FRACTEL – Design, Implementation And Evaluation of a Multi-hop Wireless TDMA System

Nirav Uchat

Faculty Mentors

Prof. Bhaskaran Raman and Prof. Kameswari Chebrolu

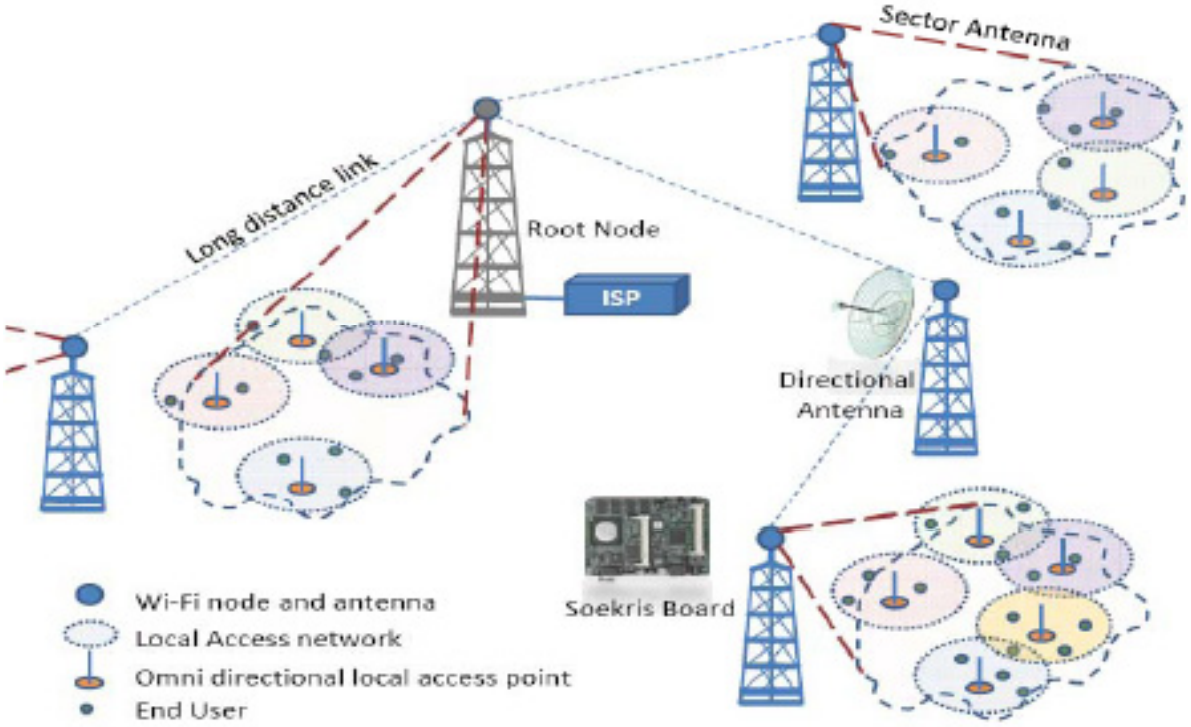
SYNERG MTP Defense Workshop

June 24-25, 2009

Department of Computer Science and Engineering
IIT Bombay

Project Vision

- Digital inclusion of remote villages
- Providing data, voice and video connectivity with QoS guarantees
- Cost effective solution by using off-the-shelf hardware, open source driver and license free band



Challenges in Wireless

- Issues in using 802.11 Wi-Fi protocol
  - Long distance carrier sensing
  - Difficult to assure QoS guarantees
  - Poor performance on long distance link
- How about using TDMA?
  - Communication with precise slot boundaries; no CSMA
  - Minimum collision due to synchronous operation
  - Guaranteed fulfillment of QoS requirements due to centralized scheduling
- TDMA more suitable than CSMA for our requirements

Problem Statement

To Design, Implementation and Evaluate Multi-hop WiFi-based TDMA System

- Using off-the-shelf hardware, open-source driver and unlicensed band
- Should support both best effort (HTTP, FTP) and real-time(voice, video) traffic
- Dynamically adapting the schedule in response to change in network load and topology

Related Work

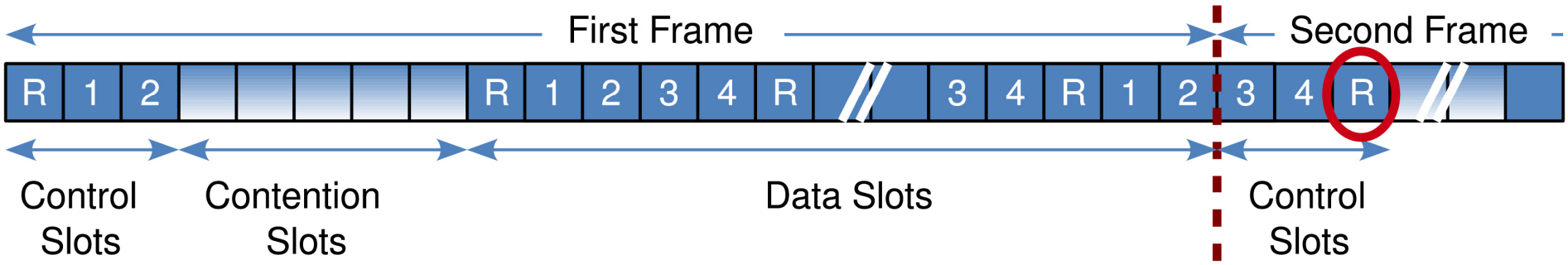
- Existing protocols provide hooks into Madwifi drivers for
  - stripping off CSMA mechanism (SoftMAC - NOV, 2005)
  - using different MAC protocols based on network conditions (MultiMAC - NOV, 2005)
  - precise time synchronization (MadMAC - SEP, 2006)
  - control over radio configuration and time critical functions (FreeMAC - AUG, 2008)
- Different Approach
  - Overlay MAC approach - works above MAC layer (JUN, 2005)
  - 2P Protocol on bipartite graph with marker packet - HostAP driver on prism chipset – (AUG, 2005)
  - SRAWAN - IIT Kanpur (May, 2006)
- To our knowledge, there is no implementation of multi-hop TDMA system yet

Our Approach

- Centralized TDMA scheduler
  - Root node creates a global schedule and disseminates it across the network
  - Adapting schedule based on bandwidth requests
- Multi-hop time synchronization and schedule dissemination mechanism
- Multi-hop packet routing
- Multi-hop TDMA implementation at MAC layer

Protocol Design 1 of 2

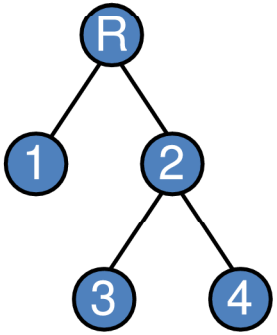
- Slot types
  - Control slot: Sending control information down the tree
  - Contention slot: Sending information towards root node
  - Data slot: Sending data across network
- Packet Headers
  - Schedule Header: Attached to every schedule packet
  - Data Header: Attached to every data packet
- Frame Structure
  - Few control and contention slots and many more data slots
  - Repeating pattern of these slots(but fixed for single frame)



Protocol Design 2 of 2

- Multi-hop Schedule Dissemination
  - Only root node has authority to create new schedule
  - All non-root nodes stores schedule that they receives from their assigned parent for multi-hop transmission
- Data Routing
  - Data header attached to data packet is used for routing packet in the network
  - Only data header gets modified while packet is being routed
- Routing Tree Elements
  - Routing tree elements sent in schedule packet is used by non-root nodes to recreate complete topology

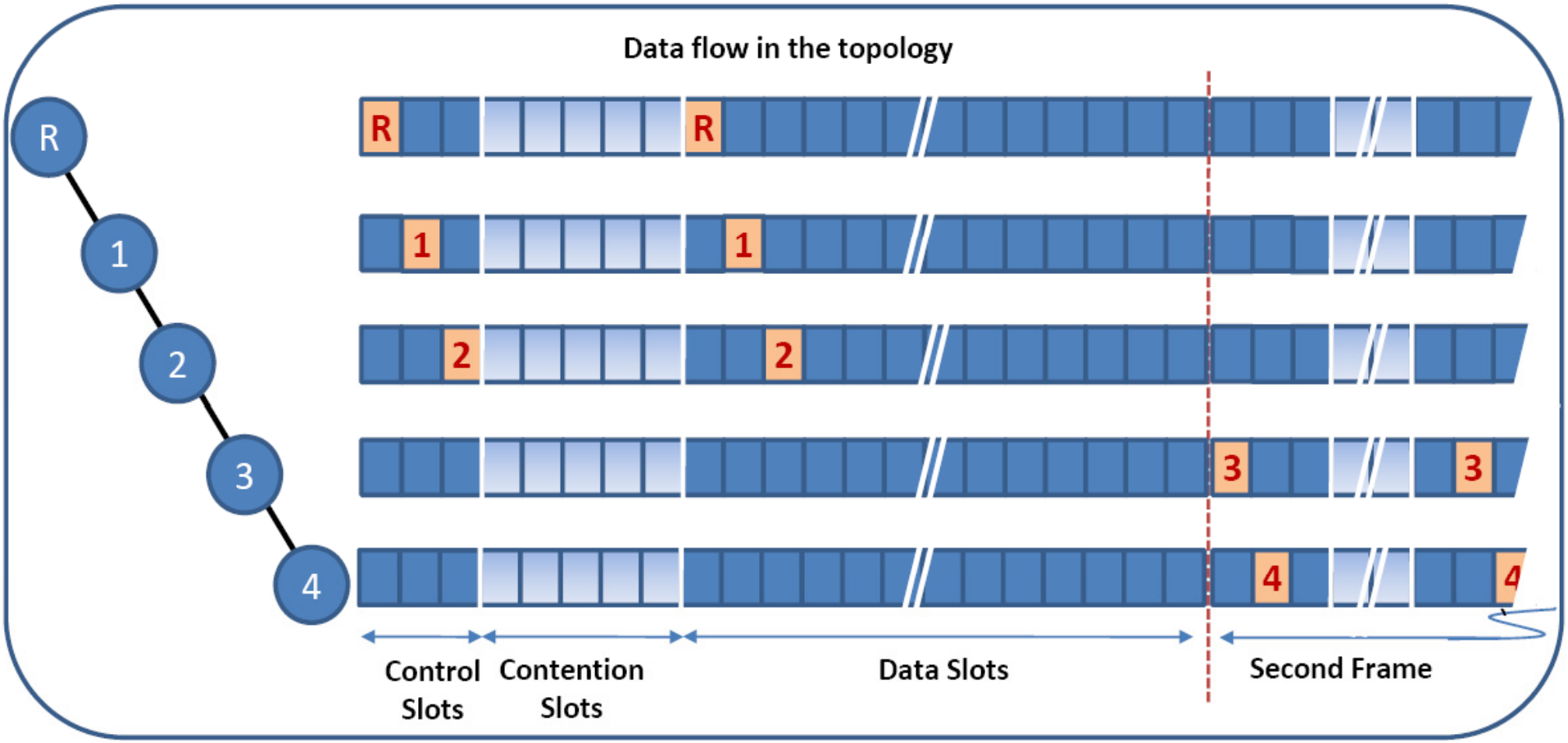
Slotting Structure



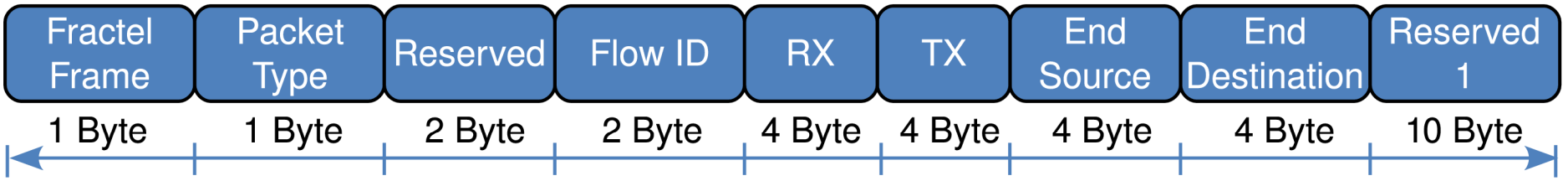
R	1	R	2	1	-	2	3	2	4	3	-	4	-
P	C	P	C	P	C	P	C	P	C	P	C	P	C

Routing Tree Elements

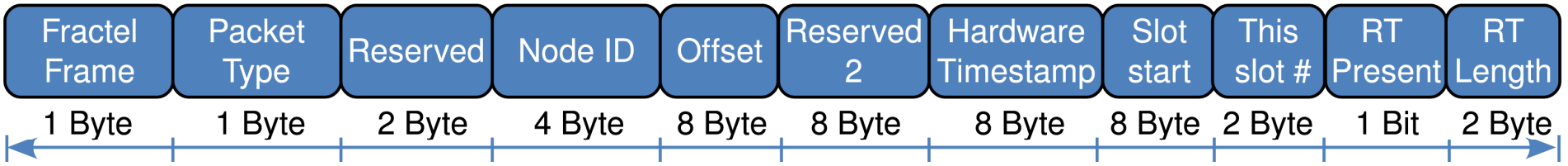
P : Parent
C : Child



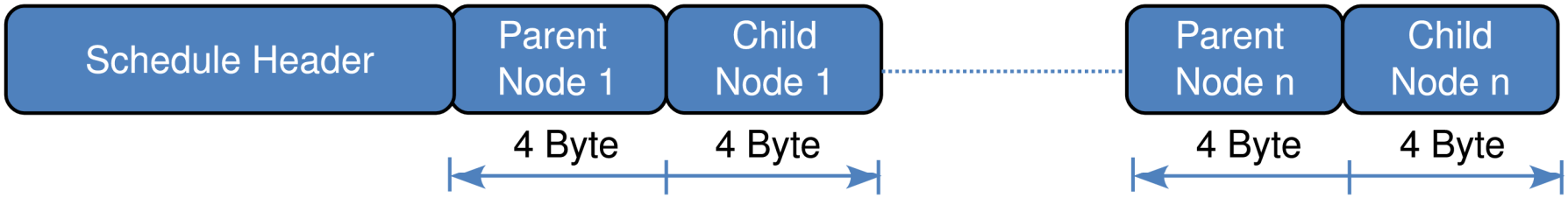
Packet Headers



Data Header



Schedule Header



Routing Tree Elements

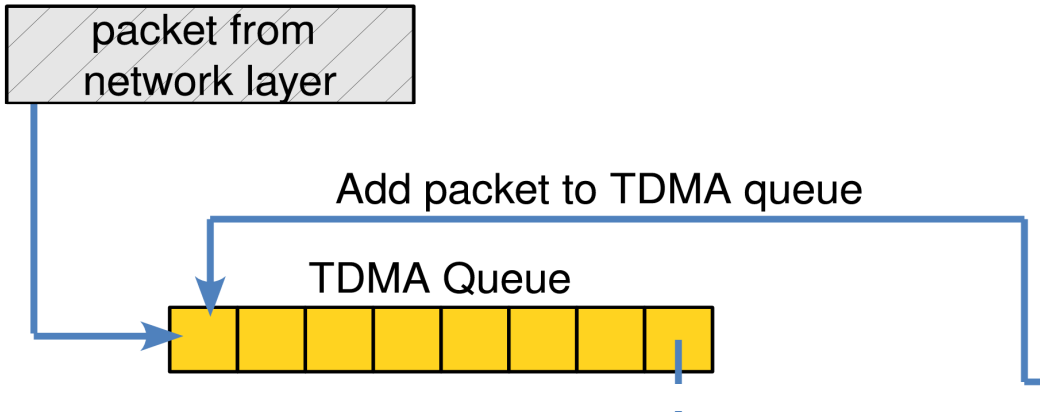
Framework for TDMA System

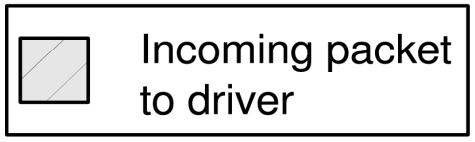
- Disabled MAC level acknowledgments - Tested
- No RTS/CTS – Tested
- Raw packet transmission; no 802.11 frame - Tested
- Disabled random/post back-off mechanism - Tested
- Tweaked CCA mechanism to always sense channel clear - Tested by Interns (Anupama and Bharat Jain) – Not implemented
- Suppress effect of NAV field and disabled sequence number printing on outgoing packets - Tested
- Generating hardware time stamped packets - Tested
- Send/Receive packets in monitor mode - Tested
- Generation of control packets at MAC layer (in monitor mode) - Tested
- Enabled channel switching from driver code - Tested

Multi-hop TDMA System

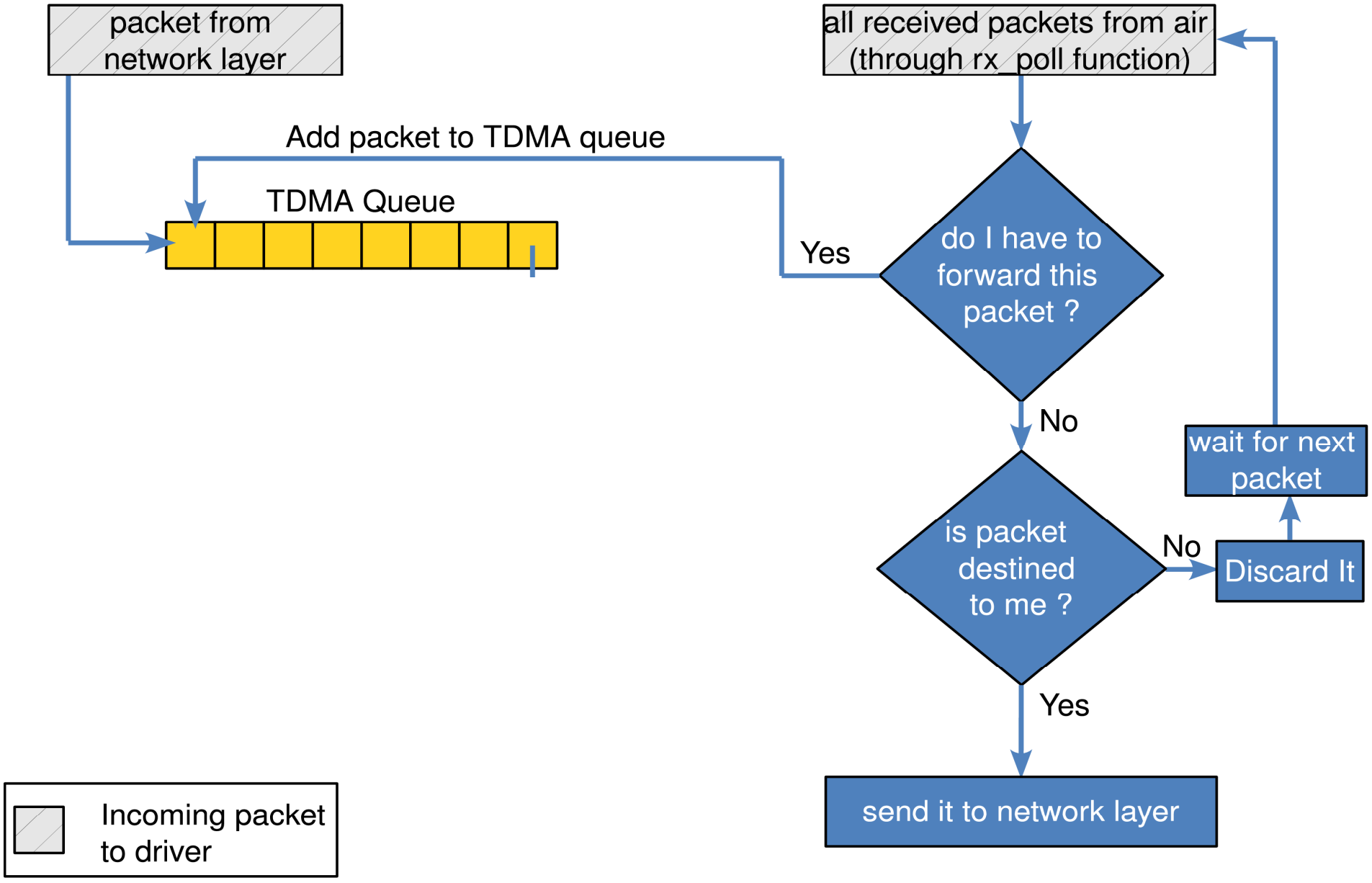
- TDMA queuing mechanism - Implemented and Tested
- TDMA slotting structure - Implemented and Tested
- Multi-hop packet routing - Implemented and Tested
- Plugged in TDMA schedule header and data header along with routing tree elements - Implemented and Tested
- Multi-hop schedule dissemination - Implemented and Tested
- Multi-hop time synchronization - Implemented and Tested
- Node join mechanism – Not Implemented
- Multiple queue implementation – Not Implemented
- Packet Filtering based on destination MAC address and discarding packet with CRC and PHY error - Tested

TDMA Queuing Mechanism

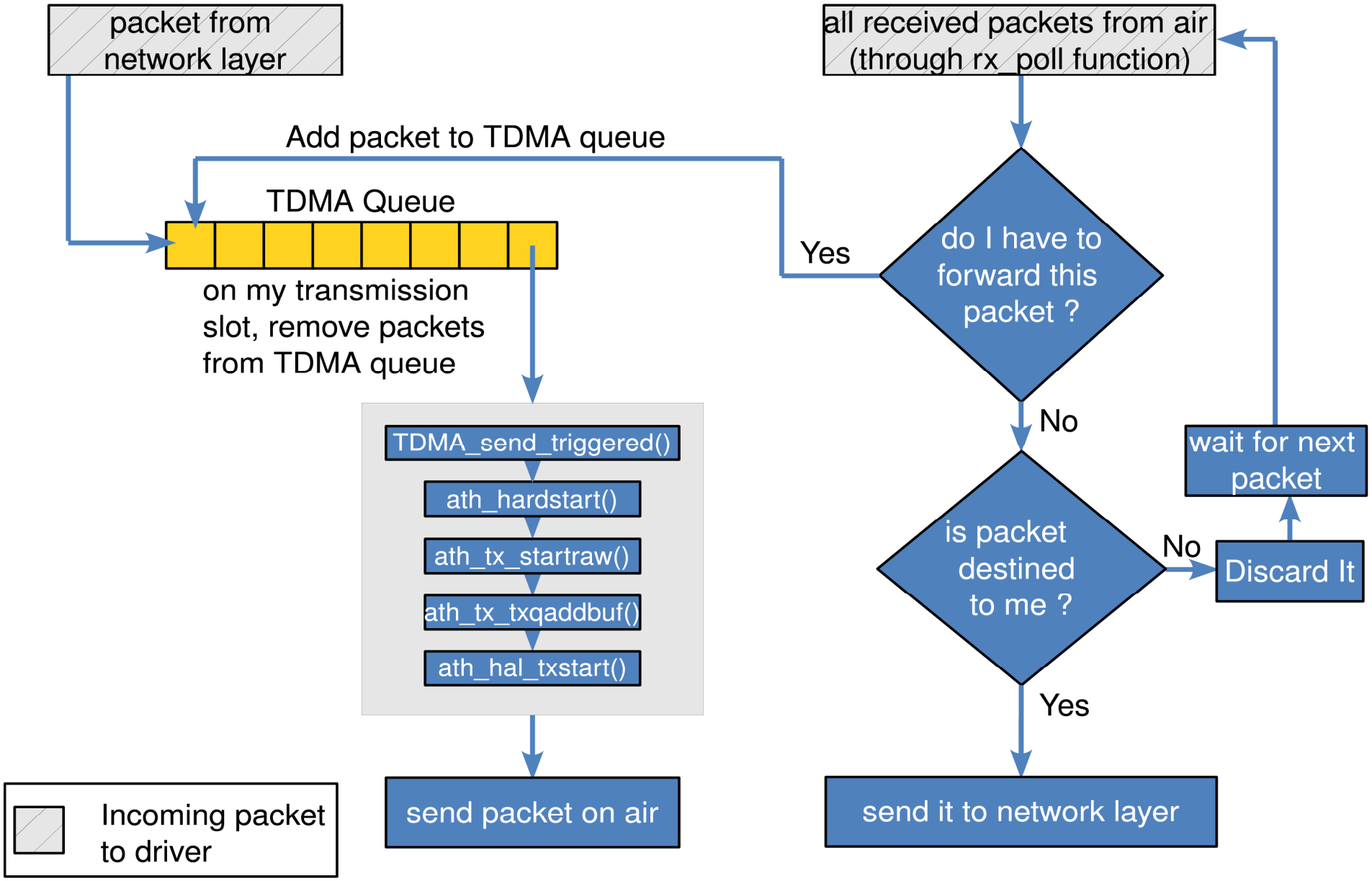




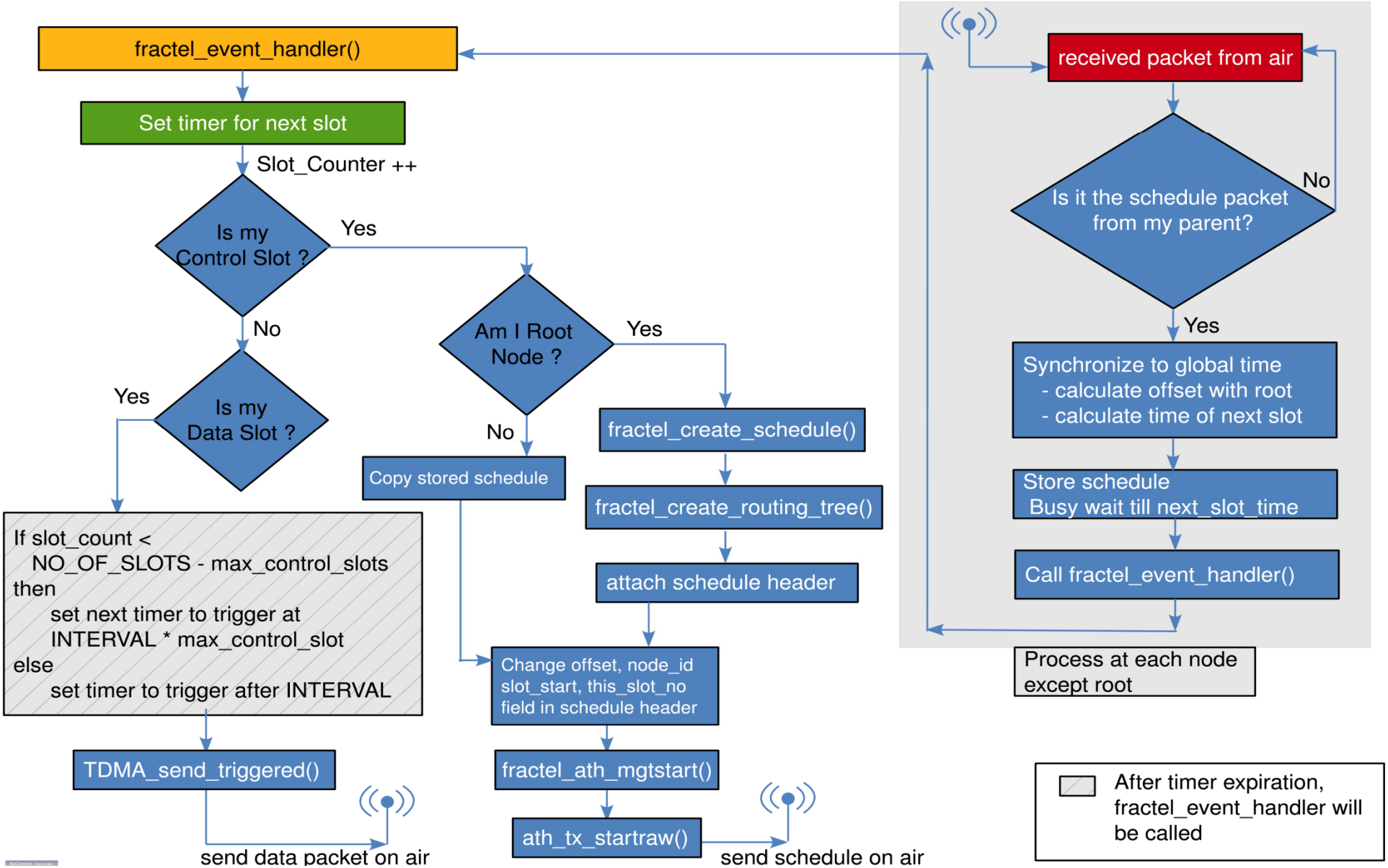
TDMA Queuing Mechanism

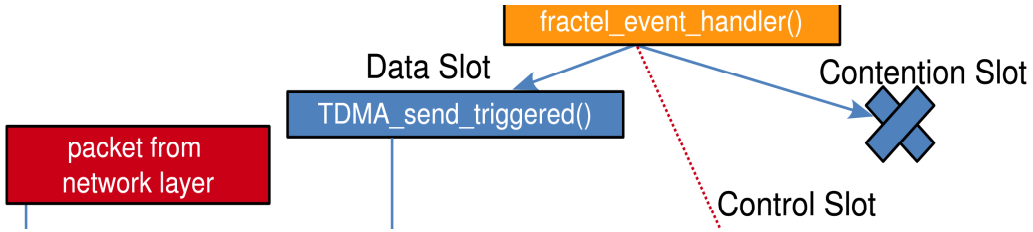


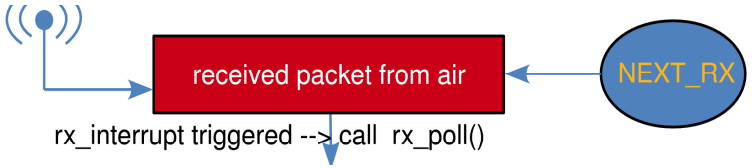
TDMA Queuing Mechanism

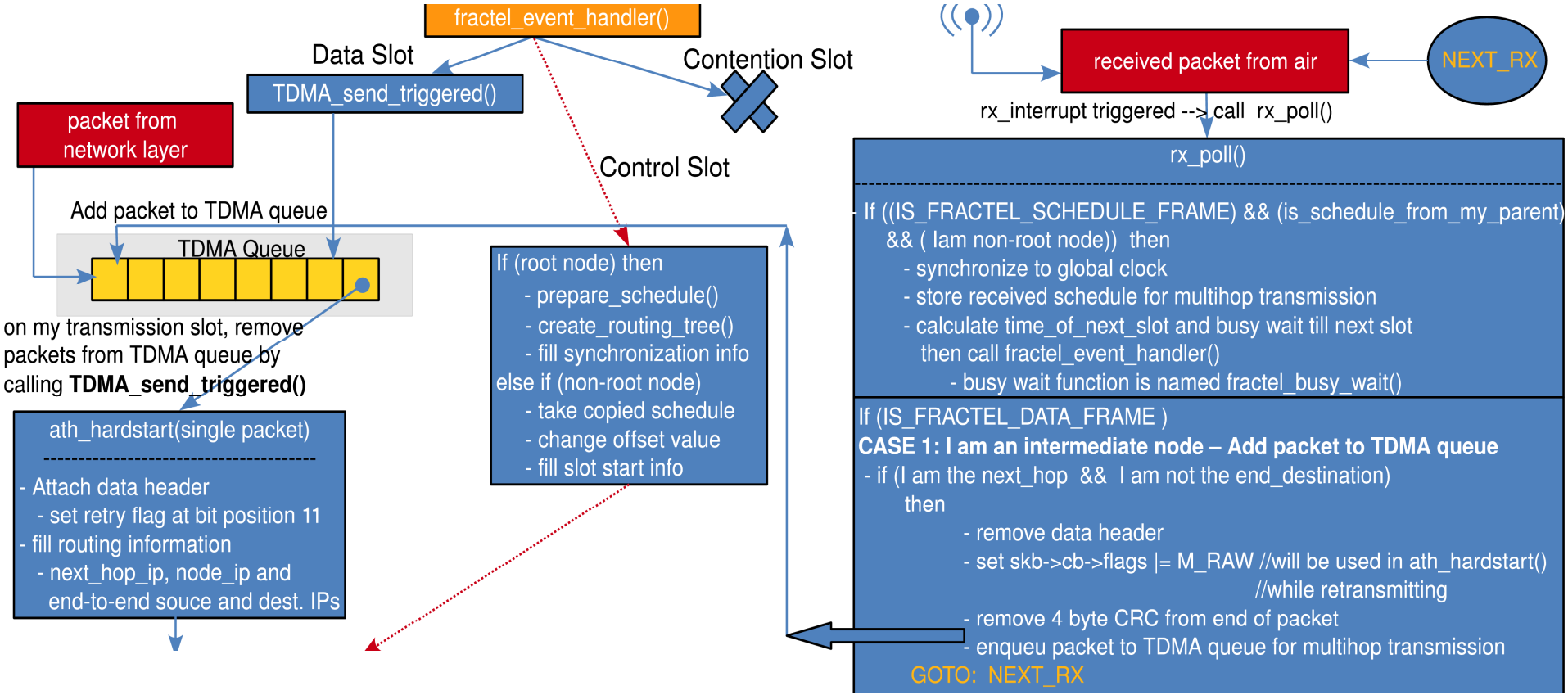


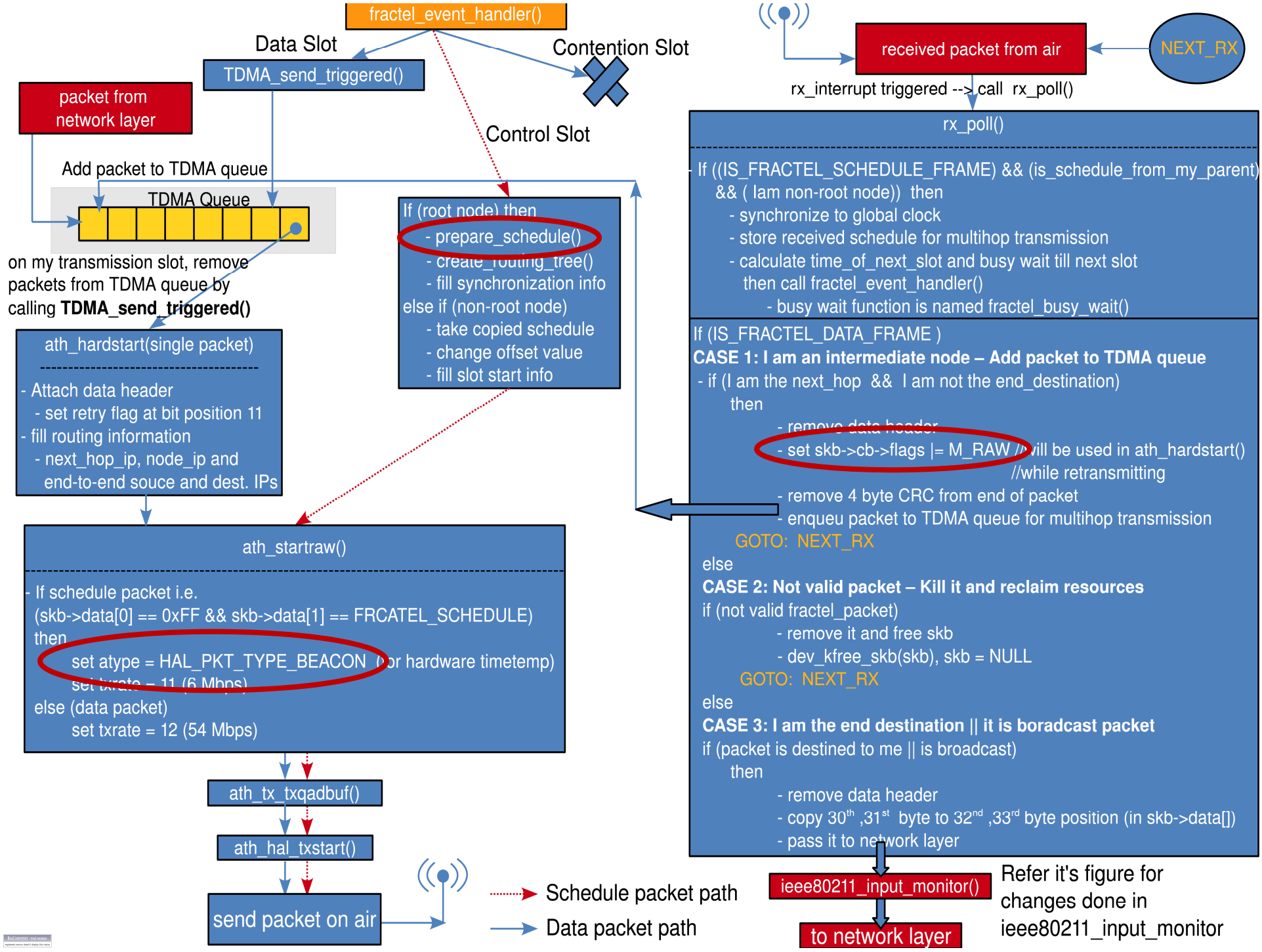
TDMA Slotting Structure




fractel_event_handler()
Data Slot
Contention Slot
TDMA_send_triggered()
packet from
network layer
Control Slot


received packet from air
NEXT_RX
rx_interrupt triggered --> call rx_poll()

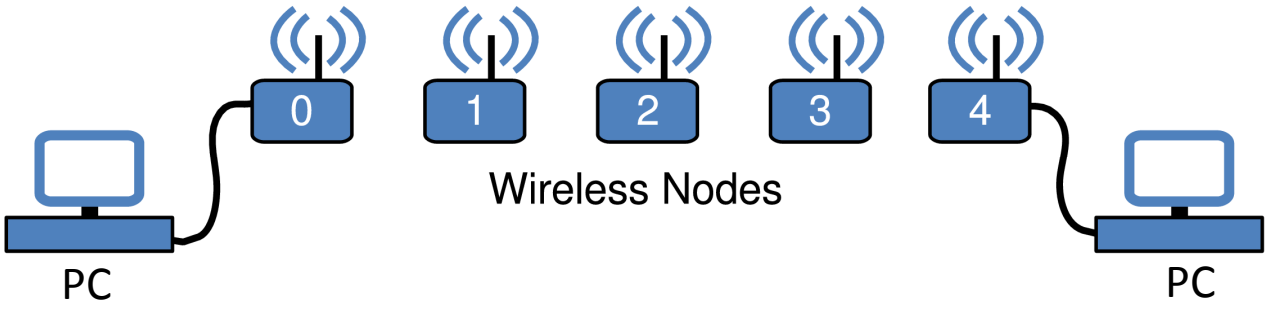

fractel_event_handler()
Data Slot
Contention Slot
TDMA_send_triggered()
packet from network layer
Control Slot
Add packet to TDMA queue
TDMA Queue
on my transmission slot, remove packets from TDMA queue by calling TDMA_send_triggered()
ath_hardstart(single packet)
- Attach data header
- set retry flag at bit position 11
- fill routing information
- next_hop_ip, node_ip and end-to-end souce and dest. IPs
If (root node) then
- prepare_schedule()
- create_routing_tree()
- fill synchronization info
else if (non-root node)
- take copied schedule
- change offset value
- fill slot start info
received packet from air
NEXT_RX
rx_interrupt triggered --> call rx_poll()
rx_poll()
If ((IS_FRACTEL_SCHEDULE_FRAME) && (is_schedule_from_my_parent) && (Iam non-root node)) then
- synchronize to global clock
- store received schedule for multihop transmission
- calculate time_of_next_slot and busy wait till next slot then call fractel_event_handler()
- busy wait function is named fractel_busy_wait()
If (IS_FRACTEL_DATA_FRAME)
CASE 1: I am an intermediate node – Add packet to TDMA queue
- if (I am the next_hop && I am not the end_destination) then
- remove data header
- set skb->cb->flags |= M_RAW //will be used in ath_hardstart() //while retransmitting
- remove 4 byte CRC from end of packet
- enqueu packet to TDMA queue for multihop transmission
GOTO: NEXT_RX


fractel_event_handler()
Data Slot
Contention Slot
TDMA_send_triggered()
packet from network layer
Control Slot
Add packet to TDMA queue
TDMA Queue
on my transmission slot, remove packets from TDMA queue by calling TDMA_send_triggered()
If (root node) then
- prepare_schedule()
- create_routing_tree()
- fill synchronization info
else if (non-root node)
- take copied schedule
- change offset value
- fill slot start info
ath_hardstart(single packet)
- Attach data header
- set retry flag at bit position 11
- fill routing information
- next_hop_ip, node_ip and end-to-end souce and dest. IPs
ath_startraw()
- If schedule packet i.e.
(skb->data[0] == 0xFF && skb->data[1] == FRCATEL_SCHEDULE)
then
set atype = HAL_PKT_TYPE_BEACON (for hardware timetemp)
else (data packet)
set txrate = 12 (54 Mbps)
ath_tx_txqadbuf()
ath_hal_txstart()
send packet on air
received packet from air
NEXT_RX
rx_interrupt triggered --> call rx_poll()
rx_poll()
- If ((IS_FRACTEL_SCHEDULE_FRAME) && (is_schedule_from_my_parent)
&& (Iam non-root node)) then
- synchronize to global clock
- store received schedule for multihop transmission
- calculate time_of_next_slot and busy wait till next slot
then call fractel_event_handler()
- busy wait function is named fractel_busy_wait()
If (IS_FRACTEL_DATA_FRAME)
CASE 1: I am an intermediate node – Add packet to TDMA queue
- if (I am the next_hop && I am not the end_destination)
then
- remove data header
- set skb->cb->flags |= M_RAW // will be used in ath_hardstart()
//while retransmitting
- remove 4 byte CRC from end of packet
- enqueu packet to TDMA queue for multihop transmission
GOTO: NEXT_RX
else
CASE 2: Not valid packet – Kill it and reclaim resources
if (not valid fractel_packet)
- remove it and free skb
- dev_kfree_skb(skb), skb = NULL
GOTO: NEXT_RX
else
CASE 3: I am the end destination || it is boradcast packet
if (packet is destined to me || is broadcast)
then
- remove data header
- copy 30th ,31st byte to 32nd ,33rd byte position (in skb->data[])
- pass it to network layer
ieee80211_input_monitor()
to network layer
Refer it's figure for changes done in ieee80211_input_monitor
Schedule packet path
Data packet path

Problems Faced

- Monitor mode communication
  - Disabling NAV field effect
  - Disabling sequence number stamping by hardware
  - Enabled receive side path for normal communication
- Hardware time stamping
  - Tweaking the hardware to consider schedule packet as 802.11 beacon
- Raw packet generation at MAC layer

Experiment Setup



Description	Bytes	Time (µsec)
UDP Payload	1470	217.77
UDP Header	8	1.185
IP Header	20	2.962
Ethernet Header	14	2.074
CRC Trailer	4	0.592
Fractel Data Header	32	4.740
PLCP Header	-	20
Total	-	249.767

Transmission Rate: 54Mbps
3 control, 8 contention and 92 data slots

Theoretical UDP throughput for 4-Hop-2msec-slot

Slot size = 1900µs (2000 – 100µs guard band)

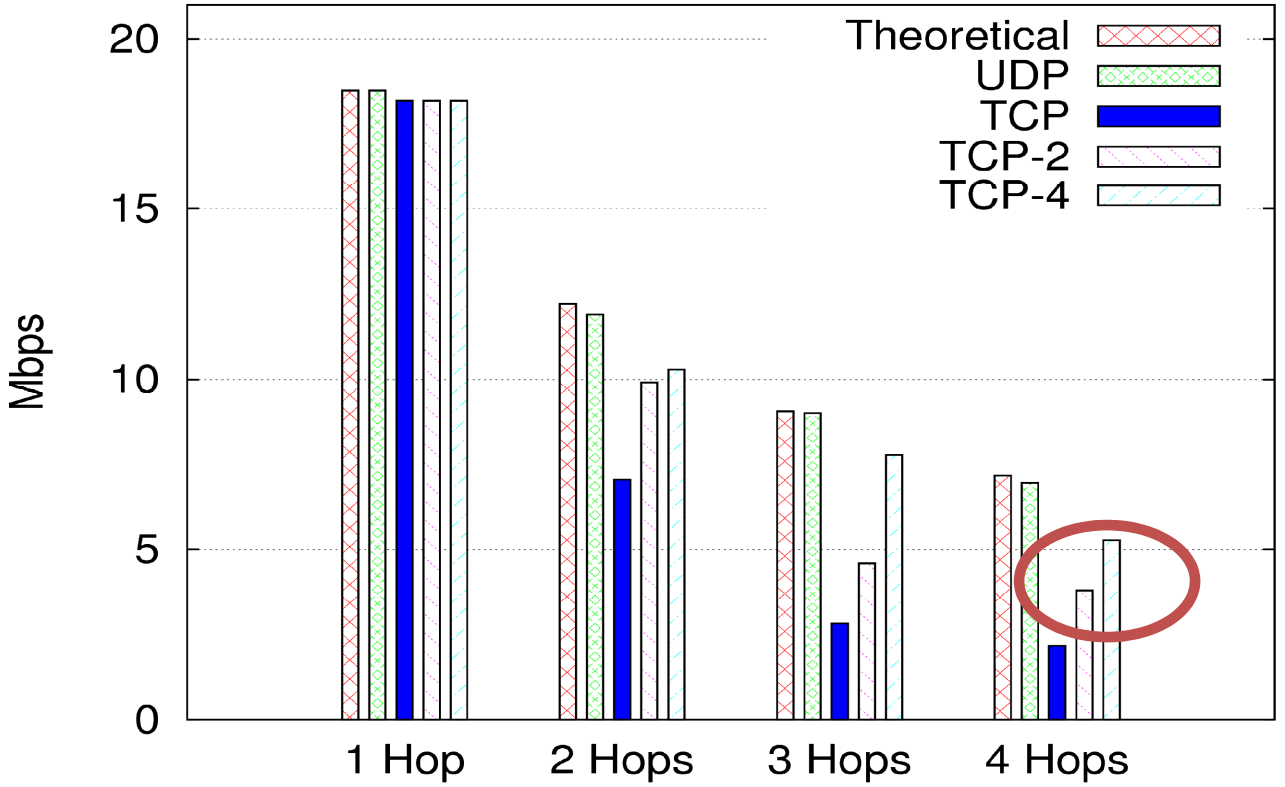
Packets/slot = 1900/249.767 = **floor(7.607)** = 7

Packets/sec = (frames/sec) * (# of slots/frame) * (pkts/slot)
= 5 * (87/5) * 7
= 609

Throughput = 609 * 1470 * 8/(10^6) = **7.16Mbps**

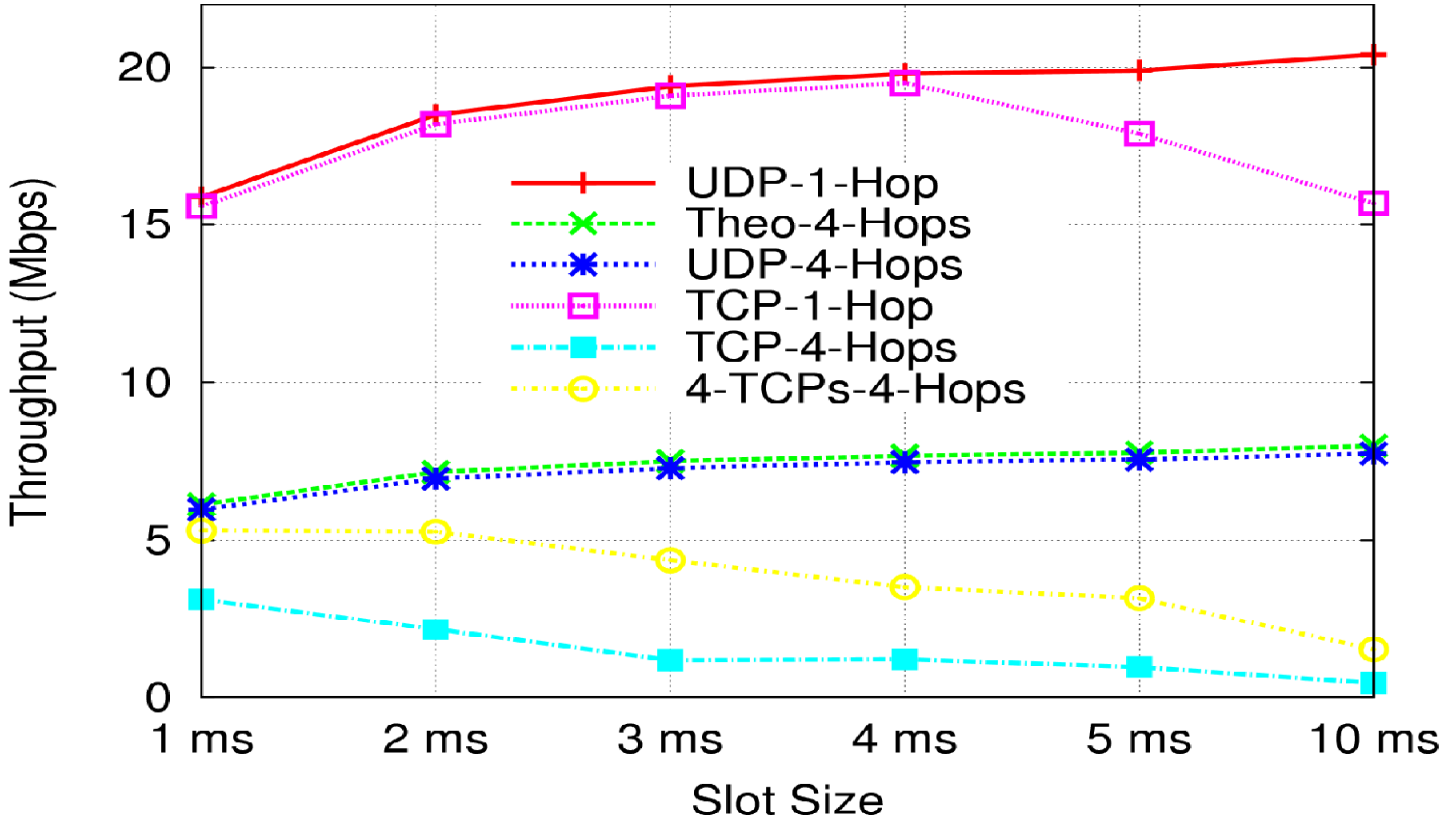
Theoretical Throughput = 7.16 Mbps
Experimental Throughput = 6.93 Mbps

Results – Throughput Vs Hops



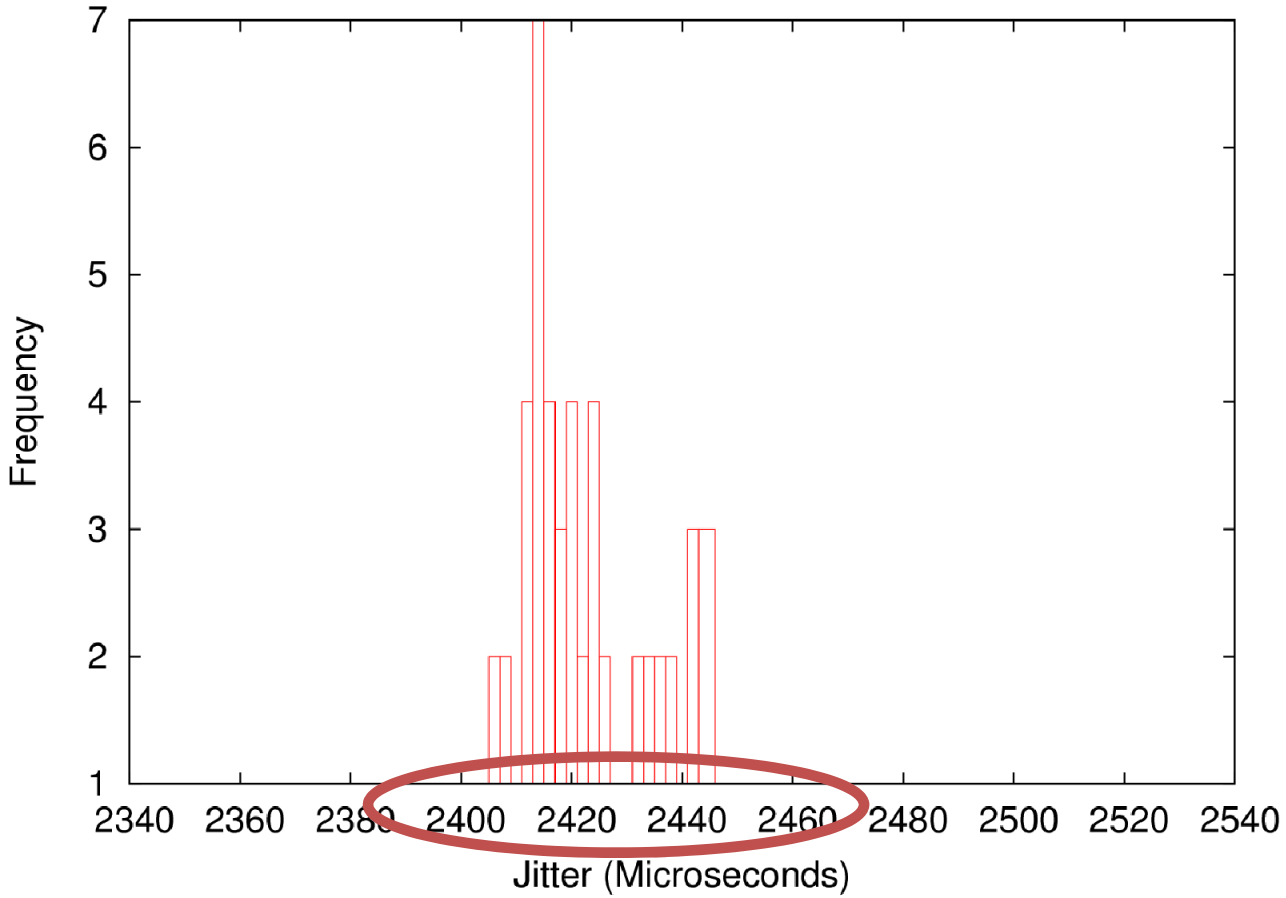
- UDP throughput decreases with increasing Hops
- TCP throughput decreases more drastically with increasing Hops
  - large RTT
  - No per link retransmission
- Multiple TCP connection though gives better performance

Results – Throughput Vs Slot Size



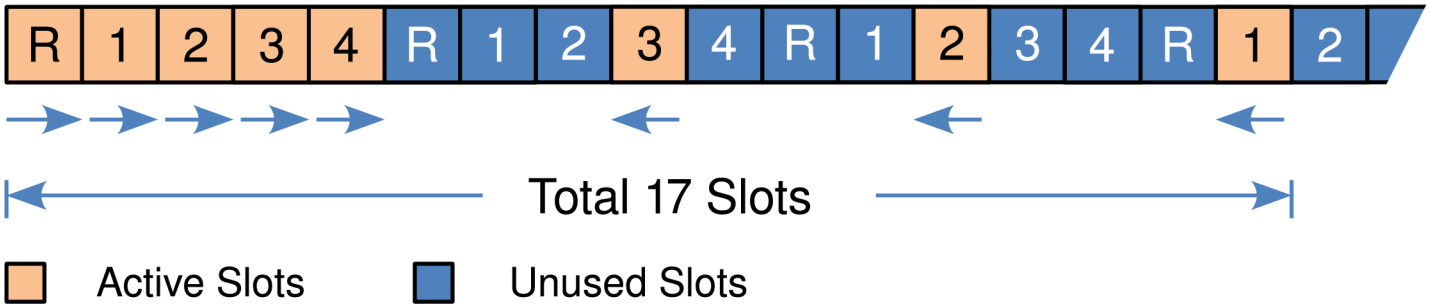
- UDP throughput increases with increase in slot size
  - Reduced overhead
- TCP throughput decreases with increase in slot size
  - large RTT
  - less effect of reduced overhead
- TCP gives good performance for small slot size

Results – Jitter for 4-Hop-5ms slot size



- Observed jitter for 60 sec UDP connection on 4-Hop-5ms slot size was less then 2.5 msec

Results - Justification for RTT



Best Case RTT = x + {(x-1) * (x-2)}

Where X= Number of nodes in the network

- Theoretical best case RTT for 4-Hop topology with 2msec slot size is @ 34 msec (17slots * 2msec/slots)
- We observed very close RTT value during our experimentation

Implications of Results

- The observed UDP throughput is very close to the theoretically calculated value.
- The throughput for single-TCP connection is quite low for multiple hops but multiple-TCP connections together can provide good performance.
- The observed jitter value is very small even for multi-hop topology.
- Avenues exist for e-learning through video conferencing, low cost telephony and internet access desirable for rural areas

Work Done 1 of 2

- Stage 1
  - Understanding madwifi driver
  - Understanding Transmit and Receive path in monitor and Ad-hoc mode
  - Prototype TDMA implementation in Ad-hoc mode
- Stage 2
  - Monitor mode communication
  - Disabling CSMA mechanism
  - Generating packets at MAC layer
  - Enabling channel switching
  - Implementing TDMA frame structure
  - Raw packet transmission; no 802.11 frames

Work Done 2 of 2

- Stage 3
  - Corrected faulty TDMA queuing mechanism (from stage2)
  - TDMA slotting structure
  - Multi-hop packet routing
  - Multi-hop schedule dissemination
  - Multi-hop time synchronization

- Future Work
  - Node Join Mechanism
  - Multiple TDMA queues
  - Design and Implementation of Scheduler

Conclusion

- Modified madwifi device driver
- Implemented Multi-hop TDMA system with
  - Schedule dissemination
  - Time synchronization
- Carried out extensive experimentation with varying slot size and hops
- Gives very good results while playing videos and making voice calls
- Still lot of work need to be done before live deployment

Thank You!