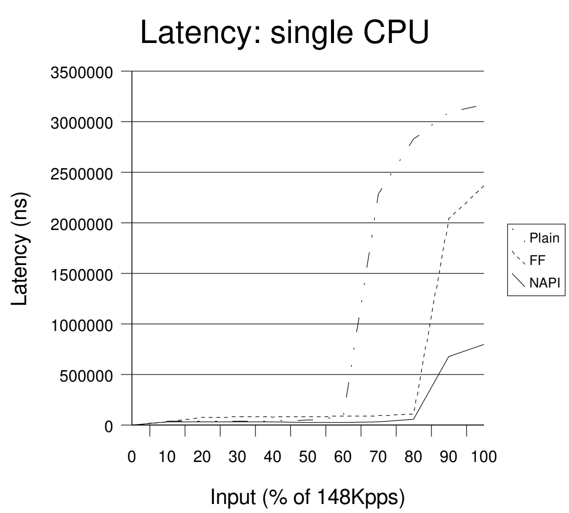


**Figure 6. Latency results for single CPU**

We observe that NAPI does very well, with latencies of less than 1 ms in the worst case. FF does show a worst case behavior of over 2 ms and the plain kernel, as is expected, the highest latencies approaching 4 ms in the worst case.

We see the same behavior in the case of SMP enabled system in figure 7. Figure 7 is extracted from the same test runs as those in figure 5.

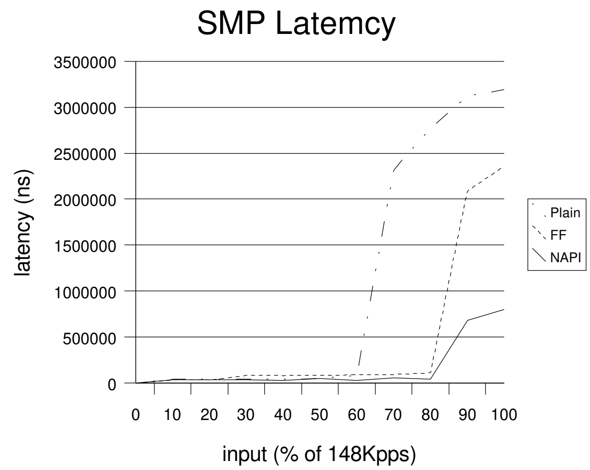


**Figure 7. SMP latency results**

## 5.3. Packet re−ordering results

In this test 17203 packets with incrementing sequence numbers are sent. On the receiving side, if any out of sequence packets are found, they are flagged (and accounted for.

As was expected there were no packet re−ordering issues with NAPI. FF does a lot better than the plain driver.

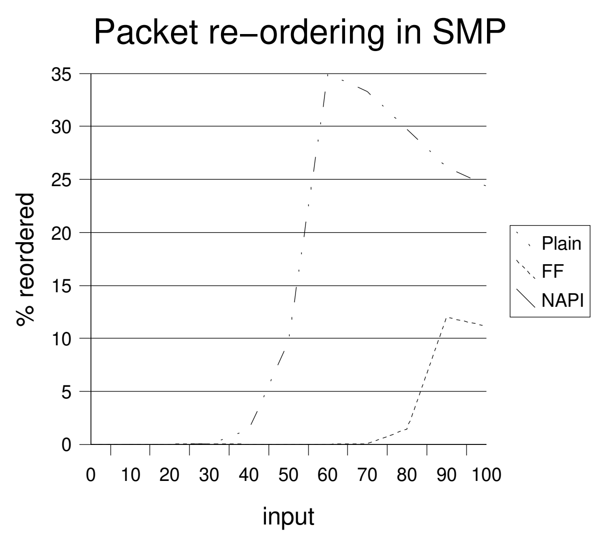


**Figure 8. SMP packet re−ordering results**

In all the tests, the throughput results matched those iu the SMP throughput tests.

## 5.4. Other results

With the Znyx 4 port card sending on two of the interfaces and receiving on the other two, we were able to reach throughput rates of 200Kpps on the SMP system used in the testbed.

Using two Intel e1000 1000Mbps NICS on a single CPU motherboard. The CPU is PIII at 933 MHz using a ServerWorks Chipset, we were able to send over 360Kpps.

None of the above tests were thoroughly analyzed and should be considered as work that is still ongoing.

## 5.5. Miscellaneous

A side benefit of reducing the number of interrupts the system experiences is a huge reduction in the system load caused by interrupts.

Profiling from a live router[**profref**] using FF in the bifrost[**bifrostref**] indicate that the interrupt setup and the interrupt code itself was the most CPU intensive.

## 6. conclusion

Linux IP packet forwarding is handled with several mechanisms, such as netfilter, netlink, the traffic control framework, and softnet. These tools have been validated in real networks[**routerref**], as Linux is commonly used for tasks such as firewalling and routing.

However, under heavy load, Linux can encounter a system congestion collapse. This limitation was noted in [FFref], a study that suggested and demonstrated an improvement to hardware flow control, termed FF. Although FF gives a definite advantage, it suffers several problems. Notably, it is hardware specific and it does not scale to an arbitrary number of NICs.

NAPI is a more advanced solution to the issues addressed by FF. NAPI features a hardware–independent design comprising interrupts and polling mechanisms. The balance between the two techniques makes NAPI effective under light or heavy loads. NAPI maintains the scaling benefits of softnet, removes SMP–induced packet re–ordering, and promotes fairness across interfaces.

## 8. References


- znyx: http://www.znyx.com/products/netblaster/zx4500open.htm
- netfilterref: http://netfilter.samba.org
- plugin: http://www.tik.ee.ethz.ch/~crossbow/rp/
- ipfilter: http://coombs.anu.edu.au/~avalon/ip–filter.html
- rfc1812ref: http://www.ietf.org/rfc/rfc1812.txt?number=1812]
- ixiaref: http://www.ixiacom.com/
- ciscoref: http://www.cisco.com/warp/public/cc/techno/protocol/tech/plicy_wp.htm
- altqref: http://www.csl.sony.co.jp/~kjc/software.html#ALTQ
- differeff: http://diffserv.sourceforge.net/
- BPS99: J.C.R. Bennett, C. Partridge, and N. Shectman, Packet Reordering is Not Pathological Network Behavior, IEEE/ACM Transactions on Networking, Vol. 7, No. 6, December 1999, pp. 789–798.
- Werner: ftp://icaftp.epfl.ch/pub/people/almesber/pub/tcio–current.ps.gz
- mindcraftref:http://www.kegel.com/nt–linux–benchmarks.html
- Ffref: http://robur.slu.se/Linux/net–development/jamal/FF–html/
- JMKKR96: J. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt–driven kernel, " in Winter USENIX Conference, January 1996.
- profref: http://robur.slu.se/Linux/net–development/experiments/010313
- RFC2883ref: ftp://ftp.isi.edu/in–notes/rfc2883.txt
- bifrostref: http://bifrost.slu.se/index.en.html
- routerref: http://robur.slu.se/Linux/net–development/papers/linuxrouters–in–use.html
- redref: http://www.aciri.org/floyd/abstracts.html#FJ93