# Data multicasting for the IJkdijk as a large-scale sensor network

H.A. Noordzij

February 25, 2008

# VOORWOORD

De eindopdracht van mijn Bachelor Telematica heb ik uitgevoerd bij TNO ICT in Groningen. Met veel plezier heb ik gewerkt aan telematica vraagstukken rondom de IJkdijk, een oerhollands project rondom dijken en watermanagement.

Graag gebruik ik deze gelegenheid om de mensen die mij enorm geholpen hebben tijdens het doen en vooral ook het voltooien van mijn bachelor opdracht, te bedanken:

- Prof. dr. R.J. Meijer, TNO, begeleider
- Dr. ir. A. Pras, UT, begeleider

Ook wil ik Peter en Kristian bedanken voor hun gouden tip om laarzen mee te nemen naar altijd gezellige bezoekjes aan de IJkdijk.

Verder wil ik mijn ouders bedanken voor hun eindeloze geduld, en Erika voor haar grenzeloze vertrouwen.

CONTENTS

# 1. INTRODUCTION

## *1.1 IJkdijk Context*

The Dutch IJkdijk is a testing facility in which dikes, sensor technology and scientific models for dikes are tested [9]. This dike is located in a remote place, east of Groningen, near the German border. The 'Nederlandse Organisatie voor toegepast-natuurwetenschappelijk onderzoek' (TNO) is one of the major participants in the Dutch IJkdijk project. Their main interest in this project is to develop a market for research related to the ICT of huge sensor networks. One of TNO's activities is the development of a sensor telecommunication system that conveys data from sensors and actuators between dikes and WAN's. As a case for designing sensor telecommunication systems, the diverse and plentiful dikes of Holland are very suitable.

*Fig. 1.1:* The IJkdijk

### *1.1.1 Dutch Dikes*

In the Netherlands there are over 17.000 km of dikes. Monitoring these dikes can improve safety and reduce construction costs. Currently, the trend is to built dikes exceedingly solid, and therefore overly expensive. The dikes are built robust in order to reduce the risks of a breach to near zero. However, by equipping dikes with sensors, one is able to continuously monitor the stability of the dikes and therefore be informed which dikes are in need of mainte-

nance or reinforcement. In the long run, this will nonetheless *reduce* the costs of both construction and maintenance whilst increasing safety. Monitoring all the dikes requires an advanced data network; this is a challenge for the currently available telecommunications structures. Densly populated area's are well connected, both with connectivity by cables, as well as through the ether. Unfortunately, most dikes are not so well connected. Moreover, today's telecom networks only provide a best effort service. When sensors guard critical systems and infrastructures, best effort might not be good enough. TNO is in the process of developing a networking concept that enables networks to adapt to specific application needs, such as guarantied bandwith and reliability. This concept is known by the name User Programmable Virtual Networks (UPVN) [5] and could prove to be very useful in this particular case.

### 1.1.2 Sensor telecommunication

TNO's expectation is that the IJkdijk case will result in a sensor telecommunication solution, suitable for high volume data streams. This solution should also be applicable to other intelligent, large scale infrastructures. TNO's approach to develop the IJkdijk telecommunication system is however practical, therefore the development has started by creating a test setup and solving the sensor telecommunication issues with this setup first.

### 1.1.3 Test setup

The test setup currently in use on the IJkdijk employs, amongst other sensors, several microphones connected to a PC with LabView §4.3. Therefore, in this case audio data will be streamed. In the ultimate case the sensor telecommunication system has to deliver 340000 streams, a sensor in every 50 meters of dike. However, the research question the current research addresses is whether UPVN or an off the shelf solution (e.g. from a large software vendor) to stream media should be used to broadcast the first data from the IJkdijk location.

## 1.2 Problem description

Monitoring all the Dutch dikes and analyzing the data from their future sensors is a complicated problem. In this bachelor thesis, the focus is on the IJkdijk case, altough the view will be broadenend to the Dutch case where relevant.

Since the IJkdijk is a testbed, the dikes will be equipped with various sensors from various interested parties. The sensors that will be installed on the short term are microphones, waterpressure sensors and a weather station. TNO will provide facilites for telepresense. The idea is to provide a platform on which sensor data can be streamed, stored and processed. The latter two are relatively simple: this is a matter of providing enough resources. For streaming, a solution is probably not trivial. Currently, we are unaware of a solution that can stream sensor data out-of-the-box.

These are the options TNO is currently considering as streaming solutions:

1. National Instruments (NI) Data Acquistion systems and LabView software

2. Microsoft Windows Media Server
3. User Programmable Virtualized Networks
4. A combination of the above

NI LabView is a populair program for data acquisition and processing. It is well known for its powerful graphical programming environment and extensive analysing abilities. It is expected that LabView can facilitate most of the storage and processing needs on the IJkdijk. A topic of investigation is the networking ablities of LabView and their applicability for streaming data.

Microsoft Windows Media Server is a software package providing all the parts needed to stream audio and video, such as encoders, distribution servers, decoders and a player. It is widely used for radio and television broadcast via internet. A topic of investigation is the ability to stream sensor data instead of audio and video.

A difficulty at this stage in the IJkdijk project is that it is not exactly known what it is we are looking for. The expectation is that listening to a dike with microphones will give us information about the condition of the dike. There are threads that can be detected in a short timespan, like cracking sounds, or the noises that digging 'muskusratten' (big rats) make. Then there is the changing of environmental sounds over time due to changing conditioning like the wetness of the dike. Having some idea about what is going to be analyzed is important for making a choise for e.g. samplerate, resolution and ways to correlate this data to other information.

At the start of this bachelor assignment, May 2007, the first physical dike, was completed at the IJkdijk location. This dike is is equipped with a *luisterbuis* (a listening tube). This 50 meter long, horizontal tube has a microphone at each end. With this we can listen to what is happening in and around the dike. As stated before, dike monitoring is currently done without the assistance of sensors. As a concequence, not much is known about observing dikes with the help of sensors. One of the goals of the IJkdijk project is finding ways to do so. There is good reason to believe that placing microphones in the dike will reveal a great source of information about what is going on inside a dike. Geophone (low frequency microphones) are invaluable for research and early warning systems for volcano eruptions, earthquakes, etc. Furthermore, constructions like bridges, can be equiped with highly sensitive sensor that can hear cracking in bridge cables or steel components.

## 1.3 Research questions

The main question is: *"How do we stream and multicast audio data from the IJkdijk to several internet attached analysis systems?"*

More refined questions are:

1. Are there compareable cases tot the IJkdijk case? If any, What are there solutions?
2. What is the current state of the art in streaming and multicasting?
3. Can off the shelf (OTS) solutions, such as LabView or Windows Media be deployed? If yes, how?

4. Can User Programmable Virtual Networks (UPVN) be deployed? If yes how?
5. What could be a suitable development path for TNO's data acquisition and data publishing setup?

## 1.4 Research approach

TNO provided National Instruments hardware and software as well as Windows Media software. UPVN is a network infrastucture concept, of which TNO has a rudementary prototype running. TNO has the desire to incorporate these systems into the solutions developed for the IJkdijk case.

In order to answer the research questions, the following approach for this assignment was chosen:

1. Perform a literature study to investigate the current state in streaming and multicasting
2. Perform a literature study to investigate cases similar to the IJkdijk case
3. Create basic LabView application for audio acquisition
   (a) Evaluate and demonstrate LabView communication facilities
   (b) Compare and evaluate available streaming solutions
4. Build or adapt an UPVN prototype to fit the IJkdijk case
5. Formulate communications concept for the IJkdijk

## 1.5 Structure

In Chapter 2 a short survey of similar sensor networks is presented, with a focus on distributing and managing sensor data. This chapter should answer the first research question. Chapter 3 explores relevant networking technologies such as streaming and multicasting, which should provide us with a background for working on the second research question. In Chapter 4, the IJkdijk context is further investigated, as to wether TS can be deployed. The prototype is described in Chapter 5.

## 1.6 Intended Audience

This text assumes basic computer networking knowledge at the level of a computer science bachelor or similar. The text targets people who work with the measuring and telematics aspects of the IJkdijk.

# 2. SURVEY OF SENSOR NETWORKS

## 2.1 Introduction

In this chapter a short survey similar sensor networks is presented. The survey is complete nor exhaustive, but does scetch the architecturale challanges for sensor networks on a large scale.

Here we briefly scratch the surface of a few sensor network related projects and inverstgate if and how they relate to the IJkdijk case. These sensor networks are a hot topic now days. The advances in wireless communication and energy conservative miniature devices make the deployment of large-scale, low power, low cost, sensor networks possible.[11] Dike monitoring as envisioned for the Dutch case is also a very large scale sensor network. Therefore an overview of sensor network cases that have something in common with our case is at it's place.

## 2.2 Tsunami detection

On 26th December 2004 a tsunami struck the unmonitored Indian Ocean and caused the death of over 100,000 lives in 11 countries. Although a tsunami warning system is already in place for the pacific ocean since 1965, only after this event initiatives have been deployed to build such a system for the Indian Ocean. Within 18 month the new system has been made operational.[10]

Tsunami detection systems deploys buoys with global positioning system and satellite communication.[4] The data from the roughly 30 buoys is monitored at special centers, currently in Japan and Hawaii. Tsunamis can be detected within seconds after the wave passes the buoy. The effectiveness of the warning system is however greatly depended on the local governments to act upon those warnings. Tsunami detection is an example of one of the earliest technology assisted warning systems against environmental powers of destruction. However, the technology bears little resemblance to our problem.

## 2.3 LOFAR

LOFAR is an acronym for the LOw Frequency ARray. LOFAR is a very low frequency radio telescope for astronomical observations. It is build as a distributed sensor network, with 160 stations distributed along five spiral arms. These stations contain a few hunderd antennas. The first stage of the signal processing takes place at the stations. Signals are then transported to a CEntral Processing (CEP) facility for further processing. This CEP is an IBM Blue Gene supercomputer at the RuG.[8]

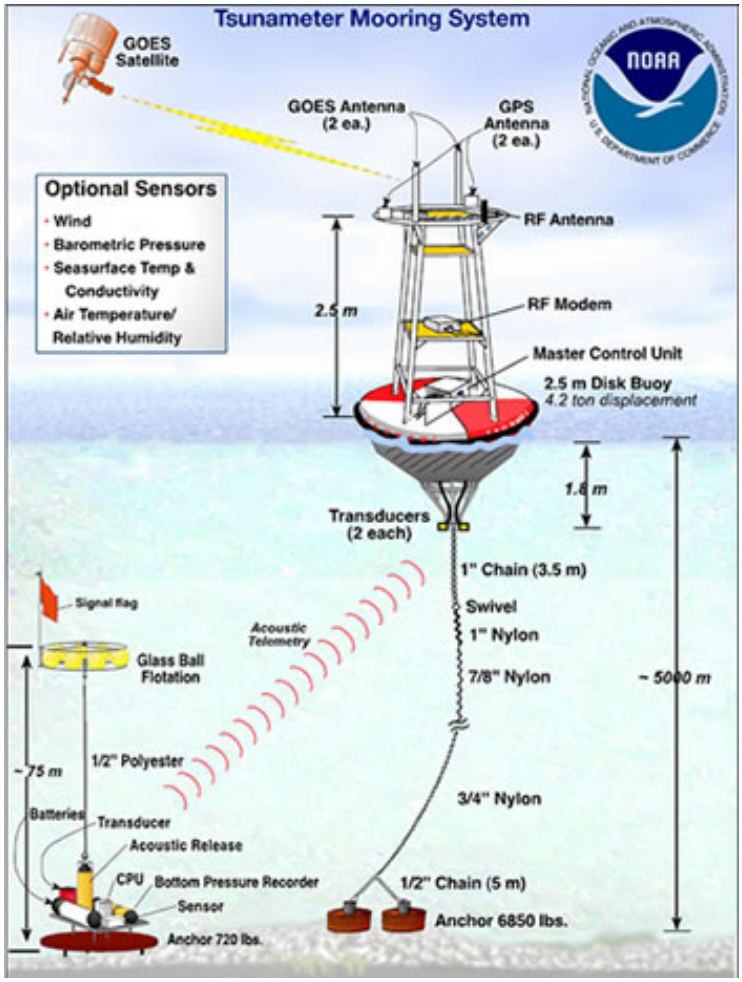


*Fig. 2.1:* Tsunami Detection Buoy

*Fig. 2.2:* LOFAR

A total of 13,000 antennas produce a datastream of 2 Gbit/sec each. The expected maximum data rate from all remote stations will be more than 26 Tbit/sec. In the initial fase of the project they connect the stations with 10 Gbit ethernet. In a later fase they will upgrade to advanced WDM / OTDM data transport technologies.[1]

Allough it might not be obvious at first glance, there are some similarities with our case. First of all, broadband connections in rurals areas are required. Furthermore, the operation of the telescope is based on the streaming of thousends of sensors, antenna's connected to custom build DA converters. As a comparision, the DA converters are 12 bit, the sample rate goes up to 200 Mhz. So the resolution is comparable to audio, the sample rate much higher.[3]

## 2.4 Seismic data in ROADnet

ROADnet stands for Real-time Observatories, Applications, and Data management Network, a project form the University of California. According to their website they are "building upon currently deployed autonomous field sensor systems, including sensors that monitor fire and seismic hazards, changing levels of environmental pollutants, water availability and quality, weather, ocean conditions, soil properties, and the distribution and movement of wildlife. ROADNet scientists are also developing the software tools to make this data available in real-time to a variety of end-users, including researchers, policymakers, natural resource managers." So their mission bears many resemblances to the IJkdijk.

They have build a framework based on Virtual Object Ring Buffers (VORB). Various types of seismic and environmental sensors can be connected to their ORBs. These ORBs will run identical software enabling the provision of real-time or near real-time data streams/packets from any number of specified data sources. The ORB receives said data via a specified port, expects data packets

to be named a certain way, and doesnt care about the internal content of the packets. The ORBs are then connected to VORBs, which can be queried via an interface, which will then return a combination of metadata, real-time data streams, database records, etc. from the VORB, depending on the request.[6]

## 2.5 NEES

The American Network for Earthquake Engineering Simulation(NEES) has a large IT infrastructure to connect earthquake simulations sites throughtout the USA. They have a lot of experience with teleprence on testsites. Interestingly enough, much of their documentation and tools are available on the web, e.g. clearly documented LabView DAQ code, a LabView control server, and examples of their usage of the Creare Dataturbine. Figure 2.3 shows the concept of a dynamic data server, used for static data and streaming data. The next section discusses DataTurbine in more detail.


*Fig. 2.3:* DataTurbine, Ring Buffered Network Bus concept

## 2.6 Creare RBNB Dataturbine

Creare Ring Buffered Network Bus (RBNB) DataTurbine, or DataTurbine in short, is a software server that provides a buffered network data path between suppliers and consumers of information. It stores all data and metadata in a uniform way, so that it can easily be accessed. An example of the easy access with the RDV viewr GUI is given in Figure 2.4.

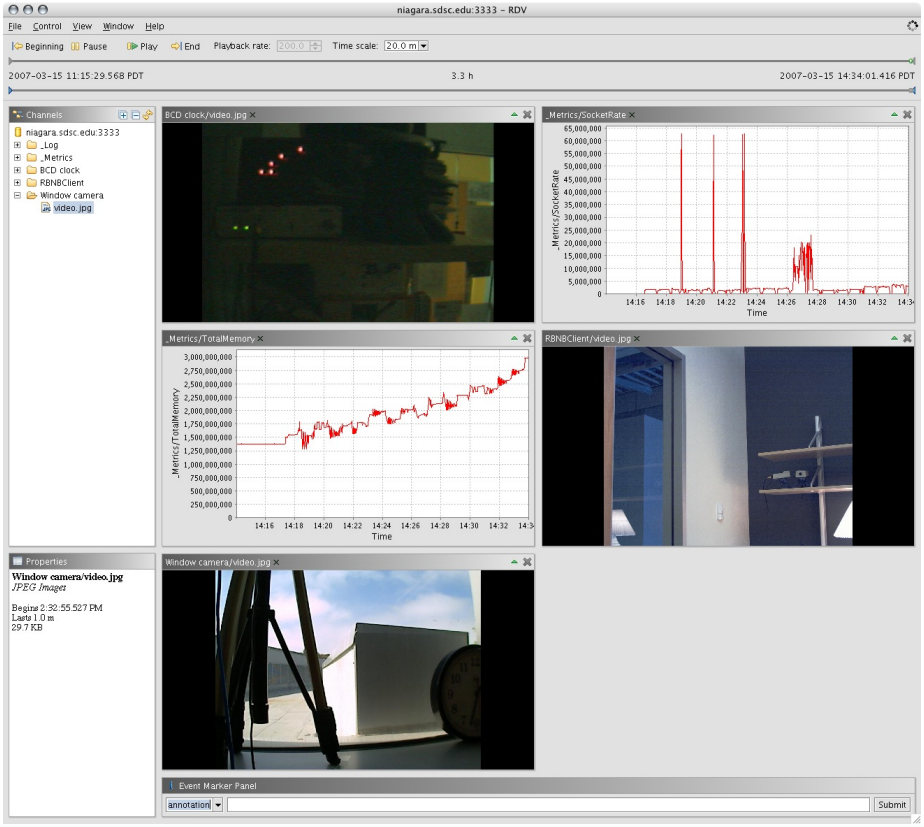


*Fig. 2.4:* DatTurbine RDV viewer

## 2.7 Summary

TNO has the intention to develop expertise in the field of telepresense, but also the desire to develop innovative network technologies. A lot can be learned from similar cases, since common elements can be extracted from the aformentioned cases. Most of them use a hierarchical setup. Data is stored and cached close to the sensors, the caches are then connected to caches higher in the hierarchy, closer to the interested user. The middleware software in use, such as VORB and DataTurbine are either completely open source or at least well documented. It is relatively easy to connect new types of sensors to the middelware. Furthermore, while these middleware solutions do support realtime or near realtime streaming, they partition their data into packets or chuncks, making it more easy to handle them. It is easy to find example of cases which use LabView for DAQ, as well as examples which build custom DAQ solutions, tailered to their needs and their budgets.

# 3. STREAMING AND MULTICASTING DATA

## 3.1 Introduction

The focus of this assignment is on the continuous transport, or streaming, of sensor data from the IJkdijk location to interested parties elsewhere. It is probably the case that the number of interested internet connected parties is more than one.

In this case then, it is useful to deploy multicast as a way to distribute the sensor data to those parties. However, deploying multicast is far from trivial. In this chapter, multicast and streaming deployment is investigated in more detail.

## 3.2 Multicast background and principles

### 3.2.1 Unicast

The most common way to send data from one host to another is unicast. A single connection or path is setup between sender and receiver, possibly a two-directional path. If a host, say 'Server', sends data to two hosts, say 'computer 1' and 'computer 2' , it wil setup two connections. The Server now has to send all data twice. When the data send to computer 1 and computer 2 is different, than unicast is the best way to transfer data. It is clear however, that this solution does not scale very well, the load on the Server is directly related to the number of client computers. Furthermore, in many cases the same data travels multiple times over the same network connection, where only one transfer could have been sufficient. This is shows in figure 3.1. When the data send to the different computers is identical, optimizations seem possible.

### 3.2.2 Broadcast

The most straightforward way to send the same data to multiple destinations is broadcast. Well known examples are both cable and satallite television. The television channels are all send to everyone with a cable connection or a satelite dish, it is up to the user to make a selection, or tune in, to the desired channel. This works well if there are many interested parties.

In a computer network, broadcast is also possible. However, while the ether and the coax cable are regulated by either the government or the cable company, the computer network is the shared responsibility of the operators *and* the users. The latter do not only *receive* data, as is the case with television, but also *send* data. Computer networks, such as Local Area Networks (LAN), are commonly devided into broadcast domains. These broadcast domains are connected by routers, which form the barrier of a single broadcast domain.

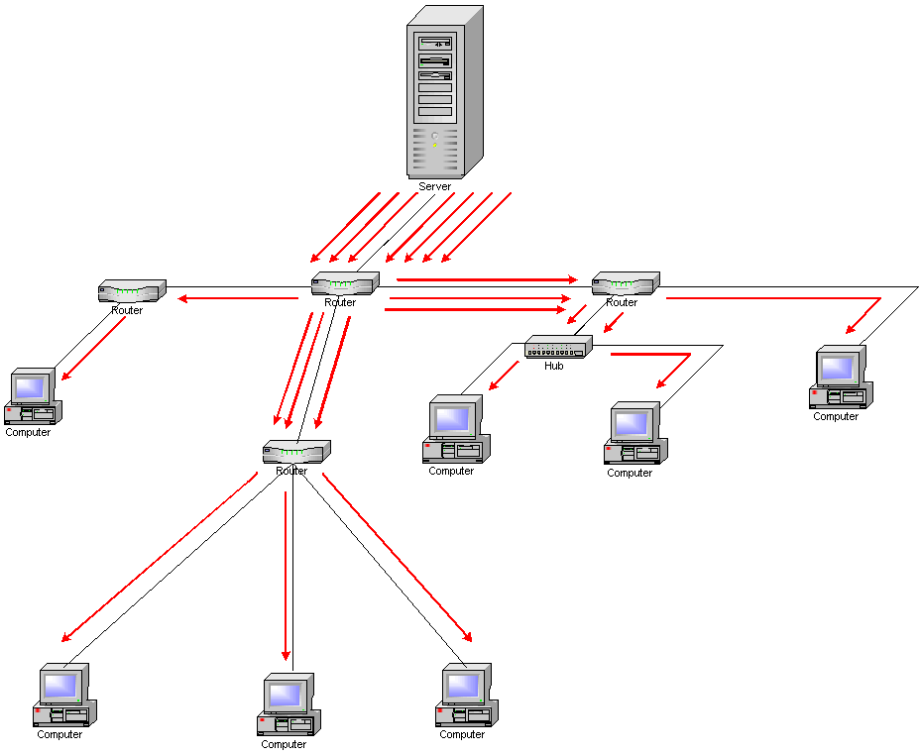


*Fig. 3.1:* unicast (source: surfnet.nl)

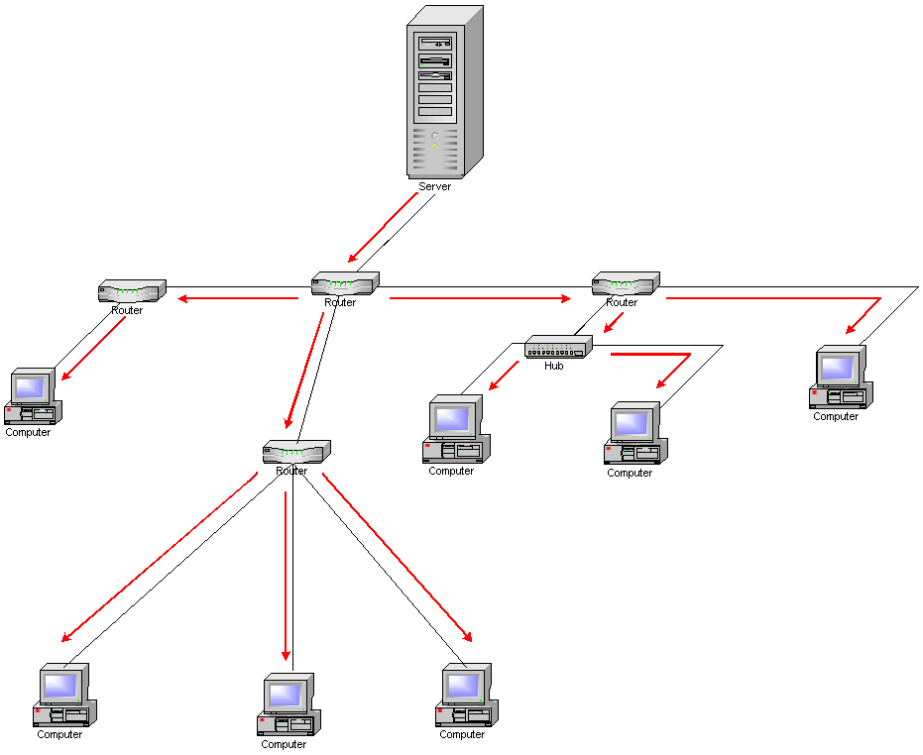


*Fig. 3.2:* multicast (source: surfnet.nl)

The broadcast domains can contain anywhere between one and a few thousand hosts. When a Server sends identical data to all other computers in the broadcast domain, then broadcast is a good options. However, when only a few hosts are interested in the data, then it is a waist to send the data to all, since the available capacity in a LAN is limited. In practise, broadcasts in a LAN are only used for discovery and configuration processes, such as finding a configuration server (e.g. a DHCP server) in a broadcast domain.

### 3.2.3 Multicast

Multicast is very interesting from the application developers point of view. The distribution is very simple, all the complications are moved form the end-host to the network.

The transport of identical information to a group of interested receivers is known as multicast. Multicast aims at the most efficient delivery of the information by sending the messages only once over each link in the network, by creating copies only when the links to the destinations split, as is illustrated in figure 3.2. The routers need to be more complex, since they now have to create the optimal distribution path. They need to build a spanning tree to get the packets to their destinations. In order to let the routers coordinate their effort towards creating the tree, several different protocols are needed. They are discussed in the next session.

## 3.3 Multicast Protocols

The most relevant form of multicast is IP multicast. Different types of multicast exist, e.g. for ATM networks, but the focus here is IP multicast. The IP network can distingish multicast traffic from normal unicast and broadcast traffic since different address classes are used.

### 3.3.1 Multicast addresses

| Address range | Mask | Description |
|---|---|---|
| 224.0.0.0 224.0.0.255 | 224.0.0/24 | Local Network Control Block |
| 224.0.1.0 224.0.1.255 | 224.0.1/24 | Internetwork Control Block |
| 224.0.2.0 224.0.255.255 | - | Ad hoc Block |
| 224.1.0.0 224.1.255.255 | - | Unassigned |
| 224.2.0.0 224.2.255.255 | 224.2/16 | SDP/SAP Block |
| 224.3.0.0 231.255.255.255 | - | Unassigned |
| 232.0.0.0 232.255.255.255 | 232/8 | Source Specific Multicast Block |
| 232.0.0.0 232.255.255.255 | 233/8 | GLOP Block |
| 234.0.0.0 238.255.255.255 | - | Unassigned |
| 239.0.0.0 239.255.255.255 | 239/8 | Administratively Scoped Block |

The table with multicast addresses raises several issues. First of all, when building a multicast application, it is not so obvious what address to choose for the multicast address. For application running in a single domain, the Ad hoc block could be used. Two other interesting ranges are the Source Specific Multicast Block, which is discussed in the multicast protocol and the GLOP block. The latter block is intended to be used for global multicast. Every autonomous system (AS) has a 255 GLOB addresses of the format 232.AS.number.0.

### 3.3.2 Internal Multicast Protocols

#### Internet Group Management Protocol

The protocol used by receivers, or hosts, to join a multicast group is called the Internet Group Management Protocol (IGMP). The most common version, version 2, is specified in RFC 2236. The most recent version is version 3, specified in RFC 3376. The first version only had a join message, which is send to the nearest multicast router to request to join a multicast group. A timeout mechanism was used to discover the groups that are of no interest to the members. The second version had al leave message added to the protocol. Version 3 is a major revision of the protocol. It allows hosts to specify the list of hosts from which they want to receive traffic from. Traffic from other hosts is blocked inside the network.

#### Protocol Independent Multicast

There are many varitions on the Protocol Independent Multicast (PIM) protocol, such as PIM Sparse Mode (PIM-SM); PIM Dense Mode (PIM-DM); Bidirectional PIM and PIM Source Specific Multicast (PIM-SSM). We focus on PIM-SM and PIM-SSM, since they are most commonly used in real networks.

PIM Sparse Mode (PIM-SM) explicitly builds shared trees rooted at a Rendezvous Point (RP) per group, and optionally creates shortest-path trees per source. PIM-SM can use any routing protocol to populate its multicast Routing Information Base (RIB). PIM-SM generally scales fairly well for wide-area usage. The protocol is specified in RFC 4601.

PIM Source Specific Multicast (PIM-SSM) builds trees that are rooted in just one source, offering a more secure and scalable model for a limited amount of applications (mostly broadcasting of content). In SSM, an IP datagram is transmitted by a source S to an SSM destination address G, and receivers can receive this datagram by subscribing to channel (S,G). specified in RFC 3569.

PIM-SSM in combination with IGMPv3 is very suitable for one-to-many multicast, such as internet radio or the IJkdijk case. PIM-SM is more suitable for many-to-many multicast, such as video conferencing.

### 3.3.3 Border Multicast Protocols

The aformentioned algorithms are designed to work in a single domain or intranet. When multicast traffic crosses a network border, e.g. form one provider to another, other protocols are required. The protocols which are in use today are the Multiprotocol Border Gateway Protocol (MBGP) and the Multicast Source Discovery Protocol (MSDP).

## 3.4 Alternatives to multicast

### 3.4.1 Proxy

In order to reduce bandwith usage, especially on the server, there are alternatives to multicast. The use of a caching proxy is one of them. In this case, the data that is streamed to multiple receivers, is copied and stored on a server in between the server and some of the receivers. Often such a cache is placed at an ISP or a company. The streamgate case at the end of this chapter is a clear example.

### 3.4.2 Peer-to-peer

A relatively new technology, that surpasses multicast and proxies in popularity, is peer-to-peer networking. Peer-to-peer turning into a powerfull technology for broadcasting multimedia. It was not considered relevant for the IJkdijk case in this stadium. This might change in the future.

## 3.5 Streaming software in practise

In this section, Microsoft Windows Media (WM) is introduced. WM includes all the parts that are common for streaming software, such as encoders, distribution servers, decoders and a player.

Microsoft Windows Media(WM) is chosen as media solution to investigate as a part this assignment, because it was available at TNO and is relatively easy to deploy. Other options to evaluate could be Realplayer and their Helix server, or combination of several open source alternatives. In later chapters, examples of integration which several open source applications are given.

Currently, YouTube is the even more well known streaming application. It is however based on Adobe/Macromedia Flash and it uses HTTP as an underlaying protocol to transfer the stream. While this is suitable for transferring stored multimedia, it is less suitable for realtime streaming, such as radio or sensordata.

## 3.6 Microsoft Windows Media Streaming Software

Windows Media software consists of these parts:

- Audio/Video Capture
- Audio/Video Encoder
- Audio/Video Distributor
- Audio/Video Decoder/Player

In figure 3.3 a possible setup as scetched by Microsoft is shown.

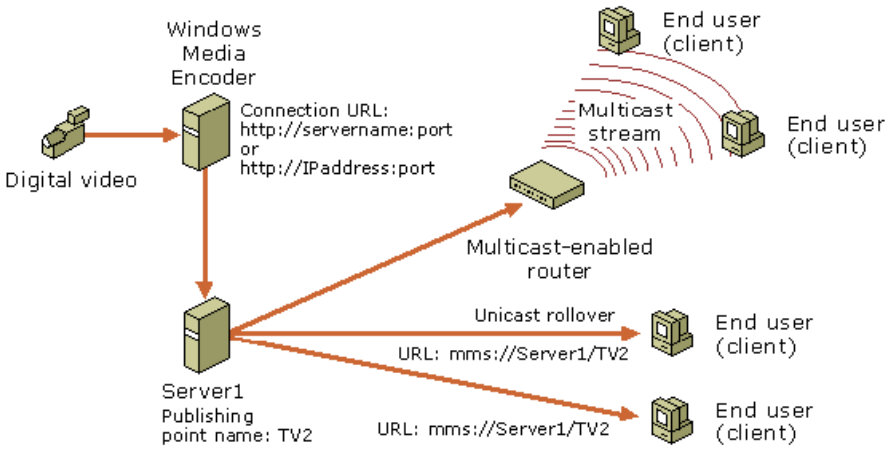


*Fig. 3.3:* Windows Media

There are several reasons why it could be usefull to use Windows Media or similar software in the IJkdijk case:

- Compression
- Encryption
- Scalability:
  - Multicast
  - Proxy/Cache

### 3.6.1 Compression

Data compression is the process of encoding information in such a way that it requires less storage, often expressed in bits. Interesting examples of compression in the session layer are audio MP3 compression in e.g. internet radio or zip compression of webpages in HTTP transfers. The first closely resembles our situation of streaming seismic audio data. The second is very common, but not so interesting for the IJkdijk case.

MP3 and similar compression algorithms can reduce the bitrate of CD quality audio of about 700 KByte/sec to 192 Kbyte/sec or less. However, this compression has its price. The biggest problem is the 'loss of quality': the decompressed signal is not identical to the original signal, some frequencies are lost. Other problems are the extra processing power requirerd to compress the signal, and to a lesser extend, to decompress the signal. Furthermore, an extra delay is introduced, since MP3 compression works best on larger blocks of samples, these blocks need to be buffered first, thereby causing extra latency.

The fact that lossy compression, such as MP3 compression, removes information from the original signal makes it unusable for most scientific measurements, such as those planned for the IJkdijk. When analyzing signals, it must always be clear (as clear as possible) whether or not the results are related to the events measured or introduced by the compression.

An alternative to lossy compression is lossless compression. Zip, a popular algorithm for file and website compression, is an example of lossless compression, however the zip algorihm is very ineffective for reducing audio files. Effective lossless algorithms do exist for compressing audio, e.g. FLAC, Microsoft WM lossless, and others. However, the compression rate is highly dependant on the nature of the originial signal. Moderately complex signals, e.g. music, can roughly be decreased by a factor two, so a CD quality stream would on average become 350 KByte/sec. How well various types of sensor data can be compressed is an interesting topic for further research.

### 3.6.2 Encryption

Encryption is the process of encoding data in order to prevent unauthorized parties to access that data. Encryption is particularly interesting for secret or copyrighted data. At this stage it is not a priority for the IJkdijk case.

### 3.6.3 Scalability

Scalability, the ability to handle growing amounts of work in an efficient manner, is an important criterion for comparing multimedia streaming solutions. To cope with this demand, there are several approaches: use multicast or us proxies.

Multicast can be enabled on WM, however it is only available when WM is installed on the more expensive Windows server editions: Enterprise Edition and Datacenter Edition. When this requirement is met, it is very simple to enable multicast for a stream. WM has support for both IGMP version 2 and 3.

An interesting feature of WM is the proxy/cache function. A WM installation can be configured to function as a proxy. An example of the WM proxy follows in the next section.

| Parameter | Soundcard DAQ | Dedicated DAQ |
|---|---|---|
| Input Range | $\pm$ 400 mV | wide, variable range, e.g. $\pm$ 1 mV up to $\pm$ 24 V |
| Input Impendance | 600 $\Omega$ - 47 $k\Omega$ | 1 $M\Omega$ to 10 $M\Omega$ |
| Input Frequency | $\pm$ 200 Hz to 20 kHz | Direct current, miliHz to 200+ kHz |

*Tab. 3.1:* Comparisson Soundcard DAQ and dedicated DAQ

### 3.6.4 Windows Media Input

The aformentioned features of WM could be useful for the IJkdijk case. So how can we deploy WM on the IJkdijk? Experimenting with WM and reading the documentation [2] revealed that there are three ways to insert data into Windows Media:

- Multimedia files
- Soundcard
- DirectShow filter

Using files is not very useful, since we are interested in the steaming of realtime data. Input via a soundcard could be used for the luisterbuis, one of the sensors on the IJkdijk. A soundcard is however not useable as a generic DAQ card, as is shown in comparisson table 3.1. The soundcard is unusable in the lower frequency range, which is exactly what we are interested in when listening to the luisterbuis, which captures vibrations from 1 kHz to the sub 1 Hz range. Furthermore, a soundcard is unusuable for all sensors but mirophones. Geophones, water presure sensors, etc. cannot be samples with a soundcard. This leaves the DirectShow filter as the only viable option. This filter basically is the API by which programmers can insert their own components into the WM framework. It could be used to inject data acquired by LabView into WM. However, building a DirectShow filter requires substantial knowledge of the WM framework and Windows COM programming. Therefore, this option is not futher investigated.

## 3.7 Multicast support by ISPs

The Dutch broadcast organisation (NPO) deployed a proxy based solution to distribute its online content to the customers of the biggest five Dutch consumer providers. The construction is known by the name streamgate[7]. Those providers do not have multicast enabled on their (consumer) customer network. The most popular content is not only stored at the NPO servers in Hilversum, but also on caching servers located at those providers. This illustrated in figure 3.4. Advantages are the decreased costs for bandwith for NPO and the lower load on interconnects (at an internet exchange). Disadvantage is the effort required to install and configure the servers at the ISPs.

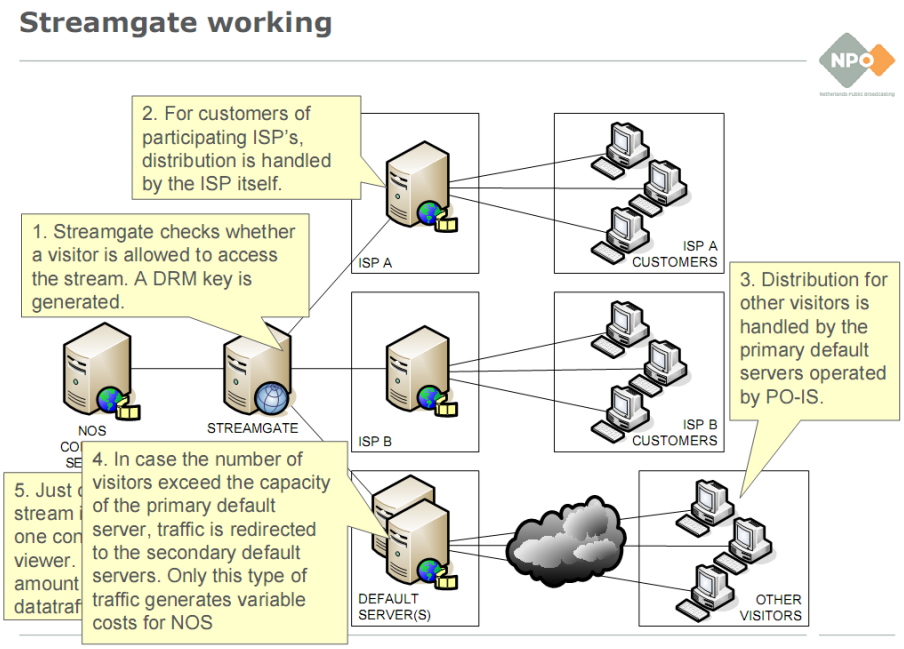


*Fig. 3.4:* Streamgate setup

## 3.8 Summary

In the IJkdijk case, where many streams have to be streamed to a few analyzer systems, multicast would be very effective in reducing the bandwith usage.

# 4. IJKDIJK MONITORING COMPONENTS, OFF-THE-SHELF AND CUSTOM BUILD

## *4.1 Introduction*

In this section, the major components of the IJkdijk are described. These are: the dike itself; the sensors in it; the hardware to sample the sensors and the software to process and transport the samples. The dike itself needs no further introduction, so first the most interesting sensor is discussed: the listening tube(luisterbuis). Secondly, the sample theory and sample hardware are discussed. In the third and forth section several commercial off-the-self sofware solutions for data acquisition are presented.

## *4.2 Luisterbuis*

The IJkdijk is equiped with a listening tube, or luisterbuis in Dutch. This luisterbuis is a horizontal drilled tube, with a microphone on each end, thus making it possible to listen to the dike. The microphone, just before it is installed at the end of the luisterbuis, is more or less visible in figure 4.1(t is covered by blue isolation). The grey box is part of the amplifier. The signal is amplified and send over to a measurement station, about 100 meters further. The luisterbuis itself is the horizonatal gray tube in figure 4.2. For this assignment, the input form the luisterbuis is sampled. The sampling of other sources can be achieved in a similar fashion.

## *4.3 Data Acquisition with National Instruments*

Data acquisition, see appendix A, is the sampling of the real world to generate data that can be manipulated by a computer. Data acquisition or DAQ in short, typically involves acquisition of signals and waveforms, e.g. sound or temperature, and processing these signals to obtain desired information. Most DAQ is done with special DAQ interface boards. For the IJkdijk case, TNO choose the following hardware:

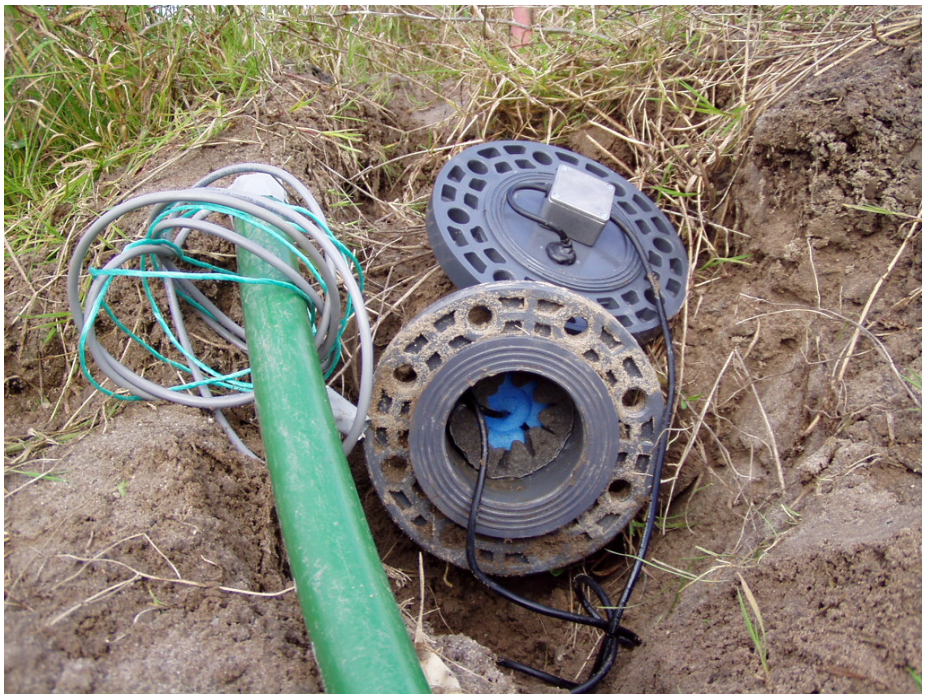

*Fig. 4.1:* luisterbuis microphone

*Fig. 4.2:* Luisterbuis exposed, after a succesfull test

### 4.3.1 Hardware Specifications

| Component | Function |
|---|---|
| National Instruments Controller | Robust casing, PC with: 1.5 GHz Celeron, 30 Gbyte hardrive, 1 Gbit ethernet, USB, serial |
| NI 4461 PXI, a high-end ADC card | 2 Input, 2 Output, 24 bit, 204.8kS/s, $\pm$ 42 V to $\pm$ 316 mV, -20 to 30 dB in 10 dB increments |
| 1 multi purpose card | 16 analog in, 2 analog out, 6 digital in, 4 digital out |

The Controller is a dual boot with Windows XP and a Realtime OS (PharLap). Furthermore, LabView is installed on the Windows partiton. LabView a graphical programming language, very populair for DAQ and analyzation tasks. It is also possible to use other languages for both of these tasks, like ANSI C, MatLab, VB.NET and others. The NI 4461 PXI provides us with 92 kHz alias-free bandwidth, antialiasing and anti-imaging protection filters.

## 4.4 National Instruments Software

For more than 20 years, NI delivers both hardware and software for data acquistion and data processing purposes. They are the market leader in data acquistion. In this section follows a brief introduction of their software options. The most versatile and popular NI software is LabView, which is a graphical development environment. An alternative to LabView is LabWindows/CVI. This integrated development environment combines ANSI C with NI extensions to combine, perhaps already existing, ANSI C code with NI hardware, analysis and user interfaces. A third option is Measurement Studio. It is an addition to Microsoft Visual Studio, which also provides instrument control and analysis tools. Other software which was biefly evaluated during this assignment was DIAdem, a program for managing, analyzing and reporting stored data.


*Fig. 4.3:* National Instruments Data Acquisition System

### 4.4.1 LabView RealTime

In order to run the Realtime OS, the system must boot into the Realtime OS, and can be programmed by deploying an adapted LabView program form another system running LabView, in the same network subnet. The LabView performance of LabView Realtime is the same as LabView running on Windows XP, it is completely dependant on the hardware. It is however realtime, meaning that the timing of the loops and measurements is guaranteed. An advantage is that the running program cannot be interrupted by processes that do run in the background on windows XP (like a indexing server, virusscanner, etc.). A disadvantage is that developing for LabView Realtime is more cumbersome, since after every change, it needs to be redeployed into the target realtime system.

## 4.5 LabView Communication

A major goal of this assignment is to demonstrate the communication facilities of LabView. As it turns out, there are many. We give an overview, ordened to their counterparts in the upper layers of the OSI stack.

| **Layer** | **Function** | **LabView Ecquivalent** |
|---|---|---|
| Application | Network process to application | FTP, SMTP, CGI, Remote control, VI Server |
| Presentation | Data representation | Waveform or array, float or integer, compression, etc. |
| Session | Setup a session between hosts | Authentication, permissions, DataSocket/SVE manager |
| Transport | End-to-end connections and reliability | TCP, UDP, DataSocket, SVE |

## 4.6 Application layer alternatives and considerations

LabView has the ability to send results by FTP, SMPT, etc. However this is not so useful for realtime streaming. The VI server can be interesting for some cases. It exports the LabView GUI (the graphs, charts and controls) to a webserver. The GUI can than be monitored and even manipulated remotely by the use of a webbrowser. This obviously has its applications, but it is no realtime streaming.

## 4.7 Presentation layer alternatives and considerations

The function of this layer is to format the data to be sent across a network. By formatting the data, it can seamlessly be used across different platforms and applications. Encryption, compression and conversion from and to network format also take place here.

The issues concerning the presentation side of streaming solution are mainly how to format the data. There are serveral options to format and store the data captured by a LabView acquisition system.

The LabView dataformats are clearly described, a summary follows in the next session.

### 4.7.1 Basic datatypes

First, there are the basic datatypes:

- **LabView string**. Custom string format, prefixed by string length, ASCI encoded
- **Integers**, 16, 32, 64, always signed
- **Floating points**, single (32), double (64) and extended (depends on underlying system) precision

### 4.7.2 Composed datatypes

The basic types can be concatenated and combined in various ways:

1. Array
2. LabView Waveform
3. Custom data types, composed of fixed combination of basic datatypes

The array speaks for itself. The LabView waveform is a simple extension to an array, it includes a timestamp and a delta t. The timestamp is the time the first sample is measured, the delta t the time between consequetive sampes, also known as the inverse of the sampling frequency. Furthermore, LabView enables the programmer to compose his or her own data types. Not only is it very easy to combine datatypes into new types, it is also clearly documented how this is actually stored and transmitted. This makes it possible to use this feature to transfer data to other application written in a different programming language, e.g. it matches the struct in C.

### 4.7.3 Endianness

LabView stores all its data in a big-endian fashion, which is the way to store multi-byte data on PowerPc. This implies that the most significant byte is stored first. Intel x86 store data little-endian, so when transfering LabView data to an application running on a x86 processer, the endianess needs to be reversed.

## 4.8 Session layer alternatives and considerations

The Session layer traditionally deals with authentication and permissions. Only authorized users should be able to access the system. When using the TCP or UDP sockets in LabView, the only way to regulate access is via a firewall. For DataSocket, the Shared Variable Engine and the VI Server, there are administration tools to regulate access based on ip addresses. Access based on other credentials is not provided by NI.

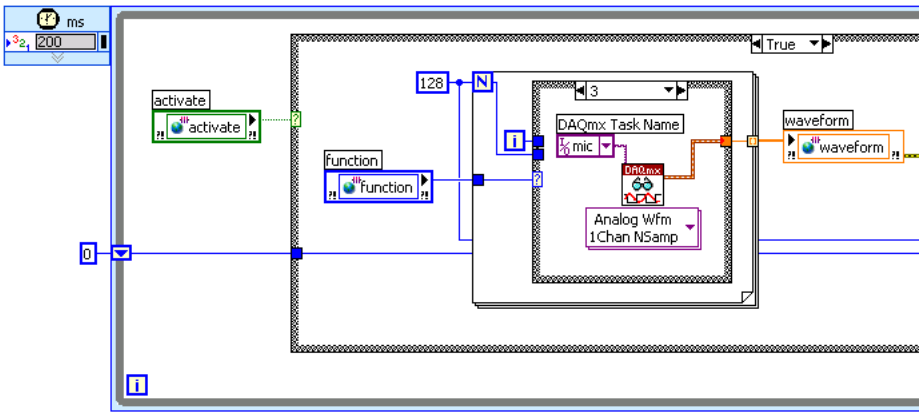


*Fig. 4.4:* LabView DAQ code example

## 4.9 Transport layer alternatives and considerations

The various ways of transporting the data itself is discussed in the following sections.

### 4.9.1 LabView Communcation tests

In order to evaluate the various ways LabView can communicate over the network, a number of testprograms were created. All program consist of at least two loops:

| Loop | Function |
|---|---|
| DAQ | Data Acquisition |
| Network | Distribution of Data |

For simplicity we choose to make the DAQ Loop a *fire-and-forget* loop. The parameters, such as which channels loop rate, sample rate, resolution and buffer-size are set and never changed at runtime.

For the Network loop there are multiple options, as presented by NI:

| Protocol | Speed | Loss | Ease | Security |
|---|---|---|---|---|
| TCP | Fast | Lossless | Difficult | - |
| UDP | Very fast | Lossy | Moderate | - |
| DS | Fast | Lossy/Lossless* | Easy | + |
| VI Server | Slow** | Lossless | Moderate | + |

* For the DataSocket, both buffered and unbuffered reads and writes are available. When using buffered reads, there is no loss, unless there is a buffer overflow. This could be detected by the application, if it choses to implement a detection mechanism. ** The VI Server is only useful for monitoring and inspecting the results of analysation. It cannot be used for real streaming.

### *DataSocket*

National Instruments (NI) DataSocket is a layer on top of TCP/IP which deals with typing and describing the information to be exchanged. The big advantage over TCP is there is no need to setup a TCP server socket and manage the individual TPC sessions. This makes it a lot faster to develop and later change applications. The biggest disadvantage is that the DataSocket libraries are only available to National Instrument programs and the protocol itself is not documented. Although DataSocket is fully supported in the current version of LabView, according to NI it is superseded by the Shared Variables Engine.

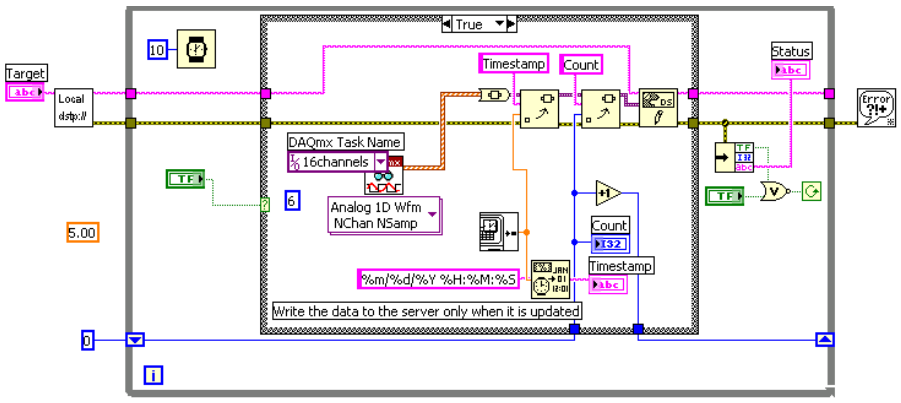


*Fig. 4.5:* DataSocket Server

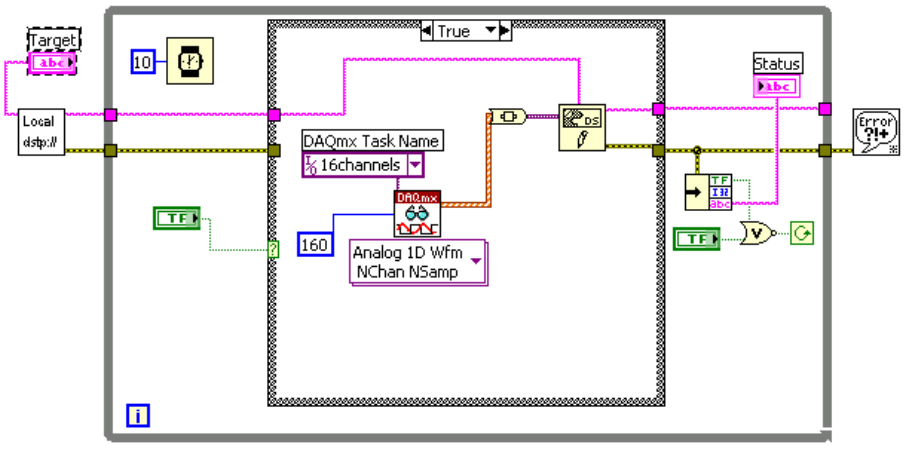


*Fig. 4.6:* DataSocket Client

### *Shared Variables*

The Shared Variable Engine (SVE) is the NI software construction to work with shared Variables. The SVE is build upon NI-PSP, which is a publish and sub-

scribe protocol.

The function and performance is similar to Datasocket, it is however even easier to work with. The idea is that any global variable in LabView code can be adapted to a Shared Variable, which is made available to other instances of LabView. SVE simplifies the process of making an existing application distrubuted by simply converting the releven variables. These variables can be used to represent the acquired data but also for control. Unfortunately, the only configuration that worked, was with a combination of running LabView Realtime and a second of instance of LabView on another computer. The requirement of using LabView Realtime is not documented, perhaps there are alternatives, but none were found during this assignment.

#### *TCP*

Building a protocol and control in LabView. On the website of NEES 2.5 the LabView code is available for their LabView server. It can accept and manage connections and can parse the data. This is not ideal, see Figures 4.7, 4.8 and 4.9.

TCP is very useful to inject data into DataTurbine via their TCPproxy. A possible disadvantage is that the LabView timestamp gets lost in this process. DataTurbine uses the arrival time as timestamp by default. Apparently, there is an ActiveX component for LabView to inject data into DataTurbine, which overcomes this problem. The details of this solution are however not investigated during this assignment.

#### *UDP*

UDP is practical for continuously streaming data, where some of the data can be missed. LabView treats UDP also as a stream. An UDP socket is simple to use in LabView, very similar to the DataSocket interface. The only downside is that handling packetloss becomes the responsibility of the programmer. There are multiple ways to detect packetloss. One option is to send both the samples and their timestamps, so two channels/arrays, one for the samle, the other for the value. Another option is to add an index to each send packet. Comparing the index of the last received packet which the index of the previously recieved packet reveals the loss of one or more packets. It is up to the application and its developers to act upon the detection of loss of information.

## *4.10 Results*

After three months of experimenting with LabView, a streaming solution that fits all criteria was not discovered. Appertantly there is no silver bullet. TNO wants to provide the infrastructure for other parties who will collaborate on the IJkdijk

### *4.10.1 Open endedness*

Both TCP and UDP are clear winners when it commes to open endedness. Via TCP and UDP interfaces it is possible to interact with other application, written

in a language of choise. With Datasocket and Shared Variables this is simply not possible.

Connecting to a TCP or UDP socket is one thing, interpreting the data is another. Building a protocol is not so easy in LabView. A more fruitful approach is to adapt other application to read LabView data. LabView datastructures are straightforward and clearly documented.

The alternatives ordened according to their open endedness:

1. UDP
2. TCP
3. DataSocket
4. Shared Variable

UDP and TCP are very similar when it comes to open endedness. UDP is slightly more easy to use in adapative environments since it does not require sessions. It is easy to extend and UDP application to support Multicast or to integrate it with peer-to-peer or UPVN networking. Furthermore, since UDP applications must be able to handle packetloss, its requirements for the quality of the network can be lower.

Both the DataSocket and the Shared Variables are not very usefull from an open endedness perspective. They are only available for NI products.

### *4.10.2 Easy of use*

The easy of use is a very subjective measure. Still, we would like metrics to compare the different options. A metric could be the time needed to build. Another metric is the effort required to adapt an application to a specific need. Yet another metric, a measurable one, is code size. In LabView however, there are no lines of code. Interesingly, the number of Virtual Instruments (VI) (the icons in the figures) can be seen as a sort of equivalent to lines of code.

Here we order the options according to the number of VI's required to stream DAQ data to another location:

1. DataSocket
2. UDP
3. TCP
4. Shared Variable

Both DataSocket and UDP are straightforwared to use. They require only basic LabView knowledge and their respective examples are easy to understand.

The TCP server socket is not so easy to use, as is demonstrated with the example code in Figures 4.7, 4.8 and 4.9.

The Shared Variable requires a LabView RealTime target and another computer running LabView in order to test with Shared Variables and real acquisitioned data. In other words, two computers, one of them running the RealTime OS, are needed. This is considered very big obstacle toward the deployment of Shared Variables.

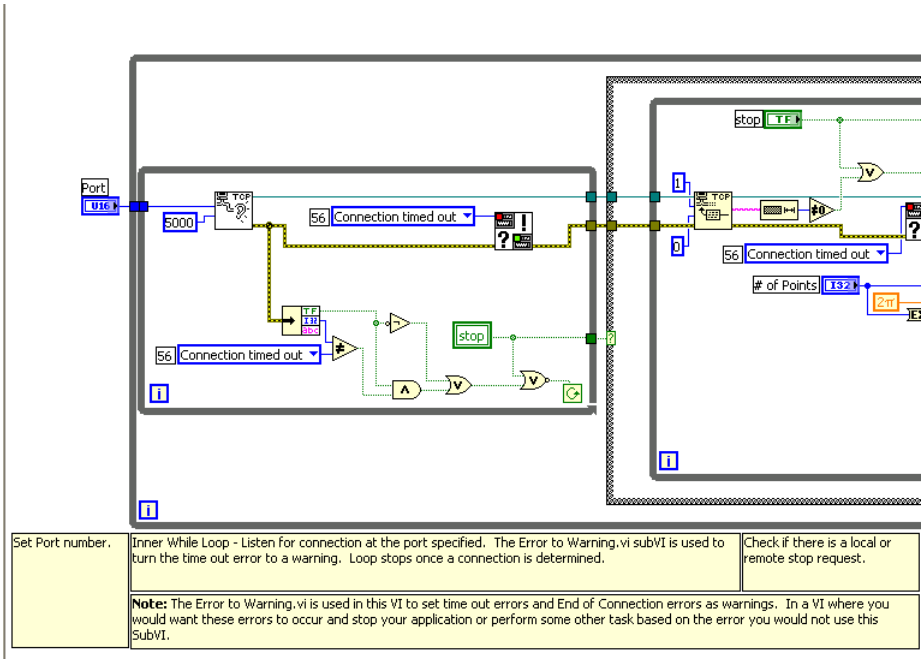


*Fig. 4.7:* TCP sever part 1

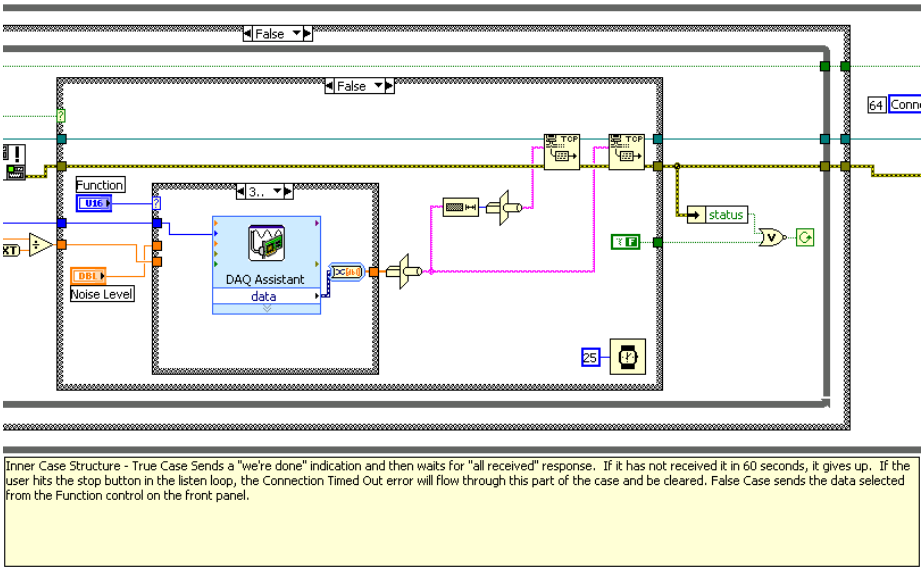


*Fig. 4.8:* TCP sever part 2

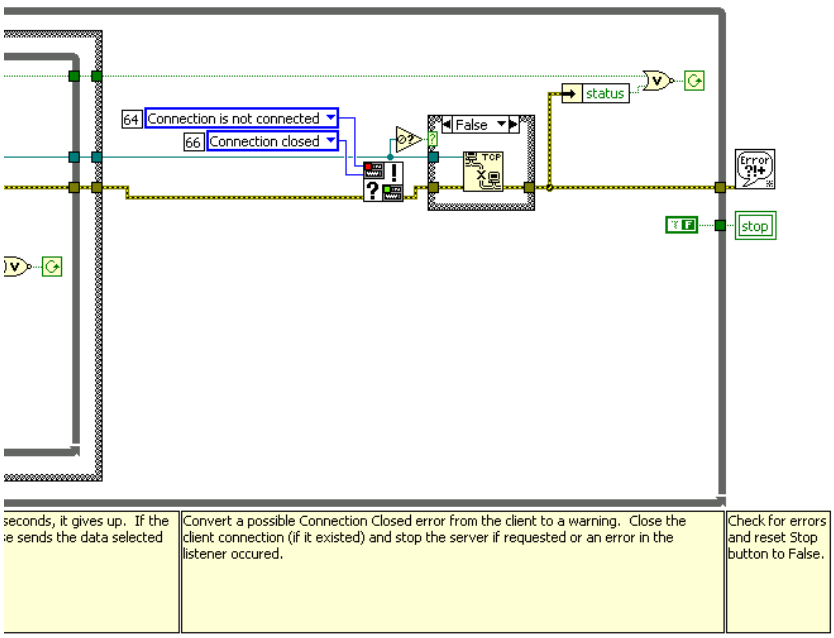


*Fig. 4.9:* TCP sever part 3

| **Method** | **VI's** | **project** | **control structures** | **RT target required** |
|---|---|---|---|---|
| TCP | 7 | - | 6 | - |
| UDP | 3 | - | 1 | - |
| DS | 3 | - | 1 | - |
| SVE | 1 | v | 1 | v |

### 4.10.3 Scalability

With the available hardware there were no bottlenecks to stream all the sampled data (400.000 samples/S) to another location, by the means of TCP, UDP and Datasocket. The tests were inconclusive for Shared Variables. It seems that the performance of Shared Variables has greatly been improved in a newer version of LabView, namely version 8.5.

1. UDP
2. DataSocket
3. Shared Variable
4. TCP

UDP is on top of the list since it is a lightweight transport protocol, support multicast and is easy to proxy/forward. DataSocket has a slightly higher overhead since it is based on TCP. A strong point is that the DataSocket Server can run anywhere. The same point applies to the SVE, the performance of the LabView 8.2 implementaion is poor however. The TPC server socket is lowest on the list, since it required that all the load is on the LabView box which is also reponsible for the DAQ.

## 4.11 Summary

For the IJkdijk realtime streaming scenario, four options remain: TCP; UDP; Datasocket and the Shared Variable Engine (SVE). The advantages of TCP and UDP are flexibility and open endedness. The disadvantages are that the details of managing sessions need to be resolved by the LabView programmer. These details are handled by DataSocket and SVE. However, the later two are only available for application build with NI software.

For cases where there is streaming to LabView stations is required, DataSocket is advised. The DataSocket Server should not run on the IJkdijk, but on system in a well connected place.

For streaming to other software, both UDP and TCP have their uses. UDP is used in the UPVN prototype described in the next section. It should be used in the cases were some packetloss can be tolerated. Care should be taken to detect this packetloss in the application, either by using a seperate channel for timestamps or by indexing each packet and calculate the elapsed time by inspecting the received index with the previously received index.

TCP should be used when packetloss cannot be tolerated. It is adviced to build a LabVew TCP *client*, which then connects to TCP server, this server should be developed with a suitable tool.

For all realtime streaming options, transferring a LabView Waveform or an one dimensional array of waveform is suitable.

# 5. UPVN

## 5.1 User Programmable Virtualized Networks

The goal of User Programmable Virtualized Networks (UPVN) is to enables programmers to program a network. The network components, such as routers, switches, interconnects and even the hosts connected to the network, should be programmable as a whole.

Currently, configuration is the domain of network operators. They define topology, type of service and other parameters. The configuation must be applied to each component individually, often by the means of a command line interface. Adding a service, such as IP Multicast implies a lot of work. An UPVN program can work as a template, create the program once, then deploy it simultaneously on all relevant network components.

User Programmable Virtualized Networks (UPVN) is a network concept which abstracts the network in such a way that an application programmer can add services to the network. With the use of tokens, which are added to packets, it is possible to build a practical security and AAA framework, see Section 2.3 of [5].

UPVN research is about how to add services to he network by programming it. Currently, when a new service is required e.g. resource reservation, a new protocol has to be developed, described, implemented, accepted and deployed. Current research is investigating ways to enable network components and the network as a whole to adapt there functionality on-the-fly. Results of the UPVN research are described here[5].

## 5.2 DataRouter, an UPVN application

At TNOs lab, an early UPVN test setup is available for experimentation. It consist of ten FreeBSD systems, which are all connected to to or three of the others systems, in order to enable different network topologies between the hosts.

For this assignment a simple datarouter was implemented to run on the UPVN setup. The idea is that LabView will stream some data into the UPVN network, which will then route it to several other hosts, running LabView or another application. The route or routes the streams will follow through the network can be programmed.

### 5.2.1 Tokens

The current state of the UPVN prototype is not yet suitable to do routing on basis of tokens on the IP layer. Therefore, we implemented a router that forwards

on basis of tokens in UDP, the transport layer. When token routing is implemented on the IP layer, we can delegate the routing to routing logic UPVN provides.

### 5.2.2 PERL

In order to create a rapid prototype, which can run on diverse hardware and software, the PERL programming language was chosen. Advantages of PERL are: portability; no need to recompile since it is a scripting language and powerful means to handle pattern matching. Disadvantages are speed, readability of the code and, as is discovered during the prototyping, weak support for SOAP (no assistance in creating WSDL descriptions). The README of the application is included as appendix B.

### 5.2.3 Route data

The datarouter operates on UDP datagrams, possibly tagged with a token. For this implementation, a four byte token at the beginning of the datagram payload is used. Either the application producing the datagram inserts the token, or the data router does so, based on the source IP address. The datarouter keeps a routing table. The table matches tokens to destionation IP address. By setting these tables on datarouters running on several, internet connected hosts, an overlay network can be build.

The performance will never be very high. Since the datarouter is currently an application, not a kernel extension, it runs in *userspace*. One of the implications is that packets need to be copied back and forward in memory, which is not efficient.

### 5.2.4 Multicast

An interesting feature of this approach is the easy to create multicast paths. Since the destination (or next hop) can be set on each of the intermediate hosts, it is also possible to set multiple destinations. Since the UDP packets are stateless, it can be received by multiple hosts.

### 5.2.5 Data manipulation

Another interesting application is the (conditional) manipulation of data. Consider a toplogy where some links are saturated or otherwise limited in capacity. If multicast is used, the bandwith use is the same for each link in the multicast tree. If the bandwith is insufficient on one link, there is packetloss for every receiver behind the link. One possibility is to reduce the bandwith needed by compressing the data. This then impacts all the receivers of the multicast stream, also the ones which do have sufficent bandwith.

With this UPVN setup it is also possible to do the compression only for the links where it is really needed. In this case, the data send by LabView was as simple as possible: an array of 32 bit floats. The datarouter was extended so that it cannot only forward but also send the packets through an application filter. In order to keeps things as simple as possible, the packet is placed in a file, the file is processed by an application which places its results in another

file, which is then read by the datarouter and forwarder to its next destination. While there are better ways to do interprocess communication, this seemed the most pragmatic start. The result is that it is possible to filter the array of floats through SOX, "the swiss army knife of sound processing programs". SOX is an open source command line tool to process sound. One of its capabilities is to resample. So, in a case where there is insufficient bandwith on a specific link, UPVN can then resample to a lower rate, on the last hop before the constrained link.

## 5.3 Summary

While UPVN is in an early stage of development, some of the ideas seem to fit on the IJkdijk case. It is possible to use UPVN concepts in combination which Data Acquisition with LabView. A simple prototype was developed to illustrate some of the possibilities.

# 6. CONCLUSIONS AND RECOMMENDATIONS

Here we present the conclusions of the research.

## *6.1 Research questions*

The main question is: *"How do we stream and multicast audio data from the IJkdijk to several internet attached analysis systems?"*

More refined questions are:

1. Are there compareable cases tot the IJkdijk case? If any, What are there solutions?
2. What is the current state of the art in streaming and multicasting?
3. Can off the shelf (OTS) solutions, such as LabView or Windows Media be deployed? If yes, how?
4. Can User Programmable Virtual Networks (UPVN) be deployed? If yes how?
5. What could be a suitable development path for TNO's data acquisition and data publishing setup?

## *6.2 Conclusions*

1. (a) Similar cases dealing with the distribution of seismic data use a hierarchical setup. A top node is discovering data sources. Data sources discover sensors. The sensor data is then stored in ring buffers. TCP/IP (the internet) and a custom transfer protocol are commonly in use. Interfaces deploy both websites and web services. The network infrastructure is treated as a commodity; its either already in place or it's bought and installed. See section §2.7

2. (a) IP Multicast is commonly enabled on backbone networks, educational networks and inside corporate networks. It is not common on the network of ISP's for home users, nor is it commonly enabled on the borders of corporate networks. §3.1

   (b) IP Multicasting is not widely used in a multi domain setup. Since the IJkdijk setup is a multi domain one, IP multicast is not a suitable solution. See §3.1

3. (a) A National Instruments Data Acquisition system (DAQ) is a suitable COTS DAQ alternative for the IJkdijk setup, since it provides all the

sensor reading and networking facilities required to stream the data from a few sensors to a few remote analyzers, see section §4.10 .

(b) TCP, UDP and DataSocket are all viable solutions to stream sensor-data. Which one to use, depends on the exact requirements, see §4.10

(c) MS Media is not a suitable COTS DAQ alternative for the IJkdijk setup, since it requires a soundcard as input device, see section §3.6.4

(d) Combining NI DAQ with a COTS streaming solution like MS Media, since it requires a custom converter/interface, which is relatively complicated to build, see section §3.6.4

4. (a) The token based routing and QOS networking, such as proposed in research UPVN's, provide interesting concepts for designing an IJkdijk network setup. A simple prototype was build, demonstrating UPVN principles applied to audio streams, see §5.2

(b) Building a simple streaming multicast application in UPVN is simple and straightforward. It is unclear wether or not UPVN can assist in the management of a very large number of streams, see §5.2.4

5. Streaming data with LabView to a few internet attached analysis systems can start as soon as an internet connection is available at the IJkdijk location.

## 6.3 Recommendations

- Study the effect of lossy compression on scientific measurement data.
- Evaluate The National Instruments LabView Datalogging and Supervisory Control (DSC) Module.
- Evaluate The National Instruments LabView Shared Variable Engine version 8.5.
- Evaluate DataTurbine.

# BIBLIOGRAPHY


[1] Jaap D. Bregman, Gideon W. Kant, and Haitao Ou. Multi-terabit routing in the lofar signal and data transport networks. *Experimental Astronomy*, 17, 2004.

[2] Tricia Gill and Bill Birney. *Microsoft Windows Media Resource Kit*. 2003.

[3] Andr Gunst, Kjeld van der Schaaf, and Mark Bentum. Core station-1, the first lofar station. *Arxiv preprint astro-ph*, 2006.

[4] Teruyuki Kato, YukihiroTerada, Masao Kinoshita, Hideshi Kakimoto, Hiroshi Isshiki, Masakatsu Matsuishi, Takayuki Tanno, Akira Yokoyama, and TakayukiTanno. Real-time observation of tsunami by rtk-gps. *EarthPlanetsSpace*, 52:841–845, 2000.

[5] Robert J. Meijer, Rudolf J. Strijkers, Leon Gommans, and Cees de Laat. User programmable virtualized networks. *e-Science2006*, 2006.

[6] Arcot Rajasekar, Frank Vernon, Todd Hansen, Kent Lindquist, and John Orcutt. Virtual object ring buffer: A framework for real-time data grid. In *HDPC Conference*, 2004.

[7] SURFnet/NPO. www.surfnet.nl/info/attachment.db?190406.

[8] Kjeld van der Schaaf, Chris Broekema, Ger van Diepen, and Ellen van Meijeren. The lofar central processing facility architecture. *Experimental Astronomy*, 17:43–58, 2004.

[9] Wikipedia. Ijkdijk. Website, July 2007.

[10] Wikipedia. Indian ocean tsunami warning system. Website, July 2007.

[11] Ning Xu. A survey of sensor network applications. *Communications Magazine, IEEE*, 2002.

# APPENDIX

# A. DATA AQCUISITION

The scope of this document is to evaulate the possible ways to transport data from the IJkdijk location to involved parties on different places over the world. In order to do so, a clear understanding of the data to be transported is needed. As a start we distingish two types of data, as shown in table A.

## *A.1 Theory of Data Acquisition*

In a sense, audio and video are sensor data too. Most sensors are electrical, their output is a voltage, which changes when the measured quantity changes. In the current fase of the IJkdijk project, TNO wants to know the suitability of LabView as well as Windows Media for acquisition of audio, video and other sensor data.

Data acquisition is the sampling of the real world to generate data that can be manipulated by a computer. Data acquisition or DAQ in short, typically involves acquisition of signals and waveforms, e.g. sound or temperature, and processing these signals to obtain desired information. Most DAQ is done with special DAQ interface boards.

### *A.1.1 Sampling theory*

**Sampling** is the measuring of the voltage at fixed point in time. **Uniform sampling** is measuring the voltage at a relugar interval. The time between each measurement is often specified as **sampling interval**, Delta t, $\Delta$t or dt. The inverse of the sampling interval is the **sampling rate** or $f_s$. Given a uniform sampling rate of $f_s$, the highest frequency that can be represented is $\frac{f_s}{2}$.

So according to the Nyquist sampling theorem **refTomasi** the sampling rate should be at least twice the maximum frequency component of the signal of interest. In other words, the maximum frequency of the input signal should be less than or equal to half of the sampling rate. These frequencies above the Nyquist frequency may then alias into the appropriate frequency range and thus give erroneous results.

| **Data Characterization** | **Examples** |
|---|---|
| Sensor data, unprocessed | Water pressure, geophone, seismometers, displacement, wind, rain |
| Multimedia, processed | audio, video |

*Tab. A.1:* IJkdijk Data

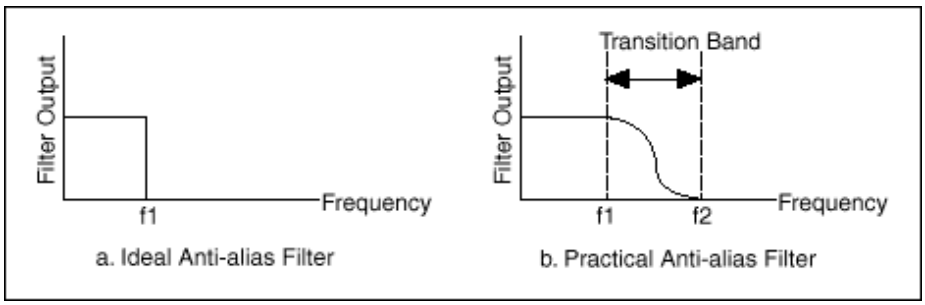


*Fig. A.1:* Anti-Alias filters (source National Instruments)

To preven this from happening, a low-pass filter is used to limit the signal. This filter is an anti-alias filter. Unfortunally, there is no perfect low-pass filter. Ideally it should fuction as a brick wall for frequencies higher than the desired input frequencies. A.1 shows the difference between a perfect and a more realistic anti-alias filter. Note the transition band, the frequencies between the desired cut-off frequency $f_1$ and the actual cut-off frequency $f_2$.

With digital filters it is possible to filter the remains of the high frequencies that pass through the analogue anti-alias filter. However, for this to work correctly, we need to select a sample rate such that $\frac{f_s}{2}$ is after the transition band. A 33 % ratio for the transitionband is simple and cheap, e.g. phones use 8 kHz sampling for a 3 kHz passband. a 10 % ratio is tougher but realizable. CD's use 44.1 kHz for a 20 kHz passband. Surprisingly, the reason that it is 44.1 kHz and not 44 kHz has nothing to do with the requirements for a anti-alias filter. The explanation is that 44,100 is a multiple of 60 (American television field rate), 50(European television field rate), 30(American television frame rate) and 25(European television frame rate).

### *A.1.2 Pulse Code Modulation*

Pulse Code Modulation or PCM is a digital representation of an analogue signal. The conversion between analogue and digital is performed by a Digital-to-Analogue converter (DA-converter). The performance and quality of a DA-converter is expressed in it's bitrate and sample rate. For audio the number of bits per sample, values between 8 and 24 bits are common, for samplerates 8 kHz to 96kHz. When storing or streaming this data, the bitrate is important. This is the number of bits multiplied by the samplerate. For example: CD quality audio is 2 channels, 16 bits, 44.1 KHz, this makes 2 x 16 x 44100 = 1.411 Mbit/s = 176 Kbyte/s.

# B. UPVN DATAROUTER README FILE

== Forward Interface

This is the documentation of fi.pl, the Forward Interface for the UDP datarouter.

== Dependancies perl SOAP::Lite fi.pl datarouter.pl

== Start the Forward Interface

—

```
$ ./fi.pl [portnumber]
Contact to SOAP server at http://edge.ict.tno.nl:1080/
```

—

The default SOAP port is 1080. When running without superuser privileges, the portnumber must be higher then 1023.

The datarouter itself is started by a SOAP call. Example in Perl:

—

```
#!/usr/bin/perl -w

use SOAP::Lite;
my $soap = SOAP::Lite
  -> uri('http://localhost/UDPForwarder')
  -> proxy('http://localhost:1080');

my $result = $soap->start('1235');

  unless ($result->fault) {
    print $result->result();
  } else {
    print join ', ',
      $result->faultcode,
      $result->faultstring,
      $result->faultdetail;
  }
```

—

The parameter of start is the UDP port where the forwarder wil operate on. The destination port is configured by the token rules. When running without superuser privileges, the portnumber must be higher then 1023.

== Configuration By default, the UDP forward daemon will drop all packets. Only when a there is an approriate token rule, it will forward to the destination from that rule.

Get the current rules, newline separated: —

```
$result = $soap->getIPRules();
$iprules = $result->result();

$result = $soap->getTokenRules();
$tokenrules = $result->result();
```

—

On the ingress nodes, there should be 1 or more IP rules. The format of the rule is: ”ipaddress: 4-char-token”:

```
$result = $soap->lockTable();
$secret = $result->result();
$soap->addIPRule($secret, '139.63.89.23: rood');
$result = $soap->unlockTable($secret);
```

On all nodes, there should be 1 or more token rules. The format of the rule is: "4-char-token: bool-strip-token ipaddress port ipaddress port ..." When the bool is non zero, the token will be stripped before sending to the destination. It is possible to add multiple ip/port tuples. When adding a rule for a token already in place, the old one will be overwritten. It is possible to avoid this by requisting the current tokens first and then pick another token.

```
$result = $soap->lockTable();
$secret = $result->result();
$tokerule = 'rood: 0 139.63.89.35 1235 139.63.89.36 1235';
$result = $soap->addTokenRule($secret, $tokenrule);
$result = $soap->unlockTable($secret);
```

It's only possible to remove all rules, possibly adding some of the removed rules afterwards:

```
$result = $soap->lockTable();
$secret = $result->result();
$result = $soap->clearTokenRules($secret);
$result = $soap->clearIPRules($secret);
$result = $soap->unlockTable($secret);
```

== Transactions

A transaction is a safe modifaction of the network. The goal is to maintain a safe state for the network as a whole. When adding a new path to the network, this should be added to all relevant nodes or none.

Transactions are currently implemented by the means of a lockTable and an unlockTable soapinvocation on all relevant nodes. When it is not possible to obtain all requisted locks, they should all be released. This is the responsibility of the application that want's to add a path.

in short, a succesfull transaction:

```
$node1->lockTable()
#save secret1, check result, ok
$node2->lockTable()
#save secret2, check result, ok
$node1->addTokenRule($secret1, $tokenrule1);
$node1->addTokenRule($secret2, $tokenrule2);
$node1->unlockTable($secret1)
$node2->unlockTable($secret2)
```

in short, an unsuccesfull transaction:

```
$node1->lockTable()
#save secret1, check result, ok
$node2->lockTable()
#check result, not ok
$node1->unlockTable($secret1)
#report failure, try again after some time, so another application
#can complete or cancel it's transaction.
```