FAKULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Degree course: Electronics

Patrik Csókás

Optimising network hardware in business messaging environments

Bachelor work

Supervisor: Ing. Pavol Malošek, PhD.

May 2009

Acknowledgement: I would like to thank my supervisor, Ing. Pavol Mališek, PhD. and Mgr. Martin Sústrik, CEO FastMQ Inc. for the patient guidance, encouragement and advice they have provided throughout creating this study. I also would like to thank FastMQ Inc. for providing the possibility of realizing my measurements.

Matter

ANOTÁCIA

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Študijný program: Elektronika

Autor: Patrik Csókás
Názov bakalárskej práce: Optimalizácia sieťového hardvéru pre business messaging.
Vedúci bakalárskej práce: Ing. Pavol Malošek, PhD.
Rok odvzdania: máj 2009

Mnoho veľkých spoločností a bánk potrebujú veľmi vysokú rýchlosť prenosu dát s cieľom deliť data s ostatnými. Cieľom tejto práce je vytvoriť domukentáciu o obchodnom správnom systéme, pretože v súčasnosti nič podobné na trhu neexistuje.

ANNOTATION

Slovak university of technology in Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY
Degree course: Electronics

Author: Patrik Csókás
Title of the bachelor thesis: Optimising network hardware in business messaging environments
Supervisor: Ing. Pavol Malošek, PhD.
Year of submission: May 2009

Many big companies, banks need very high data transfer rates to share data with other . The goal of this work is to create a documentation about the business messaging system, because at the moment nothing like that exists on the market.

List of used signs

ALU	Arithmetic Logic Unit
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ASMP	Asymmetric Multiprocessoring
CPU	Central Processing Unit
DDS	Data distribution service
ESB	Enterprise Service Bus
FSB	Front Side Bus
HBA	Host Bus Adapter
IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
iSCSI	Internet Small Computer System Interface
JMS	Java Message Service
LAN	Local Area Network
MPI	Message Passing Interface
MSI	Message Signaled Interrupts
NIC	Network Interface Card
PAN	Personal Area Network
RAM	Random Access Memory
SAN	Storage Area Network
SMP	Symmetric Multiprocessoring
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TOE	TCP Offload Engine
TSO	TCP segmentation Offload
USB	Universal Serial Bus
VPN	Virtual Private Network
WAN	Wide Area Network
XML	eXtensible Markup Language

1. Messaging introduction

Middleware is computer messaging software that connects software components or applications allowing to efficiently pass messages among them. These softwares consist of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. This technology evolved to provide interoperability in support of the move to coherent distributed architectures, which are used most often to support and simplify complex, distributed applications. It includes web servers, application servers, and similar tools that support application development and delivery. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture. Figure 1.1. shows the model of messaging, with subcategories.[1]

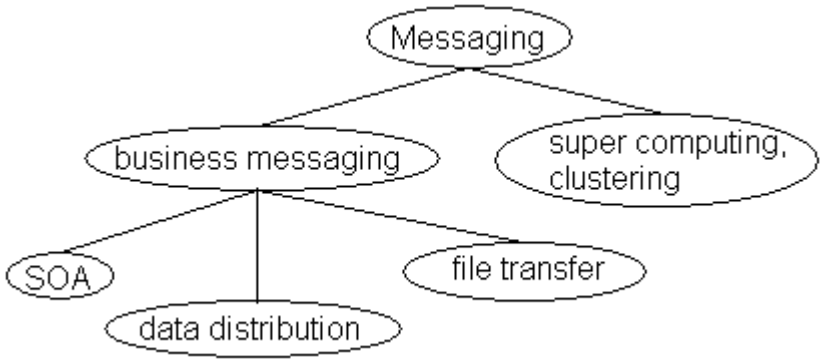


Figure 1.1. The messaging model

It sits "in the middle" between application software working on different machines, different operating systems or even on different continents. It is similar to the middle layer of a three-tier single system architecture, except that it is stretched across multiple systems or applications. Examples include database systems, telecommunications software, transaction monitors, and messaging-and-queueing software.[1]

Middleware, in the form of messaging servers and clients, is a platform on which software architects can create larger-scale applications by combining applications written using different technologies, running on different types of hardware. It is often significantly cheaper to develop independent applications and connect them using good middleware, than to develop a single all-encompassing application.[2]

All working large-scale software projects necessarily use middleware of one kind or another. In most cases ad-hoc middleware is built for the needs of the project. Many commercial products also exist. Open source products are also available. Many projects build their own middleware. The problem often seems simple ("get the data from here to there, reliably and quickly") but the middleware aspects of projects often become the most complex and troublesome parts. Successful middleware products and technologies tend to occupy a specific niche: e.g. connecting components on a single platform, or for a single language environment. The few general-purpose commercial middleware products (e.g. IBM MQ Series, BEA Tuxedo) are typically very expensive and very complex.[3]

Lastly, there are some open source (mainly Java) middleware applications but these tend to be featureware, not focused on standards, or rather, implementing so many different standards that there is no guarantee of interoperability even when using a single product. There are some middleware standards (e.g. JMS, CORBA, AMQP) but these are limited in scope. JMS, for instance, is exclusively for Java applications, though some JMS providers make non-standard APIs for other languages, and CORBA uses a complex object-centered model which is unsuitable for many kinds of application. One of the promising ways to go is AMQP.[3]

The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for Message Oriented Middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security. The Advanced Messaging Queuing Protocol can be integrated into existing products or provide open interoperability for API's like JMS. The Advanced Message Queuing Protocol can be used with most of the current messaging and Web service specifications such as JMS, SOAP, WS-Security, WS- Transactions, and many more, complementing the existing work in the industry.[4]

1.1 Business messaging

1.1.1 Service Oriented Architecture

In computing, SOA provides methods for systems development and integration where systems group functionality around business processes and package these as interoperable services. SOA also describes IT infrastructure which allows different applications to exchange

data with one another as they participate in business processes. One can define a service-oriented architecture as a group of services that communicate with each other. The process of communication involves either simple data-passing or two or more services coordinating some activity. Intercommunication implies the need for some means of connecting services to each other.[6]

In order for clients to connect to services and use them well-defined communication protocol should be used over a network. In ideal case all the clients and all the services should be using the same protocol. The base information carrier in such a environment is called message.[8]

At this point everything looks simple - there is protocol, messages, services and clients that can communicate seamlessly. However, in the case of big amount of services and clients, communication problems are arising. Main problem is with the number of connections. In the case of 10 clients where each of them is using 10 services, 100 connections have to exist on a network and each client has to manage all 10 connections with respect to availability, security, queueing, load balancing etc. [8]

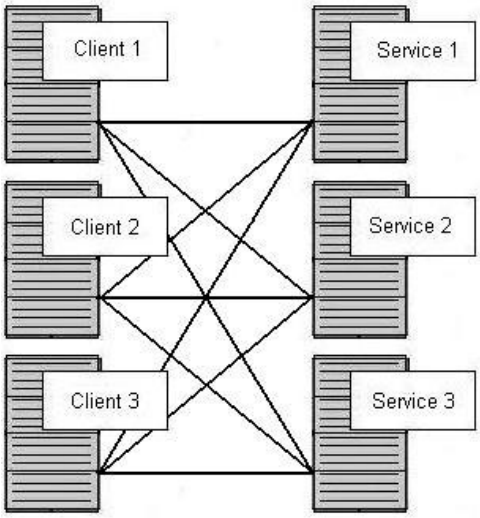


Figure 1.1.1.1. SOA model

This is simply not acceptable and therefore a centralised highly-available component has to exist on a network which is "delivering" messages from source to destination(s).

"Delivery" can have a lot of meanings and it usually refers to quite a complex algorithm implemented by so called message broker to form what is known as Enterprise Service Bus (ESB).[8]

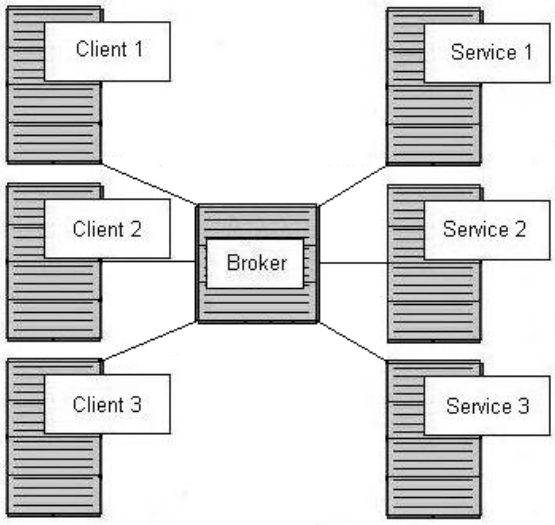


Figure 1.1.1.2. SOA model with a broker

The simplest possible architecture consists of one service, one client and a broker which is managing the communication between them. If high-availability of service is required or there is a need for load balancing, multiple services instances can exist at the same time. Also, there can be different services running at a single moment.[8]

The figure 1.1.1.2. illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

1.1.2 Data distribution

Data distribution is a specification of a networking middleware for distributed systems that standardizes and simplifies complex network programming. It proposes a publish/subscribe model for sending and receiving data, events, and commands among the nodes. Nodes that are producing information (publishers) create "topics" (e.g., temperature,

location, pressure) and publish "samples." It takes care of delivering the sample to all subscribers that declare an interest in that topic.[9]

Data distribution handles all the transfer issues: message addressing, delivery, flow control, retries, etc. Any node can be a publisher, subscriber, or both simultaneously.[9]

Data distribution supports mechanisms that go beyond the basic publish-subscribe model. The key benefit is that applications that use data distribution for their communications are entirely decoupled. Very little design time has to be spent on how to handle their mutual interactions. In particular, the applications never need information about the other participating applications, including their existence or locations. Data distribution automatically handles all aspects of message delivery, without requiring any intervention from the user applications, including:[9]

- determining who should receive the messages,
- where recipients are located,
- what happens if messages cannot be delivered.
-

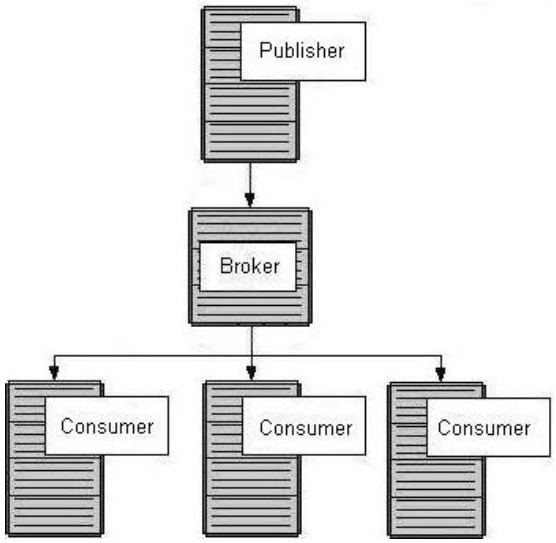


Figure 1.1.2.1 Data distribution model

The Figure 1.1.2.1 illustrates how the publisher is sending data.

1.1.3 File transfer

The file transfer method is many years old. It has a very big latency, so the usage is not efficient. It was used earlier by banks, which needed synchronization. Figure 1.1.3.1 shows the model of the method. The small banks upload their data to the central bank at 24:00, the central processes it till morning – 4:00. Than it is downloaded by the small banks.

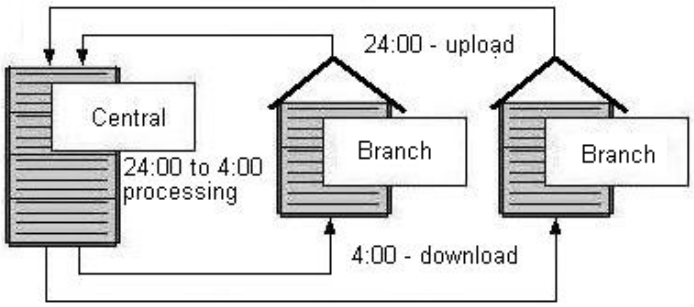


Figure 1.1.3.1 File transfer model

1.2. Super computing (clustering)

Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is used in computer clusters and supercomputers. MPI is a language-independent communications protocol used to program parallel computers. Both point-to-point and collective communication is supported. MPI "is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today.[11]

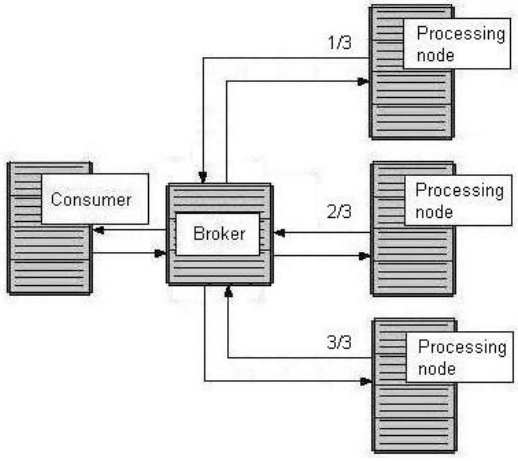


Figure 1.2.1. Super computing model

2. Hardware in business messaging

Computer hardware consists of many small components, which are connected by various buses and wires. The most commonly used platform in business messaging is x86, which refers to the most commercially successful instruction set architecture. An instruction set is a list of all the instructions, and all their variations, that a processor can execute.

2.1. Southbridge and Northbridge

Over the years personal computers and smaller servers standardized on a chipset with two parts: the Northbridge and Southbridge, as shown on the figure 2.1.1.

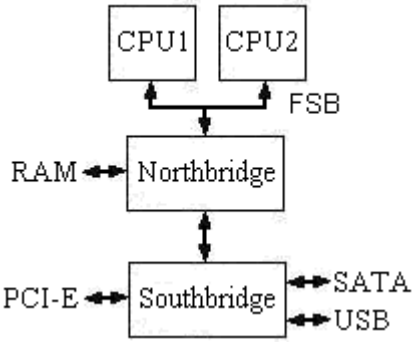


Figure 2.1.1. Structure with Northbridge and Southbridge

All CPUs are connected via a common Front Side Bus (FSB) to the Northbridge. The Northbridge contains the memory controller, and its implementation determines the type of RAM chips used for the computer. Different types of RAM, such as DRAM, Rambus, and SDRAM, require different memory controllers. To reach all other system devices, the Northbridge communicates with the Southbridge. The Southbridge, often referred to as the I/O bridge, handles communication with devices through a variety of different buses. Today the PCI, PCI Express, SATA, and USB buses are of most importance, but PATA, IEEE 1394, serial, and parallel ports are also supported by the Southbridge.[12]

2.2. Front Side Bus

The Front Side Bus (FSB) is the bus, that carries data between the CPU and the Northbridge. Depending on the processor used, some computers may also have a back side bus that connects the CPU to the CPU cache. This bus and the cache connected to it is faster than

accessing the system memory via the front side bus. The bandwidth of the FSB is determined by the product of the width of its data path, its clock frequency, and the number of data transfers it performs per clock cycle. The frequency at which a processor operates is determined by applying a clock multiplier to the front side bus speed in some cases.[14]

2.3. CPU

The computer CPU is responsible for handling all instructions and calculations it receives from other hardware components in the computer and software programs running on the computer.

2.3.1. Von Neumann architecture

The von Neumann architecture is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data.[15]

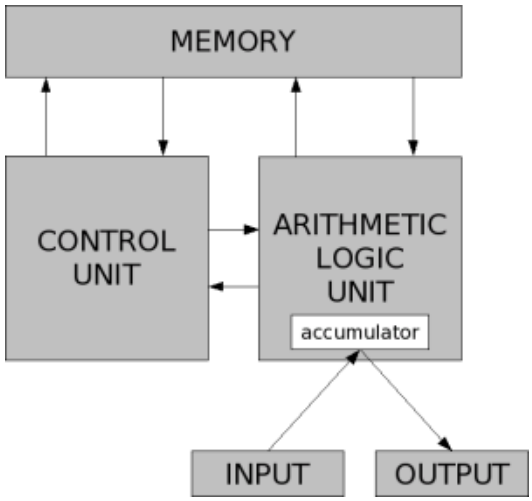


Figure 2.3.1.1. Von Neumann architecture

2.3.2. The CPU cache

A CPU cache is a cache used by the central processing unit of a computer to reduce the average time to access memory. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.[16]

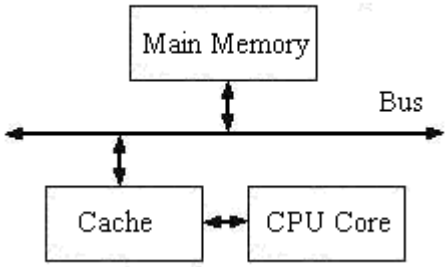


Figure 2.3.2.1. Cache configuration

Figure 2.3.2.1. shows the minimum cache configuration of a system. The connection between the CPU core and the cache is a special, fast connection. The main memory and the cache are connected to the system bus which can also be used for communication with other components of the system. Even though most computers for the last several decades have used the von Neumann architecture, experience has shown that it is of advantage to separate the caches used for code and for data.[12]

2.3.3. Symmetric multiprocessoring

In computing, symmetric multiprocessing or SMP involves a multiprocessor computer-architecture where two or more identical processors can connect to a single shared main memory. Most common multiprocessor systems today use an SMP architecture. In the case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors.[17]

SMP systems allow any processor to work on any task no matter where the data for that task are located in memory; with proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.[17]

2.3.4. Asymmetric multiprocessoring

ASMP varies greatly from the standard processing model that we see in personal computers today. Where as a symmetric multiprocessor treats all of the processing elements in the system identically, an ASMP system assigns certain tasks only to certain processors. In particular, only one processor may be responsible for fielding all of the interrupts in the system or perhaps even performing all of the I/O in the system. This makes the design of the

I/O system much simpler, although it tends to limit the ultimate performance of the system. Asymmetric parallelism is particularly attractive because it allows the use of custom processing elements optimized to execute a limited range of tasks using much less power than a general purpose processor can achieve.[18]

2.4. Memory types

There are two types of RAM which are used to store data: SRAM (static RAM) and DRAM (dynamic RAM). SRAM is much faster, and provides the same functionality, but for higher costs.

2.4.1. SRAM

Static RAM holds its data without external refresh, for as long as power is supplied to the circuit. It uses bistable latching circuitry to store each bit.[12]

2.4.2. DRAM

Dynamic RAM is, in its structure, much simpler than static RAM. All it consists of is one transistor and one capacitor. This huge difference in complexity of course means that it functions very differently than static RAM. DRAM must be refreshed many times per second in order to hold its data contents. [12]

2.5. PCI

The Conventional PCI is a computer bus, which is used of attaching hardware devices in a computer. The name is an initalism formed from Peripheral Component Interconnect. [20

2.5.1. Standard PCI

The PCI specification covers the physical size of the bus, electrical characteristics, bus timing, and protocols. The width of the standard PCI slots is 32 or 64 bits, the capacity is 133 MB/s.[20]

33 MHz PCI	133 MB/s
66 MHz PCI	266 MB/s

Table 2.5.1.1 Speeds of different slots

2.5.2. PCI Extended (PCI-X)

The PCI-X is a computer bus and expansion card standard that enhances the PCI Local Bus for higher bandwidth demanded by servers. It is a double-wide version of PCI< running at up to four times the clock speed, but is otherwise similiar in electrical implementation and uses the same protocol. The clock speed in PCI-X is 133Mhz, and supports up to 64 bits at 66MHz. Theoretical maximum amount of data exchanged between the processor and peripherals with PCI-X is 1.06 GB/s. PCI-X also improves the fault tolerance of PCI allowing, for example, faulty cards to be reinitalized or taken offline.[21]

100 MHz PCI-X	800 MB/s
133 MHz PCI-X	1 GB/s

Table 2.5.2.1 Speeds of different slots

2.5.3. PCI Express (PCI-e)

The PCI Express is a computer expansion card standard designed to replace the older PCI-X, PCI and AGP standards. PCIe is used in server, consumer and industrial applications, both as a motherboard-level interconnect and as an expansion card interface for add-in boards. The maximum speed of the PCIe standard is up to 4GHz, with a transfer rate of 8GT/s and with a 1GB/s data rate. The PCIe has many advantages. The serial interface of PCIe suffers fewer such problems and therefore requires less complex and less expensive designs. PCI-X buses, like standard PCI are half-duplex bidirectional whereas PCIe buses are full-duplex bidirectional, and PCI-X buses run only as fast as the slowest device whereas PCIe devices are able to independently negotiate the bus speed.[22]

PCI Express 1x	250 MB/s
PCI Express 2x	500 MB/s
PCI Express 4x	1000 MB/s
PCI Express 8x	2000 MB/s
PCI Express 16x	4000 MB/s
PCI Express 32x	8000 MB/s

Table 2.5.3.1 Speeds of different slots

2.5.3.1. Message Signaled Interrupts

Message Signaled interrupts are a way of generating an interrupt. Traditionally, a device has an interrupt pin which it asserts when it wants to interrupt the host CPU. While PCI Express does not have separate interrupt pins, it has special messages to allow it to emulate a pin assertion or deassertion. Message Signaled Interrupts allow the device to write a small amount of data to a special address in memory space. The chipset will deliver the corresponding interrupt to a CPU. There are two different extensions to support Message Signaled Interrupts: MSI, MSI-X. [23]

MSI-X interrupts are enhanced versions of MSI interrupts that have the same features as MSI interrupts, with the differences, that there are a maximum of 2048 MSI-X interrupt vectors supported per device, address and data entries are unique per interrupt vector, and MSI-X supports per function masking and per vector masking. [24]

2.6. Network Cards

A network interface card (NIC) works by sending and receiving signals over some type of media and therefore allow computers to communicate. This media can be a Ethernet, Fiber optics, Wireless or even Powerful line connection. When the computer performs some action, the signal gets converted to a series of bits of information called packets. There are protocols, or particular software, that controls the way these signals are transmitted. This protocol is also known as TCP/IP and is the most popular protocol used on networks and on the Internet. When a cable is connected to the network interface card, and the other end is also connected, one computer can communicate to the other computer simply by the method of conversion. Whatever process is undertaken, it gets sent to the network interface card. The network interface card takes the incoming process and converts it and disassembles it into packets so it can be transported over the network. [25]

2.6.1. TCP offload engine

The TCP offload engine (TOE) is a technology, which is used in network interface cards to offload processing of the entire TCP/IP stack to the network controller. It is used, where high-speed is recommended (gigabit Ethernet, 10 gigabit Ethernet), mostly when the processing overhead of the network stack becomes significant. At the moment most end point hosts are PCI based, but the number of PCIe based hardwares are increasing. [27]

The main types of TCP/IP offload are:

- **Paralell Stack Full Offload** – it consists of two stacks, from which the first is included with the hosts OS, and the second one is connected between the Application layer, using the Internet Protocol Suite naming conventions, and the Transport layer using a "vampire tap". The vampire tap intercepts TCP connection requests by applications and is responsible for TCP connection management as well as TCP data transfer.[27]
- **HBA Full Offload** – it is found in iSCSI Host Bus Adapters, which present themselves as Disk Controllers to the Host System while connecting (via TCP/IP) to an iSCSI Storage Device. This type of TCP Offload not only offloads TCP/IP processing but it also offloads the iSCSI Initiator Function. Because the HBA appears to the host as a Disk Controller it can only be used with iSCSI devices and is not appropriate for general TCP/IP Offload.[27]
- **TCP Chimney Partial Offload** – it addresses the major security criticism of Parallel Stack Full Offload. In Partial Offload the main System Stack controls all connections to the host. After a connection has been established between the local host (usually a server) and a foreign host (usually a client) the connection and its state are passed to the TCP offload engine. The heavy lifting of data transmit and receive is handled by the offload device. Almost all TCP offload engines use some type of TCP/IP hardware implementation to perform the data transfer without host CPU intervention. When the connection is closed, the connection state is returned from the offload engine to the main system stack. Maintaining control of TCP connections allows the main system stack to implement and control connection security.[27]

2.6.2. TCP checksum offload

The checksum offload is used in network adapters. It is an option, which when is enabled, the adapter is responsible for computing the checksum of the incoming or outgoing TCP message, resulting in reduced calculations for the CPU. The CPU gives a random value to fill the checksum field of the TCP header until the message reaches the adapter. The savings vary by packet size. Small packets have little or no savings with this option, while large packets have larger savings. On the PCI-X GigE adapters, the savings for MTU 1500 are

typically about 5% reduction in CPU utilization, and for MTU 9000 (Jumbo Frames) the savings is approximately a 15% reduction in CPU utilization.[28], [29]

2.6.3. Scatter-Gather

To achieve gigabit throughput, it is important that the operating system does not copy the data in the packets before sending them (this is called *zero-copy IP*). Unfortunately, the kernel needs to put a header before the data in the packet, so not copying the data to a buffer in kernel space means that the NIC needs to be able to fetch the header from a different place in memory than the user data in the packet. This is called *scatter/gather* and is necessary for zero-copy IP.[30]

2.6.4. TCP segmentation offload -TSO

The TSO is a technique for increasing outbound throughput of high-bandwidth network connections by reducing CPU overhead. It is a stateless offload in which the TCP layer passes a very large segment (larger than the maxim segment size for the connection) through the stack to the network interface, which is expected to segment it into a number of packets. This method improves performance by reducing per-packet software overheads within the network stack. The segmentation is usually performed by the network adapter and requires complex silicon or an embedded processor to generate headers for each packet. Alternatively, it can be implemented in software at the lowest layer of the stack, as is done in Linux generic segmentation offload. A stateful offload is a network interface acceleration based on the state contained in upper-layer protocols within a sequence of frames. Where TCP is the higher-level protocol, a stateful offload adapter is known as a TCP offload engine (TOE). The TOE also contains a complete implementation of a TCP/IP stack that is distinct from that of the host OS.[31], [32]

2.6.5. Generic segmentation offload – GSO

GSO is a performance optimization, which is a generalisation of the concept of TSO. A lot of savings in TSO come from traversing the network stack once rather than many times for each super-packet. These savings can be obtained without hardware support. The key to minimising the cost in implementing this is to postpone the segmentation as late as possible. In the ideal world, the segmentation would occur inside each NIC driver, where they would rip

the super-packet apart and either produce SG lists which are directly fed to the hardware, or linearise each segment into pre-allocated memory to be fed to the NIC. This requires modifying each and every NIC driver, so it would take quite some time. A much easier solution is to perform the segmentation just before the entry into the drivers xmit routine. This concept is called Generic segmentation offload.[33]

2.6.6. Interrupt coalescing

The interrupt coalescing is a feature, which is implemented into the NIC. It allows a reception of a group of network frames to be notified to the operating system kernel via a single hardware interrupt, thus reducing the interrupt processing overhead, particularly at high packet rates. The delays introduced by interrupt coalescing can result in significant TCP throughput degradation. In interrupt-coalescing mode, a single interrupt is generated for multiple incoming packets. This is opposed to normal interruption mode in which an interrupt is generate for every incoming packet. It is widely asserted that interrupt coalescing decreases interrupt overhead at the expense of latency.[34]

3. Networking

Networks allow computers to communicate with each other and share resources, informations. Networks are often classified as Local Area Networks (LAN), Wide Area Networks (WAN), Metropolitan Area Networks (MAN), Personal Area Networks (PAN), Virtual Private Networks (VPN), Campus Area Networks (CAN), Storage Area Networks (SAN), etc. depending on their scope, purpose and scale. The method, how the data is sent between computers and networks, is defined by various protocols and models.

3.1. OSI model

The OSI model (Open Systems Interconnection Reference Model) is an abstract description for layered communications and computer network protocol design. It divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layer, as illustrated on Figure 3.1.1. It is therefore often referred to as the OSI Seven Layer Model.[35]

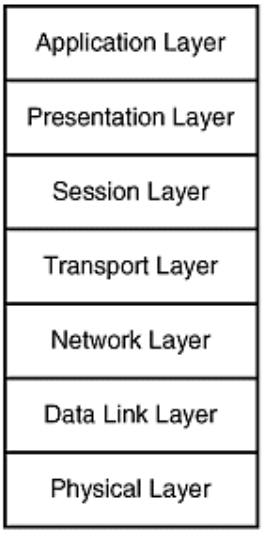


Figure 3.1.1. OSI model

Presentation layer basically takes the input, decodes, encodes, and compresses the data so it can be transmitted over the network at great speeds. The next layer that controls how the data is sent over the network is the Session layer. The Session layer sends a signal to the network interface card and tells it to open the lines of communication so the data can be sent

out. The Transport layer actually takes the data and transports it from the network interface card to another network interface card. The Network layer adds information to the data that the Transport layer sends out so other machines will know what computer sent out the data. It does this by including an IP address of the machine transmitting and the IP address of the receiving computer as well. The Data Link layer takes the data sent from the Transport and Network layers, and converts all the data to frames. The Physical layer is what manipulates and controls the way the data is sent over the network. Every one of the above layer is not an actual layer of hardware, it is just a representation of how the data is converted and transmitted from the network interface card, over the network to the other computer. [25]

3.2. TCP/IP model

The Internet Protocol Suite (TCP/IP) is a set of communication protocols, which are used for the Internet and other similar networks. TCP/IP model consists of 4 layers, from which each layer solves set of problems involving the transmission of data, and provides service to the upper layer protocols based on using services from some lower layers. In the model the IP is responsible for moving packet of data from node to node, while the TCP is responsible for verifying the correct delivery of data from client to server.[36]

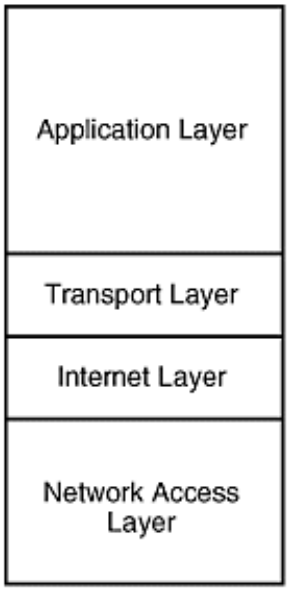


Figure 3.2.1. TCP/IP model

4. Measurements and graphs

The measurements were made under Linux operating system with the softwares Ethtool and Netperf between four servers Aladar => Aranyka (Figure 4.1) and Janos => Csaba (Figure 4.2). Hardware configuration of the servers:

Janos

Linux Debian 5.0
kernel: 2.6.26-2-amd64
CPU: 2 x Quad core Intel Xeon E5440; 2.83GHz
RAM: 4GB
NIC: Intel Pro/1000; PCIe 2.5GT/s, Width 4x

Csaba

Linux Debian 5.0
kernel: 2.6.26-2-amd64
CPU: 2 x Quad core AMD Opteron 8356, 2.3GHz
RAM: 4GB
NIC: Broadcom NetXtreme BCM5721; PCIe 2.5GT/s,Width 1x

Aladar

Linux Debian 5.0
kernel: 2.6.26-2-amd64
CPU: 1 x Dual core AMD Athlon 64 X2 3800+
RAM: 2GB
NIC: SysKonnekt SK-9E21D; PCIe 2.5GT/s, Width 1x

Aranyka

Linux Debian 5.0
kernel: 2.6.26-2-amd64
CPU: 1 x Dual core Intel Pentium 4 (HT), 3GHz
RAM: 2GB
NIC: SysKonnekt SK-9E21D; PCIe 2.5GT/s, Width 1x

Ethtool is a Linux software, which is used for querying settings of an Ethernet device and changing them. Netperf is a benchmark software, that can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency. It consists of two components: client, server. The client was installed on Aladar, Janos, the server on Aranyka and Csaba.

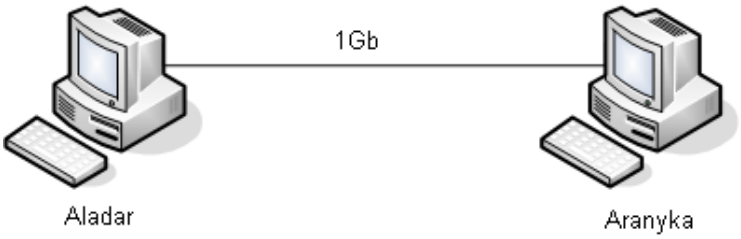


Figure 4.1. Connection between test computers 1

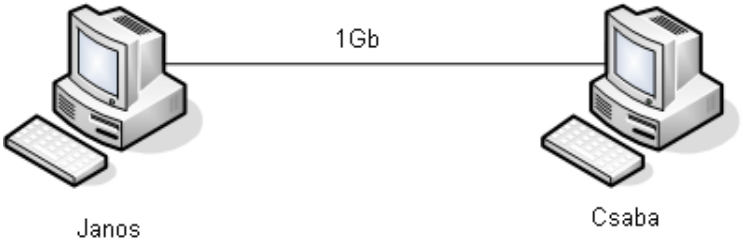


Figure 4.2. Connection between test computers 2

The connection between the computers was straight, without any other device (switch, hub, router). The speed of the network cards was 1Gb/s.

4.1. Measuring the latency

The first connection was used to measure the latency between Aladar and Aranyka. After setting up the baseline (checksum and GSO was turned off on the network card), the testing with netperf could have been started. Commands for changing the options of the network card with ethtool:

Aladar> ethtool -K eth1 tx off rx off tso off gso off sg off (turn off all)

Command for testing the request/response time of two computers:

Aladar> netperf -H 10.0.0.2 -t TCP_RR -- -r 1

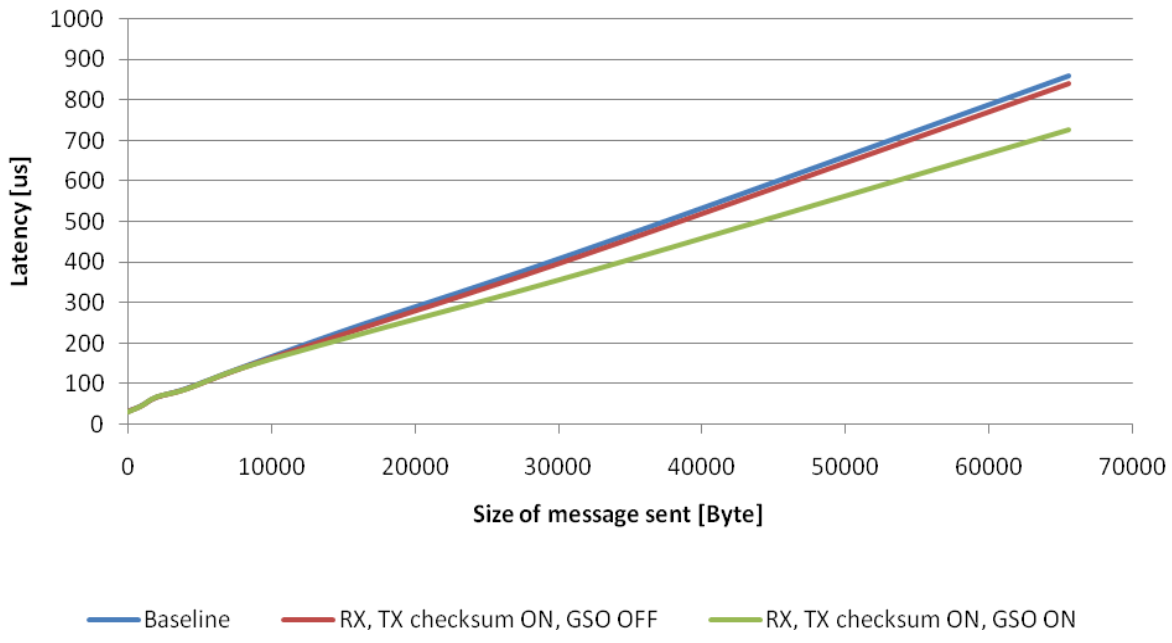
Each message size was tested five times, and from the results average was taken. The latency (the delay between the initiation of a network transmission by a sender and the receipt of that transmission by a receiver) was divided by 2 (to get one way latency) was calculated with the following formula:

$$\text{latency [s]} = \frac{1}{\text{request-response rate [s}^{-1}\text{]}}$$

Formula 4.1.1. Formula of latency

Size of message sent [Byte]	Latency [us] Baseline	Latency [us] RX, TX checksum ON, GSO Off	Latency [us] RX, TX checksum On, GSO On
1	29.96	29.97	32.21
2	30.4	29.98	32.21
4	30.21	30.08	32.18
8	30.13	30.11	32.19
16	30.39	30.2	32.39
32	30.63	30.48	32.73
64	32.26	32.06	33.28
128	32.85	32.72	33.84
256	34.93	34.7	36.03
512	38.28	37.93	39.67
1024	46.89	46.16	48.53
2048	67.15	66.88	69.79
4096	88	86.77	88.79
8192	142.97	141.28	143.1
16384	246.49	237.62	226
32768	442.11	429.9	385.59
65536	859.55	840.65	727.87

Table 4.1.1. Differences between latency



Graph 4.1.1. Differences between latency

The second measurement was made also between Aladar and Aranyka, with a complete middleware messaging system – ØMQ middleware. The command for starting the server:

Aladar:> /tmp/zmq/zmq_server/zmq_server

For measuring the latency 2 commands were used. On the client side (Aladar):

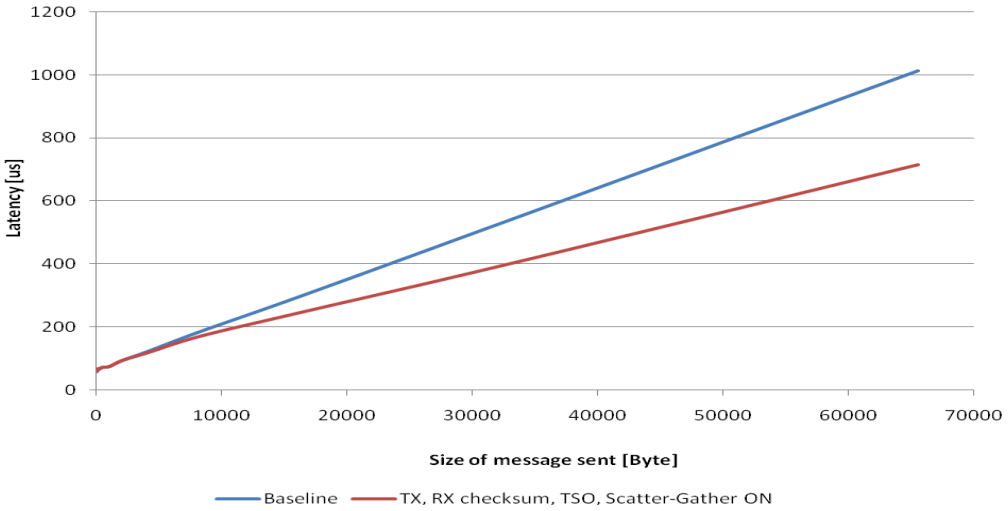
Aladar> ./perf/tests/zmq/local_lat localhost eth1 eth1 1 50000

On the remote side (Aranyka):

Aranyka> ./perf/tests/zmq/remote_lat 10.0.0.1 1 50000

Size of message sent [Byte]	Latency [us] Baseline	Latency [us] TX, RX checksum, TSO, Scatter-Gather ON
1	60.32	61.26
2	62.22	60.88
4	59.32	63.19
8	64.78	63.66
16	60.29	61.43
32	60.28	61.19
64	65.36	64.77
128	65.09	68.84
256	70.45	68.48
512	73.35	73.36
1024	76.15	75.24
2048	95.15	94.73
4096	123.95	119.18
8192	185.55	170.51
16384	300.23	247.81
32768	537.11	399.34
65536	1012.7	715.33

Table 4.1.2. Differences between latency



Graph 4.1.2. Differences between latency

4.2. Measuring the throughput and CPU load

The first measurement was made between Janos and Csaba. Commands for measuring the throughput and CPU load:

Janos> netperf -H 10.0.0.2 -c -- -m 1

The results are shown in the Table 4.2.1, 4.2.2 and 4.2.3.

Size of message sent [B]	Throughput [MB/s]	CPU load [%]
1	28.42	124.56
2	54.52	117.76
4	109.88	122.56
8	222.49	138.4
16	391.5	126.48
32	674.62	123.36
64	942.2	94.4
128	949.21	51.52
256	949.21	37.04
512	949.21	34.64
1024	949.21	29.04
2048	949.21	33.68
4096	949.21	26.56
8192	949.21	34.16
16384	949.21	13.52
32768	949.21	20.64
65536	949.21	23.6

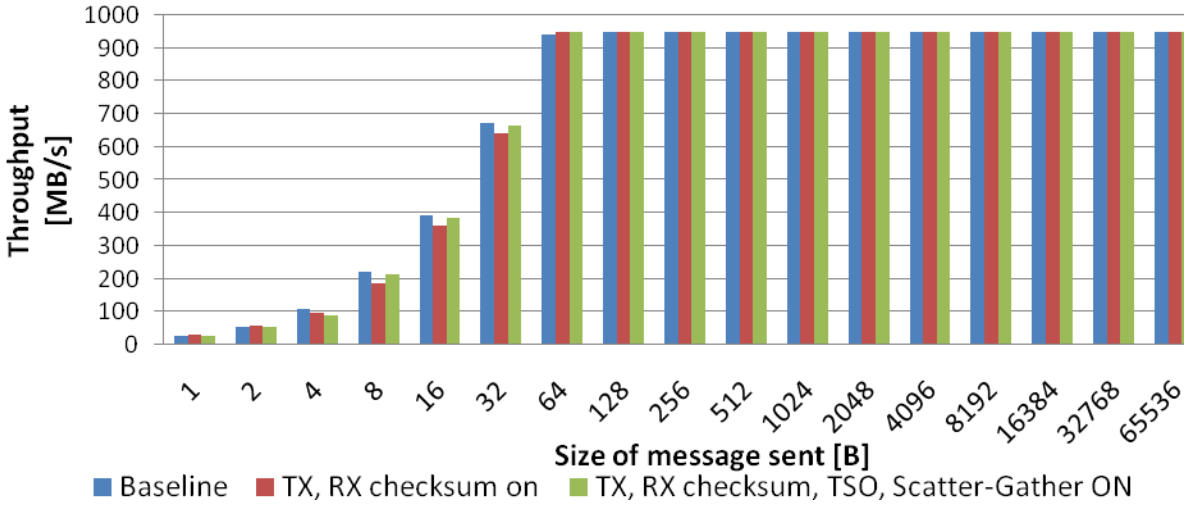
Table 4.2.1. Baseline

Size of message sent [B]	Throughput [MB/s]	CPU load [%]
1	29.25	116.24
2	58.14	110.08
4	95.97	108.4
8	188.15	134
16	360.35	124.48
32	640.75	127.76
64	949.18	76.56
128	949.21	71.52
256	949.21	44.72
512	949.21	14.64
1024	949.21	30.08
2048	949.21	20.88
4096	949.21	23.2
8192	949.21	23.28
16384	949.21	32.32
32768	949.21	13.04
65536	949.21	21.44

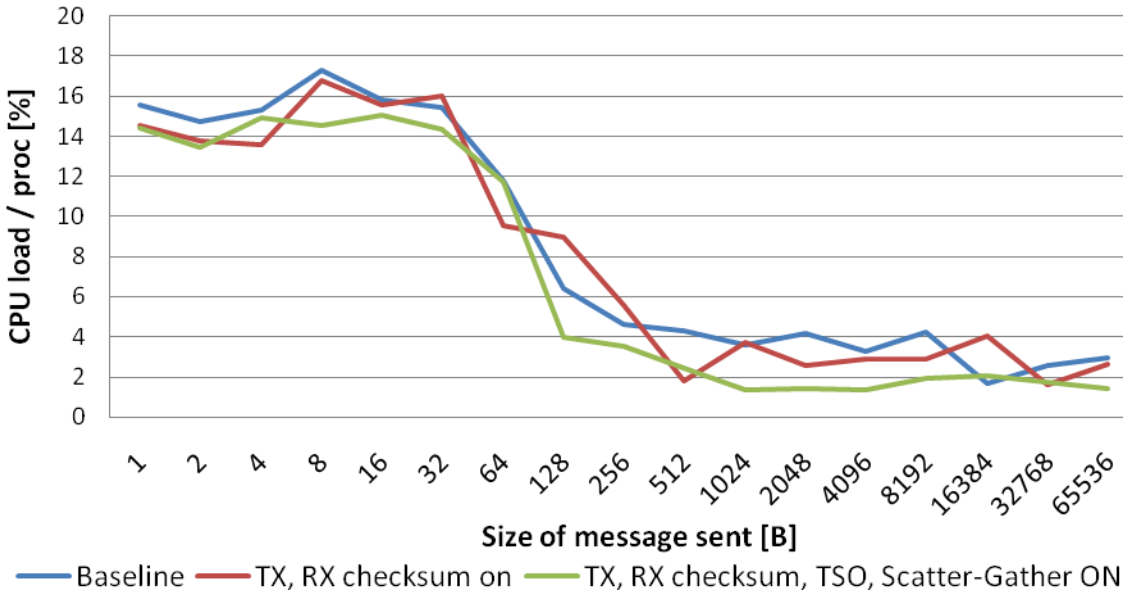
Table 4.2.2. TX, RX checksum on

Size of message sent [B]	Throughput [MB/s]	CPU load [%]
1	27.34	115.2
2	54.98	107.84
4	90.36	119.28
8	214.14	116.48
16	386.81	120.56
32	665.38	114.96
64	949.19	93.76
128	949.21	31.68
256	949.21	28.08
512	949.21	19.36
1024	949.21	10.88
2048	949.21	11.44
4096	949.21	11.04
8192	949.21	15.68
16384	949.21	16.56
32768	949.21	13.84
65536	949.21	11.44

Table 4.2.3. TX, RX checksum, TSO, Scatter-Gather ON



Graph 4.2.1. Differences between throughput



Graph 4.2.2. Differences between CPU load

The second measurement was made between the servers Aladar and Aranyka. The resulst are shown in Tables 4.2.4, 4.2.5, 4.2.6.

Size of message sent [B]	Throughput [MB/s]	CPU load / proc [%]
1	9.2	99.9
2	19.67	99.9
4	37.07	99.9
8	74.78	99.9
16	149.9	99.9
32	236.2	99.9
64	322.52	99.9
128	381.32	99.9
256	435.7	99.9
512	478.07	99.9
1024	506.46	99.9
2048	536.83	99.9
4096	553.07	99.9
8192	587.43	99.9
16384	586.72	99.9
32768	581.63	99.9
65536	566.82	99.9

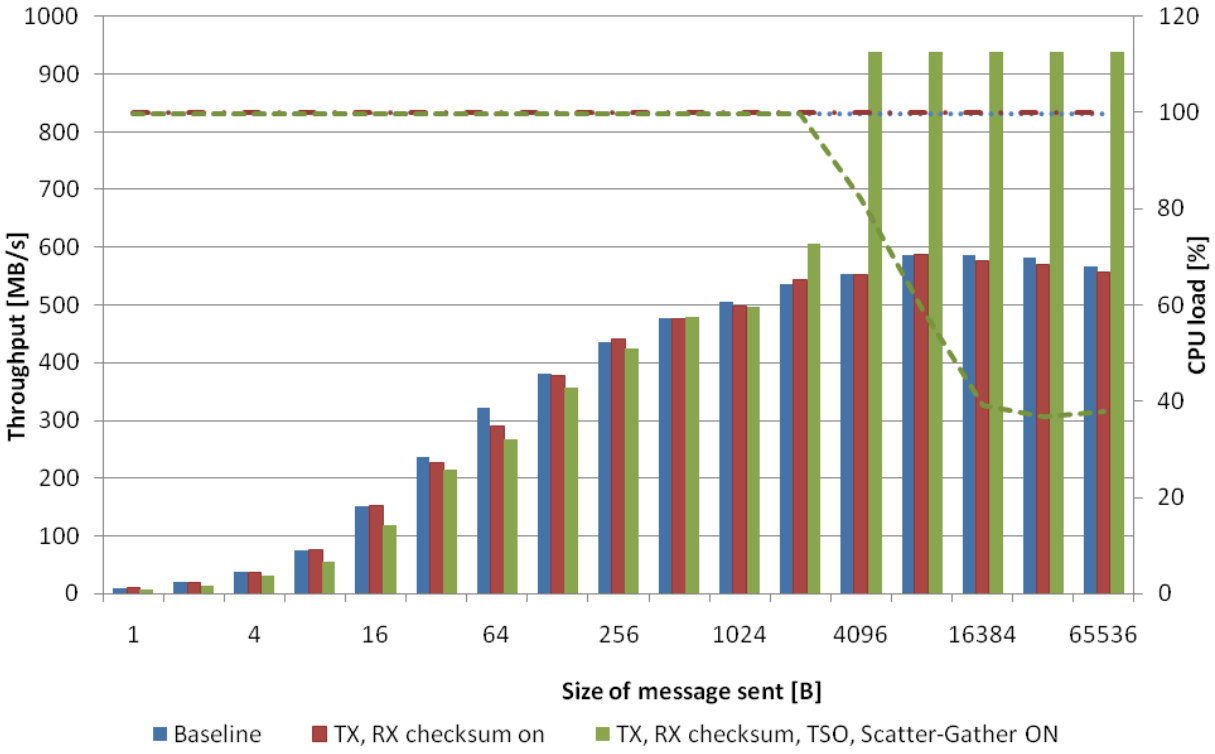
Table 4.2.4. Baseline

Size of message sent [B]	Throughput [MB/s]	CPU load / proc [%]
1	10.65	100
2	20.35	100
4	37.34	100
8	77.43	100
16	152.69	100
32	227.16	100
64	289.92	100
128	378.46	100
256	441.63	100
512	476.06	100
1024	498.41	100
2048	542.86	100
4096	553.09	100
8192	587.3	100
16384	575.41	100
32768	570.02	100
65536	556.88	100

Table 4.2.5. TX, RX checksum on

Size of message sent [B]	Throughput [MB/s]	CPU load / proc [%]
1	7.02	99.9
2	13.67	99.9
4	31.25	99.9
8	54.85	99.9
16	118.43	99.9
32	214.4	99.9
64	265.96	99.9
128	356.54	99.9
256	425.49	99.9
512	479.75	99.9
1024	496.21	99.9
2048	606.33	99.9
4096	938.94	82.3
8192	938.95	59.04
16384	938.95	39.06
32768	938.95	36.78
65536	938.95	37.74

Table 4.2.6. TX, RX checksum, TSO, Scatter-Gather ON



Graph 4.2.3. Differences between throughput and CPU load

The third measurement was made between Aladar and Aranyka, with the ØMQ middleware. For the processor statistics the mpstat module was used, with the following command:

Aranyka> mpstat -P ALL 1

Running the tests:

Aranyka> ./perf/tests/zmq/local_thr localhost eth1 eth1 1 50000000

Aladar> ./perf/tests/zmq/remote_thr 10.0.0.2 1 50000000

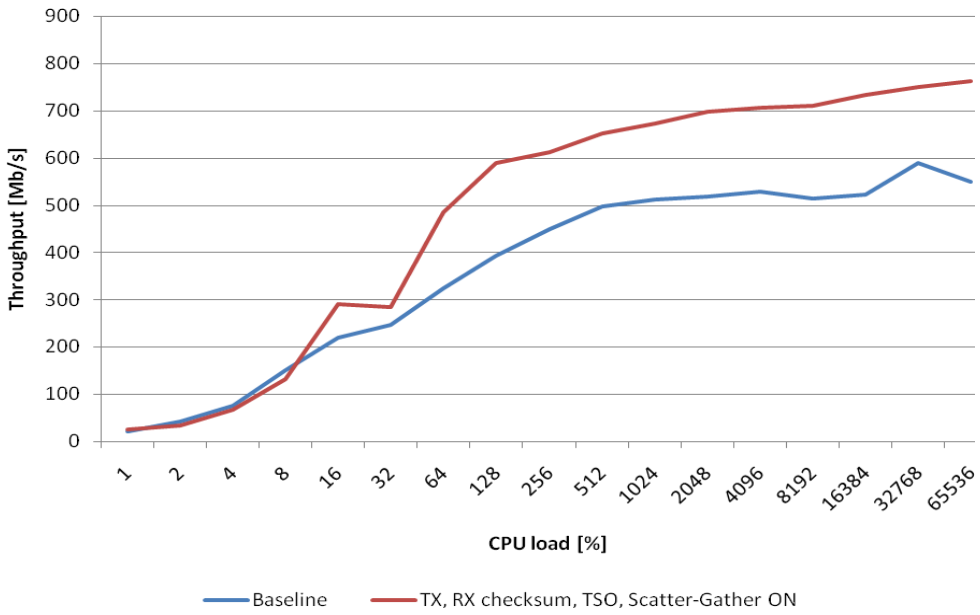
The results of the measurement are shown in Tables 4.2.7. and 4.2.8.

Size of message sent [Byte]	Throughput [MB/s]	Throughput [msg/s]	CPU load [%]
1	20.33	2606934	84.9
2	41.33	2614850	80.46
4	75.33	2374849.66	79.50
8	149.66	2341569.67	77.98
16	218.33	1707506.33	85.54
32	246	963705.33	83.47
64	324	634318	97.47
128	393	409307	94.06
256	448.33	219136.33	94.88
512	497.33	121506.33	104.8
1024	511.66	62526.66	104.12
2048	517.33	31605.33	108.48
4096	528	16128	97.96
8192	514	7846	79.02
16384	522.33	3990.33	70.64
32768	588.66	2247.66	101.93
65536	549.66	1049	97.48

Table 4.2.7. Baseline

Size of message sent [Byte]	Throughput [MB/s]	Throughput [msg/s]	CPU load [%]
1	24.33	3098845	53.94
2	33	2082542	72.69
4	66.66	2091455.66	108.67
8	132	2068998.33	73.02
16	290	2267505.66	82.18
32	283.33	1108615.33	99.49
64	485.66	949367.66	120.6
128	590.33	577023	123.54
256	612.66	299433.66	114.82
512	652.33	159329	109.89
1024	674	82327	109.08
2048	698.66	42666.33	108.6
4096	706.33	21569	112.48
8192	710.66	10850.33	114.4
16384	734.33	5605.66	106.89
32768	750	2862.33	104.5
65536	763	1457	102.7

Table 4.2.8. TX, RX checksum, TSO, Scatter-Gather ON



Graph 4.2.4 Differences between througput

5. Evaluation

The difference measurements of the network cards' settings showed visible changes in the performance. The results are illustrated in graphs.

Measuring the latency with netperf showed insignificant differences. At small message sizes there was no latency difference. From 1500B message size with checksum on, GSO on latency started to get smaller. At the end the difference was 263us with the 65536B messages. With ØMQ middleware the difference was more conspicuous, checksum TSO and scatter-gather decreased the one way latency.

Measuring the throughput and the CPU load was more complicated, because they have a strong dependency on each other. If the processor of the computer is fully loaded, the maximum limit of the throughput cannot be reached. The first measurement regarding the bandwidth was made on Janos and Csaba, which have fast CPUs. The graph of the throughput (Graph 4.2.1) shows no difference with options turned on, because the processors were capable of managing the interrupts without problems. Hardware checksumming took a bit from the CPU load as shown in Graph 4.2.2.

The second measurement was more interesant, because it was done on Aranyka and Aladar, which have weaker processors. In this case CPU load was on maximum with software checksumming, and the maximum throughput could not have been reached (Graph 4.2.3). With hardware checksumming the network interface card took the weight from the operating systems' kernel, and it was able to reach the maximum throughput – 1Gb/s.

The third measurement was made with ØMQ middleware. In Graph 4.2.4. the difference is visible from 8kB messages (when the sending buffer is full).

6. Conclusion

In computing optimization plays a significant role. Modifying a system to work more efficiently or use fewer resources will provide more benefit. In networking throughput, CPU load and latency are often mentioned concepts. The goal of this work was to create documentation about optimising network hardware for business messaging, which was fulfilled and confirmed by measurements. The detailed description of hardware creates a good overview to understand how things are working in computer systems. The OSI and TCP/IP model describes how a packet is transmitted over a network. If the kernel of an operating system is doing the operations needed to transmit data, it will increase the CPU load, and decrease the throughput. When these operations are moved to hardware level, the kernel can work on other operations, that way throughput becomes bigger, and the CPU load smaller. The ØMQ middleware helped to analyze the differences of hardware and software checksumming. If the message size is small, there is no difference regarding the hardware and software, but when they start to get bigger - the send buffers are filled – the network card manages the load. So when investing into top line computers to achieve high bandwidth rates, both CPU and the network cards need to be chosen well, because they depend on each other.

Literature

[1] Wikipedia 2009. Middleware
http://en.wikipedia.org/wiki/Middleware

[2] Zeromq 2008. Background to AMQP
http://www.zeromq.org/whitepapers:amqp-analysis

[3] Zeromq 2008. Background to AMQP
http://www.openamq.org/doc:amqp-background

[4] Wikipedia 2009. Advanced Message Queuing Protocol
http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

[5] Zeromq 2008. Market analysys
http://www.zeromq.org/whitepapers:market-analysis

[6] Wikipedia 2009. Service-oriented architecture
http://en.wikipedia.org/wiki/Service-oriented_architecture

[7] Barry & Associates 2009. Service-oriented architecture architecture
http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

[8] iMatix Corporation 2009. The ESB cookbook
http://www.openamq.org/tutorial:soa

[9] Wikipedia 2009. Data Distribution Service
http://en.wikipedia.org/wiki/Data_Distribution_Service

[10] CrossTalk 2007. Using Switched Fabrics and Data Distribution Service to Develop High Performance Distributed Data-Critical Systems 2007
http://www.stsc.hill.af.mil/crosstalk/2007/04/0704Joshi.html

[11] Wikipedia 2009. Message Passing Interface
http://en.wikipedia.org/wiki/Message_Passing_Interface

[12] Ulrich Drepper, Redhat 2007: What Every Programmer Should Know About Memory
http://people.redhat.com/drepper/cpumemory.pdf

[13] Wikipedia 2009. Southbridge (computing)
http://en.wikipedia.org/wiki/Southbridge_(computing)

[14] Wikipedia 2009. Front Side Bus
http://en.wikipedia.org/wiki/Front-side_bus

[15] Scribd 2008. Von Neumann Computer Architecture
http://www.scribd.com/doc/1532910/Von-Neumann-Computer-Architecture

[16] Wikipedia 2009. CPU Cache
http://en.wikipedia.org/wiki/CPU_cache

[17] Wikipedia 2009. Symmetric multiprocessoring
http://en.wikipedia.org/wiki/Symmetric_multiprocessing

[18] Wikipedia 2009. Asymmetric multiprocessoring
http://en.wikipedia.org/wiki/Asymmetric_multiprocessing

[19] Wikipedia 2009. Instruction set
http://en.wikipedia.org/wiki/Instruction_set_architecture

[20] Wikipedia 2009. Conventional PCI
http://en.wikipedia.org/wiki/Peripheral_Component_Interconnect

[21] Wikipedia 2009. PCI-X
http://en.wikipedia.org/wiki/PCI-X

[22] Wikipedia 2009. PCI express
http://en.wikipedia.org/wiki/PCI-Express
[23] Wikipedia 2009. Message Signaled Interrupts
http://en.wikipedia.org/wiki/Message_Signaled_Interrupts
[24] Sun Microsystems 2009. MSI-X Interrupts
http://docs.sun.com/app/docs/doc/819-3196/6n5ed4gv6?a=view
[25] Creating Words Writing Studio 2008. How does a network card work
http://www.creatingwords.com/samples/ghostwriting/computers/HowDoesaNetworkInterfaceCardWork.pdf
[26] Kioskea 2009. Network cards
http://en.kioskea.net/contents/pc/carte-reseau.php3
[27] Wikipedia 2009. TCP Offload Engine
http://en.wikipedia.org/wiki/TCP_Offload_Engine
[28] Wireshark 2009. TCP Checksum offload
http://wiki.wireshark.org/TCP_checksum_offload
[29] IBM 2009. TCP Checksum offload
http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/tcp_checksum_offload.htm
[30] Fefe: Choosing an Ethernet Nic for Linux 2.4
http://www.fefe.de/linuxeth/
[31] Wikipedia 2007. Kernel Tap, Linux: TCP Segmentation Offload (TSO)
http://kerneltrap.org/node/397
[32] Wikipedia 2007. Large segment offload (LSO)
http://en.wikipedia.org/wiki/Large_segment_offload
[33] The Linux foundation: Net: GSO
http://www.linuxfoundation.org/en/Net:GSO
[34] Marko Zec, Miljenko Mikuc, Mario Žagar 2002: Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput
http://www.tel.fer.hr/zec/papers/zec-mikuc-zagar-02.pdf
[35] Wikipedia 2009. OSI model
http://en.wikipedia.org/wiki/OSI_model
[36] Wikipedia 2009. Internet protocol suite
http://en.wikipedia.org/wiki/Internet_protocol_suite
[37] Wikipedia 2009. iSCSI
http://en.wikipedia.org/wiki/ISCSI
[38] Microsoft 2001. Background Windows Network Task Offload
http://www.microsoft.com/whdc/device/network/taskoffload.mspx
[39] Wikipedia 2009. X86
http://en.wikipedia.org/wiki/X86
[40] Steven Pope, David Riddoch 2007: Use hardware-based acceleration wisely
http://www.eetasia.com/ART_8800472106_590626_NT_c8e37e93.HTM
[41] Wikipedia 2009. Comparison of latency and throughput
http://en.wikipedia.org/wiki/Comparison_of_latency_and_throughput
[42] Marko Zec, Miljenko Mikuc, Mario Žagar 2002.: Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput
http://www.tel.fer.hr/zec/papers/zec-mikuc-zagar-02.pdf