

CS 168 VAN Simulator Guide

Overview

The VAN (Virtually Active Network) is a network simulator. It provides a simple API for creating interfaces onto different types of links, and using these interfaces to communicate between multiple nodes. It will allow you to create virtual nodes (workstations, routers, etc), define factors like delay and drop rate on the links between them and then start communicating.

The VAN code is provided as shared library. Your code will also be written as a library which sits on top of the VAN. For each part of the assignment you will produce a new library layer, and a program (or programs) which uses these libraries to demonstrate your work. This driver program will need to fully exercise the functionality of your code. The driver is critical to the testing of your code. Without a good driver, it is difficult to impossible to have a successful demo.

In order for you to understand what it is you need to do, it's first important to understand the VAN. This will walk you through everything you need to know about how to configure and compile the support code, so that you can focus on "your code."

The Important Files

There are basically only three important files that you need to have to get up and running with the support code. They are all under the `/course/cs168/` directory. The library itself is `libvan2.so`, found in the `lib/` subdir. The header files, `van2/van2/van.h` and `van2/van2/van.H` (for C and C++ respectively), are in the `include/` subdir. In addition, there are interface-specific header files for the ethernet, point-to-point, and loopback interfaces (`eth.h`, `p2p.h`, and `loop.h` respectively) which you need for the interface option flags that they contain (see the Interfaces section further on down the page.) Finally, the configuration file `sample-netconfig` is in the `etc/` subdir. Additional sample netconfig files may be found at `/course/cs168/van2/netconfig`.

You will check out your own copy of the simulator when you copy the IP code out of our course directory, and build it. You need not understand the intricacies of the build process for the simulator, but you should understand how to interact with it and how to link your code against the library it produces. Your IP and TCP code will be in subdirectories of the total project. We provide the VAN code so as to minimize the "magic" of the support code and so that you can look into how it works to learn more, or to tell us if you think there are ways it could be improved.

NOTE: We will only be discussing the C interface in this document. Once you've understood this material, the C++ interface should be straightforward. In addition, during 168, you only need to worry about P2P and WIRELESS interfaces and may safely ignore other types of interfaces.

The `netconfig` file is especially important to understand, since it's used to define your network topology. Other than this however, you should have no reason to interact with it in your program. The support code uses it for setup, and your code does not need to interact with it at all. That said, you should play with the default `netconfig` file and then create your own network layouts to test various conditions.

The support code itself is pretty straightforward. You can use either C or C++ for the projects in this class (though you really should pick one at the outset and stick with it). The header file defines all the functionality that the VAN exports. This includes support for reading/writing on the network, bringing nodes up/down, setting interface options, etc...

netconfig

The netconfig file is what the support code uses to setup the network. This file defines the virtual topology and conditions of the network. The VAN will support up to 256 nodes in a network configuration. The layout of the file is defined as follows:

```
link <type> <drop> <garble> <delay base> <delay delta> <mtu> [type-specific options]
...

node
    if <type> <num> <rate> [type-specific options]
    ...

node
...
```

What does that mean? Here's an example:

```
# link definitions
link eth 0.0 0.3 3.0 0.25 64
link eth 0.1 0.2 1.0 1.0 128
link p2p 0.0 0.0 8.0 0.8 32 localhost 10000

# node definitions
node
    if eth 0 256
    if eth 1 256

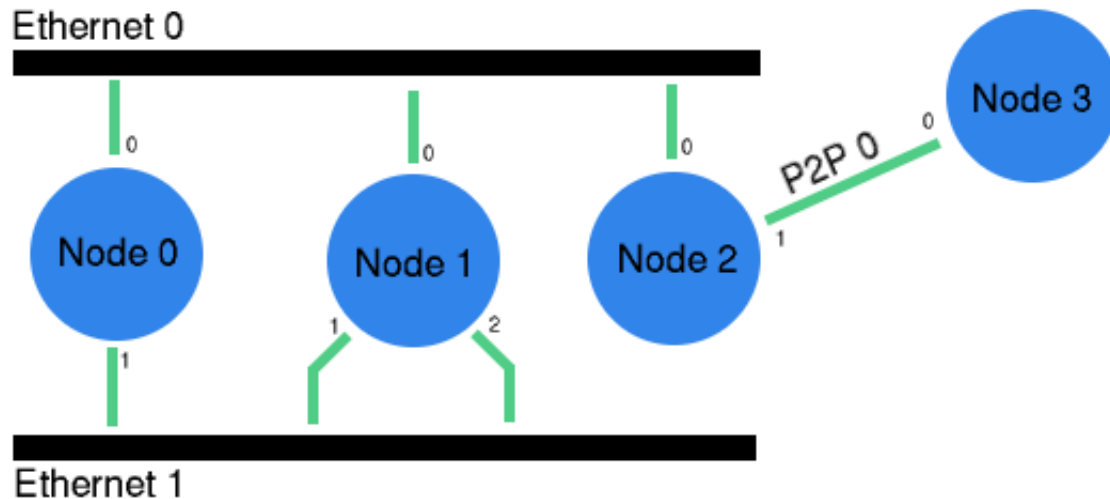
node
    if eth 0 128
    if eth 1 128
    if eth 1 64

node
    if eth 0 32
    if p2p 0 128 a

node
```

```
if p2p 0 256 p
```

This network diagram represents the file on the previous page.



The first section of the file defines the set of links that comprise the connections between the various nodes. First we have the definition of the first ethernet link, eth0. It has been given a drop rate of 0%, a garble rate of 30%, a delay ranging 2.75 - 3.25 seconds and an MTU (maximum transmissions unit) of 64 bytes. Likewise, eth1 has a drop rate of 10%, garbles 20% of its packets, has a delay in the range of [0, 2] seconds, and an MTU of 128 bytes. The third link is the first point-to-point link, p2p0. It is completely reliable (no drop or garble rate), has a delay of [7.2, 8.8] seconds and an MTU of 32 bytes. The second 2 arguments to this links are p2p-specific options specifying the host and port which the active party will connect to.

NOTE: If you attempt to bring up a node that has a passive interface on a p2p link, you must make sure that it's running on the host specified by the netconfig file. This is due to way that the VAN models p2p links. Also, it can take several seconds for the P2P links to become active. Your code should not attempt to transmit over the P2P links for 10 to 20 seconds!¹

As seen above, Node 0 has two interfaces: 0 on eth0 and 1 on eth1. The third argument to each interface is the drop rate for the individual interface. More on this later, under the congestion section.

netconfig Type-Specific Link Options

For the purposes of CS168, we are only concerned with the type-specific interface options of P2P and WIRELESS interfaces (ETHERNET interfaces have no type-specific options). For link options

¹The actual time may vary. We make no guarantees as to the exact time necessary for all the van nodes to come up

we have:

Type	Option	Description
P2P	< host > < port >	The host and address of the node running the passive end of a P2P link
WIRELESS	None	Wireless links have no type-specific options. Note that wireless links correspond to communication frequencies, and thus a wireless interface can only communicate with another wireless interface if they share the same link. Consequently, you will generally only need a single instance of a wireless link.

netconfig Type-Specific Node Options

For nodes options we have the following:

Type	Option	Description
P2P	<communication mode>	A P2P communication mode can be either "a" or "p" for active or passive respectively. The node in passive mode should be run on the host specified by the corresponding P2P link's type-specific options (see above).
WIRELESS	<x-coord> <y-coord> <radius>	The <x-coord> and <y-coord> of the options specify the x-coordinate and the y-coordinates of the node. The <radius> field specifies the transmission radius of the node's wireless interface. Another node can only receive transmissions from this node if the receiving node is at most <radius> units from this node's location.

Congestion

The third value specified to the interface directive gives the maximum number of bytes that an interface can receive in any given 10 second period. This is used to simulate dropping packets due to congestion. For example, say this interface receives a sequence of packets arriving one second apart with the following lengths:

14, 29, 31, 17, 18, 26, 45, 35, 19, 25, 31

For simplicity, we assume that the interface had been inactive for the previous 10 seconds. The first 9 packets arrive without incident. At this point, the interface has received 234 bytes in the previous 10 seconds. When the next packet arrives, the the congestion simulation kicks in. $234 + 25 = 259 > 256$, so the packet is dropped. When the next packet arrives, the window shifts and we have received $234 - 14 = 220$ bytes in the previous 10 seconds, and we can accept this packet since $220 + 31 = 251 < 256$.

Using the Support Code

As described earlier, the support code is provided as a shared library and a header file which exports the available functions. For each node on the network, the following steps must be followed in order to setup and use the VAN simulator in your code. In C, you must first start the VAN by running `van_init(netconfig_filename)`. Next, you must initialize one of the numbered VAN nodes using `van_node_get` this will initialize and return a pointer to a `van_node_t` structure. In C++, both these steps are taken care of by initializing a `VanNode`.

You will be doing this once for every node in your network (so, given the sample netconfig file above, this process would be done 4 times, for nodes 0-3). It's important to understand, however, that you should only create one node per process. That is, you should start N different instances of your driver, one for each of the N nodes in your netconfig. You must create a generic driver. If you simulate the entire network in one process, your project will not be acceptable!

Now, having created a node, you can communicate with the rest of the network via two important functions: `van_node_send` (`VanNode::Send`) and `van_node_recv` (`VanNode::Recv`). Like the `send / recv` system calls used for socket communications, they take a buffer (to write out or to read into), a length (the size of data to write or the maximum number of bytes to read), a flag parameter, and an address. Currently there are no flags defined for these methods, so this parameter is ignored. In addition, they each take an interface parameter. Be careful that, when calling `van_node_send`, you provide an address with `va_type` set appropriately. For example when sending on an interface of type `IT_P2P`, make sure that `va_type` is set to `AT_P2P`.

Each connection from one node to another is called a link. An interface value describes one of the (possibly many) connections a node has to a particular link. Conceptually, an interface is modeling a single computer's hardware (e.g. an ethernet card) while the link models the network itself (e.g. Brown's ResNet). From the above netconfig file, node 0 has interfaces to two links, `eth 0` and `eth 1`, each of which would be a separate ethernet network. The links are numbered in the order in which they are defined (starting at 0) with separate numberings for each type of link. Thus, the links described in the sample netconfig above would be `eth 0`, `eth 1`, and `p2p 0`. When sending data onto the network, or receiving incoming data, the incoming/outgoing link can only be accessed by the relevant interface number. Similar to links, interfaces are numbered, starting at 0, in the order in which they are defined in the netconfig; however, for interfaces, type does not matter. In the example above then, node 0's interface 0 connects to `eth 0` and its interface 1 connects to `eth 1`. Likewise, from node 1's point of view, interface 0 connects to `eth 0`, while interfaces 1 and 2 both connect to `eth 1`. And so, for the other two.

These interface values are used to better approximate a real network environment. When a node in a real network gets data, it doesn't necessarily know who sent it, but it does know on which interface it received the data (e.g. its 802.11 card or its modem). Your code will sit on top of the van node and create tables to map these interface values into more meaningful data.

Using `van_node_setifopt` (`VanNode::SetIfOpt`) is handy in testing various network configurations and making sure that your RIP (or other dynamic routing protocol) actually works in the face of changing topology. It allows you to bring a given interface up (working) or down (not working). Additionally, you can use `van_node_put` to destroy the node entirely.

VAN provides two more helper functions. The first, `van_node_getifopt` (`VanNode::GetIfOpt`), returns the current state of a van node's options. The second, `van_node_nifs` (`VanNode::GetNifs`), returns the number of interfaces on the van node.

Sample Code

Sample code is provided in the checkout you'll do from our asgn directory. Good luck networking!