

Colorado Technical University



**Technical Report
Computer Science**

Increasing End-Point Performance in a Congested Network

Danny L. Maupin
Senior Software Engineer
FLUKE Networks Incorporated
dmaupin@tc.fluke.com

Bo I. Sanden, Ph. D.
Colorado Technical University
bsanden@acm.org

**Technical Report Number
CTU-CS-2005-005**

Increasing End-Point Performance in a Congested Network

Danny L. Maupin
Senior Software Engineer
FLUKE Networks Incorporated
dmaupin@tc.fluke.com

Bo I. Sandén
Colorado Technical University
4435 N. Chestnut St., Colorado Springs, CO 80907
bsanden@acm.org
<http://home.earthlink.net/~bosanden>

Abstract

Network congestion is mostly caused by the software in the end points. More intelligent device drivers can examine incoming packets and drop those of no interest to the own end point. A network interface management algorithm (NIMA) is proposed that resides in the network driver and selects which packets are submitted to the protocol stack.

Measurements of the end point behavior show that the maximum throughput on a 1000 Mbps network can be achieved across the physical layer. Tests at the data link layer show that the end-point stack components without NIMA can handle at most 80 Mbps and only 8 Mbps with minimum-size packets. With NIMA, the maximum throughput was 200 Mbps for large packets and 33 Mbps for small packets.

1 Introduction

Hardware network technology is improving faster than the software that handles the traffic. In particular, there are inefficiencies in the end points' software for capturing and evaluating the traffic. Latency within the protocols caused by handling traffic that is not destined for the own end point is a problem in all networking protocol implementations.

Protocols such as TCP throttle the sending end point to control the congestion, but with the increase in traffic in the higher speed networks, the implementations fall short. Excessive throttling and retries within a specific conversation can cause undue latency and therefore application performance problems and also add to the network congestion problem.

This paper shows how smarter, managed end points can increase the number of packets that are successfully processed in the end point. A *network interface management algorithm*, *NIMA*, which resides in the device driver of the network interface, is presented as a solution to the end-point congestion problem. The driver selectively presents packets to the network stack, which results in more packets being processed successfully.

The behavior of the end point with and without NIMA was measured by means of Fluke Networks™ testing equipment. The measurements show that the maximum throughput on a 1000 Mbps network can be achieved across the physical layer. Tests at the data link layer show that the end-point stack components without NIMA can handle at most 80 Mbps and only 8 Mbps with minimum-size packets. With NIMA in the device driver, the maximum throughput was 200 Mbps for large packets and 33 Mbps for small packets.

1.1 Packet handling in the end point

In most operating systems, the device driver receives an interrupt when a packet has entered the end point. In older systems, an interrupt is generated for each packet. In newer systems, the device driver disables the interrupt and retrieves packets until none are left on the network device.

Fig. 1 shows the queuing of packets by Ethernet device drivers. The devices can be network interface cards or custom hardware solutions. The device driver attempts to place each packet in the *backlog queue*. The backlog queue function notifies the driver if a packet cannot be queued. The driver then drops the packet. Most implementations of this queuing mechanism notify the driver of different levels of congestion, that is, the percentage of the queue's usage at any given moment.

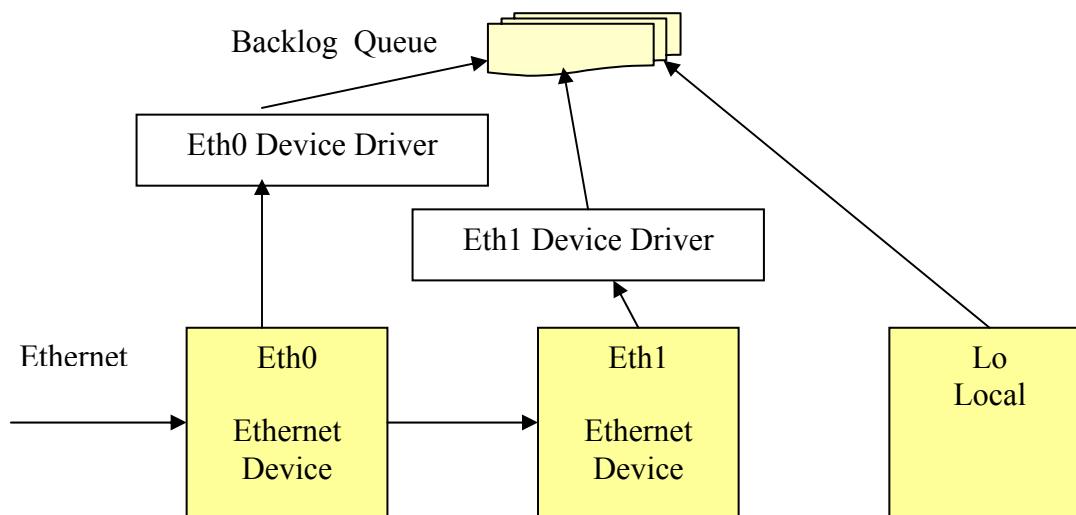


Fig. 1: Packet Movement in the Data Link Layer

Using the information contained in the packet, the backlog queue management algorithm notifies the correct network protocol by means of callback. The queue manager can either give the pending packet to the correct protocol or notify the protocol that it has a packet in the backlog queue.

IP packets are sent to the IP protocol in the network layer. Several protocols can use IP as their transport mechanism. IP sends TCP packets to the TCP layer's queue. TCP keeps track of the applications and the ports that they are communicating on. Applications that have opened a communications channel via TCP receive the packets meant for them.

As the amount of traffic increases on the physical layer, the congestion in the end point increases. In the case of TCP traffic, the increased traffic can cause TCP's congestion control mechanism to notify the sender either to slow down or cease transmitting. For traffic that is not TCP friendly, the only mechanism is that packets are dropped when the backlog queue becomes full. (A *TCP friendly* protocol is one that provides some form of congestion control.)

1.2 Measuring throughput

Throughput can be measured at different levels of the communication stack. Because of the lack of proper instruments to measure throughput on the wire it is usually measured before transmission and after reception by the end point. This can greatly distort the throughput calculations. Operating systems, processor limitations and end-point buffering inefficiencies affect the measurements.

By calculating the actual network throughput and the throughput in the end point stack this study shows that end-point congestion is the root cause of decreased throughput on higher speed networks. The major bottlenecks are not networking components such as routers, switches, hubs or network interface cards, but the end-point software.

2 Network Congestion

In a normal network, numerous end points and network components compete for the network resources. If the network has too many end points and network components contending for these resources, congestion can occur. There have been many attempts and schemes to control congestion in the end points.

Most end points operate in the *promiscuous* mode. This means that the end point allows all incoming traffic into the backlog queue. Because packets are not filtered, the queues tend to overflow more quickly. Some hardware interfaces can provide packet filtering thus rejecting any traffic that is not meant for this end point. For this, the operating systems as well as the device driver have to be aware of the feature and be able to control the hardware interface.

Queues are a main cause of network congestion. Many protocols use queues to buffer packets until the processing application can handle them. If packets arrive at a higher rate than the application can handle, the queue eventually fills, and subsequent packets start to be dropped.

Having initially larger queues or enlarging the queues dynamically works for bursts of traffic but causes excessive latency with sustained traffic flow. Larger queues require more processing of each individual packet. Large packets take up multiple allocation units, which are chained together. Each packet is also sent between the different layers in the end point's protocol stack. All this take up processor bandwidth.

2.1 Network Congestion Management

There are two general approaches to network congestion management: *congestion recovery*, which reacts when the demand exceeds the capacity of the end point, and *congestion avoidance*, which attempts to stop congestion from occurring. Congestion can be avoided either by managing end-point protocols or by managing the interfaces themselves. If you manage the protocols, you detect congestion after receiving the packet into the protocol stack. If you manage the interfaces, a packet's fate is determined in the device driver.

Most TCP implementations provide some congestion avoidance mechanism [6]. In *active queue management* [7], you resize the upper layer protocol queues (such as the TCP queues) or drop packets based on some criteria.

2.2 Flow Control Mechanisms and TCP

With the first TCP implementations, the Internet was prone to “congestion collapse” where the sending end point would often overwhelm the receiving end point’s queue(s). Van Jacobson [4] developed a congestion control mechanism for TCP based on queue usage and overflow. If a packet is dropped the receiver throttles the sender by setting the sliding-window value.

TCP can have serious performance problems when it comes to dropped packets in any one data flow. Retries can become excessive and communication timeouts can occur. If a timeout occurs while an application is communicating via TCP, the whole conversation ceases with an error condition.

The time from a packet’s transmission until acknowledgement is received is referred to as the round trip time, RTT, for the given network segment. The TCP timers are usually dynamic and based upon the RTT. They are sometimes inefficient. If they are set to short, resends can happen even if the receiver has successfully processed a packet. Setting a timer too long causes long delays if packets are being dropped.

2.3 Random Early Detection (RED)

Random Early Detection, RED, is an active queue management technique that drops packets on the probability that the queue will get full [1]. If the queue has been mostly empty recently, RED will not drop packets until the queue overflows, but if it has been full, RED starts dropping incoming packet on the assumption that the queue is going to overflow. “RED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection’s share of the bandwidth through the gateway. RED gateways are designed to accompany a transport-layer congestion control protocol such as TCP” [2]. A drawback is that arbitrary packets are dropped causing retransmission and adding to the congestion,

2.4 Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) uses a “congestion notification bit” [5] in either the IP or the TCP header, where the network components can tell the sender to slow down or halt. ECN is usually implemented in routers or other network components and not at the end points. “One advantage of ECN mechanisms is in avoiding unnecessary packet drops, and therefore avoiding unnecessary delay for packets from low-bandwidth delay-sensitive TCP connections.” [3].

ECN can only work if the end-point protocol implementations are changed to recognize the congestion bit or the ECN algorithm modifies the sliding window in the resulting ACK in TCP to reflect the congestion level. ECN implementations in the network components such as switches and routers could prove very effective, but in high bandwidth networks, this would not alleviate the congestion in the end points. The end points cannot be relied upon to correctly monitor network congestion because of the inefficiencies of the queues and the operating system internals. The end points may view the network as being congested when in fact it is performing well.

3 Traffic Flows

The network interface management algorithm, NIMA, provides for active queue management in the device driver. NIMA’s efficiency depends on the mix of packets that the end point receives. To show that

it's efficient, we have to establish a typical traffic flow. A sample was taken from a typical network with 125 end points. Several networks in larger corporations were sampled and show that the protocol statistics are representative.

The following tables show different aspects of the traffic flow over a five-minute period. Here are the statistics for the network and transport layers with a sampling interval of 5 seconds. (Transport layer protocols are indented.) TCP traffic accounts for only 9% of the total traffic on this network segment. The remaining protocols may or may not have built in congestion management.

| Protocol | % of packets | Packets |
|---------------|--------------|---------|
| MAC (All) | 100 | 20331 |
| IP-V4 | 73.6 | 14973 |
| TCP | 8.8 | 1792 |
| UDP | 39.5 | 8030 |
| ICMP | 25.3 | 5141 |
| IGMP | 0 | 10 |
| IPX | 3.0 | 604 |
| ARP | 21.7 | 4406 |
| Spanning Tree | 1.0 | 199 |
| Other | 0.3 | 65 |
| NETBEUI | 0.3 | 64 |
| CDP | 0.1 | 26 |
| Cisco VTP | 0.1 | 27 |
| EDP | 0 | 7 |

Here are the protocol statistics for the network layer with a sampling time of 1 second. Most of the traffic on the network segment is IP, which is representative of most modern networks.

| | |
|-------------|--------|
| IP | 77.11% |
| IPX | 2.17% |
| ARP | 19.19% |
| IBM NetBEU | 0.32% |
| IEEE 802.1D | 0.72% |
| CDP | 0.09% |
| DISL | 0.09% |
| Other | 0.26% |

Here are the different packet types:

| | | |
|-----------------|--------|--------|
| Frames received | 20,568 | 100.0% |
| Broadcast | 5,242 | 25.7 |
| Multicast | 295 | 1.4% |
| Unicast | 15,031 | 73.1% |

Unicast frames, which are sent to a specific end point, make up for the majority of the conversations on this network segment followed by broadcast frames that are sent to all the end points for a specific reply, such as an ARP packet, which is sent to all interfaces to retrieve the end point's MAC address from a known IP address.

The unicast frames represent most of the promiscuous traffic. Unfortunately, even if they are not required by any application, they are placed in the backlog queue before any packet filtering occurs. The percentage of the traffic that is meant for a specific end point is minimal compared to the amount of traffic that is actually on the network segment.

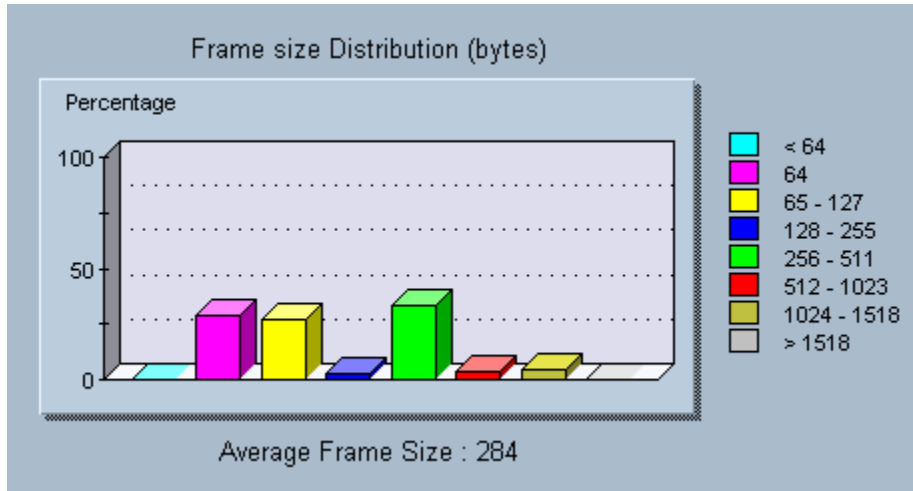


Fig. 2: Frame Size distribution

Fig. 2 shows the packet size distribution on this network segment. Previous studies have shown (Ref?) that the average packet size on all networks is approximately 300 bytes. This makes the 284-byte average packet on this network segment representative of most networks.

Fig. 3 provides the traffic flow percentages that were used. It represents the most common protocol mix that has been observed on the networks that have been tested using a protocol capture application. The different packets are as follows:

- Unicast: Traffic meant for this end point.
- Promiscuous: Traffic meant for other end points.
- Multicast: Traffic meant for more than one end point.
- IPX: Internet Packet Exchange developed by Novell.
- NetBEUI: A Microsoft specific protocol allowing network file systems to view each other.
- ARP: Address Resolution Protocol.
- Broadcast: Packets that are not ARP but contain the broadcast address.
- Misc.: Miscellaneous traffic.

| Unicast | Promiscuous | Multicast | IPX | NetBEUI | ARP | Broadcast | Misc. |
|---------|-------------|-----------|-----|---------|-----|-----------|-------|
| 5% | 60% | 2% | 3% | 1% | 20% | 5% | 4% |

Fig. 3: Protocol Mix Percentages for Network Tests

Using the protocol mix percentages in Fig. 3, the network throughput versus the end-point throughput can be examined with a deterministic network traffic flow. The protocol mix allows the examination of the effects upon the network end points under the normal traffic.

4 Network interface management algorithm, NIMA

NIMA accepts and rejects packets based upon certain criteria that are extracted from the incoming packets. NIMA primarily parses the layer 2 and layer 3 headers. It performs upper layer parsing only if the received frame is a broadcast packet. If it is a broadcast packet and NIMA cannot determine whether to accept or reject based upon layers 3, it accepts the packet. (The algorithm could easily be augmented to use the layer 4 header as well.) The algorithm is as follows:

```
IF Destination is ours (00:c0:17:c0:00:06) THEN
  Send Packet to Backlog Queue
ELSE IF Destination is Broadcast (ff:ff:ff:ff:ff:ff) THEN
  IF ARP Packet (0x0806) THEN
    IF Our Address THEN
      Send to Backlog Queue
    ELSE
      Reject Packet
    ENDIF
  ELSE (Can't determine by Layer2 or Layer3)
    Send Packet to Backlog Queue
  ENDIF
ELSE IF Destination is our Multicast THEN
  This is a Special Address that requires all end points
  subscribing to this multicast to accept the packet.
  Send Packet to Backlog Queue
ELSE
  Reject Packet
ENDIF
```

5 Measuring the physical layer throughput

The tests were performed on a small network with two end points. Each end point consisted of a custom handheld network test device, EtherScope, developed by Fluke Networkstm. The networking component was a LinkSys switch that operates at 10/100/1000 Mbps and was used to provide a deterministic network for all the measurements.

The throughput at the physical layer was measured to show that the physical layer is not the bottleneck in the end-to-end congestion. Using dedicated network testers measuring the physical layer, a true measurement of end-point-to-end-point throughput was realized.

An algorithm in the sender generated the traffic and one in the receiver ran unobstructed as a device driver with the sole mission to measure the percent of utilization before the end point's data link layer. The purpose of the algorithms in the end-point test instruments was to eliminate the operating system and protocol stacks.

The physical layer under measurement included all the components that resided between the traffic generation device and the measurement device. In this instance it was the physical devices of both end points and a LinkSys Gigabit switch. The physical layer for the throughput measurement was extended to include the 10/100/1000 BaseT transceiver within the end points themselves. This device connected to the Data link Layer, or MAC, is also referred to as the physical chip. It identifies the device, which is usually an integrated circuit, as belonging to the physical layer.

The physical layer testing was performed on a 1000 Mbps Ethernet network. Three sets of tests were performed with the typical network protocol mix, one with the minimum packet size of 64 bytes, one with the average packet size of 300 bytes and one with the maximum packet size of 1518 bytes. A set of tests comprised of transmitting packets at 8 different data rates for one minute was used for the physical layer tests. The data rates were 10, 100, 500, 600, 700, 800, 900 and 1000 Mbps. All the tests were performed using Ethernet frames.

Fig. 4 shows the transmitted and received packets per second for the three test sets and also shows the stress introduced into the network by transmitting all minimum sized frames. At 1000 Mbps, 1,488,095 packets / second are transmitted onto the physical layer. This is compared to the 81,274 packets / second that are transmitted if all the packets are the maximum size.

| Test | Bytes | Transmitted Packets per Second | | | | | | | |
|------|-------|--------------------------------|-------|-------|-------|--------|--------|--------|--------|
| 1 | 64 | 15 K | 149 K | 822 K | 893 K | 1042 K | 1316 K | 1340 K | 1488 K |
| 2 | 300 | 4 K | 39 K | 195 K | 234 K | 274 K | 313 K | 352 K | 391 K |
| 3 | 1518 | 812 | 8127 | 40637 | 48764 | 56892 | 65019 | 73156 | 81274 |

| Test | Bytes | Received Packets per Second | | | | | | | |
|------|-------|-----------------------------|-------|-------|-------|--------|--------|--------|--------|
| 1 | 64 | 15 K | 149 K | 822 K | 893 K | 1042 K | 1316 K | 1340 K | 1488 K |
| 2 | 300 | 4 K | 39 K | 195 K | 234 K | 274 K | 313 K | 352 K | 391 K |
| 3 | 1518 | 812 | 8127 | 40637 | 48764 | 56892 | 65019 | 73156 | 81274 |

Fig. 4: Physical Layer Throughput Results

The received bits per second equal the transmitted bits per second indicating the physical layer was capable of full speed data transfers with no losses, 100 % utilization. This provided the baseline for demonstrating that the congestion that is evident in a network results from the inefficiencies in the end points.

6 Measuring the data link layer

The data link layer under measurement included all the components residing between the traffic generation device and the measurement device. This test also included the interaction with the device driver and the backlog queue. It was first performed with a driver without NIMA.

Using a known network traffic generation device, the end-point throughput could be determined as the data rate at which the end point's Ethernet backlog queue notifies the driver that it is full.

The device driver was a modified version of the Linux driver that handles the network communications to and from the physical layer. It was modified to monitor the effects of increased traffic on the backlog queue and to calculate the percent of utilization that caused the backlog queue to reject incoming packets.

When congestion starts to occur, the backlog queue returns a code to the device driver to make it aware that the queue is approaching the full mark. The backlog queue also sends a NET_RX_DROP code to the device driver when it is dropping packets.

6.1 Measuring without NIMA

Measuring was first performed without NIMA in the driver. The protocol mix for a typical network was generated at the eight different transmission rates used to evaluate the physical layer.

The transmitted data rates in Fig. 5 are considerably less than the 1000 Mbps used for the physical layer test. Presenting the same amount of data to the data link layer would have caused the instrument to cease operation because of the enormous number of interrupts as well as the amount of memory allocations and packet movement to and from the stack components.

| Test | Bytes | Transmitted Packets per Second (100 Mbps Network) | | | | | | | |
|------|-------|---|-------|------|------|-------|-------|-------|-------|
| | | Received Packets by the Backlog Queue per Second (100 Mbps Network) | | | | | | | |
| | | Received % of Utilization (1000 Mbps Network) | | | | | | | |
| 1 | 64 | 1.5 K | 15 K | 82 K | 89 K | 104 K | 131 K | 134 K | 148 K |
| | | 1.5 K | 12 K | 12 K | 12 K | 12 K | 12 K | 12 K | 12 K |
| | | 0.1% | 0.8% | 0.8% | 0.8% | 0.8% | 0.8% | 0.8% | 0.8% |
| 2 | 300 | .4 K | 3 K | 19 K | 23 K | 27 K | 31 K | 35 K | 39 K |
| | | .4 K | 3 K | 11 K | 11 K | 11 K | 11 K | 11 K | 11 K |
| | | 0.1% | 0.85% | 2.9% | 2.9% | 2.9% | 2.9% | 2.9% | 2.9% |
| 3 | 1518 | 81 | 815 | 4065 | 4880 | 5690 | 6500 | 7315 | 8125 |
| | | 81 | 815 | 4065 | 4880 | 5690 | 6500 | 6500 | 6500 |
| | | 0.1% | 1% | 5% | 6% | 7% | 8% | 8% | 8% |

Fig. 5: Data link Layer Throughput Results without NIMA

Fig. 6 gives the relationship between the number of Mbps transmitted by the traffic generation device and what was received at the receiving end point expressed in percent of utilization of the 1000 Mbps network. The maximum utilization is 8%, which translates into 80 Mbps. This is the maximum data rate that the receiving end point can process. It decreases to less than 10 Mbps (less than 1% utilization) for all minimum size packets.

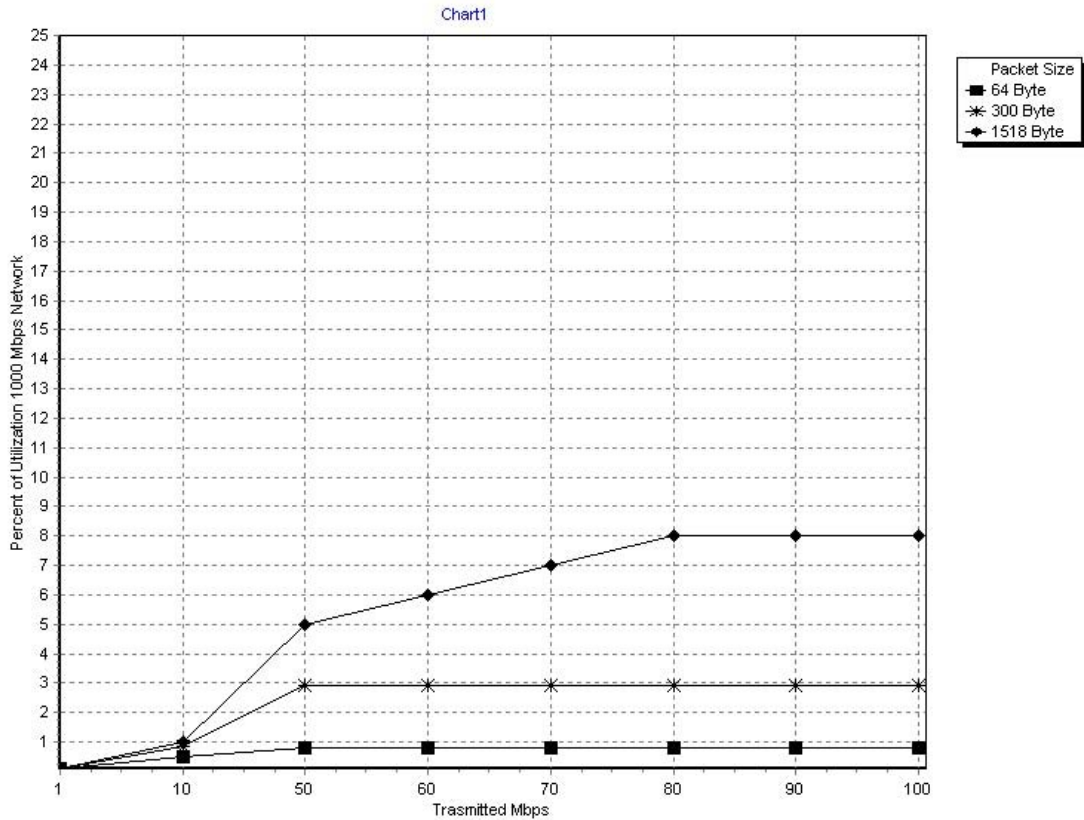


Fig. 6: Data Link Layer Throughput without NIMA

For each given bit rate, the effective throughput increases with larger packet size. This is due to the decreased number of interrupts that the system has to handle as well as the decrease in operations that the stack components have to exercise. Capacity is also lost at the sender side because of the interframe gap. This is the minimum amount of time the transmitting node must listen on the physical layer before sending its next frame. If there is activity on the physical layer the sending node must wait some period of time determined by a backoff algorithm before listening again.

6.2 Measuring the Data link Layer with NIMA

The data link layer was also tested with the NIMA in the driver. Fig. 7 shows the transmitted and received packets per second for the three test sets. The resulting packet reception increased by a factor of 4 for smaller packets and a factor of 3 for maximum size packets. NIMA also allowed the transmission of 1,000 Mbps without causing undo stress on the operation of the device. The device was able to perform without any noticeable degradation of the performance of the applications that are executing on the platform.

| Test | Bytes | Transmitted Packets per Second (1000 Mbps Network) | | | | | | | |
|------|-------|--|-------|-------|-------|--------|--------|--------|--------|
| | | Received Packets per Second (1000 Mbps Network) | | | | | | | |
| | | Received % of Utilization (1000 Mbps Network) | | | | | | | |
| 1 | 64 | 15 K | 149 K | 822 K | 893 K | 1042 K | 1316 K | 1340 K | 1488 K |
| | | 15 K | 48 K | 48 K | 48 K | 48 K | 48 K | 48 K | 48 K |
| | | 1% | 3.3% | 3.3% | 3.3% | 3.3% | 3.3% | 3.3% | 3.3% |
| 2 | 300 | 4 K | 39 K | 195 K | 234 K | 274 K | 313 K | 352 K | 391 K |
| | | 4 K | 39 K | 41 K | 41 K | 41 K | 41 K | 41 K | 41 K |
| | | 1% | 10% | 10.2% | 10.2% | 10.2% | 10.2% | 10.2% | 10.2% |
| 3 | 1518 | 812 | 8127 | 40637 | 48764 | 56892 | 65019 | 73156 | 81274 |
| | | 812 | 8127 | 18570 | 18570 | 18570 | 18570 | 18570 | 18570 |
| | | 1% | 10% | 20% | 20% | 20% | 20% | 20% | 20% |

Fig. 7: Data link Layer W/NIMA Throughput Results

Like Fig. 6, Fig. 8 gives the relationship between the number of Mbps that are transmitted by the traffic generation device and the utilization of the 1000 Mbps network capacity realized at the receiving end point. The maximum network utilization that the end point can process rose to about 20% (200 Mbps) for large packets and 3.3% (33 Mbps) for minimum-size packets. So, the amount of traffic received increased by about 4 times for the smaller packets and 3 times for the larger packets with the introduction of NIMA.

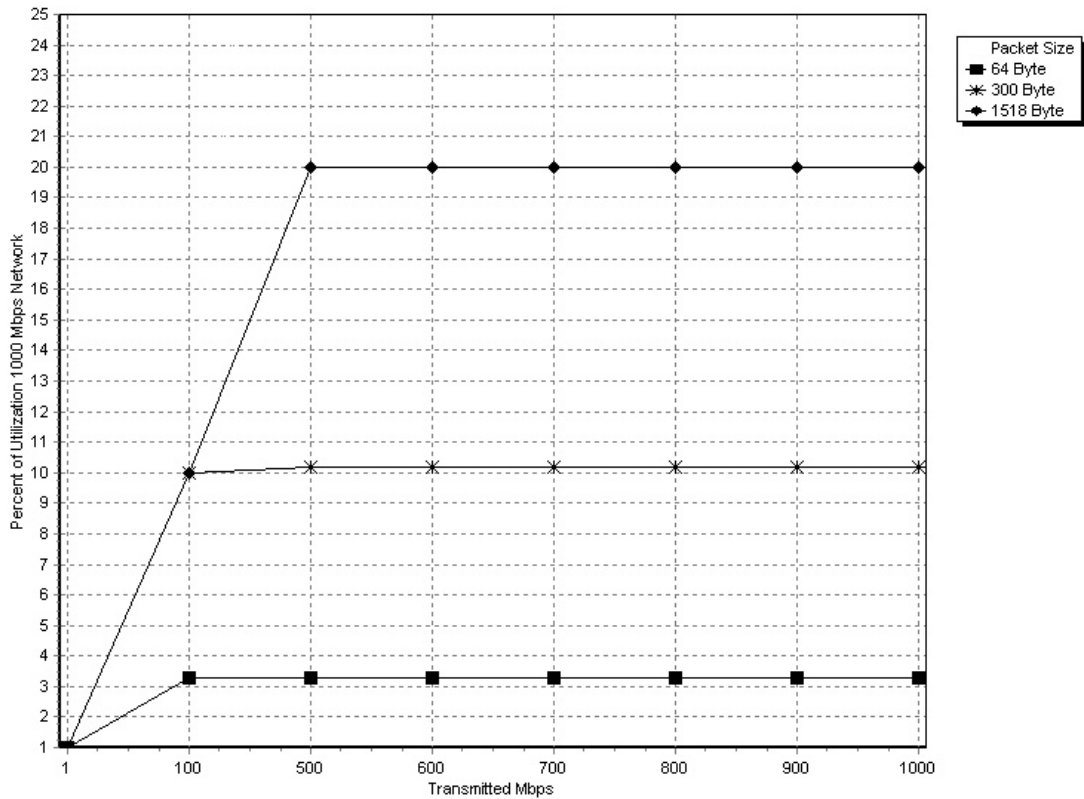


Fig. 8: Data link Layer W/NIMA Throughput

7. Conclusion

In the past, the native congestion management of the TCP protocol was enough to prevent congestion in the mostly TCP-friendly networks but is insufficient with the increased speeds and the increased traffic on modern networks. TCP's slow start and retry mechanisms add to network congestion and end-point performance problems.

Adding congestion control algorithms to the end points' network interfaces as NIMA does gives the effect of managing network traffic and not the protocol queues. The managed interfaces provide an increase in the effective throughput at the end points and therefore produce the same effect as if the network traffic were being managed at the physical layer.

The tests show that the maximum throughput on a 1000 Mbps network can be achieved across the physical layer. The end-point stack components can handle at most 80 Mbps and only 8 Mbps with minimum size packets without filtering. With NIMA installed in the device driver, the maximum throughput was 200 Mbps for large packets and 33 Mbps for small packets.

This research shows that managing the end points' data collection not only increases the throughput to the end point without the necessity for increasing the processor speed. By managing the end point's network interface, more packets are processed leading to less retries on the physical network itself, thus increasing the effective throughput of the entire network.

References

- [1] Sheldon, T., Encyclopedia of Networking & Telecommunications, Osborne/McGraw-Hill, 2001
- [2] Floyd, S., and Jacobson, V., Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, 1:4, August 1993, pp. 397-413.
- [3] Floyd, S., TCP and Explicit Congestion Notification. ACM Computer Communication Review, 24:5, October 1994, p. 10-23.
- [4] Allman, M., Paxson, V., and Stevens, W., TCP Congestion Control, Network Working Group, Request for Comments 2581, April, 1999
- [5] Ramakrishnan, K. and Floyd, S, A Proposal to add Explicit Congestion Notification (ECN) to IP, Network Working Group, Request for Comments: 2481, January 1999
- [6] Mahdavi, J., Semke, J., Mathis, M., The Macroscopic behavior of the TCP congestion avoidance algorithm, Computer Communications Review, 1997
- [7] Firoiu, V., Borden, M., A Study of Active Queue Management for Congestion Control, Nortel Networks, 2000