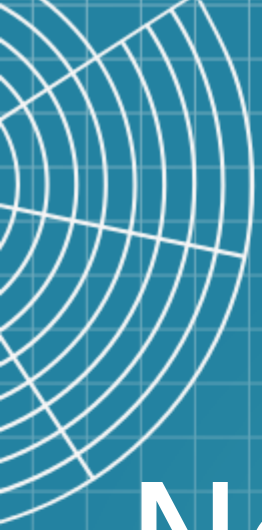
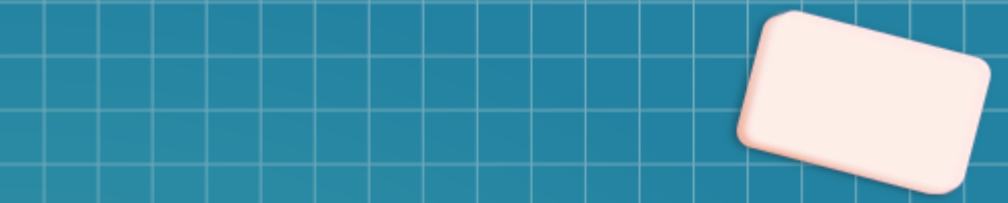
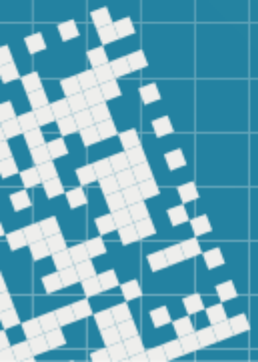
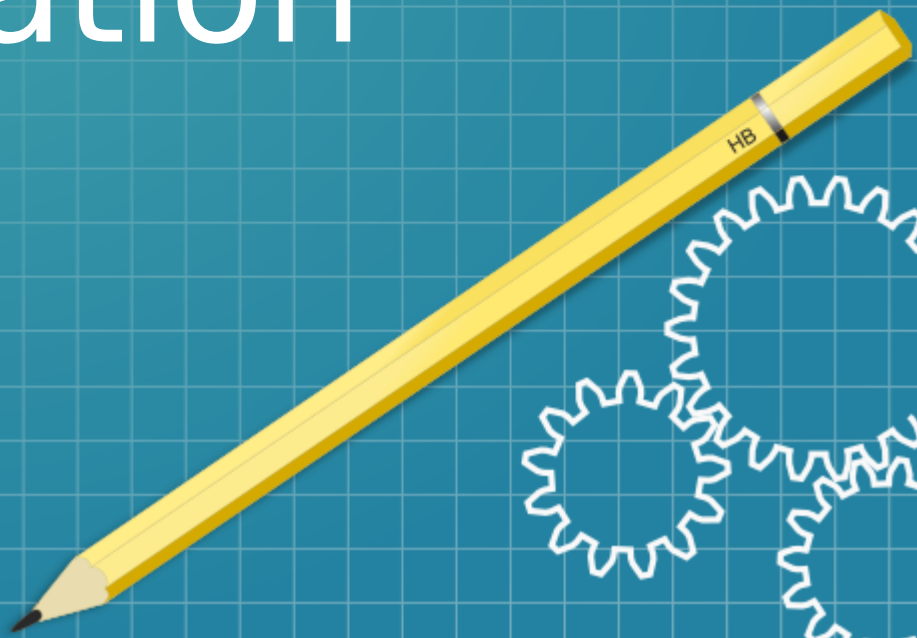
Software Defined Networking: Distributed Systems and Trust Computation





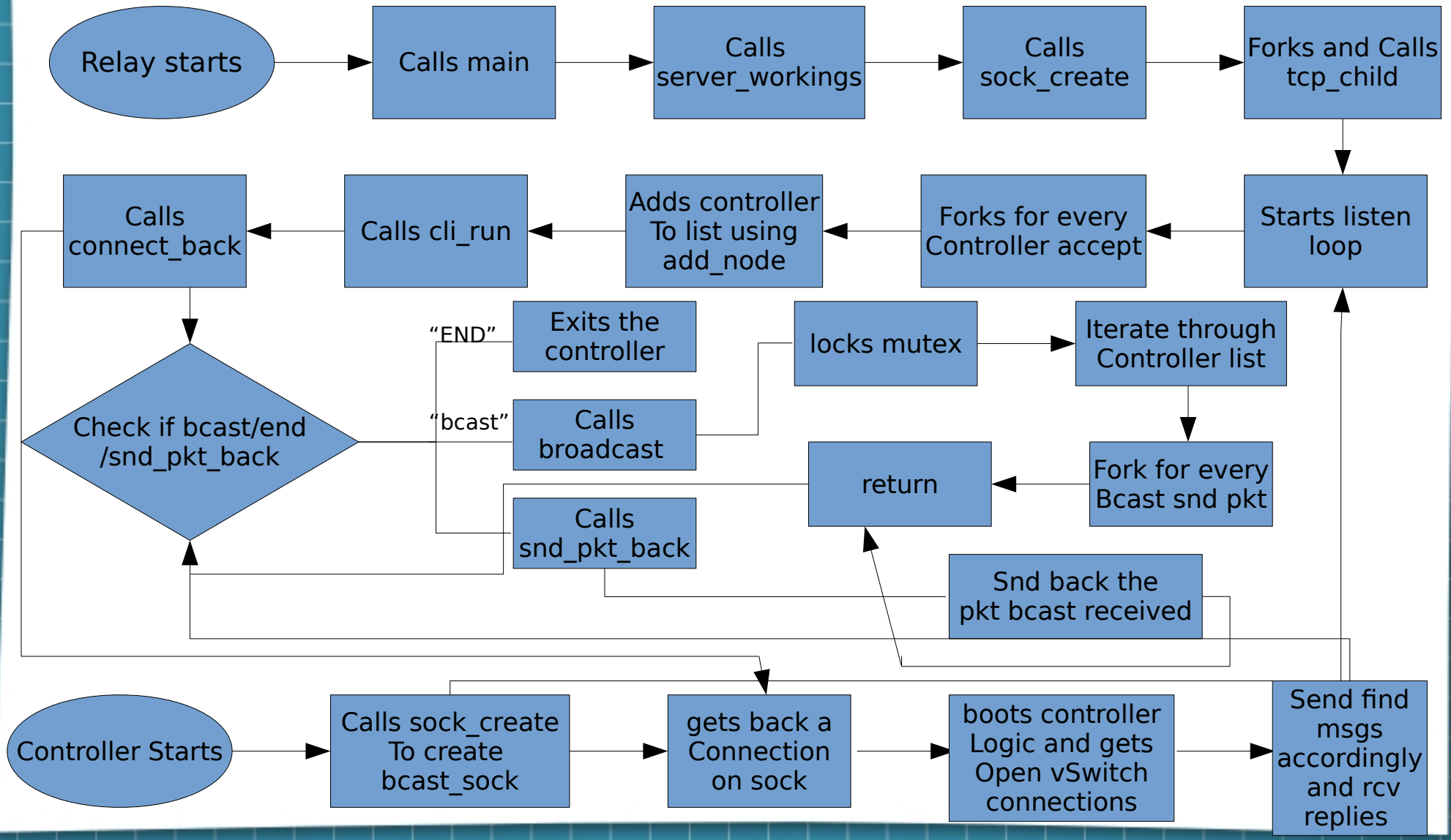




What is happening?

- The lack of support to distributed network in Openflow standard.
- Innovative way to interconnect multiple controllers.
- Acts like a messenger system between controllers.
- Controllers pose queries of certain network node addresses.
- Replies are sent by corresponding controllers that own those nodes.
- Written in pure C for better efficiency.
- Flexible library is provided to make the controller docile to relay protocol.
- Single entry, multi process and single exit structure.

Overview


Relay starts
Calls main
Calls server_workings
Calls sock_create
Forks and Calls tcp_child
Starts listen loop
Forks for every Controller accept
Adds controller To list using add_node
Calls cli_run
Calls connect_back
Check if bcast/end /snd_pkt_back
"END"
Exits the controller
"bcast"
Calls broadcast
Calls snd_pkt_back
locks mutex
Iterate through Controller list
Fork for every Bcast snd pkt
return
Snd back the pkt bcast received
Controller Starts
Calls sock_create To create bcast_sock
gets back a Connection on sock
boots controller Logic and gets Open vSwitch connections
Send find msgs accordingly and rcv replies

The Relay Modules

main.c

- int init(int argc)
- int main(int argc, char *argv[])

server.h

- int server_workings(char *argv)

sock_create.h

- int sock_create(char *addr, int flag)

allocate.h

- void *allocate(char *type, int size)
- void deallocate(void *a, char *type, int size)

tcp_child.h

- void tcp_child()
- void cli_run(union node *client)
- int connect_back(struct controller *sender)

broadcast.h

- int broadcast(struct controller *sender, char *cmds)
- void broadcast_run(struct broadcast_struct *b_struct)
- void *cleanup_run(void *a)

list.h

- void add_node(union node *new, union node *start, int flag)
- void general_equate(union node *a, union node *b, int flag)
- union node *find_node(union node *start, int tag)
- int del_node(union node *start, int flag, int tag)
- int list_len(union node *start)

list.h

- int snd(struct controller *cli, char *cmds, char *reason, char *retval, int free_it)
- char *rcv(struct controller *cli, int sock, char *reason, char *retval)

The Relay Structures

struct controller

- int id
- int bcast_sock
- int sock
- struct sockaddr_in addr

struct bcast_msg_sock

- int id
- int done
- char *msg
- struct controller *sender

union node

- int tag
- struct controller *ctrlr
- struct bcast_msg_node *bmn
- union node *nxt
- union node *prev

struct broadcast_struct

- struct controller *cli
- char *cmds

The Controller Modules

tcp_connector.h

- int get_connection_back(int sock)
- int send_to_relay(int sock, int flag, char *addr)
- char *rcv_bcast(int sock)

sock_create.h

- int sock_create(char *addr, int flag)

snd_rcv.h

- int snd(int sock, char *cmds, char *reason, char *retval)
- char *rcv(int sock, char *reason, char *retval)

allocate.h

- void *allocate(char *type, int size)
- void deallocate(void *a, char *type, int size)

TODO

- Create a good mininet topology script to handle dynamic connection as well.
- Add dynamic detection of new hosts and automatic update in controller database.
- Add caching module in relay for faster access.
- Add redundancy module in relay for higher rate of availability.
- Add sub-relaying support.

Thank You!

Prepared and Presented by:
Naman Arora
RA1511003010235