# Achieving Gigabit Performance on Programmable Ethernet Network Interface Cards

Wallace Wadge

May 29, 2001

**Abstract**

The shift from Fast Ethernet (100Mbit/s) to Gigabit Ethernet (1000Mbit/s) did not result in the expected tenfold increase in bandwidth at the application level. In this dissertation we make use of programmable Ethernet network cards running at gigabit speeds to identify present bottlenecks and also propose two new techniques aimed at boosting application performance to near-gigabit levels whilst maintaining full compatibility with existing systems. Furthermore, we investigate the performance of the PCI bus and the throughput "on-the-line" when using different frame sizes.

## 1 Conventional Systems

In a conventional system, the Network Interface Card (NIC) performs a very small part of the work required, leaving the rest for the host to handle. A typical NIC makes use of interrupts to signal to the host that new data has been received from the network. It is then the responsibility of the OS to issue the appropriate calls to fetch the data from the correct memory location, perform further processing on it (such as TCP protocol handling), and eventually copy the payload (that is, the useful data) for the processes requesting it. This system contains a number of problems, some of which are detailed below:

**Interrupt Driven** The whole system is interrupt driven which means that context switching between kernel mode and user mode is required for each packet received[1] from the network, resulting in unnecessary overheads. Under heavy load, the host might find itself flooded with interrupt requests.

**Inappropriate resource accounting** The NIC has no knowledge of underlying processes and will generate an interrupt just as soon as it receives data. This results in a scenario whereby the processing time required to handle the incoming data may be attributed to processes which are not involved in the underlying transfers.

**Lack of load shedding** If the host is unable to cope with the incoming influx of packets, it has no alternative but to drop them. Unfortunately, dropping packets cannot occur without some further host processing, which may result in receiver livelock being triggered (that is, a system doing nothing except drop incoming packets).

[1]Some newer network cards are able to coalesce some packets together prior to triggering an interrupt

**Checksum calculation** Research suggests that a host system may spend as much as 15% of the total processing time required to calculate the checksum of each packet[7].

**Multiple memory copies** The running user application must explicitly perform a system call in order to retrieve the data received from the network. This has the effect of copying the internally queued data to the applications' address space and finally de-allocating the memory reserved for the copy. Trivially, such a step is logically redundant since there is never any need to maintain multiple copies of the same data. Furthermore, copying data around wastes considerable processing resources.

There are some other drawbacks in the conventional system, but these alone are enough to warrant the necessity for a completely different implementation scheme. Research has shown that, under heavy load, a host system may spend as much as 90% (and in some cases, more than that) of the total processing time available for handling communication details leaving a mere fraction of processing time for other processes[2]. Throughput also drops to just 409Mbits/s at the application level[7] (when utilizing gigabit NICs), which is a far cry from the desired 1 Gbit/s the new Ethernet standard proposes. Some vendors have proposed increasing the payload size of each Ethernet packet from the standard 1.5K length up to 9K. Since the header length of each Ethernet packet is constant, there is less overall overhead cost for longer packets. This is equivalent to travelling on a highway where only mid-sized cars are allowed. At each toll-station where thousands of cars are processed by an attendant, each car must wait an inordinate amount of time to get through. As traffic increases, so does the time it takes to get through the toll station. This time could be dramatically reduced if the passengers in cars could pass through the toll station on buses. This would reduce the number of vehicles processed and shorten overall delay for each passenger. Several independent tests have demonstrated that performance improves considerably while at the same time a significant drop in host CPU loading could be evidenced. However, many in the industry (of special note, the IEEE) are reluctant to change the tried-and-tested Ethernet standard just yet. Most agree that if such a change where to take place, then it should no longer be called Ethernet. In this dissertation we identify the major bottlenecks and attempt to get the best of both worlds, that is, maintain the standard 1.5K packet size whilst boosting performance.

## 2 Background

### 2.1 Hardware tools

New NICs are starting to appear on the market which, unlike their conventional counterparts, can be programmed extensively to perform tasks normally reserved for the host to handle. In this dissertation, we make use of such a programmable network cards (more specifically, we use the *Alteon Tigon NIC*).

The NIC contains two, RISC-based, embedded processors together with 1Mb of SRAM. It also contains two independent DMA engines (one for moving data from the NIC to the host and the other to perform the transfers in the other direction), as well as some other hardware-assist features such as byte-steering logic to allow transfers to start and end on any byte. The NIC runs a program which is downloaded upon initialization time, such program being referred to as the *firmware* to avoid confusion with the host driver. There is no need for any PROM swapping or special utilities to update this firmware; the host merely maps the firmware program to the correct memory locations. It is up to the firmware to control the hardware; including manipulating buffers and buffer descriptors, controlling the DMA engines, calculating checksums, generating interrupts to the host and instructing the hardware to transmit or receive in some format. The standard firmware supplied also performs some useful work such as calculating TCP/IP checksums on-the-fly.

In this dissertation, we modify or extend the supplied firmware to either provide additional functionality or else to perform some investigative tests.

### 2.2 PCI Bus

The NICs available make use of the standard PCI bus to affect all transfers to and from the host. The PCI standard[1] defines two possible bus speeds (33Mhz and 66Mhz) and two different bit-rates (32-bit or 64-bit), all of which are accepted by the *Acenic Tigon NIC*. However, in this dissertation we make use of the standard 33Mhz/32-bit variety. An important point to highlight regarding the PCI bus is the fact that each transfer incurs an overhead cost which is not proportional to the number of bytes transferred. In other words transfers of larger sizes are more efficient. With this bus rate, the maximum throughput possible is about 125MB/s, which is little more than our 1Gb/s target.

### 2.3 Host interaction

The standard firmware uses a model similar to the Virtual Interface Architecture standard (VIA)[6, 3] to communicate with the host. This means that whenever the host wishes to send or receive data to or from the NIC, it uses a system of message descriptors organized in a ring fashion, using doorbells to signal their presence. Interrupts can be generated by the NIC's firmware, but we mask them off to improve performance, relying instead on polling. The NIC assumes a flat memory model and therefore does not take into consideration the page swapping techniques in use by the OS. Furthermore, processes usually make use of virtual addresses rather than the physical addressing system the NIC expects. To circumvent this problem, the *bigphysarea* kernel extension together with the *consequetive* (*sic*) driver is used to pin-down a chunk of memory at boot time. The OS sees this area as just another I/O device and does not perform any page swapping techniques on it. Upon initialization, this memory is mapped into user-space making it available for user-level processes. Since it is also very easy to calculate the physical memory locations within this consecutive memory, we are able to communicate with the NIC without the need for invoking the OS at any stage. In fact, the OS does not feature in any of our tests (with the exception of setting the link up during initialization). This also enables zero-copy communication with data transfers being of order O(message_length). Zero-copy communication enables a process to avoid memory to memory copies hence boosting performance, however it comes at a price since such memory is no longer hidden and protected from other processes. Various techniques have been proposed for better control of this type of memory[4, 5].

## 3 Results

### 3.1 Theoretical Limits

A number of key areas were investigated, each of which could be a potential bottleneck. Firstly however, we calculated the maximum theoretical Ethernet throughput which could be obtained when using different frame sizes. These are partially reproduced below:

| True Frame size (bytes) | Real Payload (bytes) | Max Efficiency (Mbits/s) |
|---:|---:|---:|
| 792 | 754 | 952.0202 |
| 1048 | 1010 | 963.7405 |
| 1536 | 1498 | 975.2604 |
| 9096 | 9058 | 995.8223 |

Table 1: Ethernet Theoretical Maximum Effeciency

In the above table, the true frame size refers to what is actually being transmitted, that is, the entire packet together with the interpacket gap and preambles. The real payload refers to our "useful" data, that is, the data which will be eventually used by the processes. Finally, the maximum efficiency refers to the best throughput we can ever hope to achieve. This implies that the maximum throughput we can achieve when using the standard 1.5K Ethernet frames is 975.26 Mbits/s and around 995.82 Mbits/s when using the extended 9K frames. Having determined our maximum limits, we next verify if the hardware can live to its claims by testing out the NIC to NIC line speed.

### 3.2 NIC to NIC line speed

Overheads are involved in transferring normal packets from the host to the NIC and vice-versa. However, since we need to test the *maximum* line to line speed using different payload lengths, we modify the firmware to do no task except send and receive packets at the fastest rate possible, the packets being generated by the NIC itself.

The NIC provides a hardware timer register which may be read and set by the firmware. We make use of this register to keep track of the elapsed time in order to eventually calculate the bandwidth obtained. This is necessary since if we were to rely on the host to establish the time elapsed, the time delay for the signal to reach the host would distort our results. The firmware also keeps track of the number of packets transmitted and received. The elapsed time and the number of packets transmitted is read by the host at fixed intervals which then outputs the results.

| Payload length (bytes) | Bandwidth (Mbits/sec) |
|---:|---:|
| 1498 | 975.0565 |
| 9054 | 995.1155 |

Table 2: Line Speed throughput for different payload sizes

Results show that the hardware is able to achieve our expected theoretical limits without incurring any significant overhead costs[2]. The maximum packet size "on the line" is set at 64K, however, CRC error detection routines become less effective beyond 12K. Furthermore, larger packet sizes incur longer latencies as well as an increased chance of corruption. Nevertheless, in our tests we have verified the throughput obtained when utilizing large frames as well as the normal-length packets.

[2]Unfortunately, hardware limitations dictate the clock accuracy of all our timings.

## 3.3 PCI test results

A memory read or write requires the use of the PCI bus. To access any data, the hardware requires the setting up of a transaction involving the specification of the address to start reading or writing to, together with the number of bytes to be transferred. This "setting up" involves an initial startup cost. Results show that the rate at which the PCI bus is able to transfer data between the host and the NIC may not necessarily be equal to the rate of moving data in the other direction.

### 3.3.1 Host to NIC

In this test, we modify the firmware on the NIC to perform a number of DMA transfers (hence utilizing the PCI bus) in quick succession. The host software is implemented to give the initial command together with the appropriate parameters (DMA transfer length and the number of times to perform the test). When the NIC's firmware receives the command, it starts controlling the DMA engines each time transferring a number of bytes from the hosts' memory to it's internal memory. When all the necessary transfers are complete, the NIC sends an event to the host together with the elapsed time (as before, the NIC is responsible for establishing the time elapsed). The host then calculates the results after performing averaging operations on multiple transfers.

### 3.3.2 NIC to Host

This is similar to the test performed above except that we reverse the transfer direction. Again, it is up to the NIC to perform most of the work. Since a continuous host polling for the completion event may interrupt the PCI bus, the host is set to check for the results only after a number of seconds have elapsed. This does not affect results since the NIC would have completed the test in question with the elapsed time taken and stored, ready for retrieval.

### 3.3.3 Conclusions of test

Results clearly indicate that at lower sized transfers, performance drops considerably. For example, if we look at the NIC to Host results, we find that there is a difference of more than 38.6% between standard 1.5K packets and the proposed 9K frames.

| Transfer size (bytes) | Bandwidth (Mbits/s) |
|---:|---:|
| 1512 | 665.6390 |
| 9072 | 922.8537 |

Table 3: NIC to Host

## 3.4 Standard firmware tests

The tests mentioned in §3.3 only test out the performance of the PCI bus; overhead costs of parsing an Ethernet header and queueing it for transmission are not calculated. The next tests investigate the maximum throughput possible in transmitting complete Ethernet packets from one side to another. Since we are testing the transmission and reception routines separately, we modify the firmware to either generate the packets (as we did to test line speed), or alternatively to receive them. In other words we create a setup to send packets from the host to the NIC at the other end, or from the NIC to the host as the final destination.

### 3.5 Host to Host

With the above tests complete, we are able to test out the complete communication chain — transmitting from host to host. Figure 1 shows a summary of all our results so far (with standard 1.5K packets). The bars are ordered as follows: the Ethernet theoretical limit, card to card throughput test, throughput obtained to transfer data from the host to the NIC and vice-versa, transmitting or receiving from a NIC using standard firmware and host to host transfers using standard firmware.

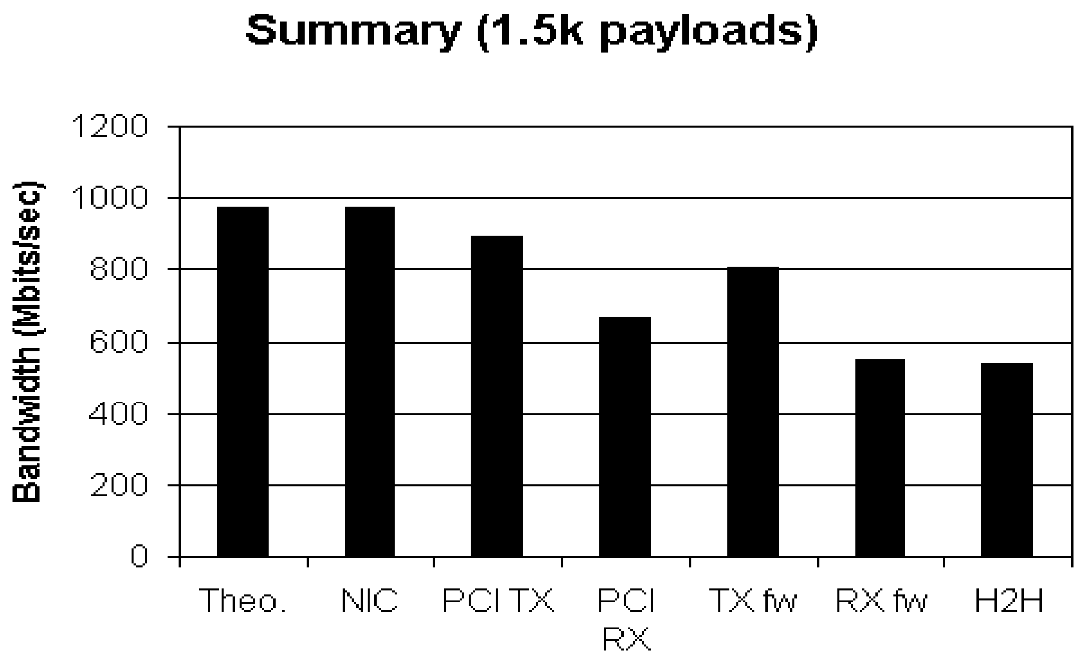

Figure 1: Summary of results (1.5K payloads)

## 4 New Techniques

It is evident that the real bottleneck in the communication cycle is the PCI bus. Normally the NIC performs a DMA transfer of the same size as the received (or transmitted) packet.

We modify this behavior by introducing two new techniques, which we call *packet fragmentation* and *packet coalescing* for the transmitting and receiving side respectively.

### 4.1 Packet fragmentation

We have shown that since PCI transfers involve an initial startup cost *for each transfer*, smaller transfers are less efficient than longer packet transfers. In this technique, the host constructs a single large “chunk” containing a number of smaller (1.5K) packets. The firmware then breaks this chunk up into a number of fragments and transmits them out on the network. Since the maximum message size which may be transferred over the PCI bus is 64K, the host may transmit up to 43 packets in one go. Message descriptor flags also make it possible for the standard system and the fragmentation technique to co-exist. Figure 2 shows that performance has been boosted right up to the Ethernet theoretical limit, mirroring the results we obtain when using large payload sizes. The host to NIC segment shows the throughput obtained over the PCI bus when transferring messages of different transaction lengths. The second graph shows the number of bytes transferred (again over the same PCI bus) when the number of fragments per chunk is altered. Notice that while the PCI throughput is able to achieve 1Gb/s, the throughput obtained by fragmentation suddenly tapers off as it hits the theoretical limit boundary.

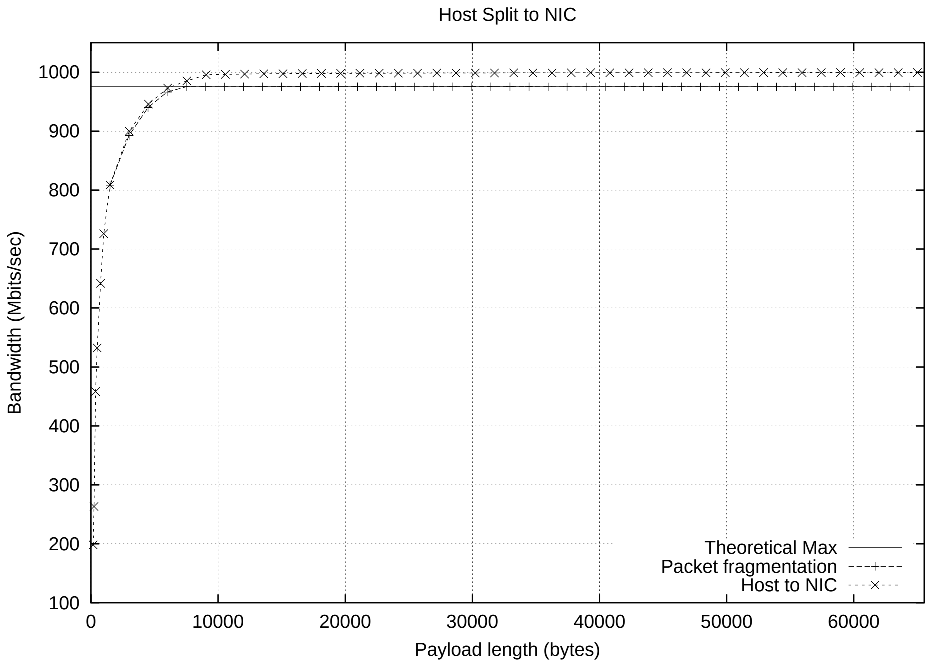

Figure 2: Packet Fragmentation vs Host to NIC

## 4.2 Packet Coalescing

For the receiving side we adopt the same rationale but use a different technique. In packet coalescing we delay transfers from the NIC to the host until a number of packets have been received. The host just allocates a larger chunk of memory and sets a special flag in the message descriptor (again this is done to allow both systems to co-exist). The NIC then “joins” a number of received packets together and transfers the packets when there is no more available buffer space or when a timeout expires. For example if the host wishes to receive 42 packets of 1.5K size in length at one go (the maximum possible), it first allocates enough consecutive memory to hold them all. Next, the NIC is informed that packet coalescing is required via a flag in the message descriptor. As a general rule of thumb, the higher the number of packets coalesced, the better performance is obtained since larger

PCI transfers are used; however, this causes increased latency. We therefore modify the firmware further in a way as to trigger an alarm (via the use of the onboard timer register) after a host-specified amount of time, this acting as a signal to transfer the pending data. For example if 3 packets were received and there is still space for 10 more, timer expiration would ensure that the host would still get those initial 3 packets without waiting for the rest. This system therefore lets the host increase performance whilst controlling latency. Once again results show that the Ethernet theoretical limit is achieved whilst maintaining full compatibility with the existing systems. Figure 3 shows the effect of packet coalescing when coupled with packet fragmentation.

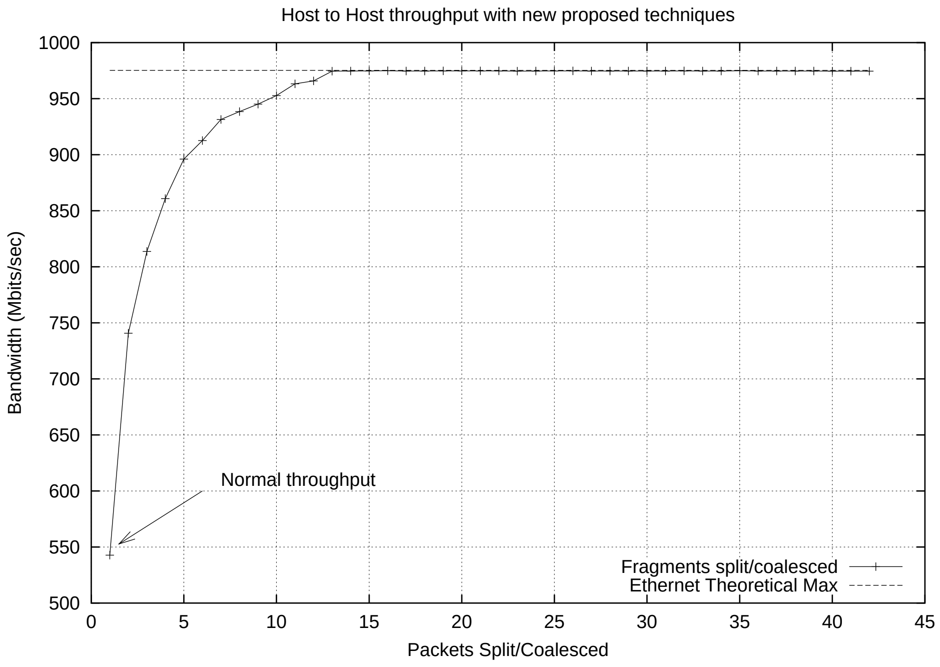

Figure 3: Packet Coalescing

## 5 Conclusions

This dissertation has verified that the Ethernet theoretical limits can indeed be reached with existing hardware. Furthermore, we have located one of the chief bottlenecks in the entire communication loop, the PCI bus. Two techniques have also been devised and implemented on programmable NICs to demonstrate that, with some minor changes, performance can be boosted dramatically without resorting to new Ethernet standards requiring a change in all intermediatory routers. The effect of varying frame sizes (including the extended Jumbo frames) on the physical line and on the PCI bus was also analysed and zero-copy techniques were implemented.

## References

[1] PCI Special Interest Group, *PCI Local Bus Specification Revision 2.1.* 1995, June

[2] Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi and Yutaka Ishikawa. *The Design and Evaluation of High Performance Communication using a Gigabit Ethernet*

[3] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. *U-Net: A user-level network interface for parallel and distributed computing.* In Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995

[4] N.J. Boden,D. Cohen, and W.-K. Su. Myrinet: A Gigabit-per-second Local Area Network. IEEE Micro, 15(1):29, February 1995.

[5] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. *Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication.* In 12th Int. Parallel Processing Symposium, pages 308–314, Orlando, FL, March 1998

[6] Compaq, Intel, and Microsoft Corporations. *Virtual Interface Architecture. Version 1.0*, December 1997. Available from www.viarch.org.

[7] Alteon Networks, *Extended Frame Sizes for Next Generation Ethernets - A White Paper*