Receive Side Coalescing for Accelerating TCP/IP Processing

S. Makineni, R. Iyer, P. Sarangam, D. Newell, L. Zhao, R. Illikkal, J. Moses

Intel Corporation

Contact: srihari.makineni@intel.com

Abstract

With rapid advancements in Ethernet technology, Ethernet speeds have increased by 10 fold, from 1 to 10Gbps, in a period of 2-3 years. This sudden increase in speeds has outpaced the rate at which processor and memory speeds have been increasing, raising concerns that TCP/IP processing will not scale to these levels. As a result, applications running on commercial servers will not be able to take advantage of the increased Ethernet bandwidth. This has led to a flurry of activity in the industry and academia focused on finding ways to scale up TCP/IP processing to 10Gbps and beyond. In this paper, we propose a novel technique called "Receive Side Coalescing" (RSC) that increases TCP/IP processing efficiencies significantly. RSC allows NICs to identify packets that belong to same TCP/IP flow and coalesce them into a single large packet. As a result, TCP/IP stack has to process fewer packets reducing per packet processing costs. NIC can do this coalescing of packets during interrupt moderation time, hence packet latency is not effected. We have collected packet traces and analyzed those to find out how much coalescing is possible in different scenarios. Our analysis shows that about 50% reduction in number of packets is possible. We have prototyped RSC on Windows and Linux to understand the benefits, and the results show that 2-7% of savings in CPU utilization is possible at 1Gbps speeds. Projection models developed to estimate processing costs at 10Gbps show that RSC can save up to 20% of the CPU.

1 Introduction

Ever increasing bandwidth needs of enterprise data centers has led to the development of 10Gbps Ethernet technology. Commercial 10Gbps Ethernet Network Interface Cards (NIC) have been available in the market for sometime now, and the deployment pace has been steadily picking up. TCP/IP is the most commonly used protocol to process data both in enterprise data centers and on the Internet. However, current implementations of TCP/IP protocol on todays and tomorrows commercial servers will not be able to process packets at 10 Gbps hence user applications can not take advantage of increased Ethernet bandwidth completely. We have measured TCP/IP performance on Intel® Xeon™ processor (2.6GHz) based server platform running windows OS. Results shown in figure 1 highlight the fact that the transmit side can achieve higher throughputs at lower CPU utilizations compared to that of the receive side processing. CPU utilizations shown in the graph are for two logical CPUs. At 65536 byte payload size, receive side throughput maxed out at 2Gbps with average CPU utilization across the two processors is around 65%. This highlights the fact that one CPU can barely achieve 2Gbps of receive throughput, which means multiple CPUs need to be expensed to achieve 10Gbps throughput.

While the TCP/IP protocol portion [4] has been proven to be not much compute intensive, there are significant other overheads that come into play when the entire processing, application to NIC and vice versa, that takes place when transmitting or receiving data by applications is considered. Some recently published papers shed more light [6,21,25,14] on these overheads by analyzing their impact.

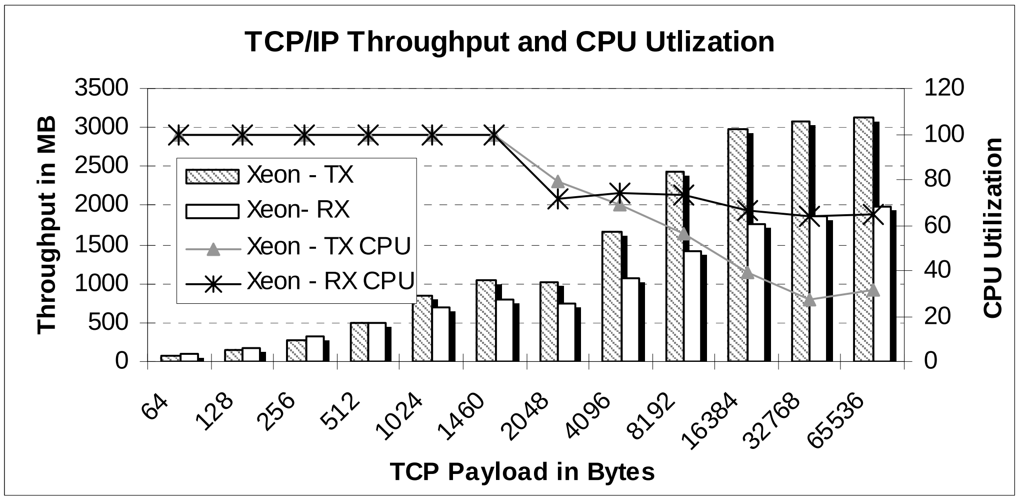

Figure 1. TCP/IP Performance

Within TCP/IP processing, transmit side processing costs have been reduced significantly over the years with the development of techniques such as zero copy transmit and Large Segment Offload (LSO). With the help of these optimizations, today's CPUs can achieve throughputs of 7-8Gbps at close to 100% utilization. In addition, the transmit side processing scales well with the CPU speeds because it is not very memory intensive. On the other hand, receive side processing is much more difficult to optimize. Today's server platforms have to spare an entire CPU to achieve just 2-3Gbps of throughput even when receiving large packets (> 1KB). CPU cycles required to receive and process a single byte of data when transferring 8KB payloads is in the order of 6 cycles. Reasons for such high cost of receive side processing are: descriptor and TCP/IP header processing and data copy from kernel to application buffer generate compulsory cache misses, per packet processing costs, OS overheads, etc. Since receive side processing is memory intensive and memory speeds increase rather slowly (as compared to CPU speeds), receive side performance does not scale as well with increases in CPU speeds. To address receive side processing challenge, several solutions have been proposed. These solutions range from techniques that target specific overheads involved in receive side processing to new protocols like RDMA [12] to complete offload solutions like TOE [1]. These techniques are reviewed in section 3 of this paper.

Our contribution in this paper is the development of a novel technique called Receive Side Coalescing (RSC) that allows NICs to identify incoming packets that belong to same TCP/IP connection (flow) and coalesce those packets to form a single large packet before sending it to the software stack for processing. This technique reduces number of packets that software has to process thus reducing overheads involved in processing packets. We have collected packet traces and analyzed those to figure out how much coalescing we can achieve when there are a large number of connections on which packets are coming. Our analysis showed that up to 50% reduction in number of packets to be processed can be achieved. In order to estimate CPU savings possible with RSC, we have implemented RSC prototype in software both on Linux and Windows platforms. Results from the prototypes showed that we can lower CPU utilization from 2-7% even at 1Gbps speeds. Analytical model developed to project savings at 10Gbps showed more than 20% of savings in CPU utilization are possible. While the savings with RSC are significant, they are not sufficient to achieve 10Gbps of receive side processing throughput without spending more than one processor core/thread entirely. But, RSC when combined with other proposed solutions such as Direct Cache Access [7] and interrupt moderation makes 10Gbps processing feasible.

Rest of paper is organized as follows: In section 2, we provide an overview of TCP/IP receive side processing and highlight various overheads involved. In section 3, we review various techniques/solutions that have been proposed to solve receive side processing challenge. In section 4, we show data from our analysis on how much coalescing is possible in the network. In section 5, we describe how RSC works, provide details on RSC SW prototypes and show the results from these prototypes. We conclude the paper with summary and future work.

2 TCP/IP Receive Processing

In this section we provide a high level overview of the processing that takes place from the time a NIC receives an Ethernet frame till the incoming data is handed over to the intended application. It is not our intention to provide a detailed description of this processing, but to provide sufficient context for readers while highlighting some major overheads involved in this processing.

Receive-side processing begins when NIC hardware receives an Ethernet frame from network. NIC extracts the TCP/IP packet embedded inside the frame by removing the frame delineation bits. It validates the frame by checking the CRC value embedded in the frame with that of the computed value. It then grabs the next available descriptor to find out where in the memory to copy the packet header and payload. Descriptor is a data structure (typically 16bytes) that the Ethernet driver and NIC use to communicate and exchange information. These descriptors are allocated by the driver and arranged in a circular ring. The driver informs the NIC through these descriptors, among other things, address of a memory buffer (NIC buffer) to store the incoming packet data. The stack allocates several memory buffers to receive incoming packets. These buffers, like the descriptors, get reused. NIC copies the incoming data into this memory buffer using a DMA engine. Once the packet is placed in memory, NIC updates a status field inside the descriptor to indicate to the driver that this descriptor holds a valid packet and generates an interrupt. This kicks off the SW processing of the received packet.

Figure 2 shows the overall flow for receive side processing. The Ethernet device driver reads the descriptor and makes sure that NIC has indicated that this is a valid packet. Driver then classifies the packet as either IP packet or some other. If it is an IP packet then it forwards it to the TCP/IP stack for further processing. Since this descriptor was updated by NIC earlier, it results in invalidating processor's copy (if found in cache). So the processor would have to get it from the main memory. If the descriptor size is 16 bytes then each cache line (64 bytes) can accommodate up to 4 descriptors. Similarly, accessing packet headers by TCP/IP stack also results in caches as this data was just placed in the memory by the NIC. TCP/IP headers combined is 40 bytes in size when there are no option fields used. So each packet header will result in one compulsory cache miss. The next step in processing is to identify the connection to which this packet belongs. TCP/IP software stores state information of each open connection in a data structure, called the TCP/IP Control Block (TCB). Since there can potentially be several thousand open connections, hence many TCBs, TCP/IP software uses a well known search mechanism called hashing for fast lookup of the right TCB. The hash value is calculated by using the IP address and port number of both the source and destination machines. Several fields (sequence numbers for received/acknowledged bytes, application's pre-posted buffers, etc.) inside the TCB are updated whenever a new packet is received. TCP/IP stack then needs to figure out where to copy the packet payload (data portion). It checks to see if the target application has already posted a buffer to receive incoming data. If a buffer is available, stack copies the data from the NIC buffer into that buffer. Otherwise, it will wait for the application to provide a buffer. TCP/IP stack may be forced to copy the data into a temporary buffer if application doesn't provide one soon enough. When the incoming data is first copied, it results in compulsory misses as the data has to be read from main memory. So for a 1460 byte copy, there will be 23 cache misses, assuming 64 byte cache line. Since user applications run in user mode, there will be one or more context switches involved when receiving data.

In the next section we explain optimizations that have been developed or proposed to help receive processing.

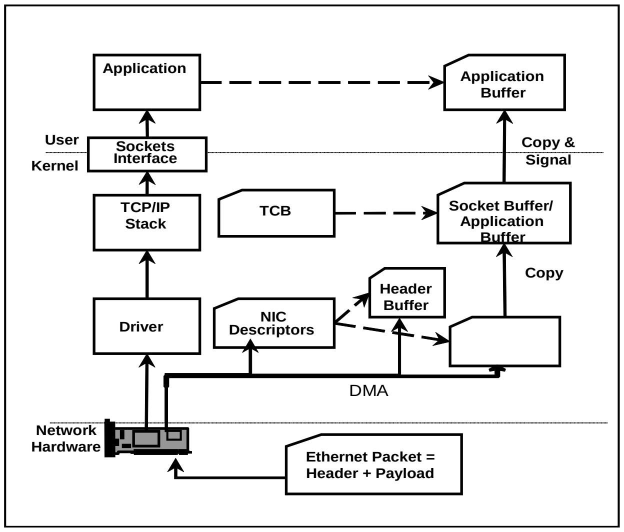

Figure 2. Data Flow in Receive-Side processing

3 Existing Techniques to Accelerate TCP/IP Receive Processing

Over the years several solutions have been proposed to improve the performance of TCP/IP receive processing. Some of these are implemented are generally available in majority of the NICs that ship today.

Interrupt Moderation

Network interrupt processing is an expensive operation even on today's machines. Interrupt moderation technique was developed to reduce the number of interrupts NIC generates. NIC instead of generating one interrupt for every incoming packet, it generates an interrupt after some number of packets are received. This interrupt moderation is typically exposed as a configurable parameter on today NICs. This is a commonly available feature in today's NICs.

Jumbo Frames

Jumbo frames technology was developed to reduce number of network packets required to transfer larger payloads. Most of today's NICs support multiple size jumbo frames. Jumbo frames not only reduce numbers of packets on the network but can also significantly reduce TCP/IP stack processing time hence CPU utilization. However, jumbo frames are not widely used because enabling jumbo frames on end machines is not sufficient - all the intermediate routers and switches need to enable jumbo frames as well.

Receive Side Scaling

Receive side scaling is relatively a new technique developed by Microsoft to allow NICs to distribute interrupts across multiple CPUs in a system. Without this feature, all the interrupts would go to one processor (typically CPU0). As a result, maximum receive throughput a machine can achieve depends on what a single CPU can achieve.

TCP Offload Engines (TOE)

TOE offloads entire TCP/IP processing from the main CPU. TOEs are typically implemented on the NIC. TOEs are expensive, require changes to operating systems and face other problems that are described well in a recent paper [17]. Given these issues, it is not known yet how this idea will succeed in the market.

Remote Direct Memory Access (RDMA)

RDMA is a set of specifications developed by the RDMA consortium to solve the problem of directly placing incoming network data into user application buffers without having to go through intermediary copies of that data. RDMA provides the ability of one computer to directly place information in another computer's memory with minimal demands on memory bus bandwidth and CPU processing overhead, while preserving memory protection semantics. RDMA over TCP/IP defines the interoperable protocols to support RDMA operations over standard TCP/IP networks.

Direct Cache Access (DCA)

DCA [7] allows NIC to place incoming packet data directly into a processor's cache. This can potentially eliminate compulsory cache misses that occur during packet processing. It also speeds up data copies as the incoming payload will be in processor's cache instead of main memory.

4 Receive Side Coalescing (RSC)

4.1 RSC Overview

RSC is a stateless and software transparent offload mechanism. It allows a NIC to identify TCP/IP packets that belong to same connection and to coalesce these packets into a single large packet before sending it up to the processor. Thus RSC reduces number of packets that a TCP/IP stack needs to process bringing down per packet processing costs significantly. RSC, in concept, is exactly opposite of Large Segment Offload (LSO) that happens on the transmit side where a large payload is handed over to the NIC for transmission and NIC handles fragmenting the payload into multiple TCP/IP packets and computing headers for these packets. Figure 2 illustrates the effect of RSC. When there is no RSC, NIC sends all the packets it receives to the stack for processing, but when the NIC supports RSC, incoming packets get coalesced into fewer larger packets. RSC requires NIC driver to allocate large buffers (4KB or higher) so that it can place payload from more than one packet in each buffer. Coalescing of packets also leads to reduction in descriptor usage as well. NIC performs coalescing of incoming packets while waiting for the interrupt moderation timer to expire. As a result, RSC does not add any extra delay in delivering the packets to the processor.

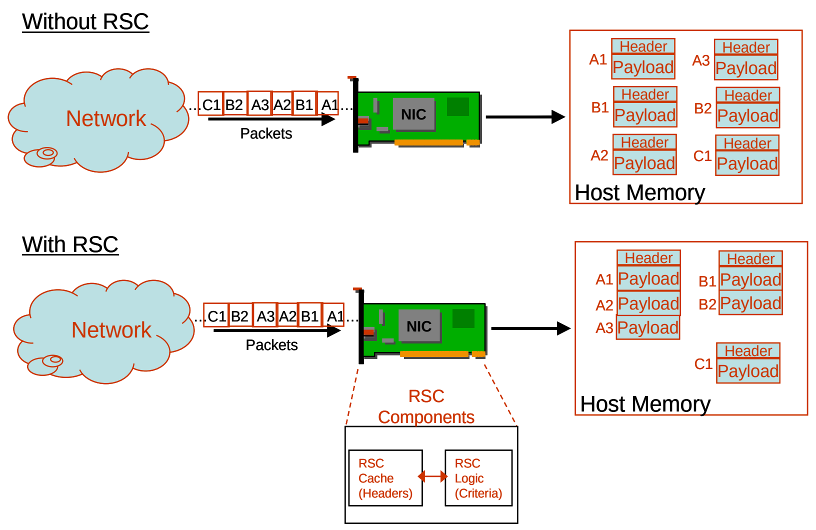

Figure 3. Illustration of RSC concept

Coalescing TCP/IP packets is possible because TCP is a byte stream protocol and applications can't make any assumptions about boundaries of a message. For example, an application can receive tail end of a message and the beginning of the next message in the same 'receive' call.

In order to perform coalescing, RSC needs to save some information (state) about each connection for which NIC decides to coalesce. This information is saved in a cache, called RSC cache. Each coalesced packet takes up about 100 bytes in the cache. The following pieces of information are stored in the cache.

- IP, TCP header info from first packet
- Descriptor number that is in use
- Starting offset in the payload buffer for the next incoming packet's payload
- Number of packets coalesced
- Current TCP checksum
- Number of bytes coalesced

RSC logic on NIC extracts TCP/IP header fields from incoming packets and does a series of tests to determine whether to coalesce this packet or to stop existing coalescing. The engine uses following criteria to make a determination.

Packet Coalescing Criteria

- TCP/IP packet (IP.protocol == 6)
- Not an IP fragment (IP.Flags.DF==1 && IP.Flags.MF==0)
- No IP (v4 and v6) header options are set
- Valid TCP Checksum (+ valid CRC for RDMA)
- Data packet (tcp.payload > 0 bytes)

- Correct TCP sequence number
- No SYN, FIN, RST and URG flags are set in TCP header
- No TCP header options other than TCP Timestamp option are set
- TCP flags other than PSH and ACK force the current coalescing to stop for this connection. An incoming packet with these flags is sent to the stack separately.
- If TCP "PSH" flag is set and no coalescing is in progress for this connection then send this packet as is. If coalescing is in progress for this connection then append this packet and close the descriptor.
- Out of order packet forces coalescing to stop
- Incoming payload is not split across multiple descriptor buffers

If an incoming packet is the first packet for a TCP/IP connection, and RSC logic decides to start coalescing, then the packet's TCP/IP header is removed and relevant information from the header is saved in the RSC cache. Packet's payload is then copied (DMA) into a buffer provided by the NIC driver. RSC does not hold onto the payload while coalescing is in progress so it does not need any additional memory on NIC. When a second packet on the same connection comes and if it meets coalescing criteria, then the entries in the RSC cache are updated (how many bytes received, starting offset in the buffer for next packet's payload, etc). TCP/IP headers are stripped from the packet and payload is copied next to the previous one in the same buffer. When the RSC logic decides to stop coalescing for a connection either because an incoming packet does not meet coalescing criteria (out of order packet, payload does not fit in the remaining space in the buffer, PSH flag in TCP header is set, etc.), then modified header in the RSC cache for that connection is written back to the memory at the location specified in the descriptor.
At the time of interrupt all the headers from the RSC cache are written back. RSC can start coalescing again after the interrupt. As a result, RSC requires only a small amount of RSC cache (1-2 KB) to store information about 3-4 connections at a time.

5 How much coalescing is possible?

It is well known in the networking community that TCP/IP traffic is bursty in nature. Several studies have been conducted and multiple papers [2, 9, 22] have been published on this subject. RSC is mainly dependent on the existence of these bursts hence the amount of benefit we get with RSC is tied to degree of coalescing that is possible. RSC with 2 or more entries of cache does not require packets on a connection to come back to back to be able to coalescing packets. It is sufficient if they come in close succession.

In order to find out how much coalescing is really possible and how it varies with number of simultaneous flows (TCP/IP connections), RSC cache size, buffer size and interrupt moderation window, we have collected several sets of network packet traces and analyzed those. Results from this analysis are presented in this section.

For all the experiments described below, we have used the following setup. We have set up a dual processor server machine as the reciever. This receiver machine is connected to a number of client machines (senders) via two 1Gbps Ethernet switches. All the client and server machines used 1Gbps NICs and ran Chariot [26] (commercially available network performance analyzer) scripts to transfer data. Each client machine sets up 1 connection with the receiver and starts transmitting data. We have used an open source program, called ethereal [5], to collect network packet traces on the receiving machine. We have then analyzed these packet traces using internally developed scripts to find out how much coalescing is possible in each scenario based on certain coalescing criteria.

5.1 Simultaneous Connections

The main purpose of this experiment is to find out how much coalescing is possible when there are multiple simultaneous TCP/IP connections. We have run an experiment where a single receiver is setup to receive data from 4, 8 and 16 client machines at the same time on a 1Gbps NIC. Packet traces were collected using ethereal program on the receiving machine while the test is in progress. We have analyzed the packet traces by assuming that we have 8KB NIC buffer available and RSC cache size is 4 entries deep. 8KB buffer allows us to coalesce up to 5 full sized TCP/IP packets (1460 bytes each). Four entry deep RSC cache allows us to keep track of up to 4 different connections. After every 20 packets, we have stopped coalescing and flushed RSC cache and started all over again. This is same as simulating a 20 packet interrupt moderation window. Results from this experiment are shown in the graph in figure 4. On the x-axis, we are showing various payload sizes for which we have collected and analyzed packet traces. On the y-axis, we are showing number of packets that RSC is able to eliminate out of a total of ten thousand packets that were captured. The results show that about 50 to 80% coalescing is possible, and there is not much variation in achievable coalescing between 4, 8 and 16 connections. One exception though is at 8KB point where the number of coalesced packets with sixteen connections is slightly smaller than that of 4 and 8 connections. This is because with the 20 packet hard stop we have for stopping the coalescing activity, we found ourselves often not able to coalesce the last packet (~890 bytes) of 8KB payload. But in case of 4 and 8 connections, more often we were able to coalesce all the packets (5 full + 1 partial) into one. Same reasoning can explain why the coalescing at 16KB through 64KB payload sizes is less than that of 8KB. Since we do not allow a packet to be split across two coalesced packets, we can only pack at most 5 full sized packets (5*1460 = 7300). For 8KB transfers, payload size in the sixth packet will not be a full sized packet, it will be around 890 bytes, hence it fits in the 8KB buffer.

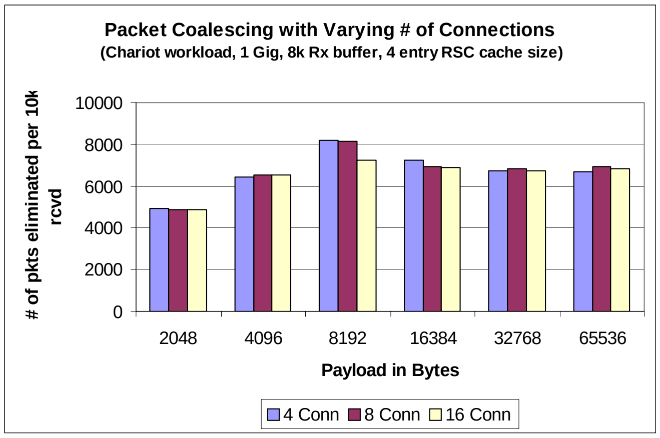

Figure 4. Amount of coalescing possible with 2, 4 and 16 simultaneous connections

Next, we ran another experiment with 16, 30, 60 and 90 clients, with as many connections, sending data to a single receiver on a 1Gbps NIC. We have collected packet traces on the receiving machine and analyzed those. Results are shown in the graph in figure 5. These results confirm further that we can still achieve more than 50% coalescing (50% reduction in number of packets) even when there are large number of connections on which packets are received. This is mainly due to the burstiness nature of TCP/IP protocol. Transmitter's user of LSO to transmit packets magnifies this effect. Note that these clients are connected to the server through two 1Gig Ethernet switches. This show that switches do not disturb the burstiness of the flows. To convince ourselves further that switches and routers do not interrupt these bursts, we ran a simple experiment where we have started a large file transfer operation on a machine with 1Gbps NIC from another machine in the same local area network. While the file transfer is progressing, we have started accessing web pages from popular Internet sites (amazon.com, yahoo.com, etc.) and captured packet traces. The captured packet traces revealed that packets from these websites came back to back most of the time confirming the burstiness of TCP/IP protocol is preserved through many routers and switches that packets might have traversed.

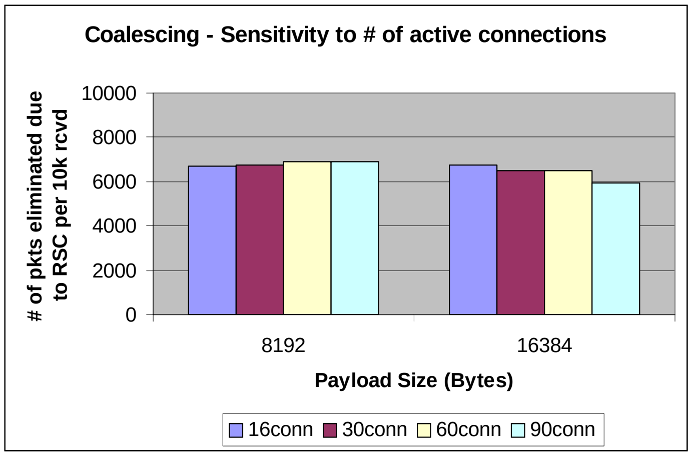

Figure 5. Data Flow in Receive-Side processing

5.2 *Sensitivity Studies*

Next we wanted to study how various key parameters such as RSC cache size, buffer size and interrupt moderation window can impact how much coalescing is possible.

The graph in figure 6 shows sensitivity study results where we have varied number of entries in the RSC cache size. RSC cache size defines how many simultaneous connections for which coalescing can be in progress. The x-axis of the graph shows different cache sizes we have studied while the y-axis shows how much coalescing was possible. We have shown data for various payload sizes. These tests are conducted with 16 simultaneous connections, 8KB buffer and 20 packet interrupt moderation window. These results point out RSC cache size of 3 entries allows us to achieve close to maximum coalescing. This data suggests that most often the receiving NIC is receiving packets for up to 3 different connections during the interrupt moderation window time even though there are 16 active connections.

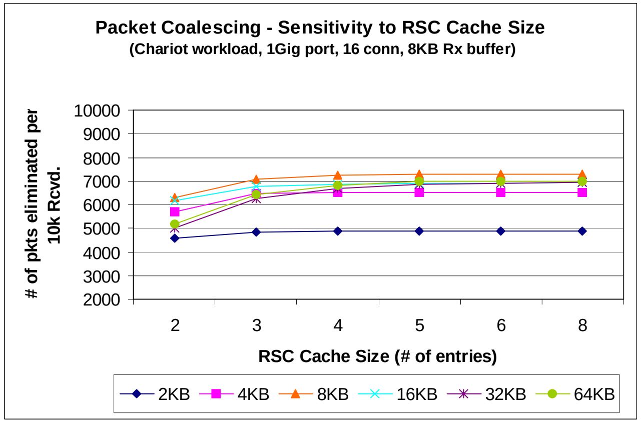

Figure 6. Data Flow in Receive-Side processing

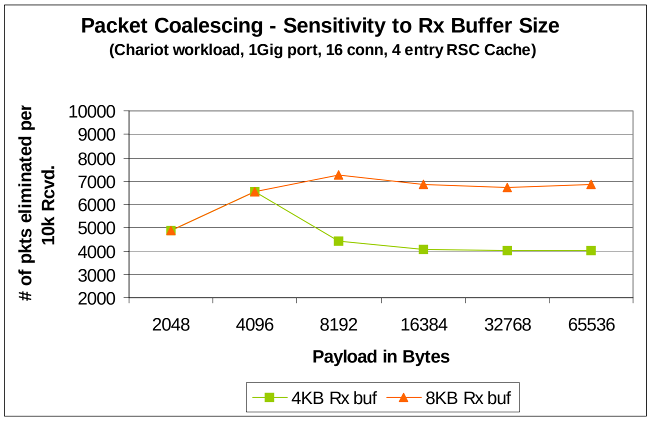

Figure 7. Data Flow in Receive-Side processing

Next, we study sensitivity to receive buffer size; size of the buffer dictates how many individual packets we can coalesce into one. The analysis of captured packets is done assuming RSC cache size of 4 entries and 20 packet interrupt window. There were 16 active connections from 16 different clients. It is clear from the results shown in the graph in figure 7 that 8KB buffer yields more coalescing for payloads larger than 4KB. The amount of coalescing is same with 4 and 8 KB buffers for 2KB and 4KB payload sizes is because TCP/IP stack typically set "PSH" flag at the end of each of each payload (2KB or 4KB in this case). Whenever the PSH flag is set in a packet, RSC coalesces that packet (if it meets coalescing criteria, of course) and stops coalescing any more packets into that buffer. Subsequent packets for the same connection will be coalesced in a new buffer. This is done to preserve the meaning and purpose of the PSH flag.

The last sensitivity study that we have done is to figure out the impact of varying interrupt moderation window. On today's 1G NICs, it is around 250us, which is equal to roughly 25 full size packets (this is the default setting on Gigabit NICs from Intel Corp.). Results of this study are shown in the graph in figure 8. Results are shown for 4, 8 and 16KB payload sizes. It can be pointed out from the data that we can achieve more than 50% coalescing even with a 16 packet window. Increasing the interrupt moderation window from 16 packets to 32 packets shows 5-10% improvement in coalescing and beyond 32 packets the rate of improvement diminishes.

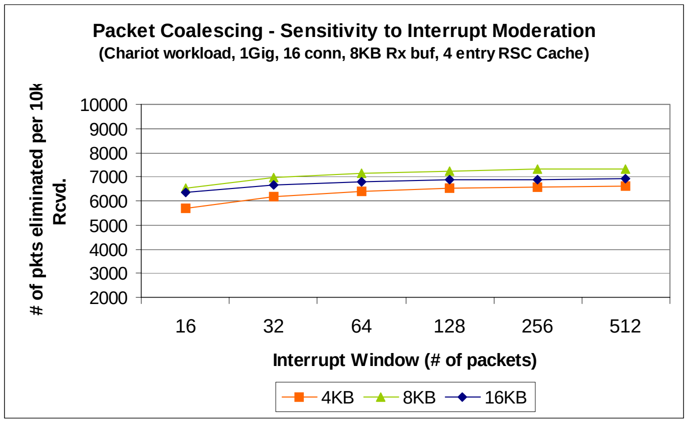

Figure 8. Data Flow in Receive-Side processing

These sensitivity studies prove that significant coalescing is possible on machines that receive significant amount of TCP/IP data. In the next section we show how this coalescing translates into CPU utilization reduction.

6 RSC Performance

6.1 Methodology

Next step in RSC evaluation is to find out how much CPU savings that we can achieve with RSC. In order to find this out, we have implemented a software version of RSC in Ethernet driver code. This allowed us to coalesce packets in the driver code before sending them up to TCP/IP stack for further processing. We have wrapped RSC code inside the driver with RDTSC instructions to measure the time spent performing packet coalescing. At the end, we have subtracted this time from the overall time to figure out the benefit of RSC. We have also measured the processing cost of each descriptor, total number of descriptors processed and the average coalescing achieved. Using this data we figured out how much time we could have saved in terms of descriptor processing if RSC was implemented in the NIC hw (RSC reduces descriptor usage as well). We have subtracted this amount also from the overall processing time. After subtracting out the RSC time and time saved in descriptor processing from the overall processing time, the CPU utilization we get is what we have used for RSC configurations.

This prototype effort also proved that Linux and Windows implementations of TCP/IP protocol can work with larger payload sizes even when the MSS is set to 1500 bytes. We did not have to make any changes to TCP/IP stacks to make them accept larger packets. There may be some corner cases where stacks depend upon the number of packets received to figure out exact response, which we are investigating. At a minimum, we think that TCP/IP stacks need to know how many packets were coalesced in each packet that they receive so TCP/IP stats like number of packet received can be maintained accurately.

To allow coalescing of multiple packets, we have forced the Ethernet driver to allocate larger buffers (4KB and 8KB). NIC does its normal processing of packets and sends them to the processor. In the Ethernet driver, we execute RSC code and decide which packets can be coalesced. We have varied the degree of coalescing to figure out how the benefits vary. We have used 3 different levels of coalescing and compared the results to a base case (denoted as “Regular” in the graphs) that does not have any coalescing. The three levels of coalescing used in our experiments are indicated in the graphs as “RSC2” (or “RSC_2”), “RSC3” (or “RSC_3”), and “RSC5” (or “RSC_5”) to mean at most two, three and five packet coalescing respectively.

In all these tests, we have a used a fast client to transmit data to the receiver machine that is under the test. The client machine has enough capacity in terms of compute power as well as network bandwidth so that it does not become bottleneck in any way. Even though we are showing results for a single stream (1 TCP/IP connection), we have collected data with 2 and 4 streams and are in the process of analyzing it. But the first indications are that RSC benefits with more streams will be the same or slightly be better than that of 1 connection numbers. We are also in the process of measuring the benefits with large number of connections.

6.2 Results

We have first implemented RSC functionality in Linux OS and took measurements with and without RSC. Results are show in the graph in figure 10. The x-axis in the graph shows different payload sizes (application buffers) that we have taken measurements for. The y-axis shows measured CPU utilization and the secondary y-axis shows efficiency of processing in terms of cycles/byte. We have used ttcp [23] program on both the client and server machines to generate TCP/IP traffic between them. Across all the payload sizes and configurations the system was able to achieve maximum possible throughput (~950Mbps) on a 1Gbps NIC.

Across all the payload sizes, RSC shows a benefit over the base case. Two packet coalescing (RSC_2) offers 2-4% savings in CPU utilization while five packet coalescing (RSC_5) saves about 5-7% of CPU when processing at 1Gbps speeds. You can also see from the cycles/byte comparison how RSC improves receive side processing efficiency.

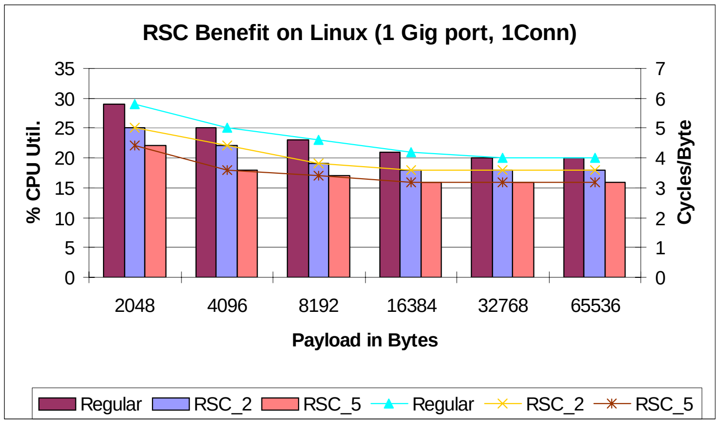

Figure 10. RSC Performance on Linux at 1Gbps

Next, we have repeated the same experiment on a server machine running windows. We have modified the Gigabit Ethernet driver code and added RSC functionality to it. We have used a program, called ntttcp [23], to generate network traffic. Results are shown in the graph in figure 11. In this graph, we are showing on the secondary y-axis, the average number of packets that we were able to coalesce in each case. Results show that RSC offers benefit with windows stack too. If we consider the 8KB payload data point, RSC_2 brings down CPU utilization from 39.5% to 37.8% and RSC_5 brings this further down to 32.6%. We have also noticed that for 16KB payloads, RSC_2 CPU utilization turned out to be (0.1%) higher than the Regular case. Up on close examination of the code, we have realized that the RSC SW prototype is causing more cache thrashing than the regular case. This is because the Ethernet driver has created 512 descriptors and buffers to receive packets. NIC circulates over these buffers and places incoming packets into these buffers. So in the base case, network processing would have used up around 512*1500 bytes (maximum TCP payload size) of cache space, where as in RSC_2 case we would be bringing in around 512*3000 bytes (since coalescing will happen in all of these buffers in a short period of time). Our hypothesis is confirmed by looking at the CPU peroformance data collected that showed there are 20 additional L1 misses and 1 additional L2 miss per packet in case of RSC_2. If we subtract out the effect of these cache misses, RSC benefit will increase by about 1-2%.

Also, we have noticed that windows TCP/IP stack seems to be lot more dynamic than the Linux stack as it seems to be doing slightly different things when receiving and processing different payload sizes. Since we do not have access to source code of windows TCP/IP stack we can not explain all the discrepancies we are seeing in the results.

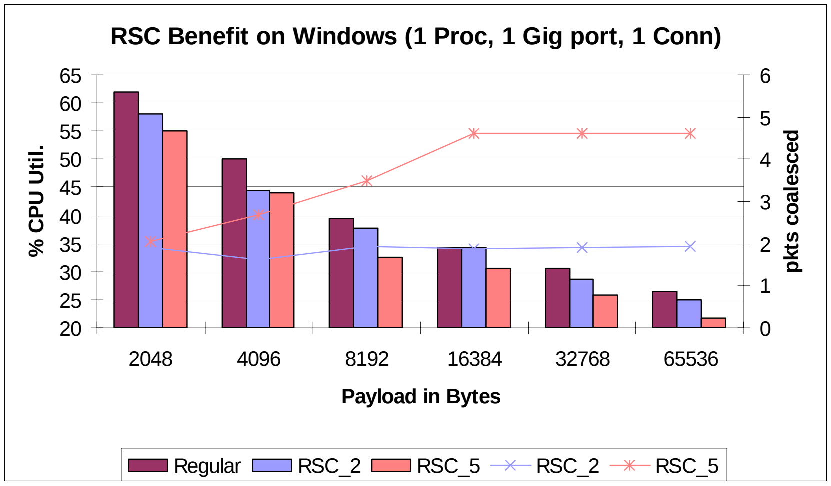

Figure 11. RSC Performance on Windows at 1Gbps

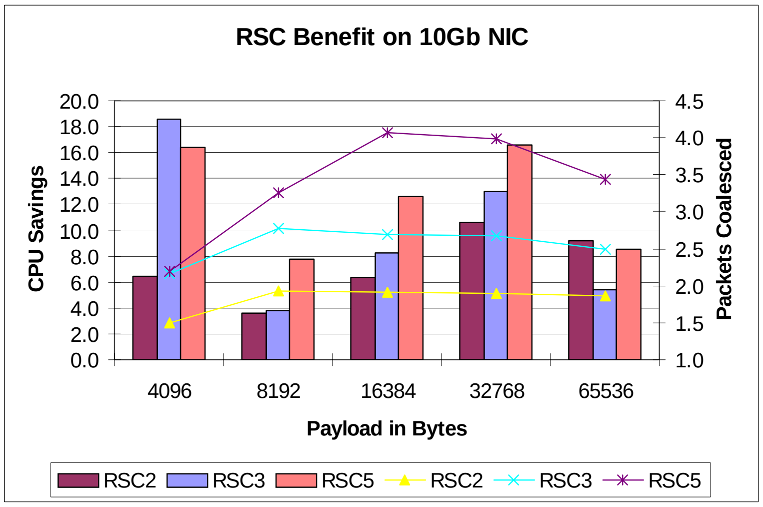

Figure 12. RSC Performance Potential when processing at multi Gbps speeds

Even though RSC showed decent CPU savings when processing at 1Gbps speeds, its benefit will be much more meaningful at multi gigabit speeds as today's systems and CPUs can not handle more than 2-3Gbps. We wanted to find out what the benefit would be at these speeds, so we have experimented with a current generation 10Gbps NIC on a windows machine. We have modified the 10G Ethernet driver and added RSC support. We have measured CPU utilization and throughput without any coalescing and with RSC2, RSC3 and RSC5 configurations. Results are shown in the graph in figure 12. Even though we have used 10G NIC, the DP server system under test could only achieve 1.6Gbps throughput at 4KB payload size and 3.1Gbps at 65KB payload size with 100% CPU utilization. As mentioned above, receive side processing can not benefit much from multiple CPUs as all the TCP/IP processing happens on the CPU that received the NIC interrupt.

This test was conducted with 1 TCP/IP stream and 1 CPU. As you can see from the graph, RSC2 benefit has gone up as compared to that of at1Gbps results. RSC2 benefit now ranges from 4 to 12%. As expected, the benefits are higher (4-18%) with RSC3 and RSC5 configurations. The benefit at 64KB payload size is less than that of the 32KB because of some abnormality with the interrupt moderation schemed implemented on these NICs. When transferring 64KB payloads, we have observed a jump in number of interrupts/sec from 7.5k at 32KB payload size to 18k. The benefit of RSC2 and RSC3 at 8KB payload size is lower than that of the 4KB payload size which is not immediately clear to us. We are investigating this anomaly. From these results, it is clear that when receiving packets at true 10Gbps speeds, even two packet coalescing (RSC2) can bring down the CPU utilization by more than 20% points.

7 Summary & Conclusions

Scaling of receive side TCP/IP processing beyond the current 1-2 Gbps has become important as the bandwidth demands in enterprise data centers have grown significantly in recent years. This requires that several overheads involved in receive side processing need to be reduced drastically. In this paper, we have proposed a novel technique, called Receive Side Coalescing (RSC) that addresses per packet processing overheads by reducing the number of packets that need to be processed by the stack. We have shown collected network packet traces and analyzed those to prove that more 50% coalescing is possible with a 4 entry RSC cache, 20 packet (~200us) interrupt moderation window and 8KB NIC buffers. In essence RSC offers similar benefits like that of 9KB Jumbo frames but without all the problems associated enabling the jumbo frames in the data centers. Next, we have shown results from our software implementation of RSC. These results showed that CPU utilization can be reduced by 1-5% with RSC2 (2 packet coalescing) and 3-7% with RSC5 (5 packet coalescing) even when processing at 1Gbps speeds. We have also shown RSC benefits increasing significantly when processing at 2-3Gbps speeds. We believe that RSC will be a key technique that allows TCP/IP processing to scale better. Going forward, we would like to implement RSC in hardware and use it in real environments so we can measure the benefits it offers for some key real world applications. This also allows us to find out any changes that may be required to the TCP/IP stacks to fully support RSC.

References

[1] "Alacritech SLIC: A Data Path TCP Offload methodology", http://www.alacritech.com/html/techreview.html

[2] E. Blanton and M. Allman, “On the Impact of Bursting on TCP Performance,” Proceedings of the Workshop for Passive and Active Measurement, Mar 2005.
[3] J. Chase et. al., “End System Optimizations for High-Speed TCP”, IEEE Communications, Special Issue on High-Speed TCP, 2000.
[4] D.D. Clark, J. Romkey, and H. Salwen, “An analysis of TCP processing overhead,” IEEE Communications, vol. 27, no. 6, pp. 23–29, June 1989.
[5] Ethereal Network Protocol Analyzer, http://www.ethereal.com/
[6] A. P. Foong, T. R. Huff, H. H. Hum, J. P. Patwardhan, and G. J. Regnier, “TCP Performance re-visited,” in Proc. IEEE Int. Symp. on Performance of Systems and Software, pp. 70–79, Austin, Mar. 2003.
[7] R. Huggahalli, R. Iyer and S. Tetrick, Direct Cache Access for High Bandwidth Network I/O,” 32nd Annual International Symposium on Computer Architecture (ISCA 2005), June 2005.
[8] iSCSI, IP Storage Working Group, Internet Draft, available at http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt
[9] R. Jain and S. A. Routhier, “Packet Trains - Measurements and a NewModel for Computer Network Traffic,” IEEE Journal on Selected Areas in Communications, vol. 4, pp. 986–994, Sept. 1986.
[10] H. Jiang and C. Dovrolis, “Source-Level Packet Bursts: Causes and Effects,” in Proceedings Internet Measurement Conference (IMC), Oct. 2003.
[11] J. Kay and J. Pasquale, “The importance of non-data touching processing overheads in TCP/IP,” in Proc. ACM SIGCOMM, pp. 259–268, San Francisco, Oct. 1993.
[12] RDMA Consortium [Online]. Available at http://www.rdmaconsortium.org/.
[13] C. Kurmann, M. Müller, F. Rauch, and T. M. Stricker, “Speculative defragmentation— A technique to improve the communication software efficiency for gigabit Ethernet,” in Proc. 9th IEEE Symp. High Performance Distr. Comp, Pittsburgh, Aug. 2000.
[14] S. Makineni and R. Iyer, “Architectural Characterization of TCP/IP Packet Processing on the Pentium M microprocessor,” Int’l Conf. on High Performance Computer Architecture (HPCA-10), Feb 2004.
[15] D. McConnell, “IP Storage: A Technology Overview”, http://www.dell.com/us/en/biz/topics/vectors_2001-ipstorage.htm
[16] J. C. Mogul. Observing TCP Dynamics in Real Networks. In ACM SIGCOMM, pages 305{317, 1992.
[17] J. Mogul, “TCP offload is a dumb idea whose time has come,” A Symposium on Hot Operating Systems (HOT OS), 2003.
[18] J. Postel, Ed., “Internet Protocol - DARPA Internet program protocol specification,” RFC 791, Sep. 1981.
[19] J. B. Postel, “Transmission Control Protocol”, RFC 793, Information Sciences Institute, Sept. 1981.
[20] M. Rangarajan et al., “TCP Servers: Offloading TCP/IP Processing in Internet Servers. Design, Implementation, and Performance,” Rutgers University, Technical Report, DCS-TR-481, March 2002.
[21] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, et al., “TCP onloading for data center servers,” IEEE Computer, vol. 37, no. 11, pp. 48–58, Nov. 2004.
[22] S. Sinha, S. Katula and D. Katabi, “Harnessing TCP’s Burstiness with Flowlet Switching,” 3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets III), San Diego, CA, Nov 2004
[23] “The TTTCP Benchmark”, http://ftp.arl.mil/~mike/ttcp.html
[24] D. Yates., “Connection-Level Parallelism for Network Protocols on Shared-Memory Multiprocessor Servers,” Ph.D. Dissertation, University of Massachusetts, Amherst, 1997
[25] L. Zhao, S. Makineni, Ramesh Illikkal, D. Newell and L. Bhuyan, “TCP/IP Cache Characterization in Commercial Server Workloads, Seventh Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-7), along with HPCA-10, February 2004
[26] Chariot – More information available at http://www.ixiacom.com/solutions/display?skey=ixchariot