3 Design

This section will introduce the design issues that were encountered during the initial designing of NTS. The goals of the simulation will be exposed, as well as the general design constraints kept in mind throughout the implementation of NTS. The different techniques used and the restrictions for each section of NTS follow.

3.1 Aim of the project

The aim to the project is to implement a variable simulated static interconnection network on a switch network, and implement the basic communication operations for this simulated interconnection network so that the effects of various network topologies on parallel processing algorithms can be investigated.

3.2 Goals of NTS

- Implement a software solution for simulating different network topologies on a switch network,
- The implementation will rely on PVM for its communication,
- Provide client code with an overloaded PVM library to use the simulated network,
- To the client code, the simulated network will be transparent, only the effects will be felt,
- The network will be dynamic, i.e., its layout can be built in an off-line mode, and still be edited while on-line,
- Principles of parallel computing will apply for the simulated network,
- Only minimal changes to a PVM program should be made for it to be able to use the NTS simulated network,
- Design for extensibility,
- A user friendly GUI will be made for the client to build the NTS simulated network and set options,
- A general help will be at hand to consult for the workings and options of NTS,
- The network will be robust.

3.3 Principals Behind NTS

Throughout the design and construction process of NTS, some basic principals and design constraints were always kept in mind. This section enumerates these, plus their consequences, mostly in the form of constraints to the use of NTS.

3.3.1 Development Language

PVM, as stated before is the communication library that will be used throughout the project. PVM is available in Fortran and C. With the addition of having the Qt GUI development toolkit available, the choice of using C++ for the development of NTS was made.

3.3.2 Deadlocks

Because of the use of PVM as the base for communications, whether internally or the proposed overloaded PVM library for clients, there will be no deadlocks in NTS communication. PVM comes deadlock free, and therefore through legacy so will NTS. This of course does not exclude that client code running on NTS may have deadlocks of their own creations, such as two clients each "listening" for a message coming from the other.

3.3.3 Equivalency with PVM

With the use of PVM internally, an overloaded PVM library is proposed to clients to access the simulated topology of NTS. This library should behave as the PVM library, with only minimal code changing for the client to convert his program from using one to the other. No configuration of NTS should be required within the client code, this should be done via the separately provided program controlling the NTS simulated network and options. The connection to the NTS simulated network should be transparent and automatic. The NTS network, as a whole should behave as a "network cloud".

3.3.4 Routing

In a store and forward network, such as the one being simulated by NTS, one of the main issues is message routing. NTS uses its own pseudo-routing algorithm based on RIP[6]. Refer to the section 3.5 ntsRouter for details of this algorithm.

[6] RIP: Routing Information Protocol

3.3.5 Dynamic Update Network

The network topology simulated by NTS is to be dynamically modifiable. The network design can be laid out in an offline mode and still modified while "running" (state in which the simulated network can accommodate client activity and communication). All modifications to the network layout should be possible in either mode, be it adding/removing nodes or links. The routing tables present in nodes of the network are to dynamically update themselves so that no routing loops or holes are present on the network. More information is present in the ntsRouter section.

This dynamic modification of the network can be used to simulate the responsiveness of the network to a node crash. All building/modification of the network is to be carried out by the specific program designed for these purposes, the ntsMaster, see section 4.6 ntsMaster.

3.3.6 Parallel Processing Network

The simulated networks of NTS being primarily used to simulate a parallel computer, such as a MPP, the principles of such internal networks are to be adopted while conceiving NTS. The main idea implemented is the fact that in a constant network as is found in a MPP or other parallel machine, all messages are known to have a final destination. Messages will therefore always reach this destination. No precautions are then taken to check how far a message still has to go or how long it currently has taken to get there. This is different from other types of network types, such as IP, which automatically delete messages that have been on the network too long, knowing that TCP will request a retransmission. In NTS, a message, which has a legal NTS destination, will keep on being forwarded, even if a path to the destination cannot be found, due to a simulated "crash" or other.

3.3.7 Usage on Network

NTS simulates a store and forward static network. NTS is supposed to be used over an all-to-all PVM network. Using NTS over a static type network or a coaxial network would bring no investigational benefits, though NTS would still run on such networks.

3.4 Definitions

- **Master**: program to setup the NTS simulated network, knows the TIDs of elements on the network,
- **Node**: any program the Master knows the TID of, i.e., Routers and Clients,
- **Router**: message forwarding program, one per computer, has a routing table with which is dynamically updated to contain all Routers and Clients on the network,
- **Client**: a program which connects to the simulated network using the overloaded PVM communications library,
- **Link**: interchange of TIDs between two nodes.

3.5 ntsRouter

The ntsRouter program is the "forwarding" daemon, which resides on each computer that is to act as a node on the NTS simulated network. Only one daemon is to be loaded per computer participating in the simulation. The ntsMaster program provides the instantiation of the ntsRouters. The ntsRouter, once setup, is an infinite loop, listening for incoming messages and processing them. Incoming messages may come from:

- **ntsMaster**: for building the initial configuration, for a dynamic change, or a change of options,
- **Client program**: Clients use the overloaded PVM communications library to connect and use the NTS simulated network, client messages are either clients wishing to join or leave the NTS network, or messages destined for other NTS user clients that have to be forwarded to the final destination (i.e., the routing process),
- **ntsRouter**: from another Router in the network to exchange routing data (i.e., a Routing Protocol is used to keep the routing tables up to date).

3.5.1 Message Forwarding Logging

For each forwarded message, the ntsRouter can log the forwarding and send the detail to the ntsMaster where it is collected for analysis. See the ntsMaster for more

details. One of the parameters of the logging is the time of forwarding the message so that the total message forwarding time through the simulated network can be calculated. For this timing to be accurate, the computers on the network must synchronize. PVM provides a facility to calculate the difference between two computer clocks, and therefore be able to synchronize them.

3.5.2 Routing Protocol

As mentioned earlier, every network has to have a routing protocol. This section describes NTS's routing protocol, which was initially implemented by Olivier Thonnard in the first incarnation: AOM. The routing protocol designed by Thonnard is such that only minor updates were necessary before it was ready for inclusion in this updated version that is NTS.

Every router and client is identified with a TID (Task Identifier) generated with PVM.

NTS uses its own Routing Protocol based on the principles of **Vector-Distance Routing Protocol**. A Vector-Distance Routing Protocol (like **RIP**, which is the simplest one) characterizes every router with a **metric**, which can be seen as the distance to go to a certain destination on the network (Vector = what is the next hop to reach the final destination, Distance = still how far it is). The metric can be very simple like a number of hops (used in RIP), but enhanced Vector-Distance Routing Protocols nowadays use composite metrics to allow every link to be precisely characterized with several parameters, like the bandwidth, the delay, the load, and the reliability. An example of such a protocol is IGRP[7], which is a Cisco proprietary Routing Protocol. IGRP uses a formula to calculate the composite metric:

$$Metric = \left(K1 \cdot Bandwidth + \frac{K2 \cdot Bandwidth}{256 - load} + K3 \cdot Delay \right) \cdot \frac{K5}{reliability + K4}$$

where the K values are constants that can be adjusted from the router console to best tune the routing process.

In a first approach, the focus on the simplest routing protocol was adopted; the metric that was used is just the number of hops (which is in fact equivalent to the

[7] IGRP: Interior Gateway Routing Protocol

"Manhattan distance"). Hence, we considered every link had the same "weight" (so all links are equivalent).

The Routing process is not a simple problem to resolve. For example, we must avoid routing loops, which can occur when two Routers are sending and receiving updates more or less at the same time while a link becomes unreachable. A routing loop can lead to the "count-to-infinity" phenomenon (which as stated in the principles followed should not happen as the network is not capable of dealing with such a case as no deletion of messages is possible) and results anyway in inconsistencies in the routing tables of the Routers.

To resolve those routing problems, routing principles from RIP were taken:

- **Split Horizon**: a Router can never send routing information (i.e., a route) back to the source from which it learned it,
- **Route Poisoning**: when a link falls down, this destination is directly set to "unreachable" in the routing tables of the attaching Routers. We say that the Routers "poison" the route, i.e., they set the path cost to an infinite value (which is 16 for RIP) and send this information so that the update propagates in the network. Of course, one can see that the maximal value for the distance (the "infinite" cost) will determine the **maximal diameter** of the network. The value of 100 was used as an infinite value, which corresponds also to the maximal number of hops,
- **Dynamic load balancing**: like in the RIP protocol, NTS implements a process that allows a packet-to-packet load balancing. When there are several routes to the same destination with the same path cost, then the Router will use alternatively these routes to send the packets. The decision of supporting load balancing for a maximum of ten alternative routes, which corresponds already to a quite large network.

The RIP Protocol has some more features to guarantee the reliability of the routing process, like Poison Reverse, update timers, flush timers, and triggered updates. Normally, RIP specifies that a Router has to respond to a Poison message with a "Poison Reverse" message (that overrides the Split Horizon principle), which is considered to be

the acknowledgment of the Poison message. This was considered none necessary, as it is redundant information. About the timers, no timers were used to avoid the overloading of the network. RIP exchanges routing information every thirty seconds by default. In the NTS implementation, only when a change occurs, will a Router generate update messages.

3.5.3 Router Example

As an example, the following figure (Figure 14) shows a Router object with four neighbors (based on Figure 8: 3-by-3 2-D mesh). Hence, the Routing Table resulting of the initial configuration and of the dynamic exchange of data is then represented (after all updates have been executed) in Table 1.

On this example, a Client code (Client#2) has "emerged" on Router 5, which propagated this information through its neighbors. Hence, there are also two possible routes to reach Client#2: via 2 and via 4 (with a number of hops equal to 3). When Router#1 receives the update (coming from 2 or 4), the first time a route is added in an empty slot, the second time the route is updated (put an alternative route in the array *Next_Hop*). In both cases, the status of the route is set to "changed" (i.e., "1"). Every route, whose status is equal to 1, will be handled when calling the function to send updates will be called. The different **Messages** (class containing an update) to be sent are then computed and the function returns a **Packet** (class containing all the updates needed to be propagated at this time) of Messages.

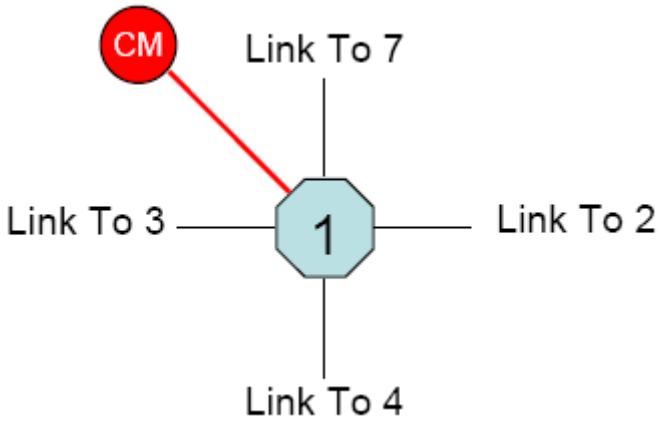

Figure 14: Detail of network shown in Figure 8

Destination	Path Cost	Next Hop	Sequence	Status	LinkOrClient
1	**0**	**[1 0 0 … 0]**	**0**	**Unchanged**	**Link**
2	1	[2 0 0 … 0]	0	Unchanged	Link
3	1	[3 0 0 … 0]	0	Unchanged	Link
4	1	[4 0 0 … 0]	0	Unchanged	Link
7	1	[7 0 0 … 0]	0	Unchanged	Link
5	2	[2 4 0 … 0]	0	Unchanged	Link
8	2	[2 7 0 … 0]	1	Unchanged	Link
6	2	[3 4 0 … 0]	1	Unchanged	Link
9	2	[3 7 0 … 0]	0	Unchanged	Link
Client#1	1	[Client#1…0]	0	Unchanged	Client
Client#2	3	[2 4 0 … 0]	2	Changed	Client
Empty(= 0)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
Empty	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*

Table 1: Example of a routing table (Router#1)

3.5.4 Message Processing

Due to limitations with PVM communications explained in the section 3.6 ntsLibrary, an ntsRouter must process the message tag of incoming messages to extract the required information. In most cases the final destination TID is calculated and then the "next hop" for this final destination is looked up in the Routing Table. The "next hop" having been determined the message can be forwarded on.

3.5.5 ntsRouter Options

ntsRouter options can be set via configuration messages sent from the ntsMaster program. Options include: printing of Router messages to the default PVM set standard out, printing the current routing table, logging of forwarded messages, working in "silent" mode (where the ntsRouter just works quietly), and modifying the network dynamically.

3.6 ntsLibrary

The ntsLibrary is the communication gateway between client code and the network simulation. It is based on overloading the PVM communication library. It is designed for easy interchange between an existing PVM program and using NTS: convert all the function calls from "pvm" to "nts", and the program will be connecting and communicating through the NTS simulated network.

3.6.1 Connecting

Initially, the program using the NTS Library, hereafter known as the Client, must connect to the NTS network via the local Router installed on his running computer to be able to use the communication functions of the network cloud being set-up. To maintain transparency of the NTS layer, and to keep the same interface as PVM, the NTS network connects automatically when any of the overloaded functions is called, the most popular one being: *nts_mytid()* which like its PVM counterpart returns the TID identification assigned to the Client program.

Using the PVM lookup table, the Client can lookup the Master's TID and therefore contact the ntsMaster program asking for the TID of its local Router. Once the Master replies, and that there is a local ntsRouter for the computer running the client code, the Client may then contact the local ntsRouter to subscribe to the NTS network. The local Router getting a request from a Client to join will add an entry for it in its routing table and send a notification to the Master before sending a connected message back to the Client. The Router will then initiate the propagation of the new destination throughout the NTS network. The Client is now connected to the NTS simulated network, and can use the full range of PVM equivalent functionality while seeing no visible effects compared with PVM other than experiencing the effects of the simulated network topology it has just joined.

3.6.2 Settings

Only one connection to an NTS network per Client is permitted therefore upon connection a single settings class is created using the singleton pattern to hold all the necessary settings for the functioning of the NTS overloaded library. Using the singleton pattern ensures that no other connection can be created. The pattern also ensures that connection does happen the first time any of the overloaded functions is called. No parameters are required nor desired from the Client for these settings, the NTS layer initialises itself on its own and a separate program, the Master, is in charge of any configuration.

3.6.3 Message Forwarding

The main function of the NTS layer is to take messages being sent by the Client and forward them through the simulated network rather then directly sending it to the final destination. For this forwarding to be able to take place a next hop parameter has got to be included into each message. PVM only uses two parameters to send a message: destination TID and message tag. The original packed message not being able to be edited for risks of compromising its integrity for the client code, and the destination field being predetermined for the destination TID, the additional information required must be concatenated within the last possible field: the message tag. The next hop having to be the destination, it is in fact the final destination and the original message tag, which have to be compressed into the message tag field. This compromises a loss of information, but the message can now be forwarded from Router to Router, each Router looking in its routing table for the next hop, until finally it arrives at its final destination.

3.6.4 Receiving Forwarded Messages

The receiving of messages in PVM is also done via two parameters, a source TID and a message tag. The original message tag having been concatenated into the new sent message tag, it is not a problem to extract this from any incoming messages. The message though originates when it is received by the destination Client not from the sender Client but from the last Router that forwarded the message through the network. The original source TID must therefore also be included in the message for proper reception.

3.6.6 Sequence

Once you have routing in a network you have sequence. Because of the use of packet-to-packet load balancing inside the network, how can one ensure that message A being sent before message B to the same destination arrives before message B all the time, while different paths through the network may be taken. NTS uses message sequence and end-to-end message reorganisation to ensure that messages are received in the correct order. For each message sent to a TID or received from a TID, the NTS layer keeps a sequence and is thus is able to determine which next message from a certain TID

is supposed to be received. This infers that the sequence must be included with the message as well. Figure 15 illustrates this question.

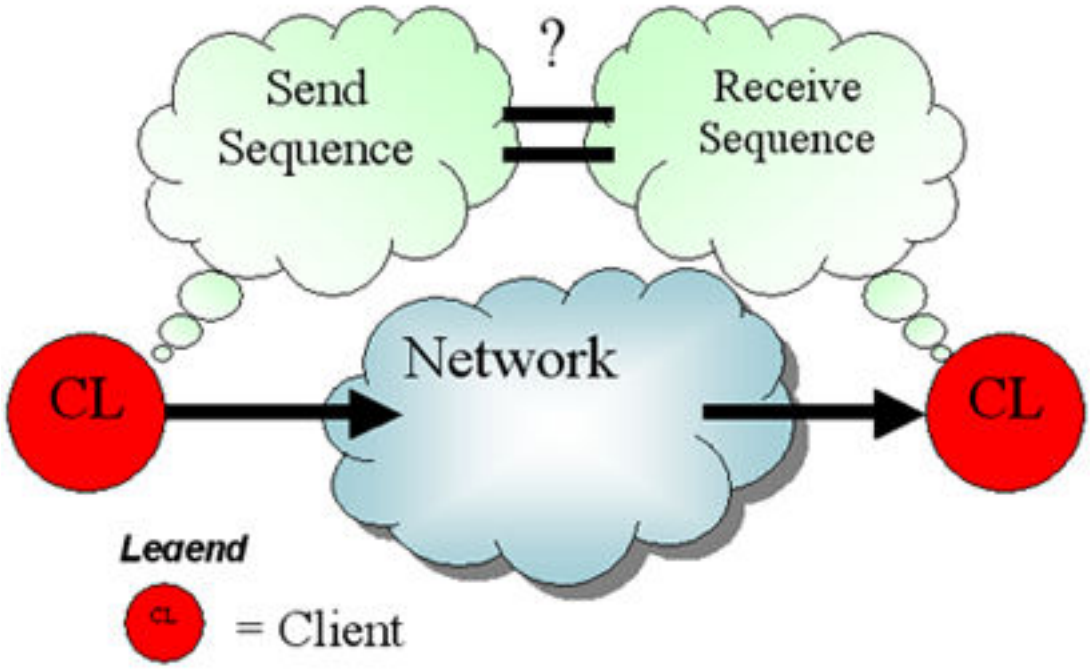

Figure 15: The AOM layer keeps the send and receive sequence for each TID to which a message was sent/a message was received from and checks the two to make sure they are the same for reordering

3.6.7 Integer Combination

Four parameters, each integers: a destination TID, a source TID, a message tag, and a message sequence, need to be inserted into a message being sent for the NTS network to function properly. As explained earlier, the message destination TID parameter of PVM being occupied by the next hop, the only location left to include these needed parameters is the message tag parameter, which happens to be of long integer type. A useful property of TIDs is that the three least significant digits should all be unique for a given session of PVM. Using this property we can cut the length of the two TIDs we have to send to only being three digits long. A sequence of ten was deemed sufficient for the target network sizes the NTS program would be simulating, therefore one digit long. This leaves as a restriction for the **message tag** of a message to be at **most two digits long** (i.e., 0 – 99). If this is the case, the four parameters can be concatenated into one. This is the solution adopted, Figure 16 shows the method used,

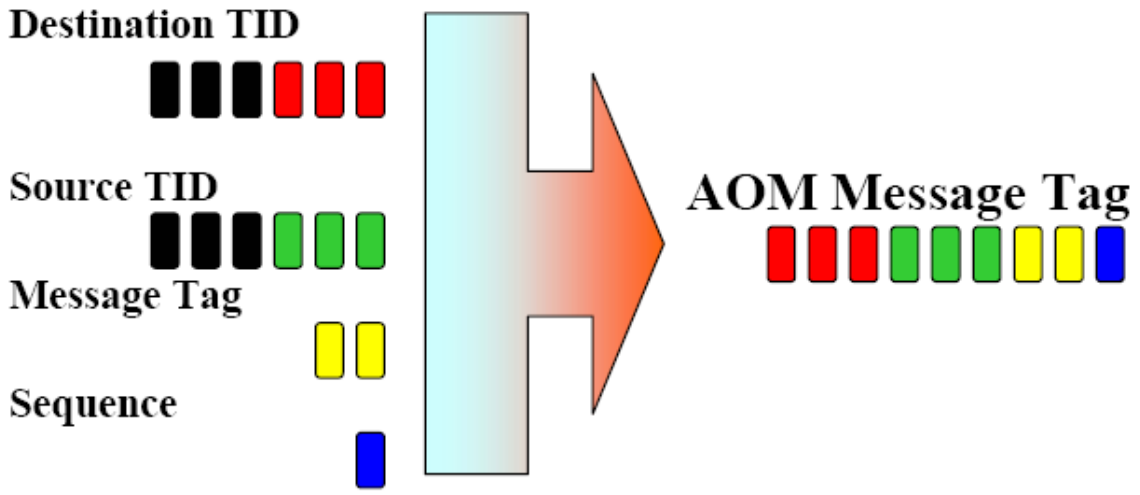

Figure 16: Integer Combination size limitation

giving for each message a unique identification passed in the message tag of the message which can be calculated at sending and reception and therefore can be used with the PVM receive functions to automatically receive reordered messages.

This use of a unique identification for each message, though permitting NTS to function properly as a store and forward network, is the one source of the most restrictions to the use of NTS. Other then the facts that client **message tags** can only be from **0 – 99** and that a total of **999 client codes** are permitted on any one run of NTS, the main restriction coming from this technique is that the use of the **"-1" listening tag** is **not permitted** to Clients. This comes from the fact that because of sequencing a specific tag is required for NTS to reorder the messages correctly.

3.6.8 Lookup Table

PVM provides a universal lookup table for each PVM session. This is used by NTS to save the TIDs of each Client and its corresponding **Id** or the three least significant digits of the TID. The id is used as a key to get the actual full TID from the table. The TID is also used as a key for the Id in the lookup table. Both of these entries are added to the lookup table upon connection of a Client to the NTS network. Routers use the former to lookup the final destination of a message once the incoming message tag has been processed. The later is a mechanism to check whether a given TID is part of the NTS network.

3.6.9 Overloaded vs. Forwarded

While the main PVM functions have been overloaded to dramatically change their behaviour to simulate the various effects of network routing, many functions in the overloaded library are just forwarded towards their PVM equivalents. For a lot of the cases this was for functionalities that did not have to be changed for the NTS simulation, but for some they are PVM advanced functionalities, which could not be incorporated into the NTS simulated network.

Note:

While not tested, it is still possible to send and receive messages with the standard PVM functions. This of course bypasses the NTS connection, and its effects. As stated though this functionality is not tested nor encouraged.

3.7 ntsMaster

The Master program is the top-layer of the NTS network simulation. This program must be run first, after PVM is launched, and will guide you through the creation of the NTS simulated network. A simple GUI interface lets you create/modify a network. The Master then iterates over the network description, creating instantiating the network for use. Once the network is created the Master permits you to dynamically change the topology of the network with the same interface to that of the creation one. The Master also allows you to view the effects of the simulated network and review some is consequences.

3.7.1 Creation

The ntsMaster permits the user to layout a network and then build it for Client use. The Master provides simple interface for adding a Router on a PVM running computer just by inputting its PVM name description (this name can automatically be chosen from a list of existing PVM running computers provided by the Master). Only one Router per computer is permitted. Once at least two Routers have been defined, the user can link Routers as neighbours to provide communication links for the network. This can easily be done using the GUI. Once the full topology of the desired network is produced, the Master will instantiate and initialise the Routers of the network for use.

The Master will first iterate over the list of Routers and **spawn** each, updating its internal structure with the TIDs of each spawned Router. The Master will then iterate over the full list of interconnections wished by the user and send to each Router the list of his neighbours. Each Router will use the information to build its initial Routing Table and then dynamic updates will propagate each table entry to the whole network. When all is ready, the Master will give the start of simulation signal.

3.7.2 The Running Network

During the running of the simulation the Master receives many messages coming from Clients or from Routers. Clients can ask for the TID of their local Router. Routers signal the Master when a new client subscribes with them, or unsubscribes. If the option is set, Routers forward information about every message that they process so that the Master can collect the data and analysis of the network performance can be performed.

3.7.3 Dynamic Rebuilding

As stated in the requirements, the Master program will not only be used to create the initial topology, but can be used while the network is running to modify it dynamically. The same interface is used for this purpose as for the initial building. In fact one could start the NTS network and then construct the topology completely while it is running, as the ntsMaster seamlessly integrates the two modes of operation. While the network Routing is such that no messages will be lost during rebuilding, it is not possible for a user to delete a Router while it still has Clients subscribed to it.

3.7.4 Master Design

The Qt toolkit is used to make the ntsMaster program with the Qt Designer doing most of the layout of the GUI. With this use, full advantages of the Qt techniques of design were implemented throughout the Master. The signal/slot mechanism was especially used to connect packages seamlessly, making the whole program easy to extend. Figure 17 shows the main packages of the ntsMaster:

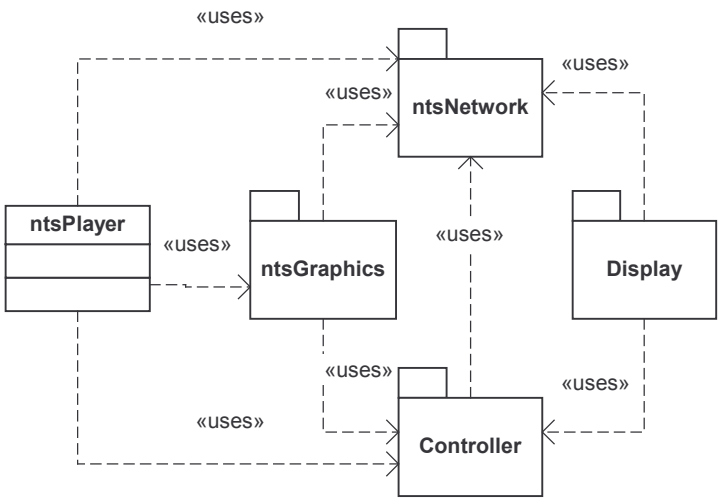

Figure 17: The main packages of the ntsMaster and their connections

While a full description of each package follows in the section 5 Implementation, the main connection theory will be stated here.

The *ntsNetwork* package is the main data structure of the program. Each package is connected to the *ntsNetwork* using the signal/slot method, in a "huge" Observer optimised pattern. Each package is connected to certain signals emitted by the *ntsNetwork*; therefore updates only happen when relevant changes to the data structure have been performed. Only the controller package is connected to the main controls of the *ntsNetwork* and can modify the main data structure of the network. Each other package: the various Displays, Graphics, and Player can set individual properties of components of the data structure but cannot change the structure such as add or delete a component.

As the Designer approach was adopted for the layout of the GUI, the Controller package is in fact the main window graphic of the ntsMaster application as well. This is also the case for each component of the Display package where each class is the graphical display plus the control to modify the options of the NTS network component being displayed. This is not the case for the *ntsNetwork* which has no proprietary display of its own (the whole ntsMaster application observer this structure to form views of it), and the *ntsPlayer* (which is just a class, and not a full package, but is a bit on its own) which uses the *ntsGraphics* package for its display (in a way the *ntsPlayer* bypasses the Controller to

use the *ntsGraphics* directly). The *ntsGraphics* is the only package to compromise a separate data structure and a view of the data.

More information on the use of the ntsMaster program and all the possible options is available in Using NTS, while more in detail information on the design of the Master is accessible in the section 5 Implementation.