ABSTRACT

SINGHAI, MRUGENDRA. Helios-2: An All-Optical Broadcast LocalArea Network. (Under the direction of Dr. Mladen Vouk and Dr. George Rouskas.)

This thesis describes the details of a new network architecture called Helios-2. It is based on the Hiper-l protocol, a reservation protocol designed to coordinate access to the various channels of a single-hopbroadcast-and-select wavelength-division multiplexing local area network. The Helios-2 network consists of a number of nodes connected through a passive all-optical star coupler which allows a broadcast network. Communication among the nodes in Helios-2 occurs using a scheduled access to the network medium i.e. optical wavelengths.

For a single-hop broadcast-and-select network to be efficient, the bandwidth allocation among the nodes must be dynamically managed. Although a number of protocols have been proposed for this purpose they all suffer from inefficiencies of operating in environments with non-zero processing, tuning and propogation delays. Helios-2 is designed to overcome these inefficiencies.

This work presents different elements of Helios-2 architecture: its state machines, communication frame formats and contents, scheduling algorithm and possible hardware architecture. These elements are optimized for a total hardware based implementation.

The Helios-2 Master State Machine, which controls the operation of a node ina Helios-2 network, has five major operation modes: Time Measurement, Join, Election, Routine and Scheduling. Master state machine controls the operation of the receive and transmit auxiliary state machines of each node.

Helios-2 uses the concept of "Virtual Receivers” to achieve non-preemptive scheduling. "Virtual Receivers” are a set of physical receivers that behave identically in terms of optical tuning. A scheduling algorithm, which is at the heart of Helios-2 network operation, is optimized for hardware based impl

ementation. Simulation was used to evaluate the behaviors of the scheduling algorithm. It was found that the scheduling algorithm proposed here produces schedules very close to optimal schedules.

Finally, an example hardware implementation for Helios-2 is outlined. Implementation would have a FPGA back end and an Optical front. FPGA back end consists of a host interface, Helios-2 state machines and transceivers that interface with optical front end.

Optical front end is based on a Dense Wave Division Multiplexing Slowly Tunable Transmitter - Fast Tunable Receiver combination. The front end may have either optical tuning or electronic tuning.

HELIOS-2

AN ALL-OPTICAL BROADCAST LOCAL AREA NETWORK

by

Mrugendra Singhai

A thesis submitted to the Graduate Faculty of

North Carolina State University

in partial fulfillment of the

requirements for the Degree of

Master of Science

Computer Engineering

Raleigh

2002

APPROVED BY:

_____	_____
_____	_____
Chair of Advisory Committee	Co-chair of Advisory Committee

Biography

Mrugendra Singhai was born and brought up in Indore, India. He received his Bachelors degree in Electronics and Telecommunication Engineering from the Department of Electronics and Telecommunication Engineering at SGS Institute of Technology and Sciences, Indore, India in 1997. He was with Bharti Telenet Inc from 1997 to 2000 where he was part of Access Network Design Group. He joined the Department of Electrical and Computer Engineering at the North Carolina State University, Raleigh, NC in fall of 2000. He is currently working towards completion of his Master's degree in Computer Engineering.

Acknowledgements

I would like to thank Dr. Mladen Vouk for having given me the opportunity for conducting research under his able guidance. His incredible knowledge is only matched by his incredible patience. I am thankful to Dr. George Rouskas for serving as Co-chair on my thesis committee and for taking interest in my work. This thesis would not have been possible without his continual support and advice. I am also thankful to Dr. Paul Franzon and Dr. Arne Nilsson for serving on my thesis committee.

I would like to thank Mr. Dan Stevenson (Director of Advanced Network Research Divison - MCNC) who provided me the opportunity to work at MCNC where I conducted most of the research presented in this thesis. I am thankful to Mr. Mark Cassada (Manager of Hardware Group - ANR, MCNC), Dr. Ilia Baldine and Dr. Pronita Mehrotra for providing vital inputs to my work.

I thank my parents for their love and incredible support, without which this would not have been possible.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Single-Hop Broadcast-and-Select WDM Networks

Wavelength-division multiplexing (WDM) is an optical communication technique that allows many nodes to transmit simultaneously on different wavelength channels, thereby achieving network concurrency and yielding substantially larger throughput than the electro-optical bottleneck. WDM thus can exploit the huge opto-electronic bandwidth mismatch by requiring that each end-node's equipment operate only at electronic rate, but multiple WDM channels from different end-node's may be multiplexed on the same fiber. A key feature of WDM is that the discrete wavelengths form an orthogonal set of carriers which can be separated, routed, and switched without interfering with each other. This use of wavelength and its processing in passive network has led to the wide array of research and development activities in the field of optical networks [1, 2, 3, 4, 5]. A number of experimental prototypes have been and are currently being developed and deployed, and tested [6, 7].

WDM networks have evolved from Point-to-Point WDM systems and Wavelength Add/Drop Multiplexer systems (mainly used for WAN and MAN applications) to Fiber and Wavelength Crossconnects systems (mainly used for LAN applications). Fiber and Wavelength Crossconnect devices fall under the following three broad categories:

- Passive Star - It is a "broadcast" device. Signal that is inserted on a given wavelength from an input fiber port will have its power equally divided among all output ports.
- Passive Router - It can separately route each of several wavelength on an input fiber to the same

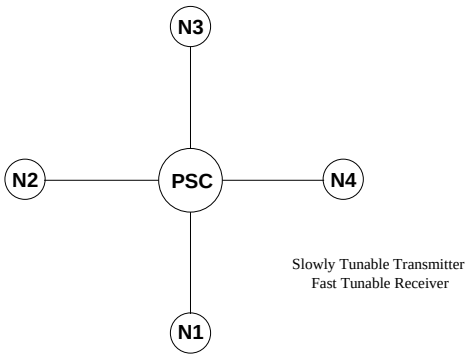


Figure 1.1: A Helios-2 Network

wavelengths on separate output fibers.

- Active Switch - It is a wavelength-routing switch.

Two general architecture forms [2, 3] that have been most widely used in WDM networks are wavelength-routed networks and broadcast-and-select networks. A wavelength-routed network consists of one or more wavelength-selective elements and have the property that the path that the signal takes through the network is uniquely identified by the wavelength of the signal and the port through which it enters the network.

A broadcast-and-select WDM optical network is constructed by connecting nodes via two-way fibers to a passive star. In this type of network architecture, all inputs are combined in a star coupler and broadcast to all outputs. Topologically, a broadcast-and-select WDM networks are either single-hop or multi-hop [6, 7]. In a single-hop broadcast-and-select WDM network, the communication between each pair of nodes is all-optical and no intermediate buffering is required i.e. single-hop broadcast-and-select WDM networks can be defined as the networks where direct transmission can be achieved from source to destination pair. Single-hop broadcast-and-select WDM networks can be implemented in various different ways, depending on whether transmitters, receivers, or both are tunable.

WDM network architecture of interest in this thesis is Single-Hop Broadcast-and-Select WDM Network. All nodes in this network are assumed to be equipped with Slowly Tunable Transmitter and Fast Tunable Receiver (STT-FTR).

1.2 Helios-2

In a single-hop broadcast-and-select WDM network, transmitter of the sending node and receiver of the destination node must be tuned to the same wavelength for the duration of the packet's transmission [6]. Thus, the problem of coordinating access to the various wavelengths of the network arises. This problem is further complicated by the fact that, at high data rates, propagation delays, processing times, and transceiver tuning times all become non-negligible.

For a single-hop broadcast-and-select network to be efficient, the bandwidth allocation among the contending node must be dynamically managed [8, 9, 10, 11]. Numerous protocols have been proposed for this purpose and they can be broadly classified into two categories: those employing pretransmission coordination [8, 9] and those not requiring any pretransmission coordination [10, 11]. These protocols suffer from inefficiencies of operating in environments with non-zero processing, tuning and propagation delays. Hiper-l is a reservation protocol designed to overcome these inefficiencies while coordinating access to the various channels of a single-hop broadcast-and-select WDM local area network [12]. Hiper-l was developed as a MAC protocol for Helios (Nodes with Fast Tunable Transmitter - Slow Tunable Receiver) network [15].

This thesis uses hiper-l as a MAC protocol. This thesis describes the details of the design of the HiPeR-l protocol as implemented in the Helios-2 (Nodes with Slow Tunable Transmitter - Fast Tunable Receiver) network (Figure 1.1). Helios-2 differs from all other WDM networks currently under development in several respects: it operates within a broadcast-and-select environment, it is collision-free, has reconfiguration and load balancing capabilities [21, 22, 23], and it is packet-switched instead of circuit-switched. Helios-2 has also been optimized for a complete hardware implementation.

The Helios-2 network will consist of a number of end nodes, which are connected to a Passive Star Coupler (PSC), a passive all-optical device that allows us to create a broadcast environment in the network. Communication between nodes will occur on multiple wavelengths; thus the Helios-2 network is a type of single-hop WDM network. The number of wavelengths utilized by the HiPeR-l protocol is assumed to be smaller than the number of nodes in the network.

Communication in a Helios-2 network is collision-free due to the use of a non-preemptive gated

scheduling protocol. We use the concept of virtual receivers, a set of physical receivers (termed as "group") that behave identically in terms of tuning, in order to achieve a non-preemptive gated scheduling protocol [17, 18, 19, 20]. All nodes in helios-2 network transmit data to a group and receive from a group. A scheduling node calculates and disseminates the transmit and receive schedules. There are two types of nodes in a Helios-2 network: candidate nodes, which are eligible to serve as the scheduling node, and slave nodes, which are not.

Each Helios-2 node will be equipped with an optical NIC to facilitate data and control communication between the nodes in the network (All communication will be done using either IPv4 or IPv6 protocol). All operations related to Helios-2 would be realized in this optical NIC through a Master State Machine and some Auxiliary State Machines.

1.3 Thesis Organization

The thesis is organized as follows. In Chapter 2 we describe high level operation of Helios-2. Master state machine and different auxiliary state machines used in Helios-2 operation are introduced. In Chapter 3 we describe the structure of superframe and different frames in Helios-2 network. Definition of different fields in each frame, length of those fields and their use is discussed too. In Chapter 4 we discuss operation of Helios-2 master state machine. In Chapter 5 we discuss operation of different auxiliary state machines used in Helios-2. In Chapter 6 we discuss scheduling algorithm for Helios-2. Performance analysis of this algorithm is also presented. Finally in Chapter 7 we conclude with a example hardware implementation of Helios-2.

2 Helios-2 Network Operation

Helios-2 network architecture is based on the assumption that nodes are equipped with Slowly Tunable Transmitter and Fast Tunable Receiver optics. All nodes in Helios-2 network are divided into groups using virtual receiver concept [17, 18]. Receivers of all nodes belonging to a group behave identically in terms of tuning. This type of network architecture facilitates non-preemptive scheduling [19, 20]. So a node either transmits to a group or receives from a group, as shown in Figure 2.1. All node communicate their per group traffic requirement to a Master node which in turn computes the transmit and receive schedules for all the nodes. This schedule is then distributed to all the nodes for their use.

2.1 Helios-2 Modes of Operation

The operation of a node in the Helios-2 network can be divided into five modes, as shown in Table 2.1. Corresponding to each mode of operation are two auxiliary state machines, the receive and the transmit auxiliary state machines. Each machine begins and ends in the idle state: they are triggered out of the idle state by a signal from the master state machine called master_controller, and they usually terminate by sending a signal back to master_controller and returning to the idle state.

When the network comes up after having been completely powered down, no master node has yet been designated, no frames are traveling, and no synchronization information is available. The first task during this initialization phase is the election of a master node; candidate nodes enter Election mode while slave nodes sleep. The operation of Election Mode assumes that candidate nodes are equipped with slowly tunable transmitters; otherwise, a network administrator must designate the master node.

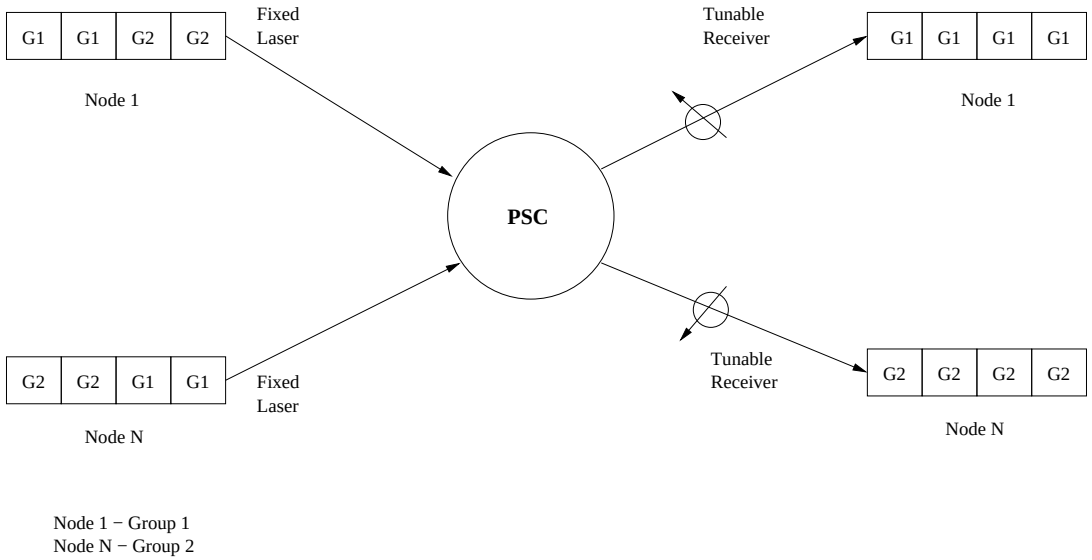


Figure 2.1: Helios-2 Operation

Mode	Function	Auxiliary State Machines	
		Transmit	Receive
Time Measurement	a new node measures its propagation delay to the PSC	**<tm>**	**>tm<**
Join	a new node contacts the master node with its bandwidth requests	**<join>**	**>join<**
Election	a candidate node vies to become the master node	**<elect>**	**>elect<**
Routine	a node transmits and receives according to the schedule	**<routine>**	**>routine<**
Scheduling	same as Routine, plus creates new schedules	**<scheduling>**	**>scheduling<**

Table 2.1: The modes of operation for a Helios-2 node

Once a master node has been elected, it circulates the synchronization & ARP information in SYNCARP frames, enabling other nodes to join the network. A node formally joins the Helios-2 network by proceeding through the Time Measurement and Join modes. In Time Measurement, a node calculates its psc_offset, the propagation delay to the PSC. All times are measured locally, and the transmissions are done in relation to the PSC time. Since collisions can occur only at the PSC, each node uses its psc_offset to ensure that its transmissions reach the PSC at the exact time prescribed by the schedule.

Following Time Measurement a node enters Join Mode. The node first lets the master node know of its presence via the JOIN frame, so that the current schedule can be expanded to include this new demand. The joining node must then wait to hear a new schedule that includes its request.

It is possible for a collision to occur when two or more nodes attempt to join a Helios-2 network at the same time. Two nodes assigned to the same transmit wavelength could experience a collision during Time Measurement (two nodes belonging to same group may transmit a JOIN frame to the master node during the same JOIN window). But the collision does not interfere with the normal operation of the rest of the network; a collision during the transmission of a TM frame (respectively, a JOIN frame) is isolated to the TM window (respectively, the JOIN window). The protocol includes backoff algorithms to resolve such contention.

After successfully joining the network, a new node enters Routine Mode, where it remains indefinitely unless an error condition occurs. During Routine Mode, the receive hardware extracts the data frames (optionally containing control information) by tuning onto different wavelengths according to receive schedule. Meanwhile, the transmit hardware transmits data frames (optionally containing control information) from its group_id queues onto its outgoing wavelength, according to the transmit schedule. These transmissions include sending an OCC payload (within data frame) to the master node, once per superframe, to communicate its packet queue occupancies; from the OCC payload the master node can calculate a schedule. Master node communicates the calculated schedule to nodes through SCHED payload (within data frame). In contrast to the Time Measurement and Join modes, Routine Mode is collision-free. The psc_offset, first measured during Time Measurement, is also measured periodically during Routine Mode, in a collision-free manner.

3 Superframe & Frames

3.1 Superframe

The time required to complete the transmissions of one full schedule in HiPeR-l is referred to as a superframe. A superframe, shown in Figure 3.1, further consists of frames, which are continuous sequences of octets transmitted by nodes on their respective transmit wavelengths. The complete list of the types of frames that may be transmitted during a superframe are shown in Table 3.1. In the following section each frame type is discussed in detail.

3.2 Frame Format

Each frame consists of a header, a variable length payload, and a trailer. Frame structure is illustrated in Figure 3.2, and each field is described in Table 3.2. The header contains the Frame Type indicator, one octet of flags, the payload length indicator, and the source and destination addresses. The trailer contains a timestamp and a CRC32 checksum field. The last column of Table 3.1 shows the possible values that can be placed in the frame type field of the Helios-2 frame header.

Table 3.2 also shows the length of each field in a Helios-2 frame. The payload length field is allocated two octets for future expansion, when it will be possible to use packets longer than 600 octets (the current maximum transfer unit). The flags field contains a number of flags, shown in Table 3.3, that are used by the nodes to indicate the state of the protocol.

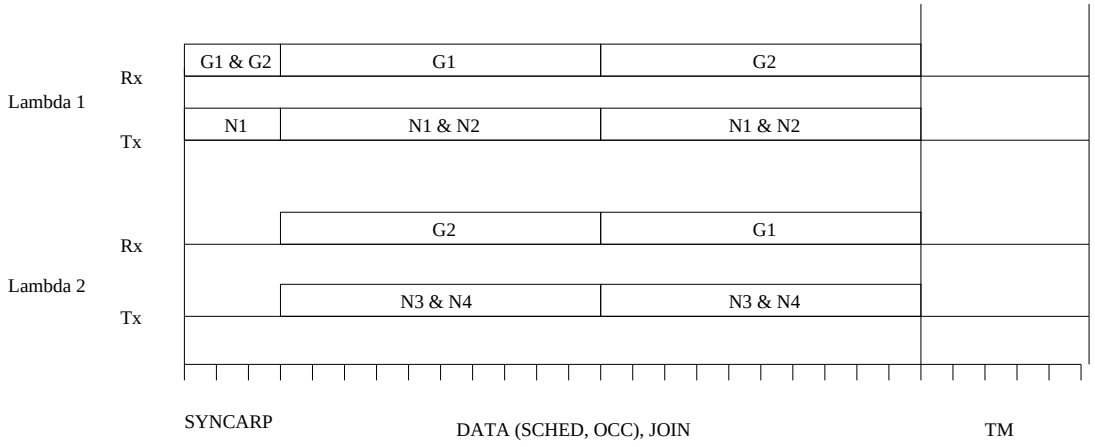


Figure 3.1: Helios-2 Superframe.

Frame	Function	Frame Type
DATA	Carries regular data, occ & sched as payload OCC transmits queue occupancies to the scheduling node SCHED transmits schedule to the slave nodes	0x01
MDATA	Carries multicast data	0x02
TM	Measures roundtrip delay to the PSC	0x03
JOIN	Transmits joining intent of a node to Master Node	0x04
SYNCARP	Carries synchronization information to the nodes & carries MAC address to group index mapping (*g*ARP)	0x05
OAM	Carries error and management information about network	0x06
AVAIL[1,2]	Announces the availability of a scheduling server to become a scheduling node during scheduler election process	0x07

Table 3.1: Frame types and functions

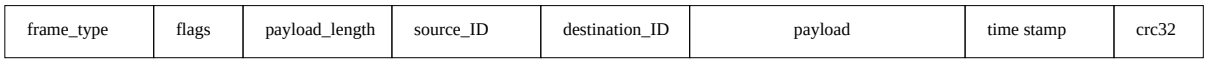


Figure 3.2: Structure of a frame in the Helios-2 network.

Field Name	Description	Field Length (in octets)
frame_type	Frame type indicator	1
flags	Frame flags	1
payload_length	Indicates the length of the frame payload	2
source_ID	MAC address of the originator of the frame	16
destination_ID	MAC address of the destination of the frame	16
payload	contains frame payload	variable $\leq$ 1456
time_out	Time stamp marking departure time of frame	4
crc32	CRC32 checksum of the entire frame	4
Total		*1500*

Table 3.2: Field lengths in Helios-2 frame

Flag Name	Purpose
more_frames	Indicates that more frames of the same type will follow this frame
sched_payload	Indicates the presence of sched payload in data frame.
occ_payload	Indicates the presence of occ payload in data frame.

Table 3.3: Flags in the Helios-2 frame header

3.2.1 Frame Addressing

The Helios-2 addressing scheme is compatible with both IPv4 and IPv6 address formats to allow direct mapping of addresses from those protocols into the Helios-2 MAC addresses. IPv6 addresses can be mapped directly onto the Helios-2 MAC addresses and used as a replacement for MAC addresses. IPv4 addresses will require padding as described in Section 2.5.4 of RFC2373. In short, an IPv4 address will be represented as eighty 0's, followed by sixteen 1's, followed by the IPv4 address of the node interface.

Similarly, multicast addresses can be used as destination MAC addresses for multicast communications in Helios-2. Helios-2 will utilize the link-local multicast addresses reserved in IPv4 and IPv6. For the all-nodes multicast group used internally for sending signaling messages, Helios-2 will utilize the following values:

IPv4: *224.0.0.250*

IPv6: *ff02:1*

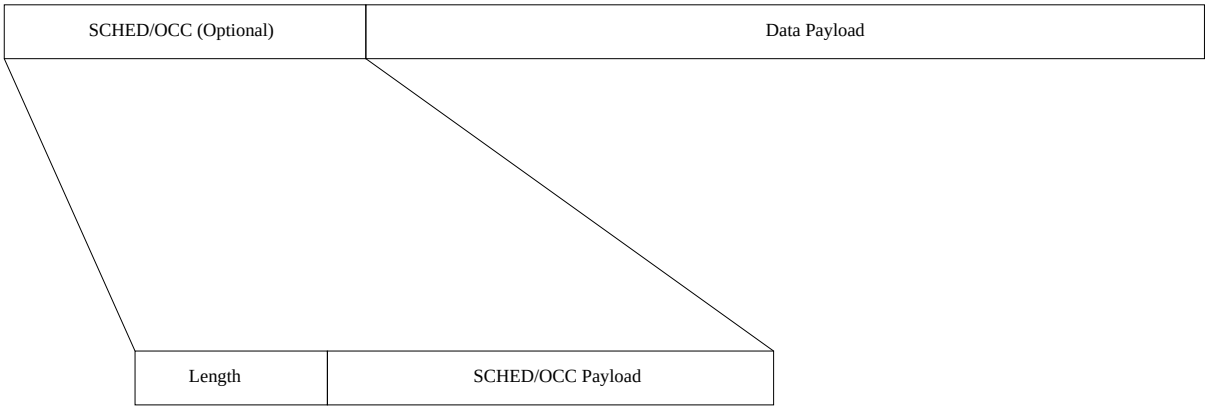


Figure 3.3: Data Frame payload.

3.3 DATA Frame

Payload of the DATA frame contains an IPv4 or IPv6 packet (Figure 3.3). Use of the timestamp field is optional. DATA frame also optionally carries SCHED & OCC payload embedded within it. A length field (1 octet) before the SCHED/OCC payload contains the length(in octets) of SCHED/OCC payload.

SCHED is sent to the each group of nodes once every superframe by the Master Node (shown Figure 3.4). SCHED transmitted for a particular group_id g_i will only contain node schedules for those nodes transmitting on wavelength g_i. Upon receipt of SCHED, each node stores its own schedule until the time comes to start using the new schedule. Special flags in the header indicate the transition phase from one schedule to the next.

As shown in Figure 3.4, SCHED payload itself consists of a header, receive schedule of the group and the individual transmit schedules of the nodes in the group. Table 3.4 describes each field in the SCHED payload. The flags field of the SCHED payload contains several single-bit flags, which are described in Table 3.5.

The structure of the OCC payload is shown in Figure 3.5. Table 3.6 describes the fields of the OCC payload in detail. Each node in the network informs the scheduling node of its packet queue occupancies by transmitting an OCC. Using this aggregate information, the scheduling node can produce a new

Field Name	Description	Field Length (in octets)
flags	Current state of the schedule and protocol	1
sched_group_id	Group_ID of group for which this schedule schedule is intended	1
switch_count	Countdown to the new schedule (along with active bit)	1
num_tx_schedules	Number of individual node transmit schedules in this frame	1
node_ID	Address of the node for which the following schedule is intended	16
num_tx_schedchunks	Number of transmit schedchunks in the nodes schedule	1
group_ID	group_ID of the transmit queue for this schedchunk	1
T_tx_start	Offset (in slots, from the start of the superframe) of the first slot in which the node may transmit on this wavelength	2
T_tx_last	Offset (in slots, from the start of the superframe) of the last slot in which the node may transmit on this wavelength	2
num_rx_sched	Number of receive schedchunks for the group	1
rcv_lambda	Receive wavelength for this schedchunk	1
T_rcv_start	Offset (in slots, from the start of the superframe) of the firstslot in which the group may receive on this wavelength	2
T_rcv_last	Offset (in slots, from the start of the superframe) of the lastslot in which the group may receive on this wavelength	2

Table 3.4: SCHED payload fields

Flag name	Purpose
active_bit	Indicates whether the information in this SCHED payload is for current (1) or future (0) use

Table 3.5: SCHED flags

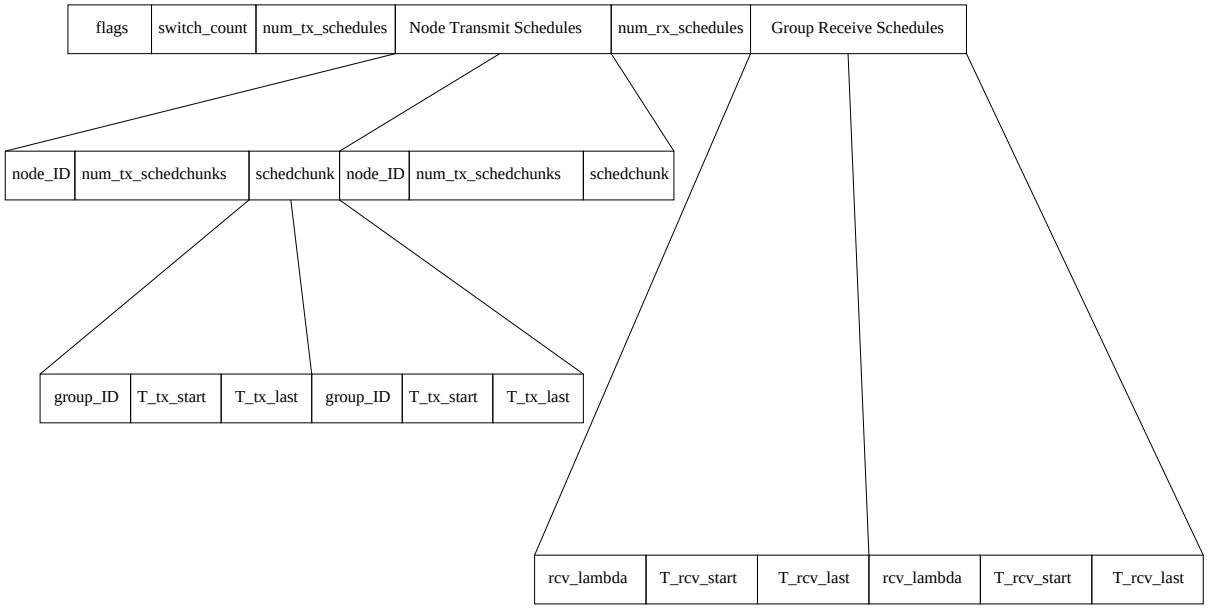


Figure 3.4: SCHED payload

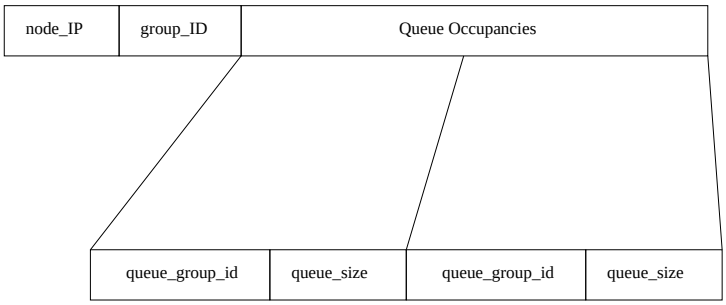


Figure 3.5: OCC payload

schedule that better accommodates node's current load demands. The scheduling node must always reserve enough time on its receive schedule for each node in the network to send its OCC information.

3.4 MDATA Frame

The MDATA frame payload contains an IPv4 or IPv6 multicast packet. Use of the timestamp field is optional.

Field Name	Description	Field Length (in octets)
node_IP	IP address of the source node	16
group_id	Group of which this node is part	1
sched_node_ID	Scheduling node's MAC address	16
sched_lambda	Scheduling node's transmit wavelength	1
queue_group_ID	Group_id of the queue	1
queue_size	Queue size of the group	2

Table 3.6: OCC payload fields

3.5 SYNCARP Frame

The SYNCARP frame is broadcast to all node at the start of superframe. It carries generic synchronization information, MAC address, IP address and group_id for all nodes in the network (shown in Figure 3.6). SYNCARP frames are transmitted on Master Node's lambda at the start of each superframe . SYNC portion of frame contains timing information of Join slots in superframe corresponding to different wavelengths while ARP portion of frame contains MAC address, IP address and group_id mapping for each node in the network. SYNCARP frame also carries tm_bit in the flags field (payload header) indicating whether a TM slot is attached at the end of superframe.

Each SYNCARP frame consists of the Helios-2 header, the SYNCARP payload, and the trailer. Table 3.7 describes each field in the SYNCARP frame payload. The flags field of the SYNCARP frame contains several single-bit flags, which are described in Table 3.8.

3.6 TM Frame

A TM window is a quiet time provided on each wavelength at the end of a schedule in order to allow nodes to measure their delay to the PSC, called the psc_offset. A node transmits a timestamped TM frame to itself during the TM window; the difference between the timestamp and the receipt time of the TM frame is the roundtrip delay to the PSC. The psc_offset is one-half the roundtrip time. A TM frame consists of the Helios-2 header, an empty payload, and the trailer.

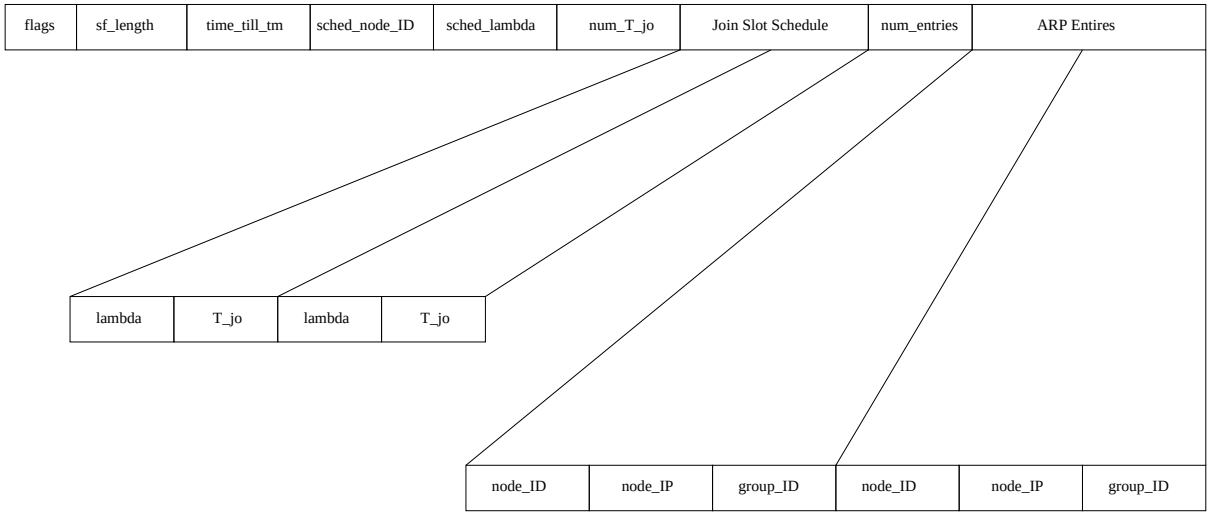


Figure 3.6: SYNCARP Frame payload

Field Name	Description	Field Length (in octets)
flags	Current state of the schedule and protocol	1
sf_length	Length of the superframe in slots	2
time_till_tm	Time (in slots) from the SYNCARP frame to the TM window (If the flags show the presence of a TM window in this superframe)	2
sched_node_ID	Scheduling node's MAC address	16
sched_lambda	Scheduling node's transmit wavelength	1
num_T_jo	Number of JOIN slots for which offset is available in this frame	1
lambda	Tx wavelength of the nodes for which the following JOIN offset is meant	1
T_jo	Offset (in slots, from the start of the superframe) of the JOIN window corresponding to the wavelength ID	2
num_entries	Indicates the number of ARP entries in this frame	1
node_ID	Contains the MAC address of the node in the mapping	16
node_IP	Contains the IP address of the node in the mapping	16
group_ID	Contains the group_ID number in the mapping	1

Table 3.7: SYNCARP frame payload fields

Flag name	Purpose
tm_bit	Indicates the presence (1) or absence (0) of a TM window in this superframe
active_bit	Indicates whether the JOIN slots in this SYNCARP are for current (1) or future (0) use

Table 3.8: SYNCARP frame flags

Field Name	Description	Field Length (in octets)
node_IP	IP address of the source node	16
tx_lambda	Transmit wavelength of node	1
queue_group_ID	Group_id of the queue	1
queue_size	Queue size of the group	2

Table 3.9: Join frame payload fields

3.7 JOIN Frame

When a new node joins the Helios-2 network, it sends a JOIN frame to the scheduler to indicate its presence. A new node that is not yet a part of the Helios-2 network transmits a JOIN frame on the node's transmit wavelength during the JOIN window in the schedule corresponding to its own transmit lambda, given by the T_jo field corresponding to the transmit lambda of the SYNCARP frame. Table 3.9 describes each field in the JOIN frame payload.

3.8 AVAIL Frames

A candidate node participating in scheduler election (Section 5.2) uses AVAIL[1,2] frames to indicate it is available to become the master node in the network. The frames consist of a standard Helios-2 header, an empty payload, and a trailer. Use of the timestamp field is optional.

3.9 OAM Frame

OAM frames are similar in spirit to OAM ATM cells. They carry additional management information between the nodes. The format and exact function of this frame type remains presently undefined.

4 Helios-2 Master State Machine

The Master State Machine also named master_controller is shown in Figure 4.1. There are six main states in master_controller, corresponding to the six modes of operation: Time Measurement, Scheduler Election, Join, Routine, Scheduling, and ERR. In addition, there are five minor states: two are related to Time Measurement (Wait On Election and TM Backoff), while the other three are related to Join (Wait On Election, Join Backoff, and Same Node Sleep). Each state is now described in turn.

4.1 Time Measurement State

The master_controller is awaiting a signal from the auxiliary state machine >tm<. A successful signal is:

- ROUNDTRIP_TIME from >tm<.

  Event: >tm< has obtained the needed data (tm_in an tm_out) so that master_controller can calculate the psc_offset.

  Transition: Move to the Join state and start >join<.

Unsuccessful signals are:

- NO_TM_WINDOW from >tm<.

  Event: None of the SYNCARP frames that >tm< encountered (out of NO_TM_MAX SYNCARP frames) indicated that a tm window corresponding to the nodes transmit wavelength was

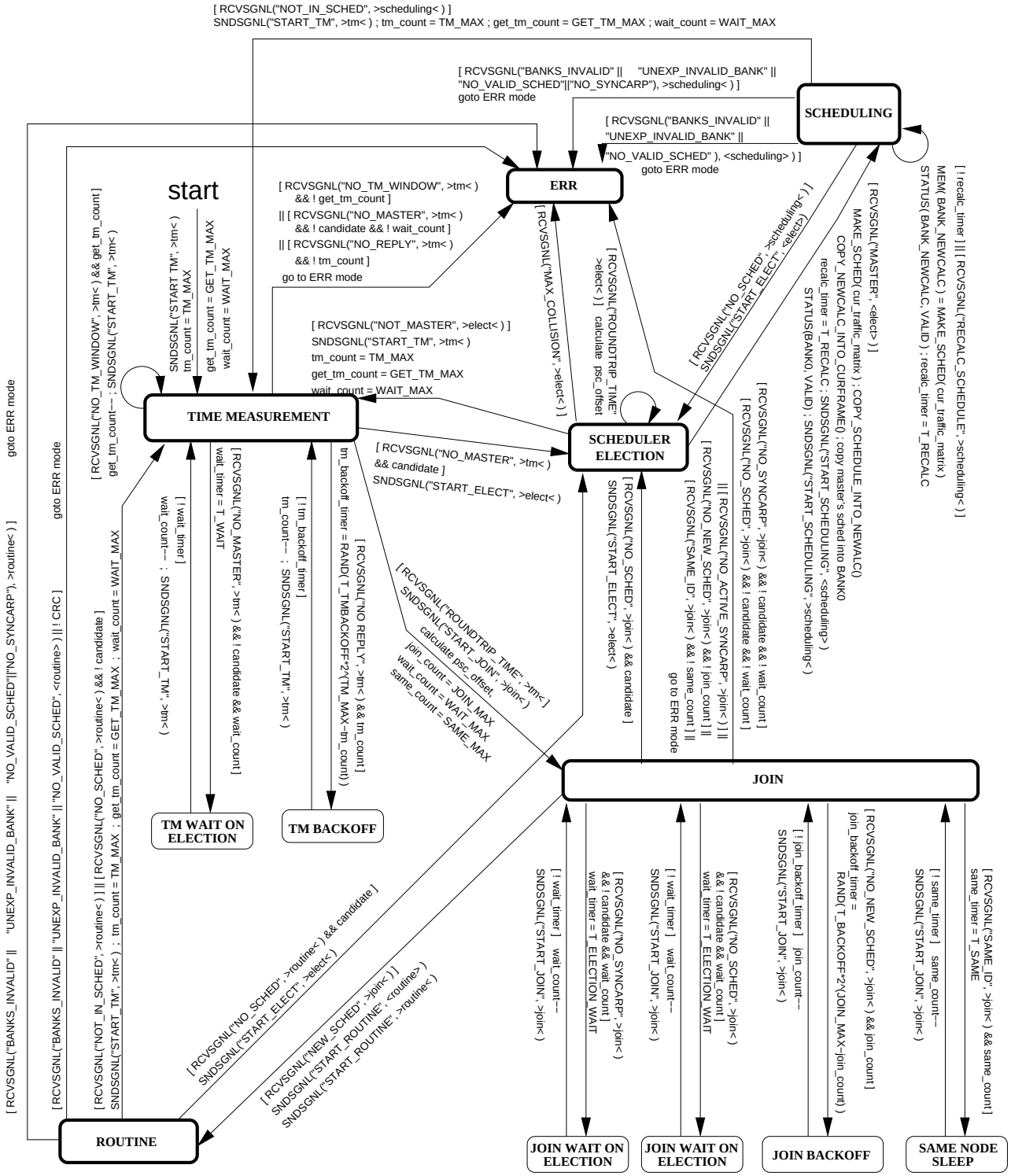


Figure 4.1: Master State Machine: **`master_controller`**

present in the superframe.

Transition: If this failure has occurred less than GET_TM_MAX times, restart >tm< (i.e., self-transition); else, move to ERR state.

- NO_MASTER from >tm<.

  Event: >tm< failed to hear a SYNCARP or DATA with OCC payload within MAX_MASTER_COUNT.

  Transition: If a candidate node, move to Scheduler Election state; else, if this failure has occurred less than WAIT_MAX times, move to Wait On Election state; else, move to ERR state.

- NO_REPLY from >tm<.

  Event: >tm< failed to hear the echo of its tm frame.

  Transition: If this failure has occurred less than TM_MAX times, move to TM Backoff state; else, move to ERR state.

4.2 Scheduler Election State

The master_controller is awaiting a signal from the auxiliary state machine >elect< or <elect>. A successful signal is:

- SCHEDULER from <elect>.

  Event: <elect> has just transmitted the second AVAIL frame, winning the election.

  Transition: Move to the Scheduling state and start >scheduling<.

Unsuccessful signal are:

- NOT_SCHEDULER from >elect<.

  Event: >elect< heard something (a SYNCARP or an AVAIL frame) which caused it to lose the election.

Transition: Move to Time Measurement state and start >tm<.

- MAX_COLLISION from >elect<.

  Event: >elect< experienced collisions for MAX_COLLISION_COUNT while trying to transmit AVAIL1.

  Transition: Move to ERR state.

4.3 Join State

The master_controller is awaiting a signal from the auxiliary state machine >join<. A successful signal is:

- NEW_SCHED from >join<.

  Event: >join< has received a DATA frame with SCHED payload that contains its own my_node_ID.

  Transition: Move to Routine state and start <routine> and >routine<.

Unsuccessful signals are:

- NO_SCHED from >join<.

  Event: >join< failed to hear a DATA frame with SCHED payload within T_GET_SCHED.

  Transition: If a candidate node, move to Scheduler Election state; else, if this failure has occurred less than WAIT_MAX times, move to Wait On Election state; else, move to ERR state.

- NO_SYNCARP from >join<.

  Event: >join< failed to hear a SYNCARP frame within T_GET_SYNCARP.

  Transition: If a candidate node, move to Scheduler Election state; else, if this failure has occurred less than WAIT_MAX times, move to Wait On Election state; else, move to ERR state.

- NO_NEW_SCHED from >join<.

Event: >join< failed to hear a DATA frame with SCHED payload that included scheduling information for my_node_ID.

Transition: If this failure has occurred less than JOIN_MAX times, move to Join Backoff state; else, move to ERR state.

- SAME_ID from >join<.

  Event: >join< heard a DATA frame with SCHED payload that included scheduling information for my_node_ID.

  Transition: If this failure has occurred less than SAME_MAX times, move to Same Node Sleep state; else, move to ERR state.

- NO_ACTIVE_SYNCARP from >join<.

  Event: None of the SYNCARP frames that >join< encountered (out of INACTIVE_MAX SYNCARP frames) had the active_bit set.

  Transition: Move to ERR state.

4.4 Routine State

The master_controller could remain in this state indefinitely, while <routine> transmits according to the transmit schedule and >routine< receives incoming data and signaling frames according to the receive schedule. Only an unsuccessful signal (triggered by an error condition in either <routine> or >routine<) will cause a transition out of the Routine state. Possible unsuccessful signals are:

- BANKS_INVALID from <routine>.

  Event: <routine> was just started, but was unable to transmit anything at all, because TX_BANK0 was invalid.

  Transition: Move to ERR state.

- UNEXP_INVALID_BANK from <routine>.

  Event: <routine> completed transmissions for the current schedule (held in cur_tx_bank) and discovered that cur_tx_bank had been marked invalid.

  Transition: Move to ERR state.

- NO_VALID_SCHED from <routine>.

  Event: <routine> was ready to switch from an out-of-date schedule to a new one, but discovered that the reserve memory bank was marked invalid.

  Transition: Move to ERR state.

- BANKS_INVALID from >routine<.

  Event: >routine< was just started, but was unable to transmit anything at all, because RCV_BANK0 was invalid.

  Transition: Move to ERR state.

- UNEXP_INVALID_BANK from >routine<.

  Event: >routine< completed transmissions for the current schedule (held in cur_rcv_bank) and discovered that cur_rcv_bank had been marked invalid.

  Transition: Move to ERR state.

- NO_VALID_SCHED from >routine<.

  Event: >routine< was ready to switch from an out-of-date schedule to a new one, but discovered that the reserve memory bank was marked invalid.

  Transition: Move to ERR state.

- NO_SCHED from >routine<.

  Event: >routine< failed to hear a DATA frame with SCHED payload within T_GET_SCHED.

Transition: If candidate then enter SCHEDULER ELECTION else enter TIMING MEASUREMENT.

- NO_SYNCARP from >routine<.

  Event: >routine< failed to hear a SYNCARP frame at the beginning of the superframe.

  Transition: Move to ERR state.

- NOT_IN_SCHED from >routine<.

  Event: >routine< heard a DATA frame with SCHED payload that failed to include scheduling information for my_node_ID.

  Transition: Start the TIMING MEASUREMENT and proceed through JOIN to rejoin the network.

4.5 Scheduling State

The master_controller could remain in this state indefinitely, while <scheduling> transmits according to the transmit schedule and >scheduling< receives incoming data and signaling frames according to receive schedule. Only an unsuccessful signal (triggered by an error condition in either <scheduling> or >scheduling<) will cause a transition out of the Scheduling state. Possible unsuccessful signals, listed below, are identical to those listed for Routine State in Section 4.4, except that they come from the state machines <scheduling> or >scheduling<.

- BANKS_INVALID from <scheduling>.

  Event: <scheduling> was just started, but was unable to transmit anything at all, because TX_BANK0 was invalid.

  Transition: Move to ERR state.

- UNEXP_INVALID_BANK from <scheduling>.

Event: <scheduling> completed transmissions for the current schedule (held in cur_tx_bank) and discovered that cur_tx_bank had been marked invalid.

Transition: Move to ERR state.

- NO_VALID_SCHED from <scheduling>.

  Event: <scheduling> was ready to switch from an out-of-date schedule to a new one, but discovered that the reserve memory bank was marked invalid.

  Transition: Move to ERR state.

- BANKS_INVALID from **>**scheduling<.

  Event: >scheduling< was just started, but was unable to transmit anything at all, because RCV_BANK0 was invalid.

  Transition: Move to ERR state.

- UNEXP_INVALID_BANK from **>**scheduling<.

  Event: >scheduling< completed transmissions for the current schedule (held in cur_rcv_bank) and discovered that cur_rcv_bank had been marked invalid.

  Transition: Move to ERR state.

- NO_VALID_SCHED from >scheduling<.

  Event: >scheduling< was ready to switch from an out-of-date schedule to a new one, but discovered that the reserve memory bank was marked invalid.

  Transition: Move to ERR state.

- NO_SCHED from >scheduling<.

  Event: >scheduling< failed to hear a DATA frame with SCHED payload within T_GET_SCHED.

Transition: If candidate then enter SCHEDULER ELECTION else enter TIMING MEASUREMENT.

- NO_SYNCARP from >scheduling<.

  Event: >scheduling< failed to hear a SYNCARP frame at the beginning of the superframe.

  Transition: Move to ERR state.

- NOT_IN_SCHED from >scheduling<.

  Event: >scheduling< heard a DATA frame with SCHED payload that failed to include scheduling information for my_node_ID.

  Transition: Start the TIMING MEASUREMENT and proceed through JOIN to rejoin the network.

- RECALC_SCHED from >scheduling<.

  Event: >scheduling< heard a JOIN frame from a node to join the network.

  Transition: Move to >scheduling< state after recalculating schedule and putting that schedule in BANK_NEWCALC (mark this bank VALID).

4.6 TM Wait On Election State

The master_controller arrives at this state from the Time Measurement state because no SYNCARP or DATA with OCC payload were heard on any wavelength and the node is a slave (i.e., cannot participate in scheduler election). The master_controller remains here for a time T_ELECTION_WAIT. There is only one opportunity to exit TM Wait On Election state:

Event: The wait_timer expires.

Transition: Move to the Time Measurement state.

4.7 TM Backoff State

The master_controller arrives at this state from the Time Measurement state, because it did not hear the echo of its TM transmission, possibly due to a collision. The master_controller remains here for a random amount of time (exponential backoff). There is only one opportunity to exit TM Backoff state:

Event: The tm_backoff_timer expires.

Transition: Move to the Time Measurement state.

4.8 Join Wait On Election State

The master_controller arrives at this state from the Join state because no schedules or no syncarp are heard and the node is a slave (i.e., cannot participate in scheduler election). The master_controller remains here for a time T_ELECTION_WAIT. There is only one opportunity to exit Join Wait On Election state:

Event: The wait_timer expires.

Transition: Move to the Join state.

4.9 Join Backoff State

The master_controller arrives at this state from the Join state, because it did not hear a DATA frame with SCHED payload containing scheduling information for my_node_ID. The master_controller remains here for a random amount of time (exponential backoff). There is only one opportunity to exit Join Backoff state:

Event: The join_backoff_timer expires.

Transition: Move to the Join state.

4.10 Same Node Sleep State

The master_controller arrives at this state from the Join state, because before the node could join the network, a DATA frame with SCHED payload was heard containing scheduling information for my_node_ID, possibly meaning that another node in the network possesses the same node ID. The master_controller remains here for a time T_SAME. There is only one opportunity to exit Same Node Sleep state:

Event: The same_timer expires.

Transition: Move to the Join state.

4.11 Error State

4.11.1 Events that trigger the transition to ERR state

A node enters ERR state from another state in the master state machine. Several events can cause the transition into ERR state.

From Master State Time Measurement:

1. The signal “NO_TM_WINDOW” was received from >tm< and get_tm_count == 0. Master_controller has repeatedly attempted to perform time measurement, and it has failed GET_TM_MAX times. Each of these failures was caused by no TM window appearing out of NO_TM_MAX superframes.

2. The signal “NO_MASTER” was received from >tm< and the node is not a scheduling server and wait_count == 0. Whereas a scheduling server immediately moves to Scheduler Election upon receiving the “NO_MASTER” signal, a slave instead transitions to the Wait On Election state for a time T_WAIT – allowing the servers time to elect a scheduler – before re-attempting >tm<. A slave allows a total of WAIT_MAX failures of this type before entering to ERR mode.

3. The signal “NO_REPLY” was received from >tm< and tm_count == 0. Master_controller has repeatedly attempted to perform time measurement, and it has failed TM_MAX times with the “NO_REPLY” error. This error results from the node having failed to receive its own transmission of the TM frame, possibly because either the transmitter or the receiver is broken. Much less likely, it is possible that a collision occurred TM_MAX times during the TM window, even though exponential backoff was used in between attempts at >tm<.

From Master State Join:

1. The signal “NO_ACTIVE_SYNCARP” was received from >join<. The >join< state machine has received INACTIVE_MAX SYNCARP frames with {syncarp.active_bit} not set. This error is likely the result of a malfunction at the master node.

2. The signal “NO_NEW_SCHED” was received from >join<, and join_count == 0. The node has received OLD_SCHED_MAX schedules that failed to contain my_node_ID. This error may have resulted from a collision in the JOIN-OCC frame, meaning that the master node never received the joining node’s request to be included in the schedule. After waiting an exponential back-off, master_controller re-attempts >join<. Master_controller allows JOIN_MAX failures of type “NO_NEW_SCHED” before moving to ERR mode.

3. The signal “NO_SCHED” was received from >join< and the node is not a scheduling server and wait_count == 0. Whereas a scheduling server immediately moves to Scheduler Election upon receiving the “NO_SCHED” signal, a slave instead transitions to the Wait On Election state for a time T_WAIT – allowing the servers time to elect a scheduler – before re-attempting >join<. A slave allows a total of WAIT_MAX failures of this type before entering to ERR mode.

4. The signal “NO_SYNCARP” was received from >join< and the node is not a scheduling server and wait_count == 0. Whereas a scheduling server immediately moves to Scheduler Election upon receiving the “NO_SYNCARP” signal, a slave instead transitions to the Wait On Election state for a time T_WAIT – allowing the servers time to elect a scheduler – before re-attempting >join<. A slave allows a total of WAIT_MAX failures of this type before entering to ERR mode.

5. The signal “SAME_ID” was received from >join< and same_count == 0. A node allows T_SAME failures of this type before entering ERR mode.

From Master State Routine Non-Scheduler:

1. CRC failure
2. The signal “BANKS_INVALID” was received from <routine>. At the start of <routine>, TX_BANK0 should hold a valid schedule, placed there by >join<. Therefore, if <routine> finds TX_BANK0 invalid, it moves to ERR mode.
3. The signal “UNEXP_INVALID_BANK” was received from <routine>. When the transmitting hardware reaches the end of the schedule in cur_tx_bank, it checks the status of cur_tx_bank. If the status is unexpectedly invalid, then >routine< has encountered an error condition and has marked cur_tx_bank invalid in order to silence the transmitter. The node then moves to ERR mode.
4. The signal “BANKS_INVALID” was received from >routine<. At the start of >routine<, RCV_BANK0 should hold a valid schedule, placed there by >join<. Therefore, if >routine< finds RCV_BANK0 invalid, it moves to ERR mode.
5. The signal “UNEXP_INVALID_BANK” was received from >routine<. When the transmitting hardware reaches the end of the schedule in cur_rcv_bank, it checks the status of cur_rcv_bank. If the status is unexpectedly invalid, then >routine< has encountered an error condition and has marked cur_rcv_bank invalid in order to stop transmission. The node then moves to ERR mode.

From Master State Routine Scheduler:

1. CRC failure
2. The signal “BANKS_INVALID” was received from <scheduler>. At the start of <scheduler>, BANK0 should hold a valid schedule, placed there during schedule calculation. Therefore, if <scheduler> finds BANK0 invalid, it moves to ERR mode.

3. The signal “UNEXP_INVALID_BANK” was received from <scheduler>. When the transmitting hardware reaches the end of the schedule in cur_bank, it checks the status of cur_bank. If the status is unexpectedly invalid, then >scheduler< has encountered an error condition and has marked cur_bank invalid in order to silence the transmitter. The node then moves to ERR mode.

4. The signal “BANKS_INVALID” was received from >scheduler<. At the start of >scheduler<, RCV_BANK0 should hold a valid schedule, placed there during schedule calculation. Therefore, if >scheduler< finds RCV_BANK0 invalid, it moves to ERR mode.

5. The signal “UNEXP_INVALID_BANK” was received from >scheduler<. When the transmitting hardware reaches the end of the schedule in cur_rcv_bank, it checks the status of cur_rcv_bank. If the status is unexpectedly invalid, then >scheduler< has encountered an error condition and has marked cur_rcv_bank invalid in order to stop transmission. The node then moves to ERR mode.

4.11.2 Response to Error Conditions

In case of error conditions, Master State Machine would present the error data to the state machine which is responsible for communicating status information to the Host. Host would process the error data and take appropriate action.

5 Helios-2 Auxiliary State Machines

5.1 Time Measurement

When a new node wishes to join a functioning Helios-2 network, it must first synchronize its system time with that of the network through a process called Time Measurement. Next, the node must execute the Join process, which lets the master node know of its presence so that the current schedule can be expanded to include the new node's traffic demands. Figure 5.1 and Figure 5.2show the receive and transmit auxiliary state machines for time measurement, respectively.

To synchronize its system time, a node must calculate its psc_offset, the time needed for a transmission to reach the PSC. The TM frame is the mechanism for achieving this goal. The master node from time to time (at least every TM_FREQUENCY superframes) will place a TM window at the end of a superframe on all wavelengths. The master node will then announce the presence of a TM window by setting a bit in the SYNCARP frame, {syncarp.tm_bit}. Further, the SYNCARP frame includes the duration of time until the TM window will appear({syncarp.time_till_tm}).

A software signal to >tm< begins the Time Measurement process. The nodes listens for either a SYNCARP frame or DATA(OCC) frame to know about the current Master Node transmit lambda. Node then tunes to this Master Node transmit lambda. The node further listens until it hears a SYNCARP frame with the {syncarp.tm_bit} set, indicating that a TM frame is attached to the end of this superframe. It then sets the tm_timer for the amount {syncarp.time_till_tm}, waits for the timer to expire, and then transmits a timestamped TM frame on its receive wavelength. When the node hears its own transmission of the TM frame, it copies the frame's timestamp and the current time into the variables tm_out and

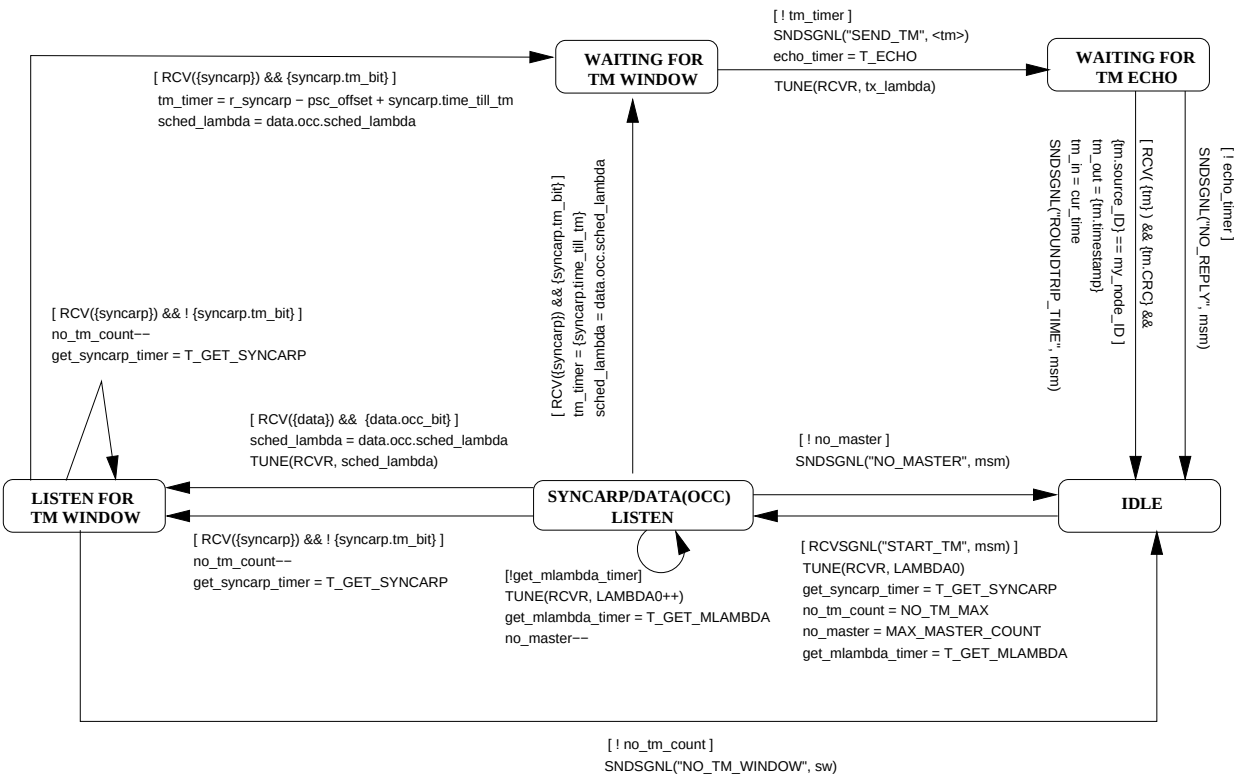


Figure 5.1: Receive auxiliary state machine for time measurement: **>tm<**

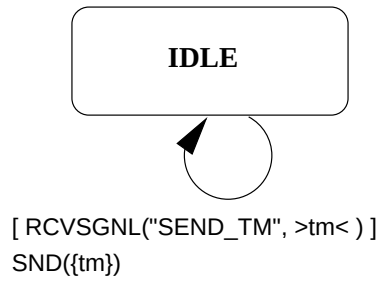


Figure 5.2: Transmit auxiliary state machine for time measurement: **<tm>**

tm_in, respectively, and signals the signaling_controller, which divides the difference of these two values by two to yield the psc_offset.

5.2 Election

Whenever a candidate node fails to detect the presence of a master node, i.e. no SYNCARP/ DATA(OCC) frames are heard, then the candidate node enters Election Mode. This situation can occur when the network comes up after having been completely powered down, or when an operational master node suddenly fails.

Slave nodes, in the meantime, are capable neither of serving as a master node nor of participating in the election of one. Therefore, whenever a slave node fails to detect the presence of a master node, it enters a sleep state for a short time. Upon emerging, it listens for SYNCARP/ DATA(OCC) frames that indicate the presence of a master node, and if none is heard, it sleeps again. A slave node may re-enter the sleep state a fixed number of times before giving up (and moving to Error Mode).

Election Mode, as it is described below, assumes that candidate nodes are equipped with slowly tunable transmitters. If candidate nodes are only equipped with fixed transmitters, then a network administrator must designate the master node.

5.2.1 Scheduler Election with Slowly Tunable Transmitters

Scheduler Election is illustrated in the receive and transmit auxiliary state machines >elect< and <elect>. A software signal to >elect< begins Scheduler Election, moving the state machine from IDLE to SILENT CONTENDER state. The node tunes both its transmitter and receiver to λ_0, listens for a SYNCARP/AVAIL1/AVAIL2 frame, which would indicate the presence of a master node or another candidate node being in ANNOUNCED CONTENDER state; in either case, the node drops out of Scheduler Election and becomes a non-scheduling node. If none is heard within a time T1, the node moves to ANNOUNCED CONTENDER. As such, the node must wait to see SYNCARP frames generated by the newly elected master node and then join the network by proceeding through Time Measurement and Join modes.

If neither a SYNCARP nor an AVAIL is heard within a time T1, the node transmits an AVAIL1

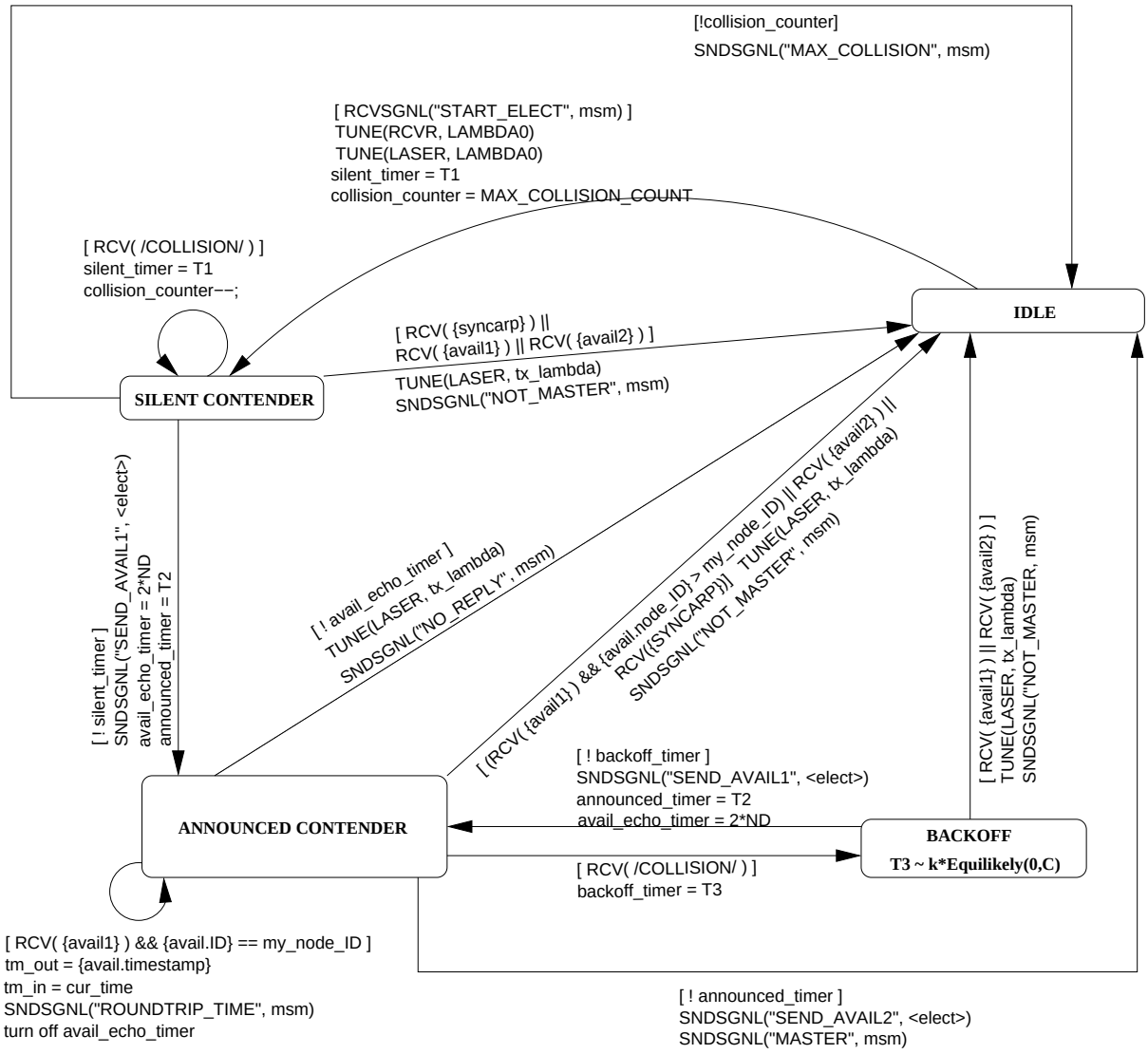


Figure 5.3: Receiver auxiliary state machine for scheduler election: **`>elect<`**

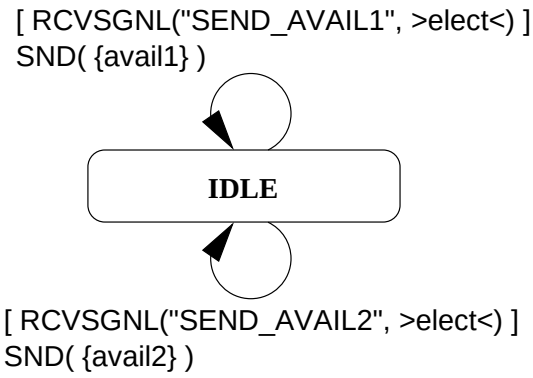


Figure 5.4: Transmitter auxiliary state machine for scheduler election: **<elect>**

frame on λ_0 and, after hearing its own transmission, becomes an ANNOUNCED-CONTENDER. Now it listens on λ_0 for a time T2; so long as the node hears no AVAIL with a higher-valued MAC address ({avail.node_ID}) during the interval T2, it will win the election and become the master node.

However, while in the ANNOUNCED-CONTENDER state, the node could hear an AVAIL with a higher-valued MAC address. In this case, the node will take itself out of the election and become a non-scheduling node; the other node with the higher MAC address has precedence in the scheduler election process.

If, on the other hand, our node detects a collision while in the ANNOUNCED-CONTENDER state, it enters the BACKOFF state for a random amount of time (T3). Other nodes involved in the collision will also enter the BACKOFF state, each choosing a different T3. The node whose T3 expires first will try again to transmit AVAIL1. (If there's a tie, a collision occurs and the involved nodes return to the BACKOFF state.) Any successfully transmitted AVAIL will cause the nodes waiting in BACKOFF to become non-scheduling nodes.

To prevent two or more nodes from mistakenly believing they have emerged victorious from Scheduler Election, the time durations T1 and T2 must obey a particular relationship. Recall that ND is defined to be the longest one-way propagation time between any two nodes. Then we have the following relationship:

$$2 * \mathtt{ND} < \mathtt{T2} < \mathtt{T1}$$

(First Inequality) If more than one node is an ANNOUNCED-CONTENDER, then this inequality en-

sures that the node with the highest-valued MAC address will win. (in particular, it ensures that all nodes with lower-valued MAC addresses will wait long enough in state ANNOUNCED-CONTENDER to hear the AVAIL from the node with highest address.)

(Second Inequality) Suppose node B is busy retuning its receiver to λ_0, transitioning from IDLE to SILENT-CONTENDER, and that the retuning is completed just after node A's AVAIL1 has passed by. Then this inequality will ensure that node B will hear node A's AVAIL2 before node B becomes an ANNOUNCED-CONTENDER itself.

5.2.2 Time Measurement within Scheduler Election

When a node reaches the SILENT-CONTENDER state, both its transmitter and receiver are tuned to λ_0. When a node then transmits AVAIL1, it becomes an ANNOUNCED-CONTENDER and sets the announced_timer for T2. Since the node should hear the echo of its own AVAIL1 transmission (provided its receiver is functional), it takes advantage of this opportunity to execute Time Measurement, that is, to calculate its psc_offset. The longest amount of time a node would have to wait to hear the echo is ND. But the announced_timer requires that the node remain in the ANNOUNCED-CONTENDER state for a time T2 before becoming the master node. Therefore, the avail_echo_timer should be set for a time longer than ND but less than T2. Since the inequality $2 * \texttt{ND} < \texttt{T2}$ must hold (Section 5.2.1), then we could set the avail_echo_timer for $2 * \texttt{ND}$.

If the AVAIL echo is heard, the avail_echo_timer is turned off. Otherwise, the avail_echo_timer will expire before the announced_timer expires, causing the node to abort Scheduler Election and then move into the ERR Mode. Bundling Time Measurement with Scheduler Election produces a master node that knows its psc_offset and has a functioning transmitter and receiver.

5.3 Join

For a new node, the Join process can be broken into two parts: letting the master node know of its presence, and waiting for the master node to include it in the schedule.

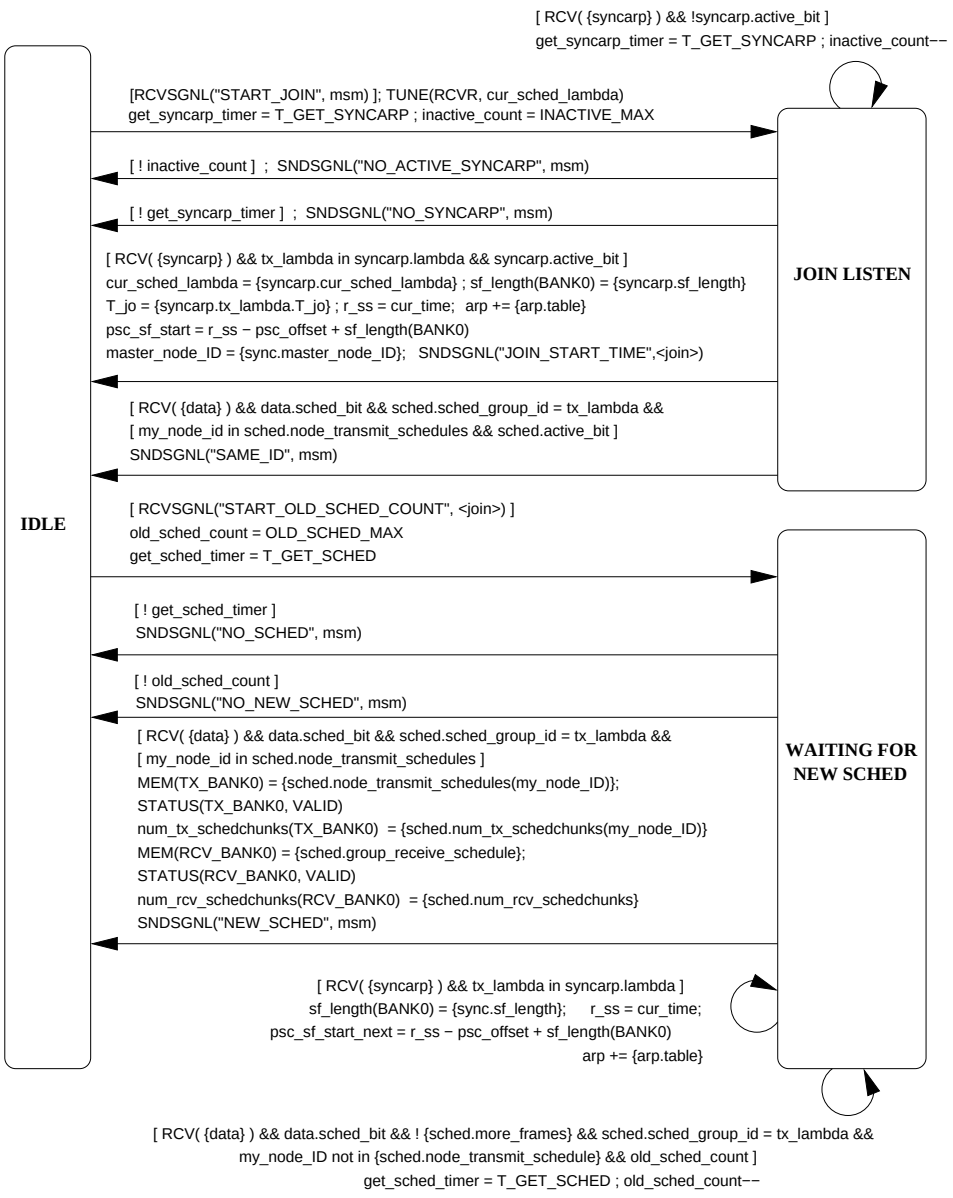


Figure 5.5: Receive auxiliary state machine for join: **>join<**

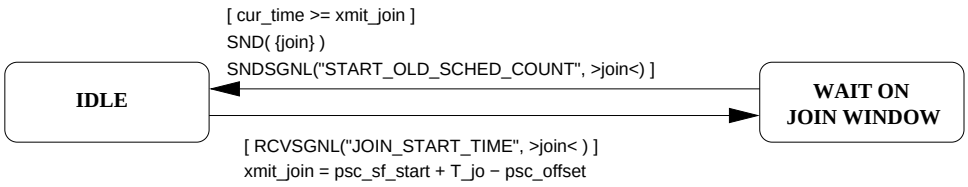


Figure 5.6: Transmit auxiliary state machine for join: **<join>**

5.3.1 Contacting the master node

The new node must learn when the JOIN window will occur, so that it can transmit a JOIN frame to the master node. It listens on Master Node transmit wavelength until it receives a SYNCARP frame with the syncarp.active_bit set. (This measure ensures that the schedule included in the SYNCARP frame contains active join windows.) From SYNCARP, it extracts the following data fields and stores them in its corresponding local variables:

{syncarp.sf_length} : the length in slots of the superframe. (gets copied into sf_length(BANK0))

{syncarp.tx_lambda.T_jo} : the offset time (relative to the start of the superframe) of the JOIN window corresponding to nodes transmit wavelength on the master node's receive schedule. (gets copied into T_jo)

Additionally, the node stores the time from its local clock that the SYNCARP frame arrived (the "receive timestamp") in the local variable r_syncarp. From these values, the node can calculate the time (from its local clock) that the start of the superframe occurred at the PSC:

$$\texttt{psc\_sf\_start} = \texttt{r\_syncarp} - \texttt{psc\_offset} + \texttt{sf\_length}(\texttt{BANK0})$$

The node can now calculate the time (from its local clock) that it must transmit a JOIN frame in order to hit the JOIN window:

$$\texttt{xmit\_join\_occ} = \texttt{psc\_sf\_start} + \texttt{T\_jo} - \texttt{psc\_offset}$$

The node must include a checksum in the JOIN frame so that the master node can determine whether it has received the correct information, since it is possible for a collision to occur when two or more nodes attempt to send a JOIN frame at the same time.

5.3.2 Waiting to be included

Although the new node can directly detect a collision in the JOIN window but once the JOIN frame has been sent, the sure way for the new node to learn that it has successfully been included in the network is to receive a new schedule (via the DATA frame with SCHED payload) which includes its own MAC address (my_node_ID). This new schedule will indicate the windows in which the new node may transmit on his wavelength and windows in which it can receive on different wavelengths.

To handle the case of a collision, the new node sets a counter (old_sched_count) to the value OLD_SCHED_MAX after it transmits a JOIN frame. While waiting to hear a new schedule containing its own MAC address, the node decrements old_sched_count each time it hears a data frame with SCHED payload destined for its group but lacks its MAC address. If the counter should reach zero, the new node notifies the signaling_controller and exits the Join process. The signaling_controller may either retry the Join process or, after repeated failures, simply give up (i.e. enter ERR mode).

If, on the other hand, the new node hears a new schedule containing its own MAC address, then it copies the necessary timing information from the SCHED payload into the corresponding local variable locations, and signals the signaling_controller that it has successfully joined the network (via the "NEW_SCHED" signal).

5.3.3 Backoff Algorithms

If a new node exits the TM receive auxiliary state machine >tm< with the signal "NO_REPLY" to signaling_controller, signaling_controller may execute an exponential backoff algorithm. (A total of TM_MAX failures of this kind are allowed before giving up on Time Measurement and moving to ERR Mode.) The tm_backoff_timer is assigned the value $\texttt{RAND}(1..\texttt{T\_TMBACKOFF} * 2^{\texttt{TM\_MAX}-\texttt{tm\_count}})$. Each time the node picks a random uniformly-distributed number whose bounds are growing larger.

If a new node exits >tm< with the signal "NO_TM_WINDOW" to signaling_controller, then signaling_controller decrements the counter get_tm_count and immediately restarts Time Measurement, without backing off. A total of GET_TM_MAX failures of this kind are allowed before moving to the ERR Mode. (A backoff algorithm could be added to the handling of this type of failure.)

If a joining node exits >join< with the signal "NO_NEW_SCHED" to signaling_controller, then

signaling_controller may execute an exponential backoff algorithm. (A total of JOIN_MAX failures of this kind are allowed before moving to ERR Mode.) The join_backoff_timer is assigned the value of $\texttt{RAND}(\texttt{T\_BACKOFF} * 2^{\texttt{JOIN\_MAX}-\texttt{join\_count}})$.

5.4 Routine

We now describe the operation of the receive and transmit hardware of a nonscheduling node, i.e. candidate and slave nodes. A new node enters Routine Mode once it has successfully joined the network; that is, during >join< it received a data frame with SCHED payload that included its own MAC address in the schedule, and it then exited >join< with the message "NEW_SCHED" to signaling_controller. The main functions of the receive hardware >routine< are to forward incoming data frames to the signaling_controller and to extract the schedule from the data frame with SCHED payload, according to the current receive schedule. The transmit hardware <routine> meanwhile transmits control frames and data frames from its wavelength queues onto the appropriate outgoing wavelengths, according to the current transmit schedule.

5.4.1 Receive Hardware

The state machine <routine> shown in Figure 5.7 can only begin after the Join process has succeeded. The final task in the Join process was to place the current schedule information into the memory bank RCV_BANK0; therefore >routine< begins by setting cur_rcv_bank to RCV_BANK0. After confirming RCV_BANK0's status to be VALID, >routine< is ready to receive the first superframe. Cur_rcv_schedule now points to the schedule contained in cur_rcv_bank (Note that the status of cur_rcv_bank has already been confirmed VALID).

At the start of any superframe, <routine> :

1. Sets psc_sf_start to psc_sf_start_next and rcv_lambda to cur_sched_lambda. Recall that psc_sf_start represents the time (according the node's local clock) that the superframe began at the PSC. The value of psc_sf_start_next could have been set in one of two ways: either >join< set the value (true only for the first superframe after the node joins the network), or >routine< set the value (true for

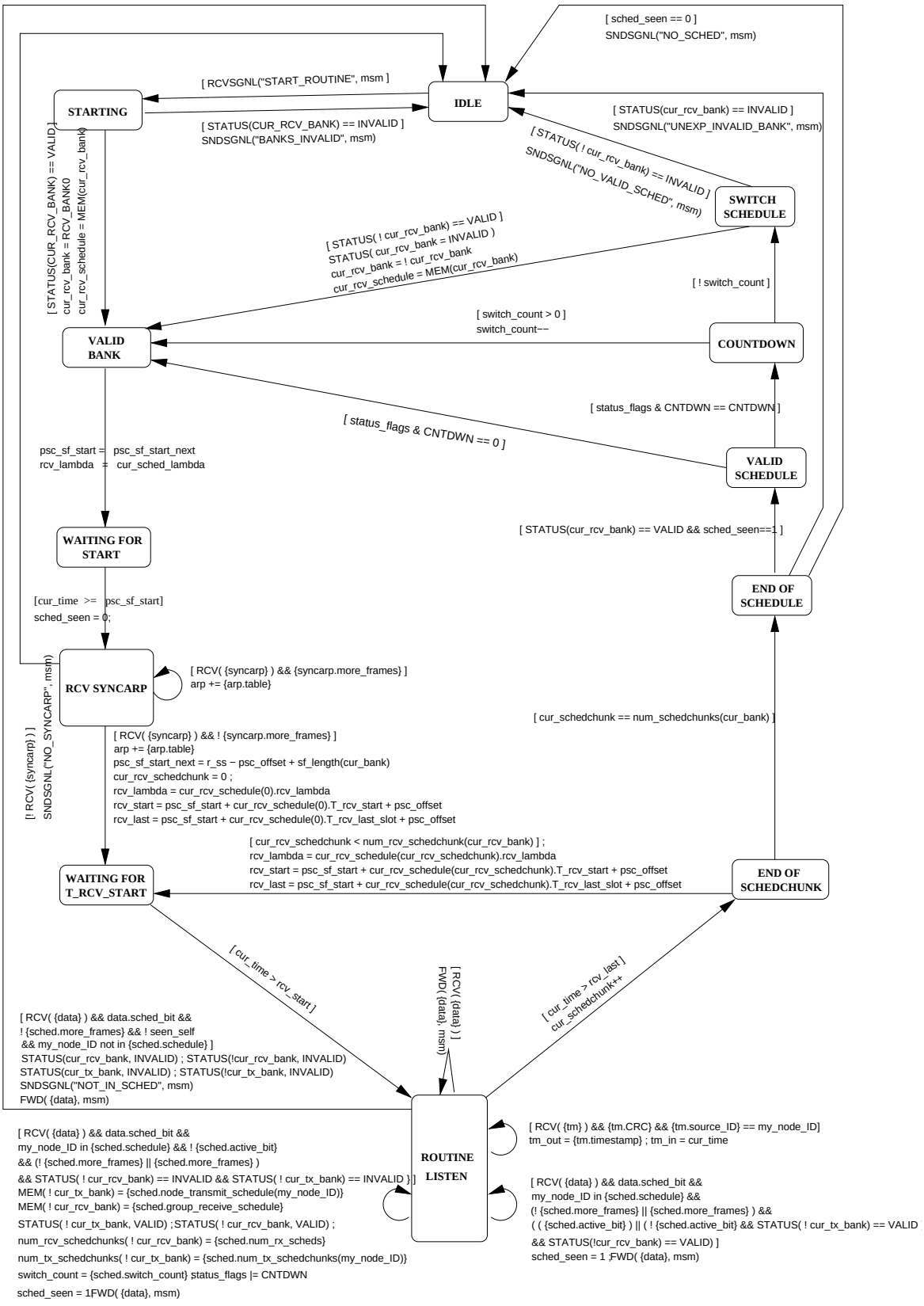


Figure 5.7: Receive auxiliary state machine for candidate and slave nodes: **>routine<**

all other superframes).

2. Waits till cur_time > psc_sf_start and then receives the SYNCARP frame on the Master Node transmit lambda. If no SYNCARP is received then the state machine returns to IDLE state. Else from SYNCARP frame copy the contents of arp into local arp table along with other synchronization information.

3. Sets the index cur_rcv_schedchunk to zero. This index will be incremented after the node completes its receiving for each successive group; the node then can recognize that it is done with the current superframe when cur_rcv_schedchunk reaches the value
   $\texttt{num\_rcv\_schedchunks}(\texttt{cur\_rcv\_bank}) - 1$.

Once these tasks have been completed, >routine< is ready to begin receiving according to the information contained in the current receive schedchunk.

At the start of any schedchunk, >routine< :

1. sets rcv_lambda to point to the queue for cur_rcv_schedule(cur_rcv_schedchunk).rcv_lambda

2. calculates two time references that govern its receive on any lambda: (a) rcv_start, the time it may begin receiving from group_id, and (b) rcv_last, the last instant at which it may start receiving from group_id.

Just prior to receive on each lambda, the node checks to make sure that the current time has not exceeded rcv_last. When rcv_last has passed, receive on this lambda must cease; the end of the current schedchunk has arrived. The index cur_rcv_schedchunk is incremented and then tested against num_rcv_schedchunks(cur_rcv_bank) to determine whether the end of the schedule has arrived. If not, >routine< proceeds to the next schedchunk.

But if >routine< has reached the end of the schedule, it next checks whether a countdown has started (counting down the superframes until the time to switch from the current to the reserve memory bank). If the countdown has not yet begun, then >routine< simply starts over at the beginning of the current schedule in cur_rcv_bank. If the countdown has begun, then >routine< must determine whether it should switch now to the reserve memory bank. This task is accomplished by considering the value

of switch_count, which gives the number of remaining superframes for which the old schedule should still be used. If switch_count is positive, >routine< decrements switch_count and starts over at the beginning of the old schedule (in cur_rcv_bank). If switch_count has reached zero, then >routine< marks cur_rcv_bank INVALID and switches to the reserve memory bank, by setting cur_rcv_bank to !cur_rcv_bank.

A node might receive different frames while listening on a wavelength. We now describe each in turn.

- Receipt of a DATA frame.

  Event: A DATA frame was received on the listening wavelength.

  Transition: Forward the frame to the frame handling layer and return to the Routine Listen state.

- Receipt of a TM frame.

  Event: A TM frame was received on the listening wavelength.

  Transition: If the TM frame was transmitted by self then perform the psc_offset calculation.

- Receipt of a SCHED payload which contains my_node_ID.

  – {sched.active_bit} was set.

    Event: The active bit in the newly-arrived SCHED payload was set, indicating that no countdown has begun to switch to a new schedule.

    Transition: Reset get_sched_timer and move to Routine Listen state.

  – {sched.active_bit} was *not* set and the status of !cur_bank is VALID.

    Event: Countdown has begun to switch to a new schedule, and the new schedule has already been copied into the reserve memory bank (!cur_bank).

    Transition: Reset get_sched_timer and move to the Routine Listen state.

  – {sched.active_bit} was not set and the status of !cur_bank is INVALID.

Event: Countdown has begun to switch to a new schedule, but the new schedule has not yet been copied into the reserve memory bank (!cur_bank).

Transition: Copy the new scheduling information into !cur_bank, save important timing information, and move to the In Schedule state.

- Receipt of a SCHED which does not contain my_node_ID.

  Event: A SCHED payload was received that unexpectedly fails to contain scheduling information for this node.

  Transition: Mark the status of both memory banks (the current and the reserve bank) INVALID, send signal NOT_IN_SCHED to signaling_controller, and move to the Idle state.

5.4.2 Transmit Hardware

The state machine <routine> shown in Figure 5.8 can only begin after the Join process has succeeded. The final task in the Join process was to place the current schedule information into the memory bank TX_BANK0; therefore <routine> begins by setting cur_tx_bank to TX_BANK0. After confirming TX_BANK0's status to be VALID, <routine> is ready to begin the first superframe. Cur_tx_schedule now points to the schedule contained in cur_tx_bank (Note that the status of cur_tx_bank has already been confirmed VALID).

At the start of any superframe, <routine> :

1. sets the index cur_schedchunk to zero. This index will be incremented after the node completes its transmissions for each successive group; the node then can recognize that it is done with the current superframe when cur_schedchunk reaches the value num_tx_sched_chunks(cur_tx_bank)-1.

2. sets psc_sf_start to psc_sf_start_next. Recall that psc_sf_start represents the time (according the node's local clock) that the superframe began at the PSC. The value of psc_sf_start_next could have been set in one of two ways: either >join< set the value (true only for the first superframe after the node joins the network), or >routine< set the value (true for all other superframes). In addition

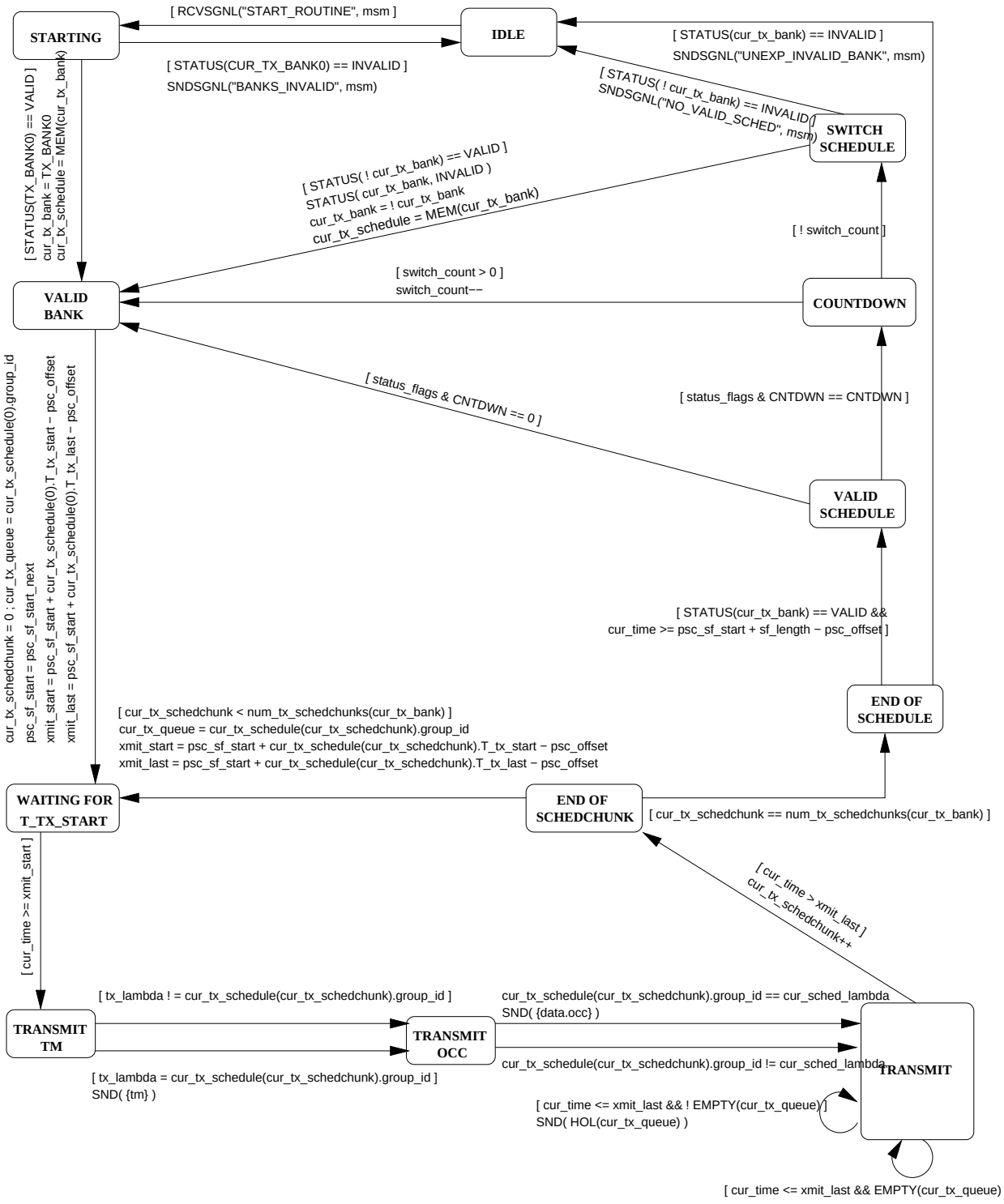


Figure 5.8: Transmit auxiliary state machine for candidate and slave nodes: **<routine>**

to the psc_offset, two other quantities are needed to calculate the start time of the superframe at the PSC, these quantities are:

- sf_length(cur_bank) : the length of the superframe in slots. The node copies the value from {syncarp.sf_length} (a field in the SYNCARP frame) into the local variable sf_length(cur_bank) associated with the current schedule bank, cur_bank.
- r_syncarp : the receive timestamp of the SYNCARP frame at the node (according to the node's clock). The node copies the current time from its own clock into the local variable r_syncarp at the instant the SYNCARP frame arrives.

Using these two quantities and the psc_offset, the beginning of the next superframe at the PSC, called psc_sf_start_next, is:

$$\texttt{psc\_sf\_start\_next} = \texttt{r\_syncarp} - \texttt{psc\_offset} + \texttt{sf\_length}$$

Once these tasks have been completed, <routine> is ready to begin transmissions according to the information contained in the current schedchunk.

At the start of any schedchunk, <routine> :

1. sets cur_tx_queue to point to the queue for cur_tx_schedule(cur_tx_schedchunk)- .group_id.
2. calculates two time references that govern its transmissions: (a) xmit_start, the time it may begin transmitting to group_id, and (b) xmit_last, the last instant at which it may start the transmission of a frame for group_id.

At this point <routine> needs only to wait until xmit_start arrives in order to begin transmitting; the first frame it transmits depends on the value of tx_lambda (Transmit wavelength of the node):

1. If tx_lambda = cur_sched_lambda (the transmit wavelength of the master node and so the group_id of the master node), then the first frame <routine> transmits must be an {DATA(OCC)} frame, to inform the master node of its queue occupancies.

2. Otherwise, if tx_lambda = group_id (the node's own group), then the first frame <routine> transmits must be a {TM} frame, to carry out "Routine Time Measurement".

3. Otherwise, <routine> may transmit DATA frames from cur_tx_queue.

Recall that DATA frames may be of variable length, with none exceeding L_max. The node transmits DATA frames from cur_tx_queue back to back, without waiting for the beginning of a new slot. Just prior to transmitting each frame, the node checks to make sure that the current time has not exceeded xmit_last. When xmit_last has passed, transmissions for this group must cease; the end of the current schedchunk has arrived. The index cur_tx_schedchunk is incremented and then tested against num_tx_schedchunks(cur_tx_bank) to determine whether the end of the schedule has arrived. If not, <routine> proceeds to the next schedchunk.

But if <routine> has reached the end of the schedule, it next checks whether a countdown has started (counting down the superframes until the time to switch from the current to the reserve memory bank). If the countdown has not yet begun, then <routine> simply starts over at the beginning of the current schedule in cur_tx_bank. If the countdown has begun, then <routine> must determine whether it should switch now to the reserve memory bank. This task is accomplished by considering the value of switch_count, which gives the number of remaining superframes for which the old schedule should still be used. If switch_count is positive, <routine> decrements switch_count and starts over at the beginning of the old schedule (in cur_tx_bank). If switch_count has reached zero, then <routine> marks cur_tx_bank INVALID and switches to the reserve memory bank, by setting cur_tx_bank to !cur_tx_bank.

5.5 Scheduling

5.5.1 Receive Hardware

The receive state machine >scheduling<, shown in Figure 5.9, retains all the functionality of >routine<, but possesses two extra transitions to aid in the collection of information needed to compute the schedule. Each of the additional transitions is a self-transition from the Routine Listen state:

- Receipt of an data frame with OCC payload.

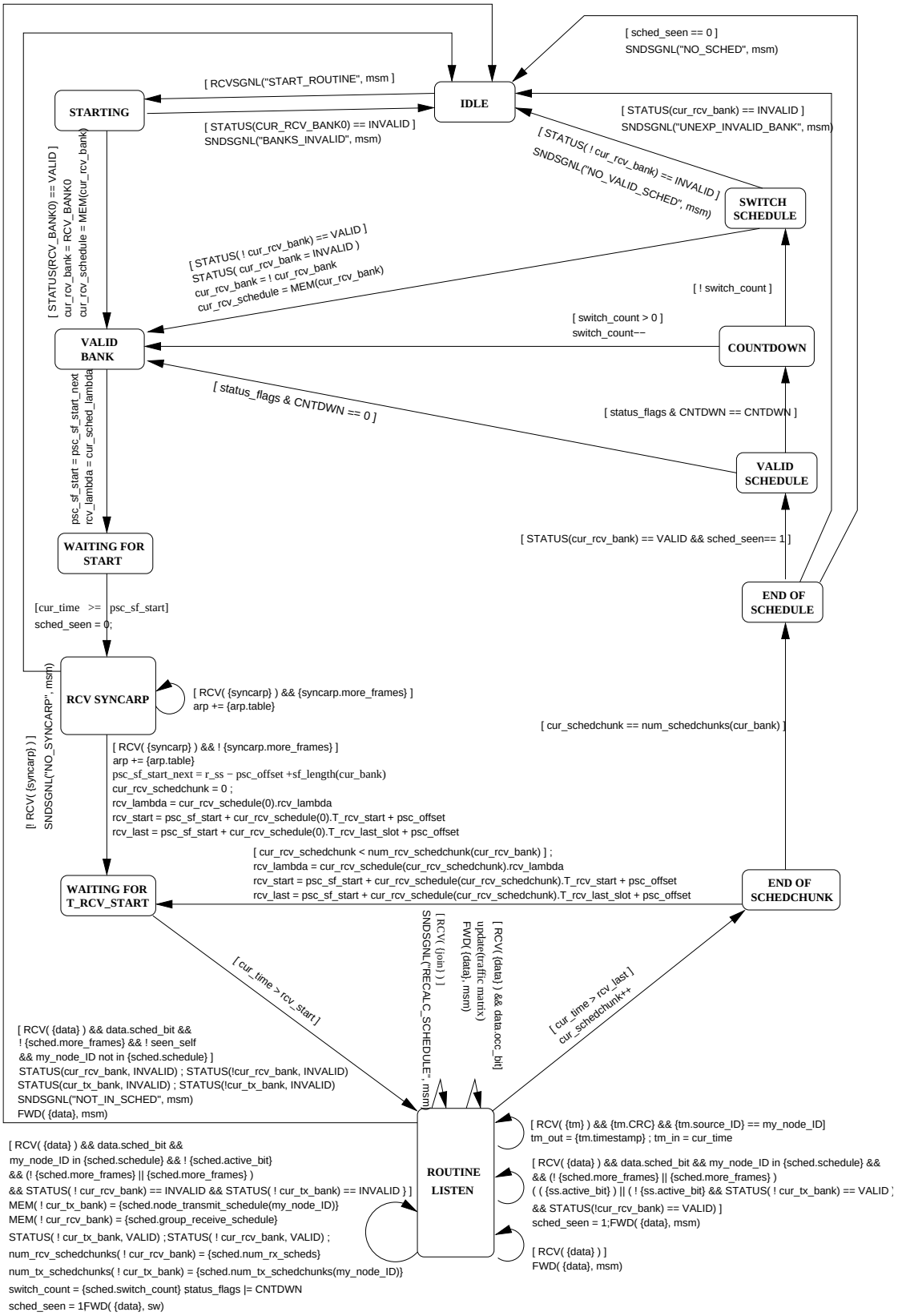


Figure 5.9: Receive auxiliary state machine for the master node: **>scheduling<**

Event: An OCC payload embedded in data frame was received on the current listening wavelength.

Transition: Forward the frame to the signaling_controller and return to the Routine Listen state.

- Receipt of a JOIN frame.

  Event: An JOIN frame was received on the current listening wavelength.

  Transition: Forward the frame to the signaling_controller and return to the Routine Listen state.

Also, the transition from the End of Schedchunk state to the End of Schedule state becomes split into two, in order to aid in the transmission of a newly-calculated schedule. Both transitions first check to make sure the end of schedule has been reached, by verifying that cur_schedchunk+1 equals num_schedchunks(cur_bank). Next, the status of BANK_NEWCALC is checked. If INVALID, no action is taken. If VALID, then a newly-calculated schedule has been placed in BANK_NEWCALC by the software; <scheduling> copies the new schedule into BANK_CURFRAME so that it can be disseminated in the next superframe.

5.5.2 Transmit Hardware

The transmit state machine <scheduling>, shown in Figure 5.10, retains all the functionality of <routine>, but possesses one extra transition. At the start of the superframe, indicated by cur_time > psc_sf_start, it transmits the SYNCARP frame from the syncarp queue.

Also, the transition from the End of Schedchunk state to the End of Schedule state becomes split into two, in order to aid in the transmission of a newly-calculated schedule. Both transitions first check to make sure the end of schedule has been reached, by verifying that cur_schedchunk+1 equals num_schedchunks(cur_bank). Next, the status of BANK_NEWCALC is checked. If INVALID, no action is taken. If VALID, then a newly-calculated schedule has been placed in BANK_NEWCALC by the software; <scheduling> copies the new schedule into BANK_CURFRAME so that it can be disseminated in the next superframe.

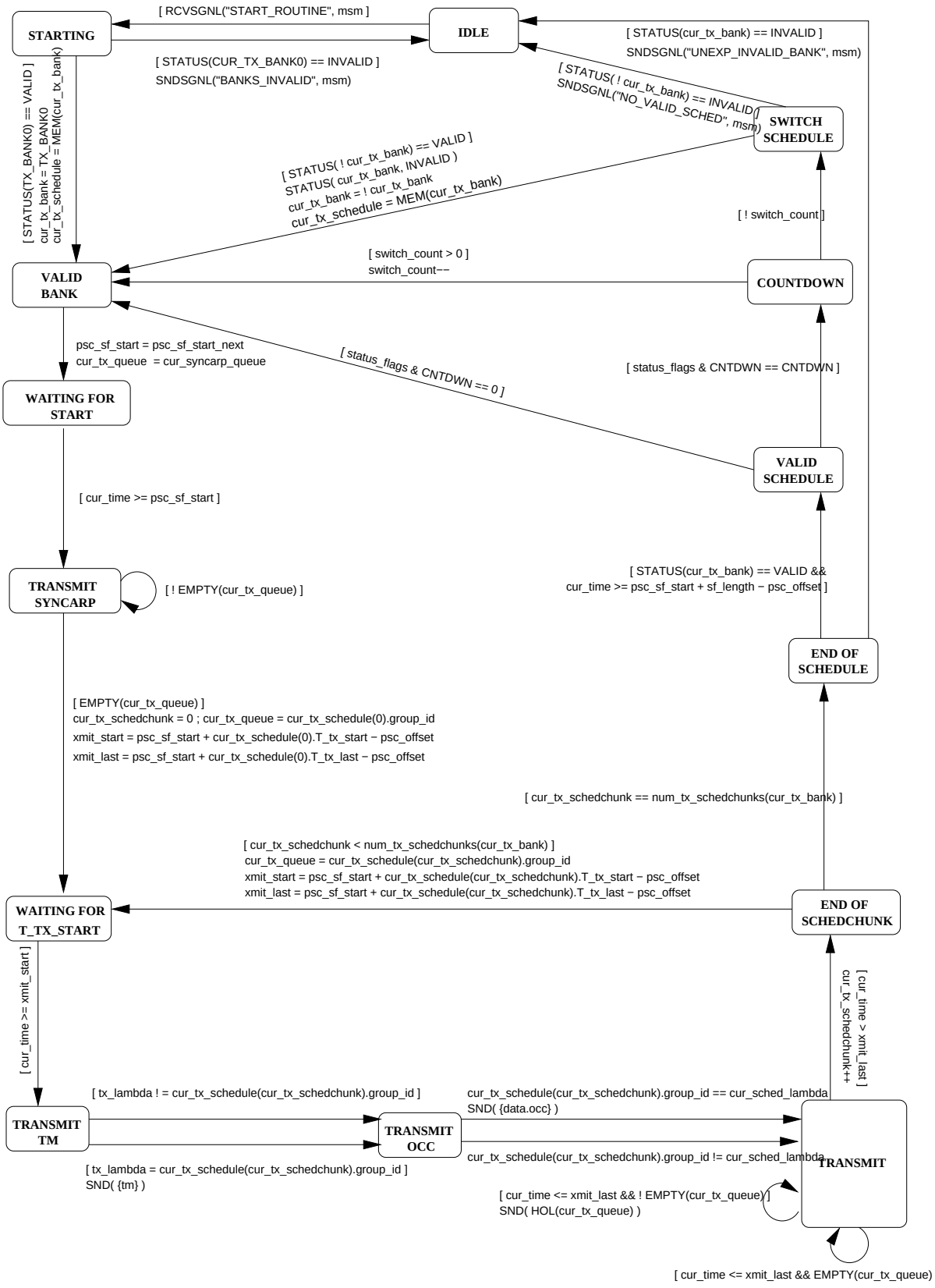


Figure 5.10: Transmit auxiliary state machine for the master node: **<scheduling>**

6 Helios-2 Scheduling Algorithm

6.1 The Helios-2 Scheduling Algorithm

Helios-2 network would typically consist of N nodes and C wavelengths, where C $\leq$ N. Each node is equipped with one slowly tunable transmitter (tuning time of ~100ms) and one fast tunable receiver (tuning time of ~10μs). Slowly tunable transmitter is tunable only for the purpose of electing a master node or for the load balancing and reconfiguration, it is fixed for all other purposes. The tunable receiver can tune to, and listen on any of the C wavelengths.

We divide the receivers in a Helios-2 network into groups, discussed in [17], termed as virtual receiver groups. These virtual receivers are a set of physical receivers that behave identically in terms of tuning. Thus, from the point of scheduling the tuning of virtual receivers to the various channels, all physical receivers in a virtual receiver group can be logically thought of as a single receiver.

Master node in Helios-2 network is responsible for assigning a group identification to a new node joining the Helios-2 network. A nodes group identification can also be changed during the routine operation of a Helios-2 network in response to the changes in traffic pattern. One such study on load balancing and reconfigurations is presented in [21, 22].

In Helios-2 master node receives an OCC payload, containing packet queue occupancies, from each node once per superframe. The master node may also receive a JOIN frame, containing packet queue occupancies, from a new node wishing to join the network. From this information, the master node can build the traffic matrix A, an $G \times C$ matrix, where G is the number of groups in the network, C is the number of wavelengths, and entry a_{ij} represents the number of slots needed for carrying traffic by

	λ_1	λ_2	λ_3	sum
g_1	4	1	3	8
g_2	2	3	2	7
g_3	3	2	1	6
g_4	2	2	1	5
g_5	1	1	1	3
sum	12	11	8	

Table 6.1: Example traffic matrix

wavelength λ_j to group g_i . For a network of $C = 3$ wavelengths and $G = 5$ groups, a sample traffic matrix is shown in Table 6.1.

Helios-2 uses a one-pass greedy scheduling algorithm, the pseudo-code for which is shown in Algorithm 1. The algorithm creates a schedule from $t = 0$ forward in time without backtracking, always attempting to schedule the highest priority node on the highest priority wavelength. Higher priority is assigned to groups (respectively, wavelengths) that have higher corresponding row-sums (respectively, column-sums) in the traffic matrix A. In the sample traffic matrix, the groups have been renumbered in order of largest row-sum to smallest, such that g_1 has the largest row-sum and g_G has the smallest, with ties being broken arbitrarily. The same was done for the wavelengths: λ_1 has the largest column-sum and λ_C has the smallest. The traffic matrix gives rise to two lower bounds on the schedule length. The maximum column-sum plus G tuning latencies is the channel bound and the maximum row-sum is called the group bound. The maximum of the channel and node bounds is the greatest lower bound on the schedule length.

6.2 Performance

The schedulers developed in previous works [13, 14, 15] produces schedules very close to the lower bound in length, but requires a prohibitively long runtime. In particular, the scheduler for Hiper-l [13, 14] has a worst-case runtime of $O(CN^4)$ while the scheduler for Helios [15] has a worst-case runtime of $O(C^2N^2)$. The scheduler developed for Helios-2 is a straightforward greedy scheduler that has a worst-case runtime of $O(C^2G^2)$. This speedup is substantial because the number of groups would be

Algorithm 1 Helios-2 Scheduling Algorithm

```
/* initialize each entry in the receive and transmit schedule to 0 */

for (t = 0...2(glb)) { /* the schedule length will not exceed 2(glb) */

        for (λ = 1...C) {
                rcv_schedule[t][λ] = 0 ;
                tx_schedule[t][λ] = 0 ;
        } /* end for */

} /* end for */

/* ********************************************************************/

/* initialize remainingDemand to the sum of all the a_λg's */

remainingDemand = 0 ;

for (λ = 1...C) {

        for (g = 1...G)
                remainingDemand = remainingDemand + a[λ][g] ;

} /* end for */

/* ********************************************************************/

/* begin scheduling */

t = 0 ;

while (remainingDemand > 0) and (t < 2(glb)) { /* while there is still unmet demand */

        for (λ = 1...C) {
                if (rcv_schedule[t][λ] == 0) { /* if no task has been assigned yet to this λ at this slot */
                        g = 1 ;
                        while ((g ≤ G) AND ((unavailable[g][t] == 1) OR (a[λ][g] == 0)))
                                g = g + 1 ; /* find an available group with unfulfilled demand on this λ */
                        if (g ≤ G ) {
                                for (i = t to t+a[λ][g]−1)
                                        rcv_schedule[i][λ] = g ;
                                for (i = t to t+a[λ][g]−1+tuneLatency)
                                        unavailable[g][i] = 1 ;
                        } /* end if */
                        n = 1 ;
                        for (n = 1...N) {
                                while ((n ≤ N) AND ((tx_lambda(n)!=λ) OR (b[n][g] == 0)))
                                        n = n + 1 ;
                                for (j = t to t+b[n][λ]−1)
                                        tx_schedule[i][λ] = n ;
                                b[n][g] = 0 ;
                        } /* end for */
                        a[λ][g] = 0 ;
                        remainingDemand = remainingDemand - a[λ][g] ;
                } /* end if */
        } /* end for */
        t = t + 1 ;

} /* end while */
```

much smaller than number of nodes and equal to or less than the number of channels. Moreover, the faster scheduler can be readily implemented in hardware, resulting in an additional gain in speed. To achieve these gains in speed and simplicity, the new scheduler produces schedules that are not as close to optimal as those produced by the original scheduler. However, the faster scheduler's results are "reasonably close" to optimal.

We consider uniform and bimodal traffic pattern for the numerical analysis of proposed scheduling algorithm. Use of uniform traffic pattern for analysis is justified when the Master node in Helios-2 network is also running load balancing and reconfiguration algorithms. If the master node in Helios-2 network is not running any load balancing and reconfiguration algorithms then it is suitable to consider network traffic as bimodal. There have been many studies [28] of typical TCP/IP Internet traffic, and a common trait is the bimodal network traffic distribution.

The histograms shown in Figure 6.1 - 6.12 plot number of simulation runs against the percentage variation of computed schedule from the lower bound. Simulation run length for all simulations was 10000. In simulations with various patterns of network traffic demand, the new scheduler produces schedules within 3% of the lower bound, approximately 90% of the time.

For the results shown in Figure 6.1 - 6.8, the elements of traffic matrix A were chosen, with equal probability, among the integers 1 through 25 (uniform distribution). We show four sets of two figures each, corresponding to the four values of the set $\{C,G\}$ as $\{4,8\}$, $\{4,16\}$, $\{8,8\}$, and $\{8,16\}$. Within each set we use two different values for tuning latency T, $T = 1,8$. At data rates of 1 Gigabits per second, and packet sizes equal to the typical IP packet (1500 bytes), the packet transmission time is $12\mu s$. Hence the two values of T correspond to transceiver tuning times of approximately $10\mu s$ and $100\mu s$ respectively; these two values correspond to possible implementations of optical filters as discussed in Chapter 7.

For the results shown in Figure 6.9 - 6.10, the elements of traffic matrix A were chosen using bimodal distribution:with probability 0.5 an element is chosen from the uniform(1, 15) distribution and with probability 0.5 from the uniform(12, 25) distribution. We show one set of two figures, corresponding to the value of the set $\{C,G\}$ as $\{4,8\}$. Within this set we use two different values for tuning latency T, $T = 1,8$.

Finally, for the results shown in Figure 6.11 - 6.12, the elements of traffic matrix A were chosen,

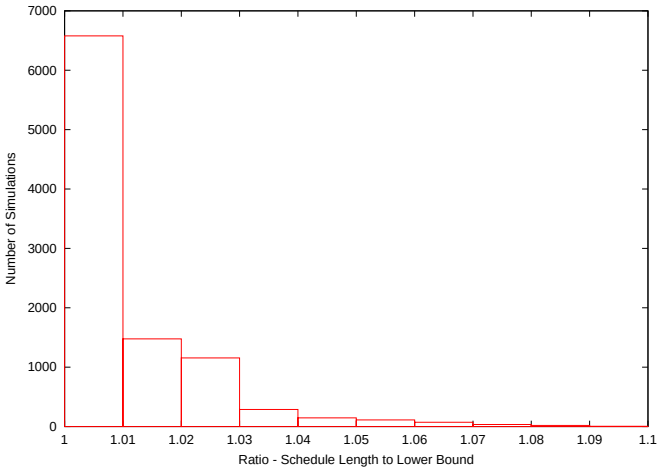


Figure 6.1: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 4 channels, G = 8 groups and T = 1 tuning slot

with equal probability, among the integers 1 through 50 (uniform distribution). We show one set of two figures, corresponding to the value of the set $\{C, G\}$ as $\{4, 8\}$. Within each set we use two different values for tuning latency T, $T = 1, 8$.

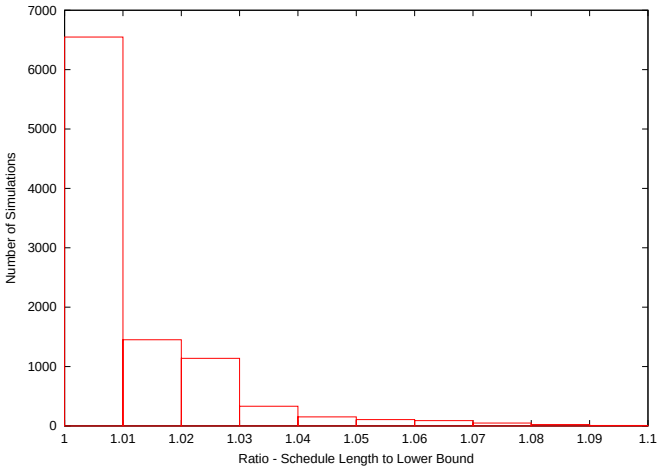


Figure 6.2: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 4 channels, G = 8 groups and T = 8 tuning slot

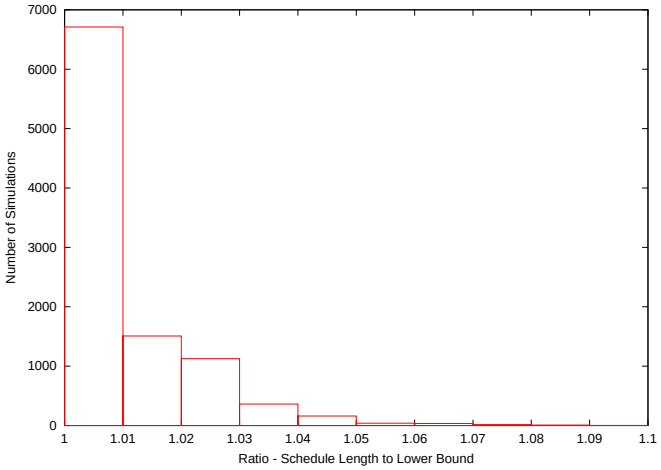


Figure 6.3: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 4 channels, G = 16 groups and T = 1 tuning slot

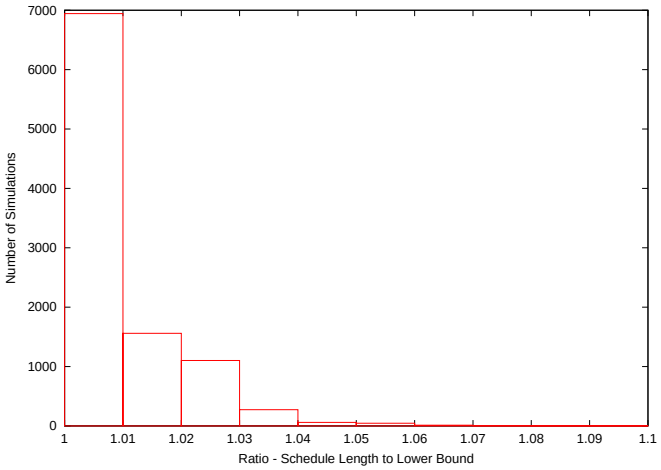


Figure 6.4: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 4 channels, G = 16 groups and T = 8 tuning slot

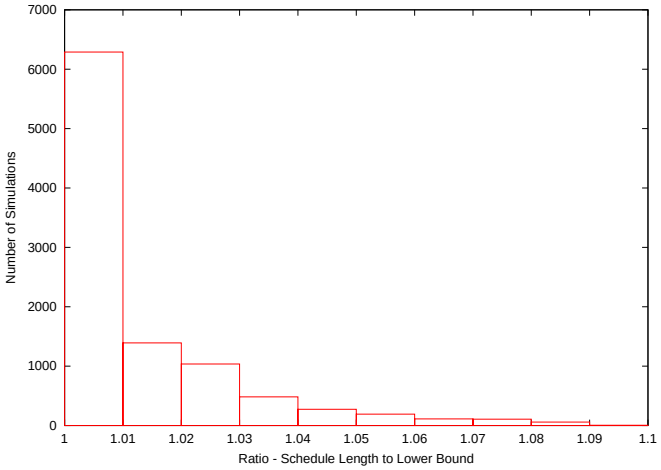


Figure 6.5: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 8 channels, G = 8 groups and T = 1 tuning slot

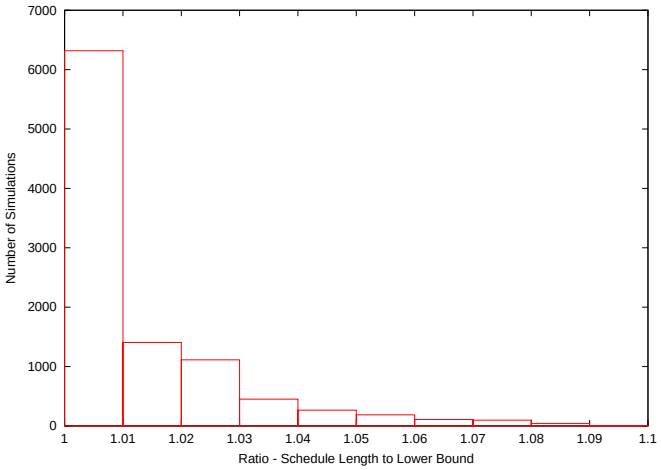


Figure 6.6: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 8 channels, G = 8 groups and T = 8 tuning slot

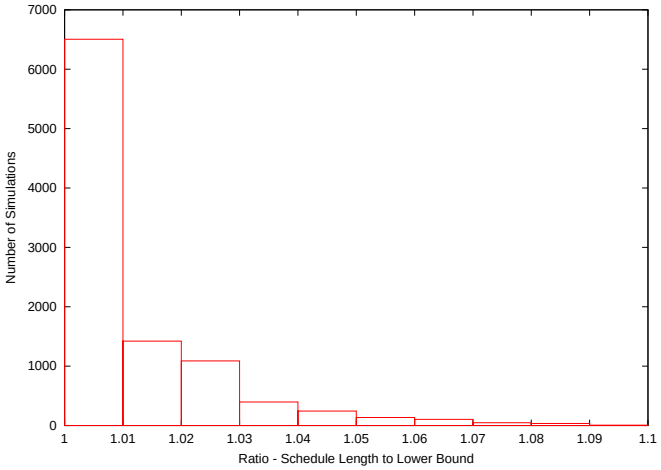


Figure 6.7: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 8 channels, G = 16 groups and T = 1 tuning slot

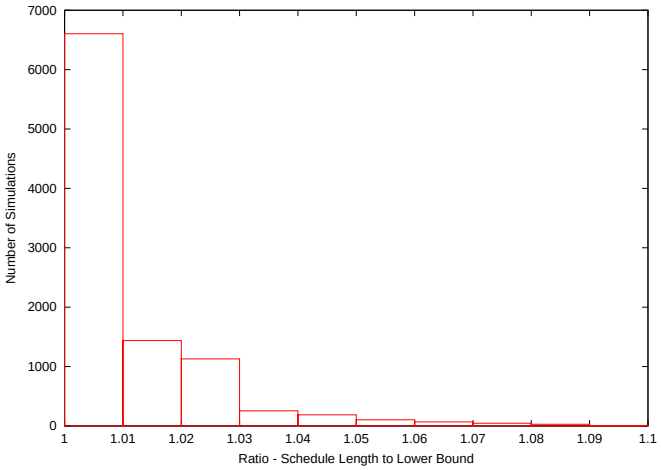


Figure 6.8: Performance of Helios-2 scheduler - Uniform(1, 25) Distribution

For C = 8 channels, G = 16 groups and T = 8 tuning slot

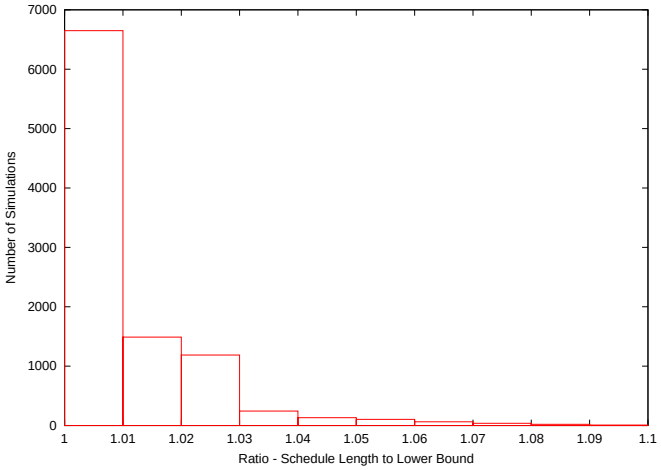


Figure 6.9: Performance of Helios-2 scheduler - Uniform(1, 50) Distribution

For C = 8 channels, G = 16 groups and T = 8 tuning slot

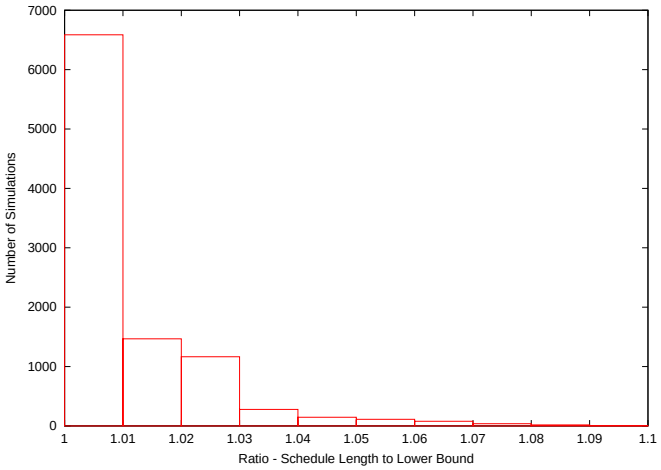


Figure 6.10: Performance of Helios-2 scheduler - Uniform(1, 50) Distribution

For C = 4 channels, G = 8 groups and T = 8 tuning slot

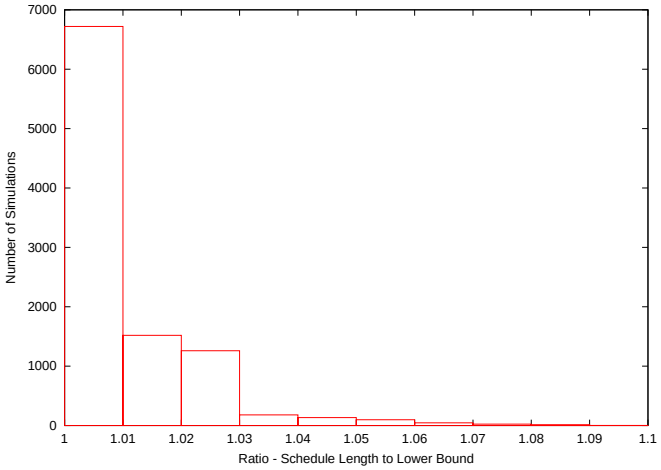


Figure 6.11: Performance of Helios-2 scheduler - Bimodal Distribution

For C = 4 channels, G = 8 groups and T = 1 tuning slot

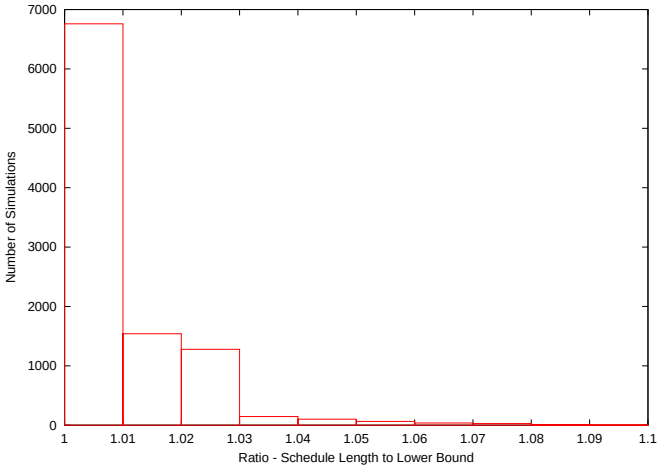


Figure 6.12: Performance of Helios-2 scheduler - Bimodal Distribution

For C = 4 channels, G = 8 groups and T = 8 tuning slot

7 Conclusion

7.1 Hardware Architecture

Hardware implementation, Figure 7.1, of Helios-2 can be broadly divided in two parts.

1. FPGA Back End - FPGA back end interfaces with host through PCI/PCI-X interface and with optical front end. FPGA back end is responsible mainly for running Helios-2 state machine, DMA controller for communication with host, Helios-2 frame parsing/generation.

2. Optics Front End - Optical front end interfaces with the Helios-2 network and with FPGA back end. Optics front end is responsible for tuning the receive and transmit wavelength during initialization/normal Helios-2 network operation.

7.2 FPGA Back End

FPGA Back End, Figure 7.2, can be broadly divided in three blocks discussed below.

7.2.1 Host Interface

Helios-2 NIC can be interfaced with Host through PCI/PCI-X interface module. This module would implement functional, timing and electrical specifications of PCI/PCI-X Local Bus specification. PCI/PCI-X configuration space module provides and appropriate set of configuration “hooks” which satisfy the needs of configuration mechanisms defined in PCI/PCI-X Local Bus specification. PCI/PCI-X interface

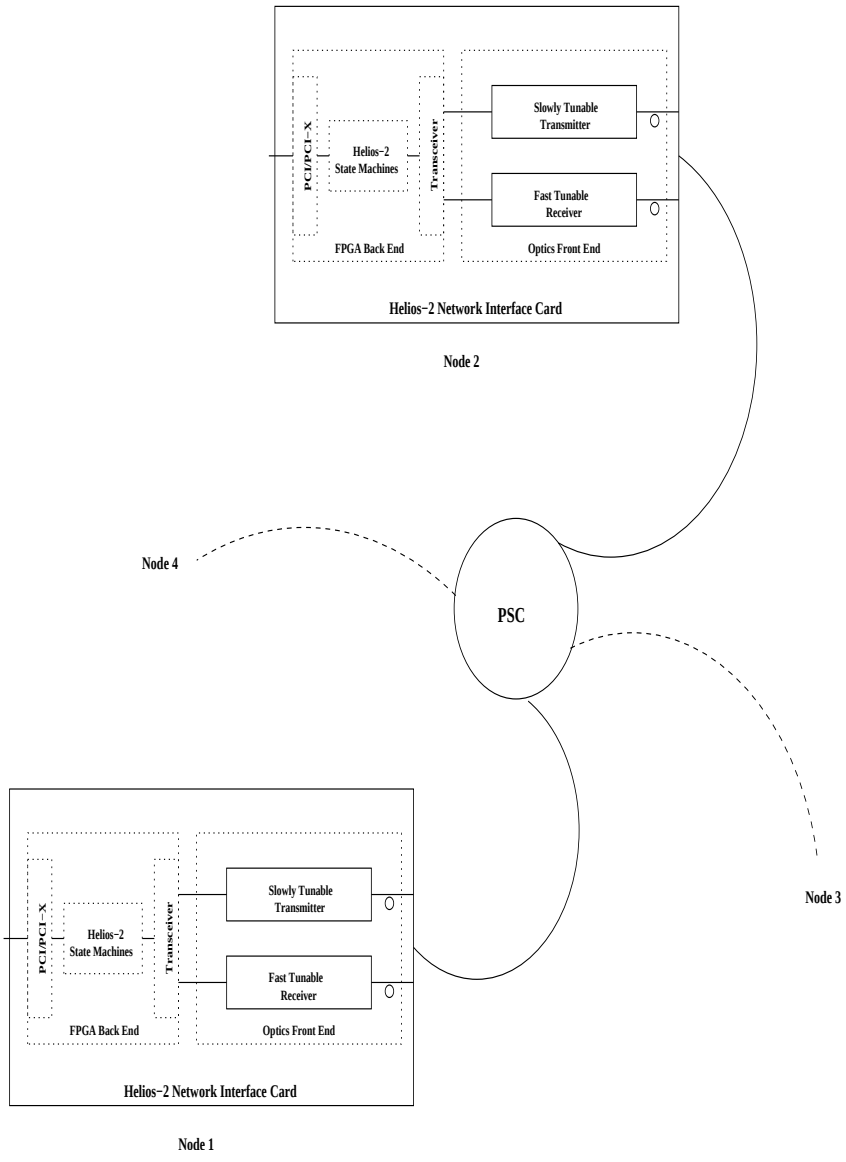


Figure 7.1: Helios-2 Network

module is readily available in form of IP core. PCI/PCI-X module would be interfaced with FIFO, DMA controllers and Memory Arbiter to assist in efficient transfer of data to/from host memory.

7.2.2 Helios-2 State Machines

Helios-2 state machines are initialized by Initialization and Control State Machine (ICSM). ICSM is responsible for following functions

- Receiving Helios-2 control information from Host during start up or during regular operation. ICSM populates Control Memory block with this information.
- Transmitting Helios-2 status and error information to the Host.

Control Memory, Schedule Memory and Signal Block is implemented using the memory available on FPGA. All Helios-2 state machines are implemented using the logic gates available on FPGA. Helios-2 Master State Machine also interfaces with On-FPGA processor (PowerPC 405 processor in Xilinx's Virtex-II Pro FPGA) [25]. This processor is used for schedule computation in the scheduler node.

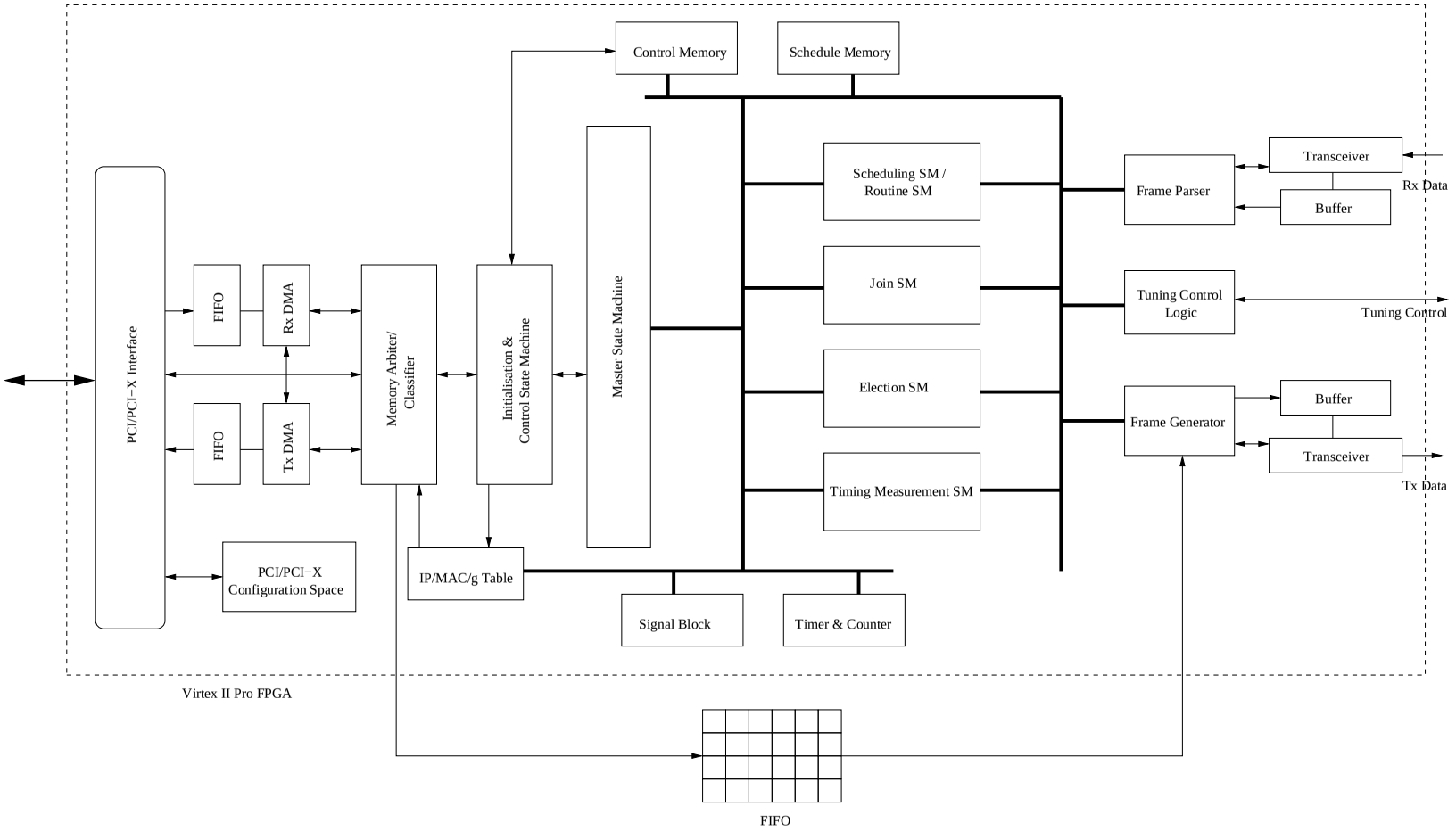
7.2.3 Optics Interface

Optics interface consists of Frame Parser module, Frame Generator module, Tuning Control Logic module and Transceivers. Frame Parser and Frame Generator modules are responsible for parsing and generating Helios-2 frames. Tuning control logic translates the transmit and receive optics tuning commands received from different Helios-2 state machines to the tuning command understood by the Optics Front End of Helios-2 NIC.

7.3 Optics Front End

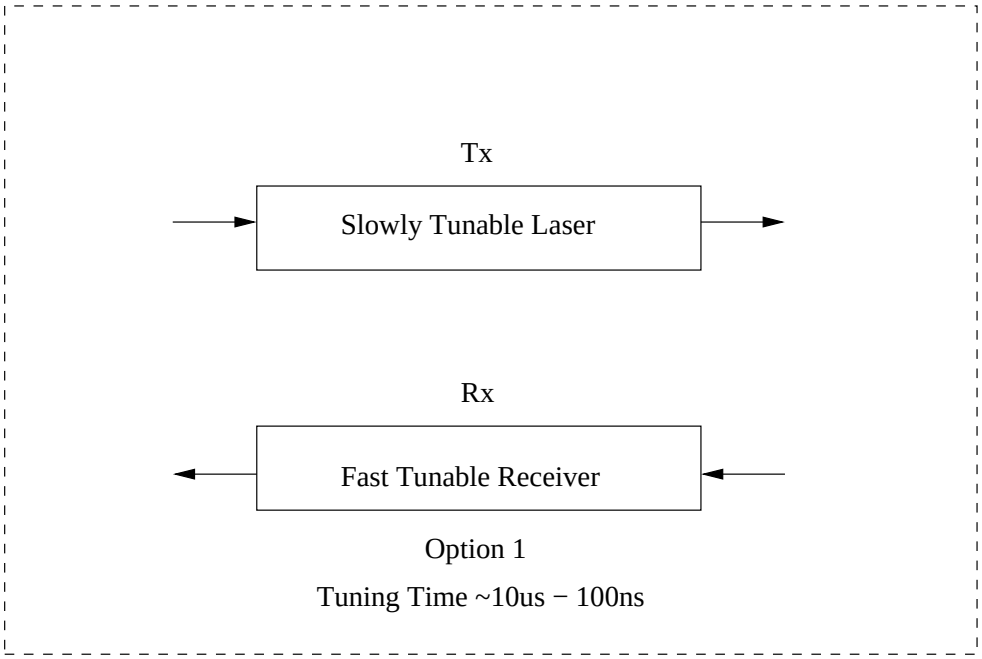
Optical Front End of Helios-2, Figure 7.3, can be implemented in two ways depending on the way Fast Tunable Receiver is realized.

1. Optically Tuned - In this option, receiver is either MZ Interferometer chain or Acoustooptic Filter.


Control Memory
Schedule Memory
PCI/PCI−X Interface
FIFO
Rx DMA
FIFO
Tx DMA
Memory Arbiter/ Classifier
Initialisation & Control State Machine
Master State Machine
Scheduling SM / Routine SM
Join SM
Election SM
Timing Measurement SM
Frame Parser
Transceiver
Buffer
Rx Data
Tuning Control Logic
Tuning Control
Frame Generator
Buffer
Transceiver
Tx Data
PCI/PCI−X Configuration Space
IP/MAC/g Table
Signal Block
Timer & Counter
Virtex II Pro FPGA
FIFO

- Receiver based on MZ Interferometer chain uses a series of splitters [24, 26]. Splitter splits the incoming wave into two waveguides and a combiner recombines the signals at the output of the waveguides. Phase difference of 180^o is created using adjustable delay in one of the waveguides. A series of these splitters can be used to select a single desired optical wavelength. Tuning time in hundred's of nanosecond range can be achieved using MZ Interferometer.
- Receiver based on Acoustooptic Filter technology a RF source to stimulates a piezoelectric crystal [24, 27]. RF waves change the crystal's index of refraction and this enables the crystal to act as grating. By changing the RF waves a single optical wavelength can be selected. Tuning times in the range of ten's of microsecond can be achieved using Acoustooptic Filter.

2. Electronically Tuned - In this option, receiver is built using discrete optical modules. Input optical signal is first feed to a optical De-multiplexer which split out the different wavelength's. These wavelength's are then feed into different APD receiver's. Output of these APD receiver's is electronically selected using an FPGA.



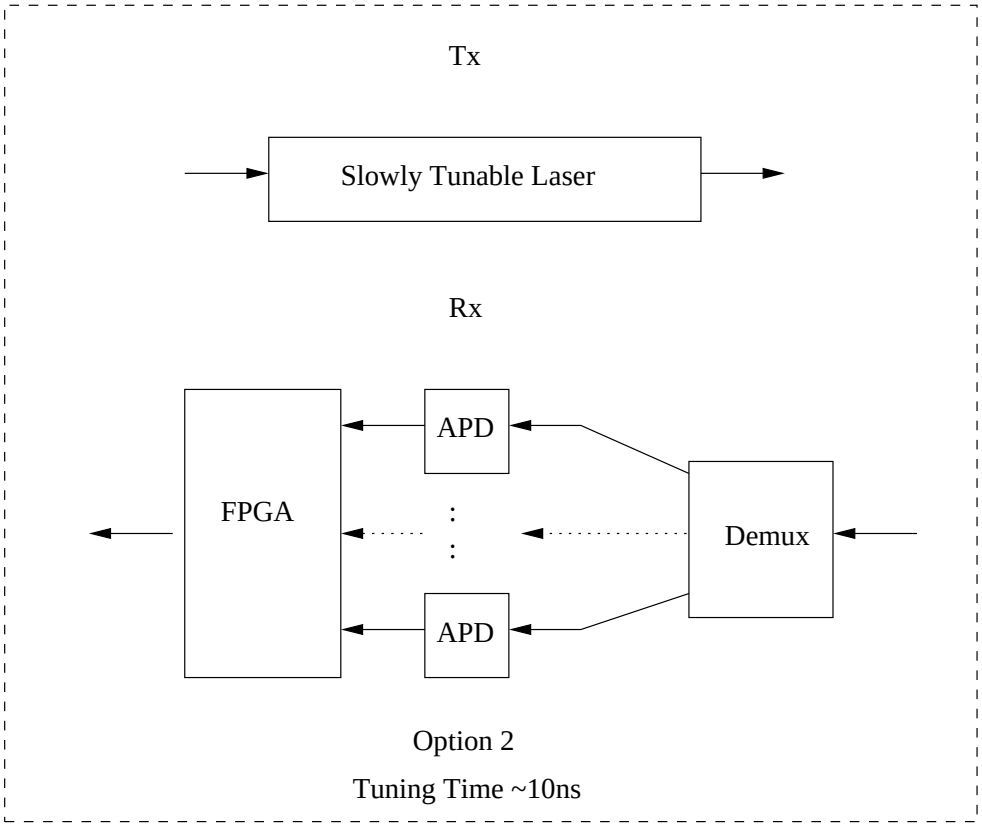


Figure 7.3: Helios-2 Optical Front End

Bibliography

[1] Paul E. Green, Jr. Optical Networking Update. IEEE Journal on Selected Areas in Telecommunications, vol. 14, no. 5, June 1996.

[2] B. Mukherjee. Optical Communication Networks: Progress and Challenges. IEEE Journal on Selected Areas in Telecommunications, vol. 18, no. 10, October 2000.

[3] Charles A. Brackett. Dense Wavelength Divison Multiplexing Networks: Principles and Applications. IEEE Journal on Selected Areas in Telecommunications, vol. 8, no. 6, August 1990.

[4] P. E. Green, Jr. Fiber Optic Networks, Englewood Cliffs: Prentice Hall, 1993.

[5] B. Mukherjee. Optical Communication Networks. New York: McGraw-Hill, 1997.

[6] B. Mukherjee. WDM-Based local lightwave networks Part I: Single-hop systems. IEEE Network Magazine, pages 12-27, May 1992.

[7] B. Mukherjee. WDM-Based local lightwave networks Part II: Multi-hop systems. IEEE Network Magazine, pages 20-32, July 1992.

[8] D. A. Levine and I. F. Akyildiz. PROTON: A Media Access Control Protocol for Optical Networks with Star Topology. IEEE/ACM Transactions on Networking, vol. 3, no. 2, pages 158-168, 1995.

[9] K. Sivalingam and J. Wang. Media Access Protocols for WDM Networks with On-line Scheduling. IEEE/OSA J. Lightwave Tech., vol. 14, no. 6, pages 1278-1286, 1996.

[10] M. S. Chen, N. R. Dono, and R. Ramaswami. A Media Access Protocol for Packet-switched Wavelength Divison Multipaccess Metropolitan Area Networks. IEEE Journal on Selected Areas in Telecommunications, vol. 8, no. 6, pages 1048-1057, 1990.

[11] P. A. Humblet, R. Ramaswami, and K. N. Sivarajan. An Efficient Communication Protocol for High-Speed Packet Switched Multichannel Networks. IEEE Journal on Selected Areas in Telecommunications, vol. 11, no. 4, pages 568-578, 1993.

[12] Vijay Sivaraman and George N. Rouskas. A Reservation Protocol for Broadcast WDM Networks and Stability Analysis. Computer Networks, vol. 32, no. 2, pages 211-227, February 2000.

[13] George N. Rouskas and Vijay Sivaraman. Packet Scheduling in Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies. IEEE/ACM Transactions on Networking, vol. 5, no. 3, pages 359-370, June 1997.

[14] George N. Rouskas and Mostafa H. Ammar. Analysis and Optimization of Transmission Schedules for Single-Hop WDM Networks. IEEE/ACM Transactions on Networking, vol. 3, no. 2, pages 211-221, April 1995

[15] Ilia Baldine, Laura E. Jackson, and George N. Rouskas. Helios: A Broadcast Optical Architecture. Proceedings of Networking 2002, pages 887-894, May 2002.

[16] George N. Rouskas and Mostafa H. Ammar. Multi-Destination Communication Over Tunable-Receiver Single-Hop WDM Networks. IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pages 501-511, April 1997.

[17] Zeydy Ortiz, George N. Rouskas, and Harry G. Perros. Maximizing Multicast Throughput in WDM Networks with Tuning Latencies Using the Virtual Receiver Concept. European Transactions on Telecommunications, vol. 11, no. 2, pages 63-72, January 2000.

[18] Zeydy Ortiz, George N. Rouskas, and Harry G. Perros. Scheduling Combined Unicast and Multicast Traffic in Broadcast WDM Networks. Photonic Network Communications Journal, vol. 2, no. 2, pages 135-154, May 2000.

[19] W. Tseng, C. Sue, and S. Kuo. Performance Analysis for Unicast and Multicast Traffic in Broadcast-and-Select WDM Networks. IEEE International Symposium on Computer and Communications, pages 72-78, 1999.

[20] Jason P. Jue and Biswanath Mukherjee. The Advantages of Partitioning Multicast Transmissions in a Single-Hop Optical WDM Network. IEEE International Conference on Communications, vol. 1, pages 427-431, 1997.

[21] Ilia Baldine and George N. Rouskas. Reconfiguration and Dynamic Load Balancing in Broadcast WDM Networks. Photonic Network Communications Journal, vol. 1, no. 1, pages 49-64, June 1999.

[22] Ilia Baldine and George N. Rouskas. Dynamic Reconfiguration Policies for WDM Networks. IEEE Infocom, pages 313-320, March 1999.

[23] Ilia Baldine and George N. Rouskas. Dynamic Load Balancing in Broadcast WDM Networks with Tuning Latencies. IEEE Infocom, pages 78-85, March 1998.

[24] M. Borella, J. Jue, D. Banerjee, B. Ramamurthy, and B. Mukherjee. Optical Components for WDM Lightwave Networks. Proceedings of the IEEE, vol. 85, no. 8, pages 1274-1307, August 1997.

[25] The Virtex II Pro FPGA. In http://www.xilinx.com

[26] Tunable Optical Filter. In http://www.optune.ca

[27] Tunable Optical Filter. In http://www.brimrose.com

[28] J. Mogul. “Observing TCP Dynamics in Real Networks”. Proceedings of ACM SIGCOMM, pages 305-317, October 1992.