No-hop Documentation

Lily Huegerich (plus epydoc)

May 5, 2021

Contents

1 Introduction

No-hop[6] utilizes the programmable data plane to offload the look-up process in a Distributed File System to the data plane. No-hop includes two versions. The first, (No-hop_finger-table) utilizes a table modeled after the Chord [5] finger table to maintain table sizes that scale efficiently as the number of hosts grow, however for this all involved switches need to be programmable. The other version (No-hop_rewrite) only needs one programmable switch on every entry path but has tables that scale less efficiently. The No-hop_rewrite tables are loosely inspired of the principle of a one hop dht [1].

1.1 Dependencies

No-hop was implemented in the P4tutorial virtual machine [8], No-hop also benefits from the use of the P4 tutorial utilities programs [8]. The switch programs are written in P4 [3] and the rest in Python. The main programs used in the virtual machine are scapy [10], the P4 software switch, compiler and grpc [2, 9, 4] as well as mininet [7].

1.2 Example Usage

Steps to run a basic time and hop test comparing the baselines to the No-hop rewrite implementation.

1. In your shell at No-hop/compare_classic_v_dataplane run:

   ```
   make
   ```

2. Mininet will build the example topology and start a mininet command prompt. To check the status of the network run:

   ```
   mininet> net
   ```

3. Now to run scripts on a particular host use a host name returned by the above command to open a terminal for that host. The below example uses a host name given by the example:

   ```
   mininet> xterm h_Rc0
   ```

4. If you want to test your own metrics you can use the functions in send_and_recieve_dht.py to build your own server and clients otherwise you can use test_time.py. To do this run:

   ```
   python test_time.py <Node Name>
   ```

   on all hosts and lastly on the client:

   ```
   python test_time.py client
   ```

   this will start the test and save the output of the tests to individual logs for all hosts in the folder test_logs.

5. Once finished run:

   ```
   sudo mn -c
   ```

   Other wise when running the next time make will not work since the links still exist from the prior run.

2 Module Data_Plane_DHT

2.1 Functions

change_order(*key_range*, *switch*)

change order of ranges, for when a range crosses 0 in the ring

predesscor(*switch*)

find ring list index of predesscor of switch

successor(*switch*, *ID*, *direction*=0)

Find succesor of switch, returns in this order, ring list index, connection port, and switch object of succesor

distance(*ID*, *switch*)

find logical distance between switches in ID space

active_ports(*switch*)

find in use ports of switch

make_string_from_connection(*c*, *host*='none')

return a connection object as a human readable string

make_bIDirectional_connection(*switch_a*, *switch_b*, *host*='none')

Make connection that is bidirectional, returns two connections if settings are set to not biderectional connections, otherwise one

2.2 Variables

Name	Description
hosts	**Value:** 0
__package__	**Value:** None

2.3 Class Ring

2.3.1 Methods

__init__(*self*, *name*, *level*=0, *RING_ID_SIZE*=6, *ip_base*='10.0.0.', *mac_base*=1)

Name is an IDentifier for the ring level is the distance from non DHT Traffic, (the farther from leaf nodes the lower the level , so the tree grows down)

free_host_ID(*self*)

find free host ID

add_switch(*self*, *switch*, *max_port_out*=30, *max_port_in*=30, *switch_to_controller_port*=1, *ID*=False, *classic*=True)

should in normal cases only be run with a string for a name, switches should only be created inside a Ring object

add_host(*self*, *connected_switches*=1, *switches*=[], *client*=False)

connected switches is the amount of switches connected to host, switches are which switches, a list of their IDs

print_switches(*self*)

Function to print status and overview of switches

2.4 Class host

Servers and clients in topology

2.4.1 Methods

__init__(*self*, *name*, *ip*, *mac*, *max_port_out*=20, *max_port_in*=20, *ID*='ending_num', *client*=False)

2.5 Class Switch

In most cases one should never manually call switch init methods, should be done by calling add_switch inside a ring object

2.5.1 Methods

__init__(*self*, *name*, *ring*, *max_port_out*, *max_port_in*, *switch_to_controller_port*, *ID*=False, *classic*=True)

For this example we used a pseudo random number gernarator. Use a hash function that is fitting to your usability requirments

switch_status(*self*)

Switch status is to be called in a for loop when looking at all switches in a ring,

make_tables(*self*, *fail*=False)

Generate table contents for switch

2.5.2 Instance Variables

Name	Description
switch_to_controller_port	connections are a tuple of (port, switch), port being an int and switch being the object of the swith connected to
mac	bmv2_connection_object only used if useing bmv2 switches

2.6 Class connection

If no ports are given in with parameters a_pot and b_port one will be assigned

2.6.1 Methods

__init__(*self*, *switch_a*, *switch_b*, *a_port*=`-1`, *b_port*=`-1`, *host*=`'none'`)

connection from switch_a to switch_b

remove(*self*)

remove a connection object from both switches connections lists

print_connection(*self*, *inout*)

2.6.2 Instance Variables

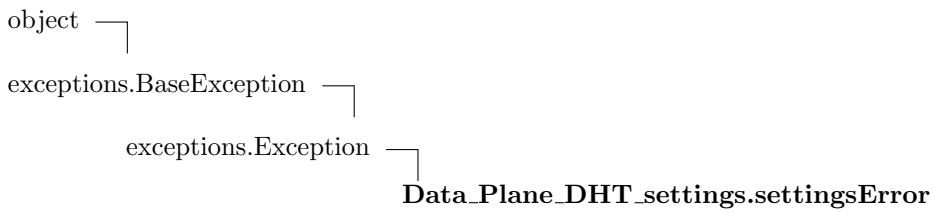
Name	Description
direction	Port Assigment

3 Module Data_Plane_DHT_settings

3.1 Variables

Name	Description
highly_verbose	very detailed output, especially usefull for debugging **Value:** `0`
verbose	standard output **Value:** `1`
generate_topo_json	1 to save generated topology to json, json is rewritten everytime that program is called if set to 1 **Value:** `1`
RING_SIZE	this must also be changed in the P4 file and the servers (if you are to change it) **Value:** `6`
bidirectional_connections	Connections in mininet are bidirectional, so should be for minient 1 if not should be 0 **Value:** `1`
log_file	where should logs be saved **Value:** `'../logs/'`
test	**Value:** `1`
Rewrite_implementation	1 for No_hop rewrite 0 for No_hop finger table implementation **Value:** `1`
__package__	**Value:** `None`

3.2 Class settingsError



This error is raised if a function is not complient with a certain setting

3.2.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

3.2.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

4 Module make_topology

4.1 Functions

create_connected_ring(*ring_name*, *level*, *amount_of_switches*, *topo*, *hosts*, *ip_base*=`False`, *switches*=`[]`, *IDs*=`[]`, *classic*=`True`)

create a ring of amount_of_switches may switches with hosts many swithes, switches are connected in a ring, returns ring object

quick_test()

simple topology for quick testing

test_ring(*failover_test*=`False`)

tree_topo(*failover_test*=`False`)

tree topology for testing classic data center structures

basic_3_ring(*level_zero_amount_of_switches*=`4`, *level_one_amount_of_switches*=`2`, *outsIDe_nodes*=`1`)

return ring_v0_1, ring_v1_1, ring_v1_2

4.2 Variables

Name	Description
__package__	**Value:** `None`

4.3 Class configurationError

object
exceptions.BaseException
exceptions.Exception
make_topology.configurationError

4.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

4.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

4.4 Class topo_tracker

Keeps track of generated topology

4.4.1 Methods

__init__(*self*, *file_name*, *file_path*='../basic/')

add_switches_to_topo(*self*, *switches*, *fail*=False)

adds generated switches to a already existing topo_tracker fail=True is purposfull failing of a link for testing purposes.

add_hosts_to_topo(*self*, *rings*, *amount*, *client*=False, *connected_switches*=[])

connect_nodes(*self*, *ring_0_nodes*, *ring_1_nodes*)

connects all nodes in ring 0 to all nodes in ring 1

formalize_table(*self*, *switch*, *fail*=False)

formalize table values to send to the software switch

generate_ipv4_lpm_table(*self*, *switch*, *fail*=False)

IPv4 tables for comparision of No_hop to classic look up proccess

path_finder_ip(*self*)

wrapper for P4 tutorial shortes path function used in LPM tables

check_links(*self*)

Verfiy correct link configuration , especially checking for duplicates

create_json(*self*)

print topo object to JSON

5 Module send_and_recieve_dht

5.1 Functions

user_input(*my_id*, *direction*)

Functions supported:

send- send a packet, options Packet id=number within id space message= (per default) "DHT message from (this id) :D" default will assign id for packet

quit end program

send(*ID*, *message*='test rewrite', *direction*=0, *gid*=1)

Send No-hop packet

print_pack(*pkt*)

recieve()

main()

5.2 Variables

Name	Description
ring_size	**Value:** 6
q	**Value:** 1
direction	**Value:** 2
__package__	**Value:** None

5.3 Class P4dht

object

scapy.base_classes.Gen

scapy.base_classes.BasePacket

scapy.packet.Packet

send_and_recieve_dht.P4dht

inhereted from scapy packet class, packets for No-hop

5.3.1 Methods

Inherited from scapy.packet.Packet

__contains__(), __delattr__(), __delitem__(), __div__(), __eq__(), __getattr__(), __getitem__(), __gt__(), __init__(), __iter__(), __len__(), __lt__(), __mul__(), __ne__(), __nonzero__(), __rdiv__(), __repr__(), __rmul__(), __setattr__(), __setitem__(), __str__(), add_payload(), add_underlayer(), answers(), build(), build_done(), build_padding(), build_ps(), canvas_dump(), clone_with(), command(), copy(), decode_payload_as(), default_payload_class(), delfieldval(), display(), dissect(), dissection_done(), do_build(), do_build_payload(), do_build_ps(), do_dissect(), do_dissect_payload(), do_init_fields(), extract_padding(), firstlayer(), fragment(), from_hexcap(), get_field(), getfield_and_val(), getfieldval(), getlayer(), guess_payload_class(), hashret(), haslayer(), hide_defaults(), init_fields(), lastlayer(), libnet(), lower_bonds(), mysummary(), pdfdump(), post_build(), post_dissect(), post_dissection(), pre_dissect(), psdump(), remove_payload(), remove_underlayer(), route(), self_build(), setfieldval(), show(), show2(), sprintf(), summary(), upper_bonds()

Inherited from object

__format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

5.3.2 Properties

Name	Description
Inherited from object __class__	

5.3.3 Class Variables

Name	Description
name	**Value:** `'P4dht'`
fields_desc	**Value:** `[<Field (P4dht).message_type>, <Field (P4dht).gid>, <Fiel...`
overload_fields	**Value:** {<class 'scapy.layers.l2.Ether'>: {'type': 4626}}
Inherited from scapy.packet.Packet aliastypes, explicit, initialized, payload_guess, show_indent, underlayer	

6 Module classic_dht

6.1 Variables

Name	Description
__package__	**Value:** `None`

6.2 Class table_one_hop

Simple baseline implementation of a one hop DHT

6.2.1 Methods

__init__(*self*, *name*=`'h_R0'`, *file_topo*=`'topology.json'`)

evaluate(*self*, *id*)

6.3 Class table

Simple baseline implementation of CHORD DHT

6.3.1 Methods

__init__(*self*, *name*=`'h_R0'`, *file_topo*=`'topology.json'`, *preset_ids*=`True`)

successor(*self*, *id*)

evaluate(*self*, *id*)

distance(*self*, *id*, *entry*)

Index

References

[1] A. Gupta, B. Liskov, and R. Rodrigues. "One Hop Lookups for Peer-to-peer Overlays". In: *HotOS*. 2003.

[2] P4 Language Community. *P4c*. 2019. URL: https://github.com/p4lang/p4c (visited on 03/19/2019).

[3] P4 Language Consortium. *$P4_{16}$ Language Specs, Version 1.1.0*. 2018. URL: https://p4.org/specs/ (visited on 02/14/2019).

[4] P4 Language Consortium. *Simple Switch GRPC*. https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc.

[5] I. Stoica et al. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM*. 2001.

[6] L. Huegerich. *No-hop Software*. https://github.com/lilyhuegerich/No-hop. 2020.

[7] *Mininet*. http://mininet.org/.

[8] *P4 Tutorial*. https://github.com/p4lang/tutorials.

[9] P4.org. *Behavioral model repository*. https://github.com/p4lang/behavioral-model. Accessed: 2018-12. P4 Language Consortium, Oct. 2015.

[10] *Scapy*. https://scapy.net/.