Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Tomi Salminen

Flexible and transparent buffering of radio astronomy measurements:

VLBI-streamer and Flexbuff

Master's Thesis
Espoo, April 27, 2015

Supervisor: Professor Heikki Saikkonen
Advisors: Ari Mujunen D.Sc.(Tech.)
Vesa Hirvisalo D.Sc.(Eng.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Tomi Salminen

Title:
Flexible and transparent buffering of radio astronomy measurements: VLBI-streamer and Flexbuff

Date:	April 27, 2015	**Pages:**	58
Major:	Embedded Systems	**Code:**	T-106
Supervisor:	Professor Heikki Saikkonen		
Advisors:	Ari Mujunen D.Sc.(Tech.) Vesa Hirvisalo D.Sc.(Eng.)		

Radio astronomical data recording poses three challenges for its users: High volume, high bandwidth and large geographical distances. The solutions have been custom hardware and physical shipping of data sets. Performance of commercial off-the-shelf hardware for streaming data recording has been steadily improving and is starting to surpass the challenges. If the use of custom hardware can be dropped, the solution can benefit from successive hardware generations without added development efforts.

This thesis will present *VLBI-streamer* software for simultaneous read and write data streaming and *Flexbuff* as the hardware housing this software. Volatile and non-volatile memory are used as pools of resources in a user space confined best-effort relaxed architecture. Since hardware will keep changing and it takes time, developers and money to port software to different hardware, VLBI-streamer was not bound to any specific hardware. VLBI-streamer will run on virtually any Linux-system with a few package requirements and was made modular to support different stream types (UDP/TCP/RDMA) and disk I/O backends.

In this thesis Flexbuff combined with the software VLBI-streamer developed by the author is shown to be capable of 10Gb/s and beyond recording and streaming, meeting current radio astronomical data recording needs.

Keywords: network, data-acquisition, TCP, UDP, radio astronomy

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMA

Tekijä: Tomi Salminen

Työn nimi:
Joustavaa ja läpinäkyvää puskurointia radiotähtitieteen mittauksissa: VLBI-streamer ja Flexbuff

Päiväys:	27. huhtikuuta, 2015	**Sivumäärä:**	58
Pääaine:	Sulautetut Järjestelmät	**Koodi:**	T-106
Valvoja:	Professori Heikki Saikkonen		
Ohjaajat:	Diplomi-insinööri Ari Mujunen Tekniikan tohtori Vesa Hirvisalo		

Radiotähtitieteellisten kokeiden nauhoittamisessa on kolme haastetta: Suuri datamäärä, korkea kaistanleveys ja suuret maantieteelliset etäisyydet. Ratkaisuna ovat ennen olleet räätälöidyt komponentit ja tallenteiden fyysinen lähetys. Helposti saatavilla olevien tietotekniikan yleiskomponenttien suorituskyky on noussut tasaisesti ja tallennuskyky alkaa lähestyä asetettuja haasteita. Jos räätälöidyistä komponenteista voidaan luopua, yleiskomponenteille tehdyt ohjelmistoratkaisut voisivat hyötyä jatkuvasti kehittyvistä komponenttisukupolvista ilman ylimääräistä ohjelmistokehityspanostusta.

Tämä diplomityö esittelee *VLBI-streamer* ohjelman yhtaikaiseen datan tallennukseen ja lähetykseen, sekä *Flexbuff* komponenttimäärittelyn, jolla ohjelma ajetaan. Lyhyt- ja pitkäkestoisia muisteja käytetään varattavissa olevina resursseina käyttäjätilaan rajatussa, parhaan yrityksen arkkitehtuurissa. Koska fyysiset komponentit tulevat aina vaihtumaan, VLBI-streamer toteutettiin ilman siteitä mihinkään tiettyyn rautakomponenttiin, jolloin säästytään ylimääräiseltä ohjelmistokehitykseltä komponenttien vaihtuessa. VLBI-streamer on ajettavissa melkein millä tahansa Linux-järjestelmällä ja tarvitsee vain muutaman ulkoisen vapaasti saatavilla olevan ohjelmistokomponentin. VLBI-streamer on myös modulaarinen ja tukee erilaisia verkkoprotokollia (TCP/UDP/RDMA) ja kovalevytallennustekniikoita.

Tässä diplomityössä Flexbuff ja kehittämäni VLBI-streamer osoitetaan pystyvän nauhoittamaan ja lähettämään yli 10Gb/s tietovirtoja, täten täyttäen nykyisten radiotähtitieteellisten kokeiden tarpeet.

Asiasanat:	verkko, datatallenus, TCP, UDP, radiotähtitiede
Kieli:	Englanti

Acknowledgements

I'd like to thank Metsähovi Research observatory and its crew for the great opportunity to develop and grow as a software developer with this project. I'm especially grateful for the relaxed leading style of Ari Mujunen, where I had freedom to explore different options for a best solution. I had a lot of rough edges as a developer, but in an encouraging environment I think I turned out alright.

Espoo, April 27, 2015, Tomi Salminen

Abbreviations and Acronyms

10GE	10 Gigabit Ethernet
AHCI	Advanced Host Controller Interface
AIO	Asynchronous I/O
API	Application Programming Interface
ATX	Advanced Technology eXtended
BOD	Bandwidth On Demand
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
DBBC	Digital Base Band Converter
DMA	Direct Memory Access
EVN	European VLBI Network
FPGA	Field-Programmable Gate Array
FS	File System
FUSE	File system in Userspace
Gb	Gigabit
GB	Gigabyte
GNU	GNU's Not Unix
HPC	High Performance Computing
IT	Information Technology
JIVE	Joint Institute for VLBI in Europe
LFN	Long Fat Network
NAPI	New API
NCQ	Native Command Queuing
NFS	Network File System
NIC	Network Interface Card
OS	Operating System
PCI	Peripheral Component Interconnect
PPS	Pulse Per Second
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory

RDMA	Remote Direct Memory Access
RTT	Round Trip Time
SSD	Solid State Drive
SATA	Serial Advanced Technology Attachment
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VFS	Virtual File System
VLBI	Very-long-baseline interferometry
VSI	VLBI Standard Interface

Contents

Chapter 1

Introduction

1.1 Data recording in radio astronomy

Continued advances in digital signal processing enable radio astronomy instruments to generate increasingly higher resolution measurements. Increasing the sample rate allows the receiving of a larger bandwidth. These advances inevitably also increase the data rate at which the data is generated. This poses challenges on handling the bandwidth and volume of the data. Data sources generate continuous multi-gigabit streams like the ones described in 3.5.2. Any recording system receiving these streams must be able to continuously sustain the sent data rate or suffer packet loss, which results in data loss and possibly a botched observation.

Multiple stations can observe the same target at the same time. When this data is combined or correlated as it is called, the results emulate a dish the length of the distance between the stations. These dishes with diameters up to a hundred meters can be individual dishes separated by whole continents or a collection of similar dishes within visual range of each other. The former scenario is the more relevant one for this thesis, as it adds the requirement of distributing the high data rate data sets of geographically separated stations.

Recording astronomical data was started with tape drives and hardware correlators but has moved to spinning hard disks combined with software solutions and will most likely someday move to solid state drives and fully to software correlators. Distributing the data has also moved from shipping physical disks to data streams over network connections. The physical connections between the devices has also changed from bulky custom cables to basic network connections, from which the VLBI community benefits as the network industry is being heavily developed.

If there are consistent requirements in the research sector, they are affordability and compatibility. Anything saved in facilitating the data can be invested in better instrumentation for generating the data itself. Also building a solution around a single data type would restrict the solutions lifespan to that particular type of data.

1.2 A commercial off-the-shelf solution

The requirements explained previously form the basis of this thesis. The work itself is divided into a hardware and a software solution. The former of which is a collection of hardware components recommended for housing an efficient data recording system, which we will call Flexbuff . The latter is the software running on the Flexbuff, which was named VLBI-streamer.VLBI-streamer is freely available open source software developed by the author and available at `https://code.google.com/p/vlbi-streamer/`

As the IT industry drives the development of Commercial off-the-shelf (COTS) hardware, it stands to reason to develop custom hardware solutions for components. A mold for a recording element was sketched and called Flexbuff [31]. Individual components are surely replaced by each successive generation, but the overall component types will probably remain similar. There will be slow non-volatile memory, faster volatile memory and some sort of interconnectivity between machines.

Assuming the evolution of underlying hardware is driving our instruments forward, software developed for a custom hardware platform would not survive competitive in a longer time period. This would require redevelopment of the software on every hardware generation. The software VLBI-streamer was developed to abstract away the hardware it is running on, but still work efficiently with the hardware and allow advanced features when available. As the development is done on a Linux system, there is a lot of benefit to be had from advances in kernel development. This also hints that development should be restricted to user space without creating custom kernel patches, which would again require redevelopment and maintaining or the software would be bound to a moribund kernel.

The challenge of large distances between observing stations is met by the increase in network connectivity globally. To make use of this connectivity, VLBI-streamer aims to use the common network protocols efficiently, while also allowing new solutions into its architecture.

The use of Flexbuff and VLBI-streamer is not restricted to radio astronomy. There are startups selling solutions for data acquisition, which have customers in for example the auto industry [37]. The main focus is captur-

ing sensor data at high rates. Within interferometry, the target of interest can also be for example sea level variations. [10]

The hardware side of Flexbuff was researched by Esa Turtiainen, Jouko Ritakari, Ari Mujunen and Minttu Uunila[31]. VLBI-streamer was designed and developed on top of this research by me and molded by almost daily dialogue with the authors of the original Flexbuff design.

1.3 Structure of this Thesis

Chapter 2 goes through the individual hardware components Flexbuff uses and considerations in using those hardware components. Chapter 3 lists the operating system software considerations and motivations. After the background chapters, chapter 4 describes the developed software and important design considerations that were found during development. Chapter 5 has experimental results showing VLBI-streamer working with network loads up to 20Gb/s on local receive and distributed performance between several stations. The remaining chapters 6 and 7 are where the results and future uses are examined.

Chapter 2

Hardware

This chapter describes each relevant hardware component for high speed networked data recording. What exactly is relevant for this particular project was studied earlier by Esa Turtiainen et al.[31]. During the spring of 2012 the components of Flexbuff were roughly:

- 10 Gigabit ethernet (10GE) network card.
- Large pool of hard drives (24+).
- Enough serial advanced technology attachment (SATA) controllers to control the hard drives.
- 12 or more gigabytes (GB) of modern high speed memory.
- Modern multi core central processing unit (CPU).
- Advanced technology eXtended (ATX) motherboard and rack mounted chassis to house it all.

It should be noted that all of these components will probably be replaced by more advanced components shortly after, probably sporting decade larger performance values. The term Flexbuff was coined to describe this type of hardware arrangement. Next are a few issues that were relevant during the spring of 2012 and which influenced VLBI-streamers design.

The coming topics will cover different techniques along the data sets path from the network to a persistent storage. These can be segmented by looking first at the network card receiving the data packets, then at the bus connecting the network card to memory, how the data is moved from memory to the buffer of a persistent storage device and finally the techniques in writing to a persistent storage.

2.1 Network interface cards

Flexbuffers are mostly equipped with 10GE network cards for data transfer and 1GE links for management. There are a plethora of features supported by the various 10GE network cards. As will be explained in 3.4 the first part of the transfer is connectionless UDP-packets so there will be less emphasis on TCP-specific techniques.

2.1.1 Interrupt mitigation

The default behaviour for packet receiving in an operating system (OS) is invoking of an interrupt routine to handle the packets transfer from the network card to the kernel memory space. Since interrupt routines are high priority and can cause trouble if they perform too long operations, these interrupt routines usually simply set flags for the default OS threads to handle the raw data transfers. With 10GE ethernet, the rate of interrupts can rise very high as the number of packets per seconds increases. With for example 2048 byte packets: $\frac{10Gb/s}{(2048*8)b} = 655360p/s$. So there would be 655360 interrupts per second. Since network card interrupts are usually bound to a single core of a multicore processor the core would most likely be congested in context switches, which could result in packet loss and poor efficiency. [15]

The problem is addressed with interrupt mitigation, which sets a time limit during which only one interrupt is allowed. The network card will buffer the packets, until the OS in the next interrupt handles a group of packets. This reduces the amount of context switches and testing showed this feature to limit the interrupt rate to consume about 10% of the cycles of a single core. The time limit should not be too long though, as it increases the latency and with high enough bit rate the network card's buffers can overflow. [15]

On the Kernel software side interrupt mitigation is handled by the new application programming interface (NAPI), which does not require any action from the user space developer's side. [19]

2.1.2 Infiniband

Infiniband is an interconnect used mostly for high performance computing (HPC) and data centers. Infiniband potentially could provide fast interconnects abstracted away from common network I/O concepts and troubles. The largest benefit would come from remote memory accesses described below.

In VLBI the distances between stations is large, which means relying on existing network connections, Bandwidth On Demand (BOD) links and

the public Internet, in where infiniband-connections have no quarter. It should be noted though that VLBI-streamers modular architecture allows the development of an infiniband module. [21]

2.1.3 Remote Direct Memory Access

Remote Direct Memory Access (RDMA) enables sending between process and memory of interconnected nodes without the continuous involvement of either sides CPUs. It is a sort of bypass without extra copying through kernel space. Infiniband sports RDMA, but some normal NICs have similar capabilities through a protocol called iWARP, which shows promise in bandwidth utilization. [23]

The RDMA draft is quite new though and has some security considerations with exposing a memory segment in public Internet. [24] Again it is not ruled out, as a module for it can be developed for VLBI-streamer.

2.1.4 NIC tuning

Network card improvements can be gained through adjusting several parameters:

- Increasing kernel receive buffer size.
- Interrupt mitigation and interrupt intervals.
- Adjusting backlog length for TCP-connections.
- Experiment and research a suitable congestion control protocol for TCP depending on the connection

For UDP connections most of the tests have the kernel receive buffer set to 16MB.

2.2 VLBI standard interface

VLBI standard interface (VSI) described in [36], is a custom parallel interconnect for VLBI data. In addition to data pins it sports clock signals that enable connected devices to share a pulse per second (PPS) between them for synchronization. Flexbuff comes into play after the data has already been structured into packets, but a module for a peripheral component interconnect (PCI) lane connected VSI-card is a possibility.

2.3 SATA controllers

2.3.1 Advanced Host Controller Interface

Advanced Host Controller Interface (AHCI) is a data movement engine which abstracts the SATA 2 control away from the host machine and implements a standard interface. I/O requests are scheduled by signaling with appropriate AHCI ports. Completed requests are signaled by aggregated (mitigated) interrupts. [39]

2.3.2 Native Command Queuing

Native Command Queuing (NCQ) is a feature implemented in SATA 2 that runs in the disk firmware rearranging access requests to optimize throughput and reduce head seeks. [39]

NCQ for VLBI-streamer means that small request are aggregated to larger ones when they are targeted to a sequential file strip. Most of modern hard drives have implemented NCQ through AHCI and using it does not require any extra tuning.

2.4 Disk Drives

VLBI data recording has evolved from tuned tape drives, video cassette recorders to hard drives. The emergence of solid state drives might eventually replace spinning disk drives, but currently the capacity and price of spinning disk drives makes the older technology more usable. Spinning disk drives can also be used as efficient recording media if their characteristics are taken into account.

Everything described in this section can be found more thoroughly written in [6].

2.4.1 Spinning disk drives

Traditional hard drives have spinning platters, where data is preserved magnetically. Most of the performance costs occur when seeking for data from another location on the platter. The cost of physically moving the read head is very large when comparing it with CPU cycles. Also the seek head must seek to the correct location on the track.

In data access patterns this means that random accesses are very costly. Again in data recording, this encourages to use sequential writes and reads

to get the largest bandwidth. When reading or writing sequential data the data rate depends if the data is on the outer or inner tracks. The difference is due to inner tracks requiring more frequent seeks, since they contain less data per track.

There is a possibility that all disks of a system would start to converge on the inner tracks at the same time as all the disks are the same size and model with uniform performance characteristics. This would cause the total system throughput to drop dramatically. If this is seen as a real threat scenario, the suggested fix is to repartition the disks with non-uniform block division and mounting them randomly as write points. This way different tracks fill up first and the performance will stay randomly uniform during operation.

Modern disk drives also have large caches to mitigate random data access costs. This means the data might not be yet written to the hard drive, but is sitting in the cache. For our purposes, these caches have no function, since they are minuscule when compared to the data volume recorded in regular VLBI sessions.

Most spinning disk drives have also an internal request queue. Since the mechanical delay of seeking is quite large, all incoming requests for data are set in a queue, which the disk drive can arbitrarily rearrange to minimize mechanical delay. As will be explained in 3.3.1 the OS also does this rearranging and combining.

2.4.2 Solid state drives

Solid state drives (SSD) are NAND-flash based storage units. In addition to more read or write speeds, a bit advantage of SSDs is their ability to perform random access data operations almost as fast sequential ones, due to the absence of moving parts. During the spring of 2012 the capacity and price per gigabyte on SSD drives is still way behind traditional hard drives, but SSD technology might someday replace traditional hard drives.

Chapter 3

Software concepts

The path of data from the wire to a persistent storage is laden with different software boundaries and concepts. These can be roughly divided to:

1. Network protocols transferring the data.
2. Sockets as gateways between the network and volatile memory.
3. Transferring data between volatile and non-volatile memory.
4. File systems as non-volatile memory.

In this chapter we go examine alternatives and design considerations for each step. In addition to performance, the qualifying factor for an alternative was also its probable life span, which means how probable it is that the same alternative can be usable in the future without added development effort.

3.1 Network scenario and correlation

The advantage of VLBI is that a dish is emulated that is of the size of the baseline between the antennas. The dishes can be separated by thousand of kilometers. This inherently sets the network scenario to include large distances and so also increases the round trip time (RTT) of data transfers. Resolution is also affected by the measurement bandwidth, so more data bandwidth results in higher resolution data. This forces our network scenario to be a long fat network (LFN).

In a correlation, each stations data needs to be available at a single location, where data from same time spans are compared to correlate a result through processes not described in thesis. The relevant part is that data needs to be made available at a high bandwidth with matching time spans.

There are numerous on the network protocols described here and for more in-depth information I recommend [27], in which are described all the protocols in this section.

3.1.1 Transmission Control Protocol

Transmission Control Protocol (TCP) is the default transport layer protocol between nodes on the Internet. It hides packet loss and reordering of packets from applications and avoids congestion by keeping track of acknowledgements. TCP also has states as it is connection oriented.

A simple description of TCP is where the sending side has a window of packets that are en route to the destination. The size of the window determines how many packets can be en route at a time. Packets are cleared from the window when an acknowledgement is received for each packet. The classic algorithm for TCP is Reno, which has a small initial transfer rate that climbs up to match the bandwidth to the destination. If a packet is lost, the rate is dropped dramatically and the slow start is started again. [22]

Although a very well functioning protocol for open Internet usage, the problem in our domain with TCP is that it has a history of not performing optimally with LFNs. [13].

With a large RTT, achieving the networks full capability takes longer, since the sending side has longer iteration times when it tries to increase its window size. Combine this with the possibility of a few packets dropping will result in a zig-zag shape demonstrated later in 5.3.3

Many of the older problems were addressed in [32] from which spawned new TCP-algorithms to replace the Reno-algorithm. The default on Linux systems currently is Cubic described in [11]. Tests like [29] with modern TCP algorithms show promise in total bandwidth even with 40GE network, though the study had relatively fast path of 49ms. Similar tests with a very long baseline of 32372km show that TCP will start using the bandwidth optimally, but only after a fairly long ramp up time. The problem with this study is the use of Linux kernel 2.6.12, which is fairly old and most likely still using the old Reno-algorithm. [38]

An important advantage with using TCP in a correlation process is its inherent and automatic buffering and rate control. The protocol stops sending when the readers buffer is full and vice versa. In the correlation phase the correlator needs each stations data at the same rate and buffering this data would force buffering mechanisms to the receiving side, or conversely rate throttling to the sending side. For a scenario where multiple stations are sending to a correlator with different network paths and capabilities, the TCP streaming rate from each station would converge to a shared minimum

without added software complexity.

3.1.1.1 TCP with multiple streams

In addition to the already listed problems and improvements for TCP, there is also the possibility of using multiple TCP streams instead of one. This mitigates the problems of long fat networks explained in 3.1.1 and in essence divides the LFN into multiple virtual thinner networks. Also the losses in speed when cutting the individual streams speed by half is mitigated considerably.

The problem then becomes that of how to speed up individual transfers with this method. Since mostly dealing with data that is divided into packets, the data stream can be divided by the packets and then distributed into an arbitrary amount of TCP pipes. The only requirement is that both ends know how many TCP connections there are, so they know which packet belongs to which spot. This method could be named single data, multiple streams. The study [14] referred to it as cascaded TCP.

3.1.2 User Datagram Protocol

User datagram protocol (UDP) is a transport layer protocol that has only header fields for source port, destination port, length and a checksum. This means it does not resequence the packets or resend lost packets. Another way of putting it: It lets a higher level application take care of data loss and reordering. UDP also does not take congestion into account and so can cause congestion collapses on nodes with heavy load. [4]

UDP is the main workhorse in observations and important in the VLBI-streamer software itself. This itself poses a challenge with rate limiting. As there is no inherent rate limiting in the UDP-protocol, the sending side has to take the network capacity of the full path into consideration. Since recording machines themselves are locally connected to high speed network switches, having them take only the first network hop into consideration would most likely drop most of the traffic as packet loss in some hop in the network connection. So anyone sending UDP-traffic has to take the paths weakest bandwidth into consideration.

Sending UDP packets at a constant rate is not a trivial task. It was quickly noticed in the developing of VLBI-streamer that simply sleeping until the next packet should be sent is not as straightforward as it seems as the default non-preemptive Linux scheduler only wakes threads during scheduling intervals [26]. These problems are solved in section 4.4.2.

3.1.3 Stream Control Transmission Protocol

Stream Control Transmission Protocol (SCTP) is a sort of a hybrid of TCP and UDP. Since TCP abstracts the sending of data to a point where the only consideration is the number of sent bytes, there are no clear packet boundaries in TCP. SCTP adds this packet boundary consciousness. Packets are also kept in order, which makes SCTP connection oriented. SCTP Is only introduced here, but not used in VLBI-streamer or in the observations themselves.

3.2 From the network to main memory

A simplification of a packet receive is as follows:

1. A packet is read from the wire into one of the network cards internal memory queues
2. The network card generates an interrupt to signal the operating system of a new packet
3. The interrupt handler eventually copies the packet to the correct kernel space socket buffer, which was reserved for the receiving program
4. The receiving program copies the packet from the kernel space buffer to its own user space buffer.

3.2.1 Sockets

Sockets are endpoints of an inter-process communication flow. There are three types of socket types, each of which was used during this thesis:

- Datagram sockets or connectionless sockets
- Stream sockets or connection oriented sockets
- Raw sockets

From a software developer perspective, sockets are reserved kernel memory spaces, to which the Operating System (OS) copies packets. The packets are bound to specific sockets according to their port numbers. The packets can be read to user space with system calls. If the buffer is full and more packets arrive, the extra packets are dropped and the kernel registers this as packet loss.

For more in-depth information on the operating systems handling of network related functionality, I recommend [26].

3.2.2 Packet Memory Map

The Linux kernel offers in addition to traditional sockets packet memory map sockets. These specify a ring-type memory area into which the kernel can directly write packets to. After initialization the interface operates in promiscuous mode, which means it will capture all packets arriving to the interface. In practice this requires the user space program to poll for events in the memory mapped socket, after which it processes the packets and marks them as free to be used for more packets arriving for the network.[20]

Also if the socket is created with AF_PACKET and PACKET_FANOUT, the packets will be spread evenly to threads which have registered to use the interface [1].

3.2.3 PF_RING

There is also a custom packet capture module and driver named PF_RING described in [8]. Since the module had not made its way into the Linux kernel mainstream, there was concern that it could die out and take any software built upon it with it. Also it might not offer drivers for a specific network card or might force the use of an older kernel. Again it is not ruled out as a module can be easily added for it in to VLBI-streamer.

3.2.4 Splicing

Linux user space can make use of splicing, which is data transfer between kernel memory pipes. A feature that caught my attention is giving splice-commands flags, which enable it to move data between locations without copying. This means that the packets could be copied from their kernel space socket buffer efficiently directly to disk. [18]

Splicing is currently only supported for TCP [25]. Also for our purposes, splicing from the socket would require that the write point would have to keep up with the receiving process. This in turn means that VLBI-streamer would be forced to a raid-solution with fast enough write speeds. Also in redundant array of independent disks (RAID) systems it would be more optimal to perform large writes, as they can be spread to more disks due to stripe size and subsequently utilize more of the disks bandwidth.

It should be noted though that memory mapped spaces can have a file descriptor associated to them. This means that packets could be spliced directly to memory. There is some evidence that direct splicing is superior to manual send or write-commands [29].

Splicing UDP-packets to the network is supported, but with it the packet size cannot be properly determined.

3.3 From main memory to non-volatile storage

As the data is in memory, it is organized in so called memory pages of power of two in size. Operating systems often have extensive caching systems for reducing hard drive transactions. Since VLBI-streamer handles its on caching, such caches will be avoided.

3.3.1 Virtual file system

The Linux virtual file system (VFS) provides a disk cache named the page cache for keeping pages in memory which are used regularly. These pages are only flushed on request or when the system runs out of usable memory. In addition all written pages are copied to the VFS for efficient combination of writes and sharing of pages between processes. In this application, this does not serve much purpose and also causes an extra memory copy. Since its desired to utilize the maximum disk write bandwidth from the start of a recording, keeping pages in memory delays the data write and can cause a jerk when memory goes low enough for the writes to begin: The memory is suddenly full of pages that need to be written to disk and processes are blocked from getting its share of memory until the pages are written.

The write to disk can forced though, but it still might not release the memory that was allocated for the cache and does not remove the extra memory copy. [3]

3.3.2 Direct I/O

The GNU/Linux user space also provides a DIRECT_IO flag, which can be specified when opening a file. This flag will skip all page caches and write the data directly to disk. There is a requirement though: All writes must be multiples of the block size, which is traditionally 512 bytes and 4096 bytes in newer mediums. [16]

Due to this requirement, many issues must be taken into consideration:

- It might not be possible to write an integer amount of packets, as the byte count might not be divisible by the block size.

- Dummy data might have to be written at the end of a file.
- When writing to the same file, only n-packets can be written, where $n \times$ packet size is divisible by the block size. This means extra data not divisible by block size cannot be written, unless partial packets are written.
- Packets sequentially in memory cannot be stripped of header data without an extra memory copy, since the packet size minus bytes stripped most likely will not align.

A motivational graph for Direct I/O is shown in figure 3.3.1. As the data is written to a raid, it benefits from using large writes that parallelize better to the disk drives as the write is split into stripes. Without direct I/O, the blocks are automatically grouped to larger writes as they are copied into the VFS page cache. The downside of the copying is the CPU overhead of copying the data. dd is a widely used Unix command line tool for copying data. During the writes the dd-process with direct I/O consumed only about 14% of the cycles of a single CPU core, when the non-directs consistently took 100% of the cycles of a core and so was probably bottlenecked to a single cores performance.

The first dd command without reverse order performed better with smaller writes, but this was most likely due to still having free memory at the start of the test run, which did not cause cycles to be used on flushing the memory. For confirmation, the test was restarted with a reverse order, so first the very large block sizes were tested.

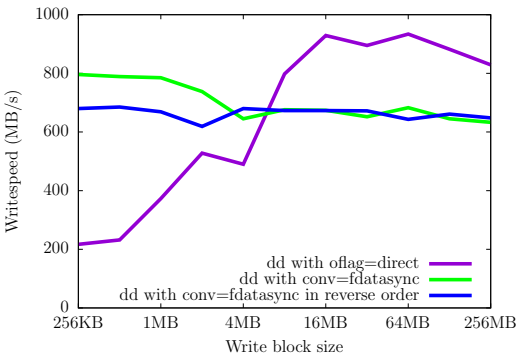


Figure 3.3.1: Write speed to 14 disk software raid 0 with 4096MB file from memory

3.4 VLBI backends

Since the project was aimed at recording astronomical data, some parts of the recording process should be explained. After the signal is focused to the receiving unit, it travels through a shielded cable to a digital back end. The digital back end does base band conversion and sampling of the data. The end results is an array of power values divided by band and time.

3.4.1 DBBC and FILA10G

The Digital Base Band Converter (DBBC) developed by Gino Tuccari et al. can be connected with VSI cables to a FILA10G boards which convert the antenna data into network packets. FILA10G uses stateless UDP traffic[30]. The conversion from a data stream to network packets is usually done by field programmable gate arrays (FPGA) that fit multiple data samples to a packet and send it to the network.

3.5 Data formats

3.5.1 Mark5b

The mark5b data format is specified by [33]. The relevant bits are:

- Frame number within second
- Julian day
- Second of day
- Fraction of second

When the receiving end gets a full seconds worth of data, the number of frames per second is known. After this the frame number and second constitute an index for the data by which the amount of missing data and the location of out-of-order packets can be determined. Each header combined with a payload constitute a 10016 byte frame.

3.5.2 FILA10G

For network transfer the rather large 10016 byte mark5b frames are split into two 5008 byte frames and the FILA10G adds a 32-bit filler and 32-bit counter in front of each frame. This means the network packets are 5016 bytes. During the spring of 2012 most correlator software could only process the mark5b frames without the net frame. For the receiver at the correlator this means it has to strip away the extra bytes from each packet.

The FILA10G had limited network functionality during the spring of 2012. It was tested by the author to generate a 4Gb/s UDP data stream correctly and its documentation showed at least the ability to configure 8Gb/s UDP streams.

Some information on FILA10G can be scraped from [12], but most parts were discovered through experimenting with the hardware itself.

3.5.3 VLBI data interchange format

VLBI data interchange format (VDIF) is a stream-based packetized data format, which is meant to standardize VLBI data storing. The relevant bits are the second from epoch and data frame number withing second, which can be used to resequence the data stream. [35]

3.6 Related work

VLBI data recording is not a new problem. Its history is mostly dominated by the Mark-series. There was also a new arrival in 2012 called Xcube.

3.6.1 Mark series

The mark series is a set of VLBI-data recording devices, which started with modified tape drives, moved to hard drives and is currently somewhat of a standard in the VLBI-community with the Mark5 disk recorders. The Mark5 series has banks of disk drives, which are controlled by StreamStor disk interface cards. The series from Mark5A to Mark5C supported input modules from VSI to Ethernet. Though sold as separate hardware, the machines are mostly COTS components.

3.6.2 Mark6

Mark6 is the newest in the VLBI recording series. The design has 4 bays for hard drives, which are each dedicated to its own NIC resulting in a total maximum recording rate of 16Gb/s. All the hardware is COTS, but with a different casing. The software is open source.[34]

Browsing the source code[9] in the autumn of 2011 showed that the software is using the PF_RING driver for packet capture. This could also be deduced from the 4Gb/s limit on individual network cards, as the interrupt rate spikes beyond that rate. The implementation uses only the PF_RING aware drivers, which might be to ensure the network card has its regular network capabilities still working.

3.6.3 Xcube

Xcube is a modular data recording and analysis platform, which is used in the automotive industry but is marketed also to the VLBI-community. The system is quite similar to Mark6. [28]

Additional features are Burst mode, which offers larger speed recording than is supported by the disk system for a limited amount of time. This is most likely simply using memory buffers that allows some flexibility in recording. The receive medium can be chosen freely, so the design is most likely modular. During a visit in the spring of 2012, a staff member informed that they were using the PF_RING aware driver.

Chapter 4

VLBI-streamer

4.1 Overview

VLBI-streamer is a software designed by the author in collaboration with the designers of Flexbuff. Development was done mostly between 2011 and 2012 by the author. VLBI-streamer can record and send high speed network packet streams. The main focus of development was the recording and storing of radio astronomy sessions and the subsequent sharing of those sessions.

4.2 Design principles

4.2.1 The Linux Kernel and hardware

Linux is a powerful operating system kernel which is widely developed. In addition to all the benefits it facilitates it also restricts certain data paths. The need for high speed transfer of data from network to disk calls for the minimization of memory copies en route to disk, but the kernel forces kernel/user space separation and so causes extra memory copies.

4.2.1.1 Packet receiving

As described in Chapter 3 the kernel copies the packet from the network cards memory to a socket buffer which is then copied with the for example the recv-command to a user space buffer. The corner cut version would be the copying of packets straight from the network card memory to disk with a direct memory access (DMA) operation. This could be done by writing or augmenting a network card driver. The openness of the Linux kernel permits this sort of augmentation to the drivers and the kernel.

With limited development time, VLBI-streamer would be restricted to a single or few network cards, which would cause the software to only run on those few cards. This would also mean more software development costs when developing the software to run on the next generation hardware. Also all ongoing development with the existing drivers and receiving would be either lost or would need manual integration with our code base. There would most likely be issues with using the card for normal network operations with the custom driver, not to mention the security risks.

Weighing all these toils together, the few extra memory copies start to look more appealing. Especially if its desired that the software lasts over hardware generations.

4.2.1.2 Hard drives

VLBI-streamer is not tied to any specific hard drive or other non-volatile media. The current implementation is geared toward utilizing large sequential writes and avoiding simultaneous operations to single targets. The non-volatile media targets are set as reservable resources, which are used by the buffer elements to read or write their data on. They can also be used as queues, for example when a certain file is required by a buffer entity, it will queue itself to the drive and be woken up when the resource is free to use again.

4.3 Architecture

4.3.1 Active file index

As the software might be receiving a stream that it wants to send simultaneously for correlation, a central data structure and access interface for file meta data is required. The active file index handles loading and saving of file meta data thread safely. This way active recordings can be mirrored to multiple sites and the transmission medium changed. This way a UDP stream from a FiLA10G can be received, while simultaneously sending it to a remote location with TCP-packets.

4.4 Modularity

Structuring software as modular can give much better results in software quality [5]. There are also many characteristics in this project that suggest a modular approach. As explained in 3.1.1 TCP does not utilize a networks

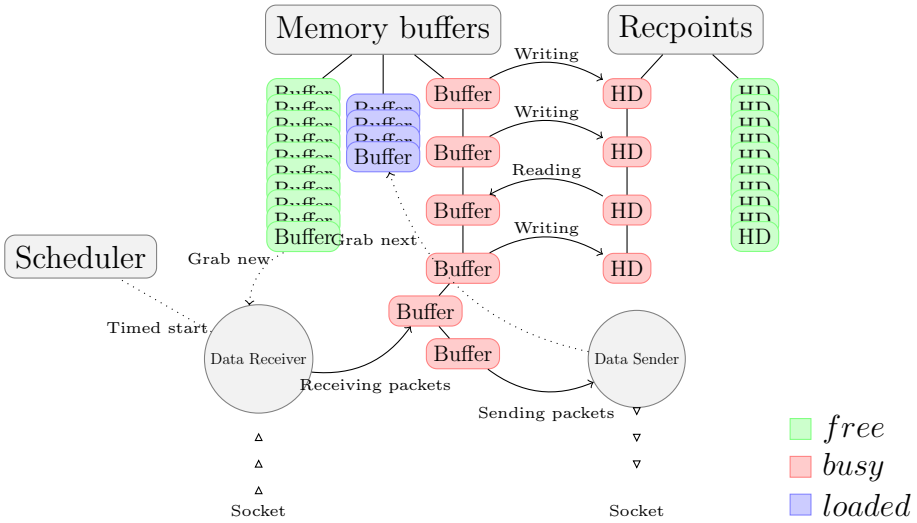


Figure 4.3.1: VLBI-streamer architecture. The scheduler starts receivers and senders. Receivers reserve free buffers from the queue as buffers filled by the receiver write themselves to available hard drivers. Senders order a number of buffers to fill themselves from disk and send them sequentially to the network.

bandwidth optimally in some scenarios. This is why the software sports a modular send and receive side supporting UDP, TCP and multi streamed TCP described in section 3.1.1.1. The transfer between volatile and non-volatile memory is also modular with different capabilities described in section 4.5.

4.4.1 Back ends for network

The network sides access to the memory buffers consists of asking for buffers and their memory space and then releasing so their threads can write themselves to disk. Other than that, the module for networking can operate quite freely. Using the modules is split to three phases: Initialize, run and close.

As existing packet packers like FiLA10G use stateless UDP packets for data transfer, the first back end developed for receiving sending was a UDP packet receiver.

4.4.2 UDP packet receiver

The UDP packet receiver creates a simple SOCK_DGRAM socket and mostly only tunes its size in the kernel memory space to the maximum size allowed. The operational parameters are a port and time spent recording or a target for mirroring, which means re-sending the packet to network immediately to a third target. Although a very simple solution in subsequent tests it showed to perform very well and did not strain a multi core even with close to $10Gb/s$ line rate packet receiving.

The challenging part was developing sending side that could regulate the speed at which it was sending. We cannot send UDP packets in a busy loop, since they cause congestion at the intermediate network nodes, which with great certainty will not have the same speed network connectivity due to the large distances between stations. If for example the network card connected to the sending machine can output 10Gb/s, but an intermediate link to the correlator has only 2Gb/s capacity, one fifth of the data will be dropped at the choke point. There must be a wait time between individual packet sends on the application level.

The wait time can be implemented with a busy loop or a sleep call. In a non pre-emptive kernel with normal priority threads, the minimum amount slept was tested to be the schedulers rate. This translated into about 50ms minimum sleep, which was too low for sensible transfer rates on a 10GE network. For example with 8888 byte packets: Rate $= \frac{\text{Size of packet in bits}}{\text{Time between packets}} = \frac{8888*8}{5\times10^{-5}}\, b/s \approx 1356\, Mb/s$

Due to the minimum sleep time, the implementation has an optional busy loop waiter for systems without a pre-emptive kernel. This waiter has dire performance consequences on systems with more sending processes, as each sending process requires a core in busy loop. This means severe scalability problems with multiple sending threads.

With a proper pre-emptive kernel, low latency scheduler timer and proper real time priority the sleeping timer works as intended and provides an optimal resource rate limiter for VLBI-streamer.

4.4.3 RX-ring

As described in 3.2.2 there is a ready load balancing packet capturer capable of writing into mapped memory areas. At first this sounded like the perfect solution for this project, not counting the forced capturing of all packets. This was implemented and tested, but it showed very high interrupt rate usage which resulted in packet loss on larger that $\approx 5\, Gb/s$ data rates.

The module source code is still in VLBI-streamer source code, but is

discontinued from development and will most likely not work. There is a chance that this interface might prove useful for some scenario or when it is developed to also use interrupt mitigation techniques.

4.4.4 TCP packet receiver

As described in 3.1.1 TCP would have many advantages for the network transfer towards correlation after the data stream has been recorded from a back end.

Since all connections from observatories to the correlation sites are unique, its not guaranteed that the data rates are uniform between the stations. The correlation itself requires the same time slices from each station and a delayed send from one stations would inadvertently delay the whole correlation and require the other streams to be buffered while the delayed stream tries to catch up. If a TCP-connection was used instead of UDP-packets, the streams would be automatically slowed through the TCP-stack to the lowest rate sender by blocking the sending thread.

There is no data loss or packet resequencing. Also in the event of a network connection failure, the sending sides would be automatically notified of this, disconnected and would stop their transfers. If there was a sudden unexpected extra traffic in switches en route, TCP could automatically react to it without extra coding efforts. The rates would be slowed and all other connections would slow down also adjust but not fail altogether.

4.5 Back ends for writing

The API for writing modules is quite simple. Upon acquiring a write element as a resource, the write end will open a new file to which write the data or open an existing file from which to read data from. The buffer will call the back ends write or read function to transfer the data.

Each of these writers have different characteristics. For example as shown in 3.3.2 a writer using DIRECT_IO requires considerably less CPU cycles, while being restricted to the underlying block size writes. While the use of less CPU cycles enables the recording of a larger bandwidth data stream, the restriction can for example prohibit the manipulation of the received data headers to make it compatible for a specific correlator as explained in 4.6.2.

4.5.1 Default writer

The default writer uses the default read() and write() system calls with the DIRECT_IO flag.

4.5.2 Asynchronous I/O writer

The asynchronous I/O writer (AIO) uses a Linux native libaio-library to query all the writes and poll for completion. The motivation is the ability to query large writes, which can fill the target drives caches efficiently for data flows.

4.5.3 Splice writer

The splice writer tries to benefit from moving data between file descriptors without needing to copy data between kernel address space and user address space. During the winter of 2012 splice had a more limited support for connectionless UDP sockets, which downplayed its importance for VLBI-streamer. The splice writer in VLBI-streamer simply splices the file descriptor associated with the memory mapped buffer into the persistent storage file descriptor and hints the file system to write the memory area to disk.

Although splicing could have been used to transfer bytes between a network socket and a persistent storage file descriptor, this would have tightly coupled the disk write end to the network receive end without the ability to buffer the data. This could have caused packet loss if the physical hard drive stalled, although the kernels virtual file system explained in section 3.3.1 could have probably handled the jolts.

4.5.4 Writev writer

The writev writer uses system calls to structure a write according to an array of iovec structures. Iovec-structures allow for gathering output writes, where a large batch of smaller memory areas are written in a single call. This is useful when stripping the headers from the start of packets is required. It is problematic though as the page cache is used, extra memory copies are made and the writes will surely not be the size of pages, which might cause data alignment overhead. [17]

4.6 Packet manipulation

4.6.1 Packet resequencing

UDP streams are connectionless, lossful and do not guarantee that the data arrives in the sent order. Therefore a mechanism had to be implemented into VLBI-streamer to take care of resequencing packets. Also its important to fill in missing packets with dummy data, as the packets from multiple stations have to aligned between themselves so the correlation does not have to realign them. Looking into the metadata is also useful for specifying a start time for data recording from the data stream.

The resequencing algorithm copies the packet from the socket to its presumed slot without looking at the metadata. If the packet is an earlier packet, it is copied to its correct spot with. If a previously used buffer is missing packets it is left to dangle on the receive end so the missing packets can be written to it. If the packet arrived before it should have, the index is moved to that packets position. This way keeping count is not required on which packets were received and which not, since already received packets ahead of our index will not be overwritten. Previous packets might arrive twice, but it is not an issue.

Filling with the non-aligned packet with dummy data or setting it as non-valid is done in the memory buffers, away from the critical receive process.

4.6.2 Header stripping

As explained in 3.5.2 VLBI-streamer has to occasionally manipulate the data. Since writes with DIRECT_IO require page size aligned data, this mode is only supported on writev.

4.7 Daemon mode

The starting of a process that requires most of the system memory can be time consuming. Also if multiple of such processes are desired to run on a single system, they should be capable of sharing their resources, since they cannot both have most of the system memory in their use. This is why VLBI-streamer was converted by the author to run as a daemon process sharing its memory buffers and disks with multiple receives and sends.

4.7.1 Scheduling

Instead of simply timing shell commands to start transfers, a scheduling system was created as the base for spawning other threads. A scheduling thread takes care of initializing all the resources and monitors a schedule file, from which it schedules the data transferring threads for receiving or sending.

4.7.2 Priority

As explained in section 1.2 we wanted to limit VLBI-streamer to run strictly in user space. It also required priority settings higher than allowed for normal user space processes. The natural way of combining these was as a regular OS service, which is started at boot time. After the software starts, it sets its priority high enough and drops its privileges.

4.8 FUSE

Filesystem in userspace (FUSE) enables mounting file systems in userspace. This relaxes the requirements for developing a custom file system and has proliferated many novel file systems, such as sshfs for mounting a file system over ssh.[7]

With this project FUSE was used to give the correlator single continuous files from the split files it had received. This was developed, but with limited development time proved a bit unstable. FUSE could have also been used to change the data format on the fly without modifying the original data. For example Mark5B network headers could have been changed to the regular non-network headers. Also this could have been interfaces with VLBI-streamer to utilized shared memory and accelerate the read process itself without the need for a raid.

4.8.1 Read acceleration

The correlating programs have a slow start, which is caused by pre-checks that try to determine the common starting point and clock skew between the recordings. If the correlator were given a number of files instead of one, it would have to do the pre-checks on each file separately and the correlation run would be a lot longer. Although VLBI-streamer supports writing to a single file, there is still reason to organize the correlation read through VLBI-streamer. If the correlator would start by reading small blocks on disks which are being written to, disk seeks would increase and total throughput would dramatically drop.

By using shared memory and communicating through local domain sockets, VLBI-streamer can serve the needs of a reading process by accelerating and policing the read process. Since the correlator will sequentially read the whole recording, it makes sense to treat it similarly to a send process. Even most of the modules can be re-used for this purpose. Currently the VLBI-streamer FUSE file system does its own reading through the mountpoints.

4.8.2 Data manipulation

Since there are pre-definable functions between the reading process and the VLBI-streamer buffers, the data read can be manipulated to different formats. Take the header stripping process described in 3.5.2. The header stripping can be done and is already implemented on the FUSE level with each recording being stripped of the headers without taking a performance toll on the receive process.

When developing VLBI-streamer it soon became apparent that using DIRECT_IO and byte stripping would not work well together. DIRECT_IO requires block size writes while byte stripping breaks up the data writes from memory to disk into packet size minus the length of bytes stripped segments. There would be several ways to go around this problem:

- Copy the data from the socket first to a temporary buffer, from which transfer only the requested amount to the actual buffer to be written do disk.
- Copy the packet segments to another memory buffer which will be written to disk without the overheads
- Receive each strippable part to a fake buffer and the data to the real buffer.

During development, no correct way was decided upon and so no implementation for DIRECT_IO writers with byte stripping was developed.

Chapter 5

Experiments

In this chapter we look at how VLBI-streamer handles different network loads in different network scenarios. The local experiments involve two machines connected either directly or through a switch to each other. Machines used in local tests were Ara and Watt in Metsähovi Radio Observatory, with either optical or CX4 10GE interconnects. These tests mostly simulate the recording of data sets from a digital backend at an observatory.

Remote tests have Flexbuff like machines on stations separated by relatively large geographical distances. These tests simulate the sending of data sets to a correlation facility after the experiment has been recorded. Also the simulatenous recording of ongoing experiments is tested.

The goal of these experiments is to test if VLBI-streamer can send and record any data rates available from the digital backends described in section 3.4. VLBI-streamers performance in these tests shows if its software architecture combined with Flexbuffs hardware housing can operate as a radio astronomical data buffer on stations. VLBI-streamers main function is local receiving of data, so the experiments are mostly focusing on local receive capabilities

The hardware of the test machines can be found in Appendix A. All tests show time as wall-clock time and speed as Mb/s. The measurements are started a few seconds before the start of the data transfer and continued long enough to check that performance is stable and would not regress over time.

5.1 Local receive with UDP

In a local receive scenario a VLBI back end is spewing data as UDP packets from a non-regular network device, that is restricted to generating and

sending the packets. In these experiments Flexbuff is used to record UDP packets in a local network with a packet generator machine and a receiver machine. Attributes of interest are the scalability according to memory and disk drives and maximum throughput values.

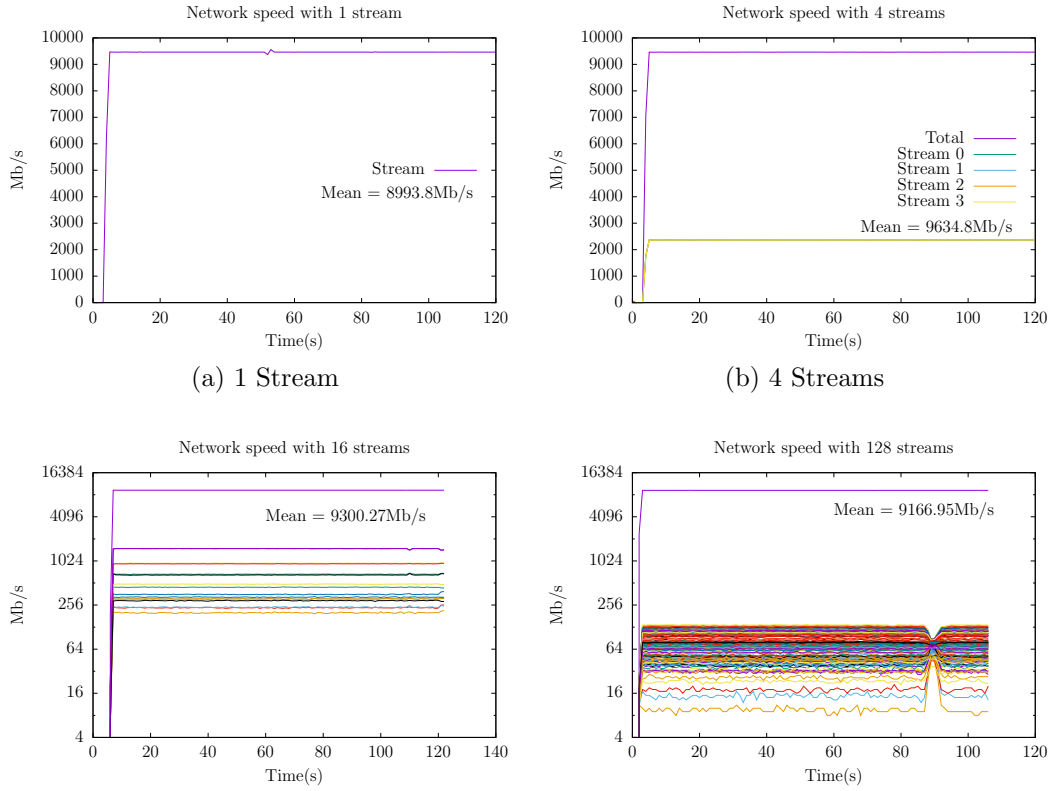
5.1.1 10GE local receive

Four different scenarios were tested for local receive: Data streaming of one, four, sixteen and 128 separate streams being received simultaneously. The results are shown in 5.1.1 with the total bandwidth in red and other lines as the individual streams. The data was sent in Metsähovi Radio Observatory between Ara and Watt through a direct 10GE CX4 connector. The program sending the data had separate threads for each stream and sent data in a busy loop. After the streams were increased from four to sixteen the sending program did not distribute the packets evenly anymore, but the total bandwidth remained the same and no packet loss was monitored on the receiving machines kernel level.

The receiving threads have a higher priority than normal user space processes. This is to guarantee a fast access to the kernel socket buffer to prevent it from overflowing. When a large amount of threads were run, as here with 128 threads, the system became slightly unresponsive but captured all packets without error. Setting the higher priority can be disabled, so when used with a large amount of threads, higher priority could be disabled to keep operability. The priority is not needed with large amounts of threads, since individual streams are relatively slow. Also since each stream continuously requires a memory buffer, the buffer size for 128 streams was dropped to 64MB. This increases the number of memory buffers, but it can also cause performance issues as each buffer has a thread running on it.

5.1.2 2x10GE local receive

The 10GE test showed no special bottlenecks, so more tests were done adding the Myricom fiber channel card and modifying the packet generator software to fanout its connections on a comma-separated list of targets. Preliminary tests showed that the increased priority in VLBI-streamer caused other services on the system to stall. This might be because the system root FS is mounted as a network file system (NFS), and might starve with large amount of high priority threads run on the system. Since the high priority threads are only needed for almost line rate single socket receives, the priority was dropped to normal.



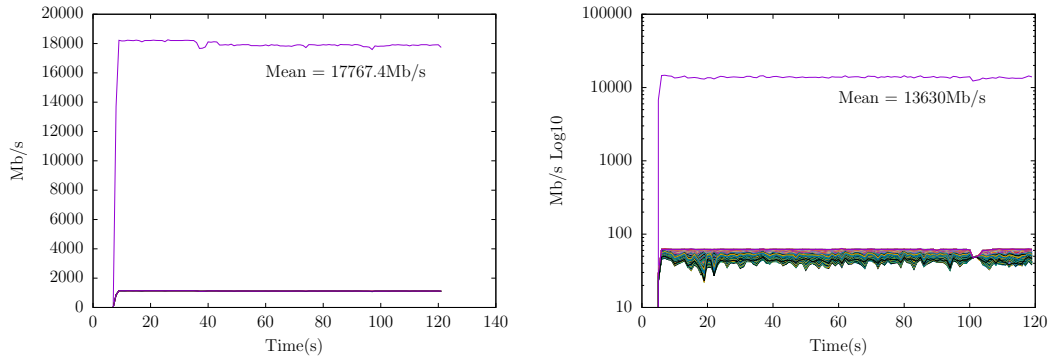
(a) 1 Stream

(b) 4 Streams

(c) 16 Streams

(d) 128 Streams

Figure 5.1.1: 10GE Local receive on Watt with dummy UDP-streams created from Ara. 64 buffers for used on all tests, except with 128 streams, where 256 smaller buffers were used.



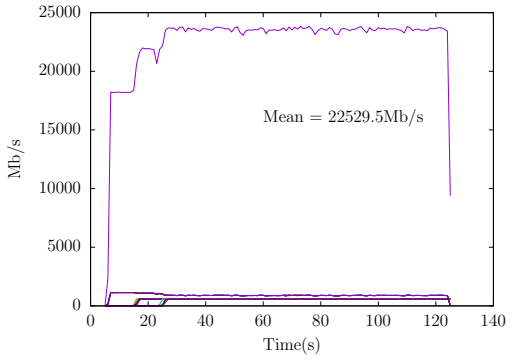
(a) 24 Streams 2 NIC (b) 256 Streams 2 NIC

Figure 5.1.2: 2x10GE Local receive. Receive on Watt with dummy UDP-streams created from Ara. The 256-stream test used 512 buffers and the 24-stream tests used the default 64-buffers.

Tests showed no packet loss on the receiving ends kernel. Results are shown in 5.1.2. With a lower number of threads the sockets started to have packet loss. Most likely acquiring a fresh buffer for writing took too long with the heavy load and the kernel socket buffers started to overflow forcing the kernel to drop packets.

5.2 Simultaneous receive and send

An important aspect of buffering astronomical data is the simultaneous sending of older or current recordings for correlation. This means Flexbuff must be able to send recordings it is currently receiving and must not compromise the receiving processes to packet loss. 5.2.1 Shows Watt receiving 16 streams and starting a send of another 16 streams through the 2 NICs. The receive process lowers from an average of 1136Mb/s per thread to 891 Mb/s. Since there was no packet loss registered on the kernel side, the receive process drop must occur on the pathways between the machines. Nevertheless the total bandwidth in and out averages at 24Gb/s, which can already start to bottleneck on the PCI express side, since the NICs are connected with PCI-E 2.0 at only 8x and have shown capping to below specified speeds in earlier tests.



(a) 16 Streams 2 NIC send and receive

Figure 5.2.1: 2x10GE Local receive and send in UDP. The red curve is the total send and receive bandwidth. Next curve is the receiving process and the lowest is the sending process. A total of 16 receiving processes and 16 sending processes.

5.3 TCP performance

On the TCP side the loss of packets is no more a concern. This allows for metrics on maximum system performance with dummy transfers. These metrics serve as a kind of hardware limit for our system, over which performance improvements from software cannot be expected. The used test scripts and programs for invoking these transfers are included in the VLBI-streamer repository. The data for total network transfers rate was logged with a bandwidth monitor named bwm_ng [2].

Before the TCP tests began, the connection between Italy and Metsähovi was tested to work at about 7.4Gb/s in UDP without packet loss. This works also as a reference value as a probable upper limit of TCP-transfers.

5.3.1 TCP reference values

It should be noted that the bandwidth monitor does not take into account the overhead of TCP-transfers and only measures the amount of bytes transferred between the nodes. Also this overhead cannot be easily factored out from the data, since variable size packets will cause a variable size overhead. One could simply measure the total amount of sent payload divided by time sent for transfer. This way each test stream would try to send its full data payload and then stop. Due to TCP-transfers being very opportunistic about their used bandwidth, the streams will exit at different times as others are faster and others slower. This will cause a test of N-streams to have variable results

as at the start there will truly be N-streams, but in the end N-1, N-2 and so on depending on how uneven the transfers are.

A small test program was developed to send the same data amount evenly over all of the streams. It was quickly noted though that this approach will not utilize the whole bandwidth efficiently as other streams have to stall while others are trying to catch up. This did not become a problem until 2 10GE NICs were used and performance started to lag behind.

Finally a small program named groupsend was modified to work in a threaded mode and renamed to groupsend_threaded. The program has a main loop which connects the TCP or UDP connections, starts the sending threads and monitors the threads for their amount of payload sent.

Since large buffers with TCP tend to cause a distortion in the values of sent data as the initially empty buffers are filled rapidly, a similar program working in reverse to receive data in a threaded mode was also developed by the author and named grouprecv. A thread is spawned for each stream, which empties the buffer in a busyloop. All test programs are included in VLBI-streamer sources.

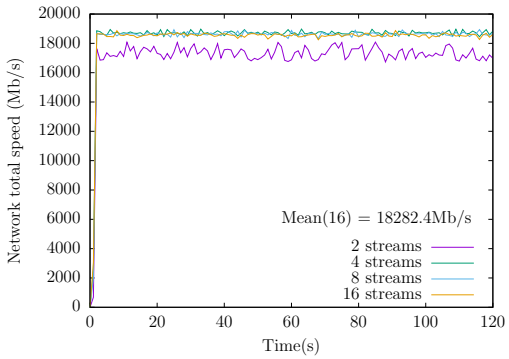
5.3.2 Local network TCP tests

In 5.3.1 the graphs 5.3.1a and 5.3.1b show the system performance on a variable number of TCP streams. This works as a good estimate for a roof value to which VLBI-streamer is tested against. 5.3.1a shows the receive speed of the data payload on two 10GE links in the local network. 5.3.1b shows how the system performs when simultaneously sending and receiving with TCP. This is a relevant test as the receive process clearly caps at a network limit, where as the input and output tests caps at some plethora of system restrictions, most likely capping of the PCI-E lanes.

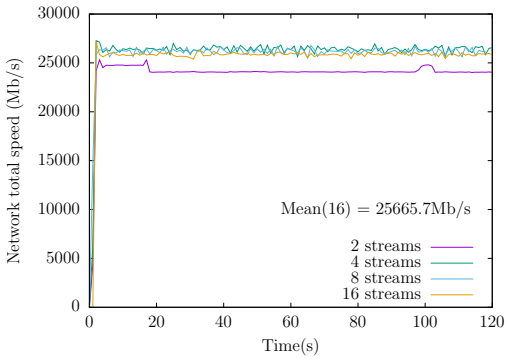
Figures 5.3.1c and 5.3.1d show VLBI-streamer working very closely and exceeding these reference limits. This means the reference tester programs were not fully stressing the system and could use some more development.

5.3.3 Long range TCP tests

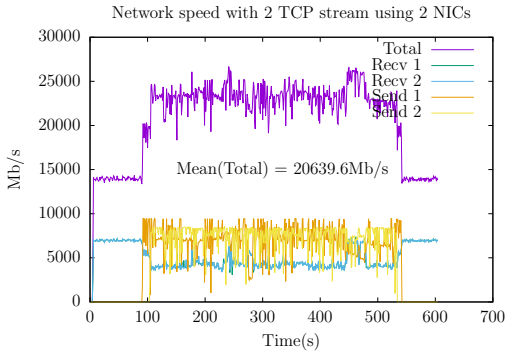
As speculated in 3.1.1.1 the dividing the TCP streams into substreams could give beneficial results on long fat pipes. Figure 5.3.2 shows just how this technique raised the mean transfer rates on a particular pipe. A single transfer stream here hits its congestion limit every 20 seconds shown by the sharp drop in transfer speed. As explained in section 3.1.1 this is the visible effect of a long ramp-up time which drops the mean transfer speed considerably. The ramp-up time increases as the probing of the TCP-path is affected by



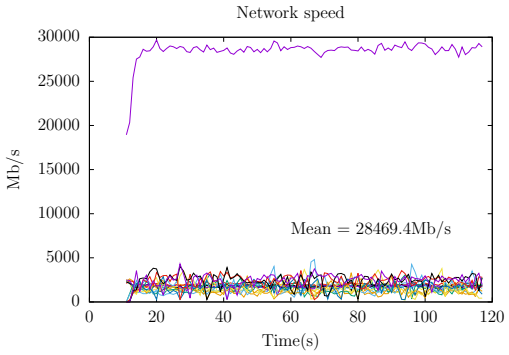
(a) 2x10GE NIC dummy receive in TCP



(b) 2x10GE NIC dummy sending and receiving with TCP

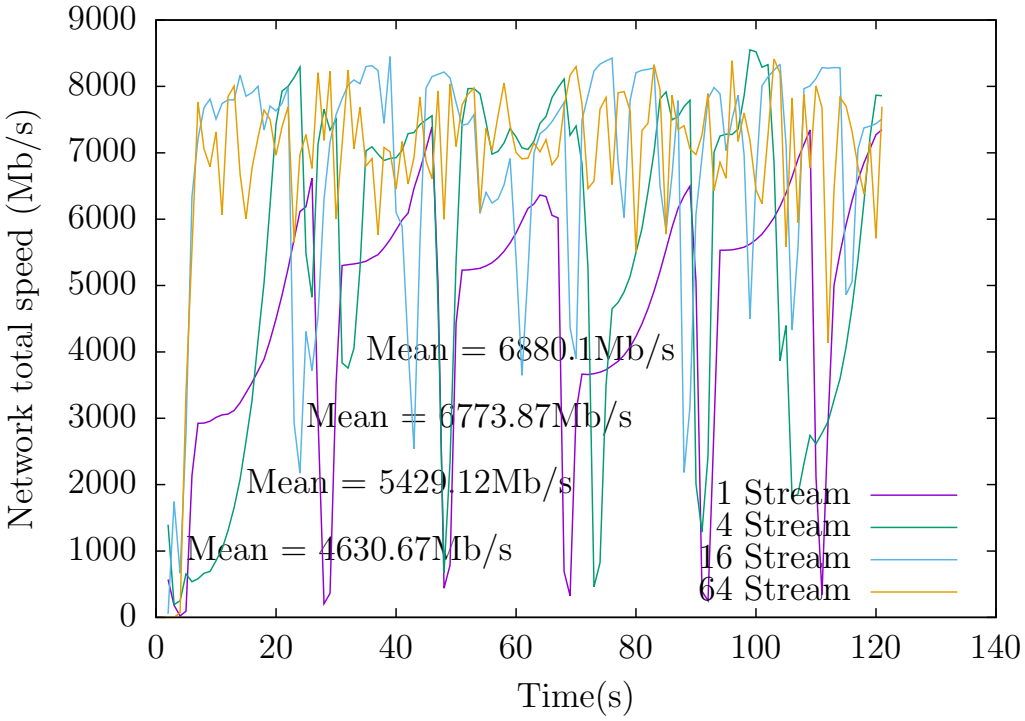


(c) 2 Streams 2x10GE send and receive in TCP with VLBI-streamer



(d) 8 Streams 2x10GE send and receive in TCP with VLBI-streamer

Figure 5.3.1: 2x10GE Local network receive and send.



(a) Variable TCP-streams

Figure 5.3.2: Long range TCP transfer rates between Metsähovi and INAF which had a 50ms delay. The mean value of the transfers rises as the number of streams increases

the delay. This is in contrast to the local receives, where ramp-up is not visible in the graphs as the TCP stream gets feedback from the TCP path almost instantaneously. The full path between the stations was measured to exhibit packet loss after using UDP transfer over a rate of 7Gb/s. This gave a good reference value against which to compare TCP results.

5.4 Distributed performance

Flexbuff is meant to operate on individual stations, which have long geographical distances between them. Flexbuff was tested running on 4 stations in addition to the central correlator. Each station was set to send a pre-recorded correlating dataset to the central correlator and record a stressing stream at the same time. This emulates the default behaviour of receiving an astronomical session locally from the FiLA10G and sending an already recorded set onward for correlation. The test setup is illustrated in figure 5.4.1

As explained in 3.5.2 the receiving machine at the joint institute for VLBI in Europe (JIVE) has to strip 8 bytes from each header. This adds some processing overhead as each packet must be written separately with the writev backend and DIRECT_IO cannot be used. The different speeds for the stressers are due to the station machines having different capabilities for receiving. The Onsala graph 5.4.2 shows a very unstable receive rate, where instead of a stable packet recording, the graph shows heavy undulation between 4.8 Gb/s to 6.8 Gb/s. This was due to packet loss, which was registered at the kernel level. The machine was probably lacking some optimization steps, but there was insufficient time to optimize the machine and it would have been quite risky to try to tune a machine 600km away just hours before the experiment. Watt was unstable due to a long FUSE development on it, that left a lot of defunct processes. The stalling behaviour would have been fixed with a reboot, but went unnoticed during the experiments.

Jodrell bank was limited to 1890Mb/s payload speed on their so called JBOD link. This set a common upload limit as the correlation was limited by the lowest upload speed. While the data was still being received in JIVE the correlation of the pre-recorded data set was tested successfully by the engineers at JIVE. As the FUSE system was already in development, it was also tested and shown to work. The FUSE-system enabled the use of single large files as the correlation data, instead of the default separate memory buffer size files. There were some lockups in FUSE though, that are likely due to a deadlock situation.

Some of the correlation load was also distributed to the receiving machine

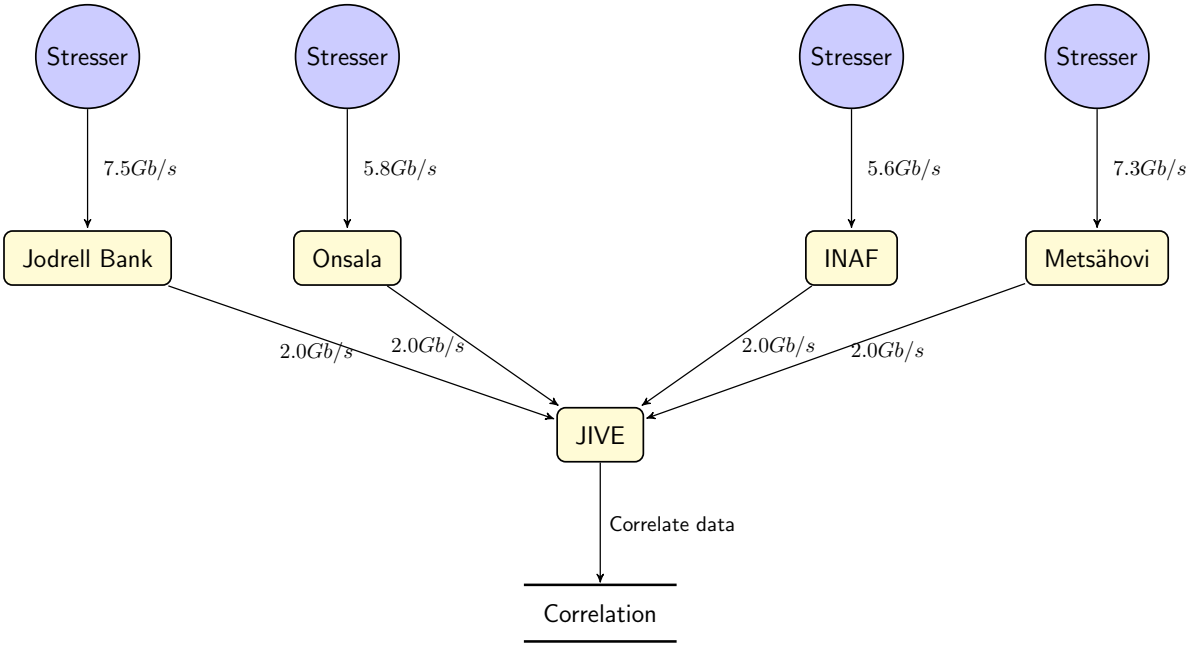
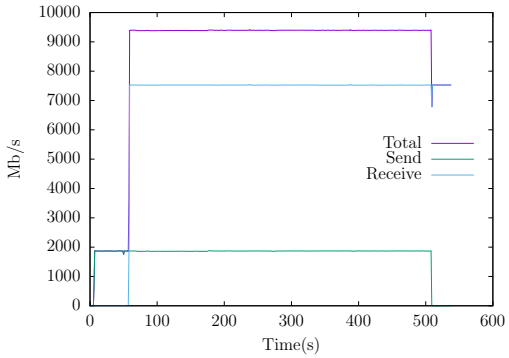
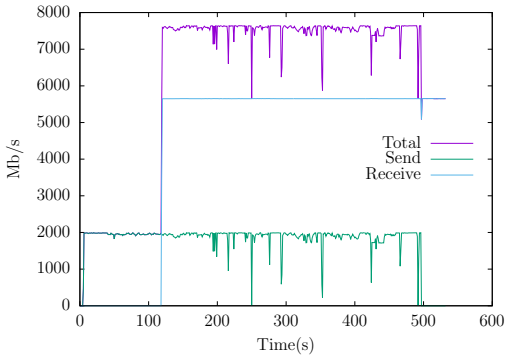


Figure 5.4.1: Distributed test setup

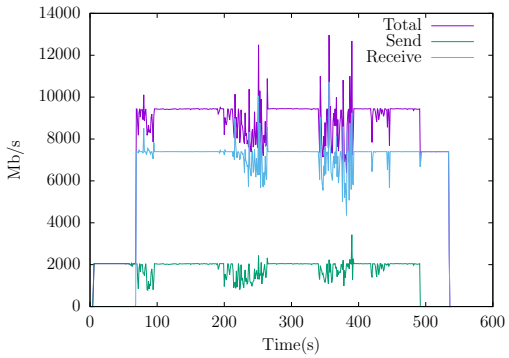
in JIVE, which showed no degradation in performance. Most likely the higher priority in VLBI-streamer protected its receive-process.



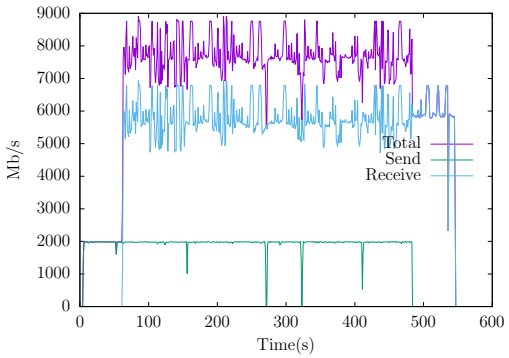
(a) Jodrell Bank



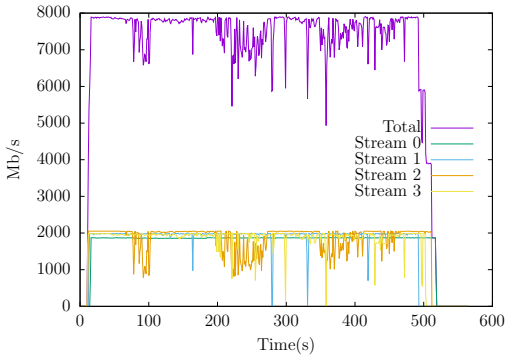
(b) Medicina



(c) Metsähovi



(d) Onsala



(e) JIVE

Figure 5.4.2: Distributed test

Chapter 6

Discussion

6.1 Performance

The desired initial performance was set to the recording of at least a 4Gb/s UDP stream described in 3.5.2 with hardware limits introduced in the introduction of chapter2. For operational purposes an added requirement was the sending of previous recordings while another recording was active, which enables recoding of an active session and the correlation of a previously recorded one. These were achieved fairly early on with the experiment in 5.4, where a 2Gb/s stream was simultaneously sent while receiving dummy data between 5.6Gb/s and 7.5Gb/s depending on the station.

After these experiments, the focus was shifted to testing VLBI-streamer against the hardware limits of its underlying system. It was the authors focus to create a non-hardware bound software solution that would give the software a longer lifetime. Using two 10GE interfaces simultaneous showed that larger than 10Gb/s transfers can be used, but seem to cap at the PCI-E bus-limits as observed in 5.2. Larger than 10Gb/s UDP streams might still result in packet loss, but in the spring of 2012 these could not be properly tested yet. During the experiments, 40GE interfaces were still too rare to get a hold of. Interface bonding could have been used, but was an unknown method to the author.

An important factor in achieving good performance happens when the packet is stored from the network into the random address buffer (RAM) buffer. The receiving side of an UDP-stream gets a continuous stream of data that has no flow control, but after the packet is in the memory buffers introduced in 4.3, the processing can be laxed from real-time to best-effort. Releasing the write to persistent storage from following a tight real-time schedule takes the physical aspect of spinning disks better into considerations,

as e.g. vibrations might cause a sudden drop in write speeds.

6.2 UDP considerations

Although all other tests went well, the ones with multiple NICs receiving a few high speed streams showed packet loss. It seemed that when adding another NIC into the tests, the packet capture immediately started to suffer from packet loss. This might be due to a contest on resources. With two NICs, the other will not have a monopoly on the structs and resources of the packet receiving kernel resources. The overhead from serializing access to these resources might be the cause of packet loss with the high speed streams. Increasing the kernel socket buffer size is one way to avoid this, but perpetually increasing a buffers size cannot be counted as a final solution.

During the spring of 2012 there existed no actual backends that could hit this limitation on VLBI-streamer. The issue could be solved with simply moving to a newer kernel with different scheduling parameters. Also a newer hardware platform would at least alleviate the symptons.

6.3 TCP considerations

Although there was not enough time to test the TCP multistreaming with real data sets, it showed that the mean transfer rate can be increased by 2.2Gb/s on the 50ms 7Gb/s line from Metsähovi to Italy. Further testing and development could useful for future developments. Also since VLBI-streamer is starting to be too large a software, this multiplication of TCP streams could be done outside of it. A simple program could be develop to either convert a single TCP stream into multiple streams or vice versa. This software could then run on both ends of a long fat pipe converting in a nearly transparent style.

Since the characteristics of the Metsähovi to INAF line suit a typical inside European VLBI network (EVN) session, this could be a superior mode of transfer in VLBI-sessions where a live recording could be available as a high bandwidth TCP-stream at the correlator within seconds. The buffering nature of TCP would also automatically limit the transfer rates to the slowest of stations, which would be the correlation speed anyway.

6.4 Software development considerations

The project was started with the assumption that memory copies should be avoided at all cost. Though they should still be avoided, there are some cases where the complexity of VLBI-streamer could have been reduced by moving its features to smaller units that work as preliminary stages for the data handling.

Examples of this are byte stripping as discussed in 4.8.2. If the byte stripping was done on a very small program that simply spliced data from an UDP socket and forwarded it via local domain socket to VLBI-streamer, the large amount of byte stripping logic in VLBI-streamer could have been avoided. Since byte stripping would have required extra memory copies anyway a solution with separate programs and memory copies in between would probably give even better performance results than implementing the feature directly into VLBI-streamer, especially from the software development viewpoint.

During VLBI-streamer development a range of different utilities were developed. These range from network testing to metadata inspection. These parts could be detached from the original project into separate toolkits for the VLBI-community.

Chapter 7

Conclusions

The purpose of VLBI-streamer was to handle the receiving of high speed astronomical data and subsequent sending of this data to remote correlation sites. Benchmark values were set to the maximal data transfer rates advertised in the spring of 2013 and after those were achieved, the target was extended to the hardware limits of the receiving system.

The experiments show that any current data rate generated by a back end for astronomical data can be captured on COTS hardware with software confined to the GNU/Linux user space. This means less software development costs and better longevity for such software solutions.

VLBI-streamer can also handle the simultaneous sending of datasets as shown in 5.2. This fills the original concept of VLBI-streamer as a flexible buffer for VLBI data and enables it to perform as a general recording device with less single points of failure.

7.1 UDP results

As section 5.1.1 showed VLBI-streamer can record any number and any speed UDP traffic on a 10GE link. It is only a question of configuring the software for the scenario, which might at this point be the softwares Achilles heel as the considerations into the correct configuration are hard to arrive at.

Tests with multiple 10GE cards showed good performance with multiple streams, but as discussed in 6.2 few very high speed streams resulted in packet loss. This shortcoming will not affect the operational capabilities of VLBI-streamer on current digital back ends. When 40GE networks become relevant for VLBI, the hardware will also most likely bring the software up to speed.

7.2 TCP results

TCP outperformed UDP on local network by a large margin. Also due to its stream-oriented and buffering nature, it should be an obvious choice for short to medium range transfers. The stream splitting feature was not tested enough, but should be further researched.

Bibliography

[1] *AF_PACKET fanout support*. http://lists.openwall.net/netdev/2011/07/05/30. July 2011.

[2] *Bandwidth Monitor NG*. https://sourceforge.net/projects/bwmng/. Apr. 2013.

[3] D.P. Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly Media, 2008. ISBN: 9780596554910. URL: http://books.google.fi/books?id=h0lltXyJ8aIC.

[4] Bob Braden et al. "Recommendations on queue management and congestion avoidance in the Internet". In: (1998). RFC 2309. URL: https://tools.ietf.org/html/rfc2309.

[5] C.A. Conley and L. Sproull. "Easier Said than Done: An Empirical Investigation of Software Design and Quality in Open Source Software Development". In: *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.174.

[6] Yuhui Deng. "What is the future of disk drives, death or rebirth?" In: *ACM Comput. Surv.* 43.3 (Apr. 2011), 23:1–23:27. ISSN: 0360-0300. DOI: 10.1145/1922649.1922660. URL: http://doi.acm.org/10.1145/1922649.1922660.

[7] *Filesystem in Userspace*. May 2014. URL: http://fuse.sourceforge.net.

[8] José Luis García-Dorado et al. "High-Performance Network Traffic Processing Systems Using Commodity Hardware". In: *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 3–27. DOI: 10.1007/978-3-642-36784-7_1.

[9] *Github: MIT Haystack VDAS (Mark6) public repository*. Nov. 2013. URL: https://github.com/MITHaystackObservatory/VDAS.

[10] R.E. Glazman. "An experimental implementation of interferometric techniques for sea level variation measurements and reflection coefficient phase determination". In: *Oceanic Engineering, IEEE Journal of* 7.4 (1982), pp. 155–160. ISSN: 0364-9059. DOI: 10.1109/JOE.1982.1145536.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". In: *SIGOPS Oper. Syst. Rev.* 42.5 (July 2008), pp. 64–74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105. URL: http://doi.acm.org/10.1145/1400097.1400105.

[12] *Hat-Lab home page*. http://www.hat-lab.com/hatlab/. 2013.

[13] Van Jacobson and RT Braden. "TCP extensions for long-delay paths". In: (1988). RFC 1072. URL: http://tools.ietf.org/html/rfc1072.

[14] U. Kalim et al. "Cascaded TCP: Applying pipelining to TCP for efficient communication over wide-area networks". In: *Global Communications Conference (GLOBECOM), 2013 IEEE*. Dec. 2013, pp. 2256–2262. DOI: 10.1109/GLOCOM.2013.6831410.

[15] Ilhwan Kim, Jungwhan Moon, and Heon Y. Yeom. "Timer-Based Interrupt Mitigation for High Performance Packet Processing". In: *In Proc. 5th International Conference on HighPerformance Computing in the Asia-Pacific Region, Gold*. 2001.

[16] *Linux Programmer's Manual: open (2)*. Feb. 2013.

[17] *Linux Programmer's Manual: readv (2)*. Nov. 2010.

[18] *Linux Programmer's Manual: splice (2)*. May 2012.

[19] *New API driver design*. http://www.linuxfoundation.org/collaborate/workgroups/networking/napi. Nov. 2009.

[20] *Packet MMAP*. http://www.mjmwired.net/kernel/Documentation/networking/packet_mmap.txt. Feb. 2013.

[21] Gregory F Pfister. "An introduction to the InfiniBand architecture". In: *High Performance Mass Storage and Parallel I/O* 42 (2001), pp. 617–632.

[22] Mrs Reena Rai and Maneesh Shreevastava. "Performance Improvement of TCP by TCP Reno and SACK Acknowledgement". In: *Performance Improvement* Vol. 2.1 (March) (2012).

[23] M.J. Rashti and A. Afsahi. "10-Gigabit iWARP Ethernet: Comparative Performance Analysis with InfiniBand and Myrinet-10G". In: *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. 2007, pp. 1–8. DOI: 10.1109/IPDPS.2007.370480.

[24] Allyn Romanow and Stephen Bailey. "An Overview of RDMA over IP". In: *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003)*. 2003.

[25] *StackOverflow: Linux splice() returning EINVAL ("Invalid argument")*. `http://stackoverflow.com/questions/7084254/linux-splice-returning-einval-invalid-argument`. Aug. 2011.

[26] William Stallings. *Operating Systems: Internals and Design Principles*. 6th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN: 0136006329, 9780136006329.

[27] W. Richard Stevens. *TCP/IP Illustrated (Vol. 1): The Protocols*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993. ISBN: 0-201-63346-9.

[28] Mikael B Taveniku and Jeffrey E Smith. *SYSTEM AND METHOD FOR HIGH-SPEED DATA RECORDING*. US Patent 20,130,091,379. Apr. 2013.

[29] Brian Tierney et al. "Efficient data transfer protocols for big data". In: *E-Science (e-Science), 2012 IEEE 8th International Conference on*. IEEE. 2012, pp. 1–9.

[30] Gino Tuccari et al. "DBBC2 backend: Status and development plan". In: *IVS General Meeting Proceedings*. 2010.

[31] Esa Turtiainen et al. "Hardware design document for simultaneous I/O storage elements". In: (2011). URL: `http://www.jive.nl/nexpres/lib/exe/fetch.php?media=nexpres:2011-02-28_wp8-d8.2.pdf`.

[32] D. Borman V. Jacobson R. Braden. "TCP Extensions for High Performance". In: (1992). RFC 1323. URL: `http://tools.ietf.org/html/rfc1323`.

[33] Alan R. Whitney and Roger J. Cappallo. *Mark 5B design specifications*. Tech. rep. Massachusetts institute of technology, 2004.

[34] Alan R Whitney et al. "Demonstration of a 16 Gbps Station-1 Broadband-RF VLBI System". In: *Publications of the Astronomical Society of the Pacific* 125.924 (2013), pp. 196–203. DOI: `10.1086/669718`.

[35] Alan Whitney et al. "VLBI Data Interchange Format (VDIF)". In: *Proceedings of the 8th International e-VLBI Workshop, PoS (EXPReS09)*. Vol. 42. 2009.

[36] *VLBI Standard Hardware Interface Specification – VSI-H*. Aug. 2000. URL: `http://www.vlbi.org/vsi/docs/VSI_H_paper_for_IVS_TOW.pdf`.

[37] *XCube Website*. http://www.x3-c.com. Nov. 2013.

[38] Takeshi Yoshino et al. "Analysis of 10 Gigabit Ethernet using hardware engine for performance tuning on long fat-pipe network". In: *Proceedings of PFLDnet 2007 (Fifth International Workshop on Protocols for FAST Long-Distance Networks)*. 2007, pp. 43–48.

[39] Young Jin Yu et al. "NCQ vs. I/O Scheduler: Preventing Unexpected Misbehaviors". In: *Trans. Storage* 6.1 (Apr. 2010), 2:1–2:37. ISSN: 1553-3077. DOI: 10.1145/1714454.1714456. URL: http://doi.acm.org/10.1145/1714454.1714456.

Appendix A

Appendix

A.1 Test machines

Name	CPU	OS	Motherboard
Ara	AMD Phenom II X6 1090T	Debian 6.0.7 Kernel 3.8.6	Crosshair IV Extreme
Watt	2x Intel Xeon E5620	Debian 6.0.7 Kernel 2.6.32	Supermicro X8DTH-i/6/iF/6F
JIVE	2x Intel Xeon E5620	Debian 6.0.7 Kernel 2.6.32	Supermicro X8DTH-i/6/iF/6F
Jodrell	2x Intel Xeon E5620	Debian 6.0.7 Kernel 2.6.32	Supermicro X8DTH-i/6/iF/6F
Medicina	2x Intel Xeon E5620	Scientific Linux	X8DTH-i/6/iF/6F
Onsala	AMD Phenom II X6 1090T	Debian 6.0.7 Kernel 2.6.32	Crosshair IV

Name	Memory		Drive controllers	HD	Network	
Ara	16GB DDR3	1333Mhz	LSI Logic / Symbios Logic	24	2x10GE + 2x1GE	
Watt	24GB DDR3	1066Mhz	Fusion-MPT SAS-2 Falcon	36	2x10GE + 2x1GE	
JIVE	24GB DDR3	1066Mhz	Fusion-MPT SAS-2 Falcon	36	2x10GE + 2x1GE	
Jodrell	24GB DDR3	1066Mhz	Fusion-MPT SAS-2 Falcon	36	2x10GE + 2x1GE	
Medicina	12GB DDR3	1066Mhz	Supermicro 3ware Inc 9750 SAS2/SATA-II RAID PCIe	12	1x10GE + 2x1GE	
Onsala	16GB DDR3	1333Mhz	Fusion-MPT SAS-2 Falcon Extreme	24	2x10GE + 2x1GE	

A.2 Used network cards

Manufacturer	Model	Capacity	PCI-Express	Stations
Chelsio	T240-CR	10GE	2.0 x8	Jodrell Bank
Intel	82598EB	10GE	2.0 x8	JIVE, INAF, Metsähovi
Intel	82599EB	10GE	2.0 x8	Metsähovi
Myricom	Myri-10G-PCIE-8A	10GE	2.0 x8	Onsala, Metsähovi