Radio Device API Addendum: Support for TDMA Radios

Version: 11 April 1998; v0.5

Authors: Dave Beyer[1]
dave@rooftop.com
Rooftop Communications

Jason Erickson
jerickson@CCGATE.HAC.COM
Hughes Aircraft

1 Purpose

This addendum to the Radio Device API provides additional capability in the area of TDMA-oriented channel access. Some of the added definitions also mention security-related issues. Time-oriented security algorithms are often tied to the transmit/receive scheduling process. As such, the "RF side" of this API may be a cryptographic module controlling the radio. <<Still true? Should this be reworded?>>

2 Architecture

The position of the "TDMA Radio Device API" presented here is fundamentally the same as its position in the core Radio Device API document, at the "Transceiver Frame Control Interface" (TFCI), shown in Figure 1. However, this document differs slightly from the core specification in the division of responsibilities between the radio device "below" the API and the protocols "above" the API, in particular with regard to the precise scheduling of transmission on the channel. In the core specification all channel scheduling is performed by the protocol software. In contrast, this addendum assumes that the fine-grain scheduling of packets, and/or dynamic radio waveform characteristics, with tight timing requirements is performed by hardware or software situated below the API, typically under the direction of the protocols above. This shift in responsibilities may be appropriate due to reasons that include:

- channel scheduling requirements that exceed the processor scheduling latency for the operating system and/or CPU running the protocol software,
- the delay of processing primitives between the radio device and protocols across the API, or
- concerns regarding the red/black security boundary where a crypto-module situated close to the radio hardware below the API, provides the precise packet scheduling.

For completeness, we list below the functions that this addendum assumes are the responsibility of the radio device, i.e., situated "below" the TDMA Radio Device API, with the new responsibilities highlighted by italics:

- RF & IF radio stages (mixers, filters, power amplifiers, low-noise-amplifiers, ...),
- Modulation,

[1] This work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Small Business Innovative Research Program (SBIR). Refer to the *Acknowledgments* section for details

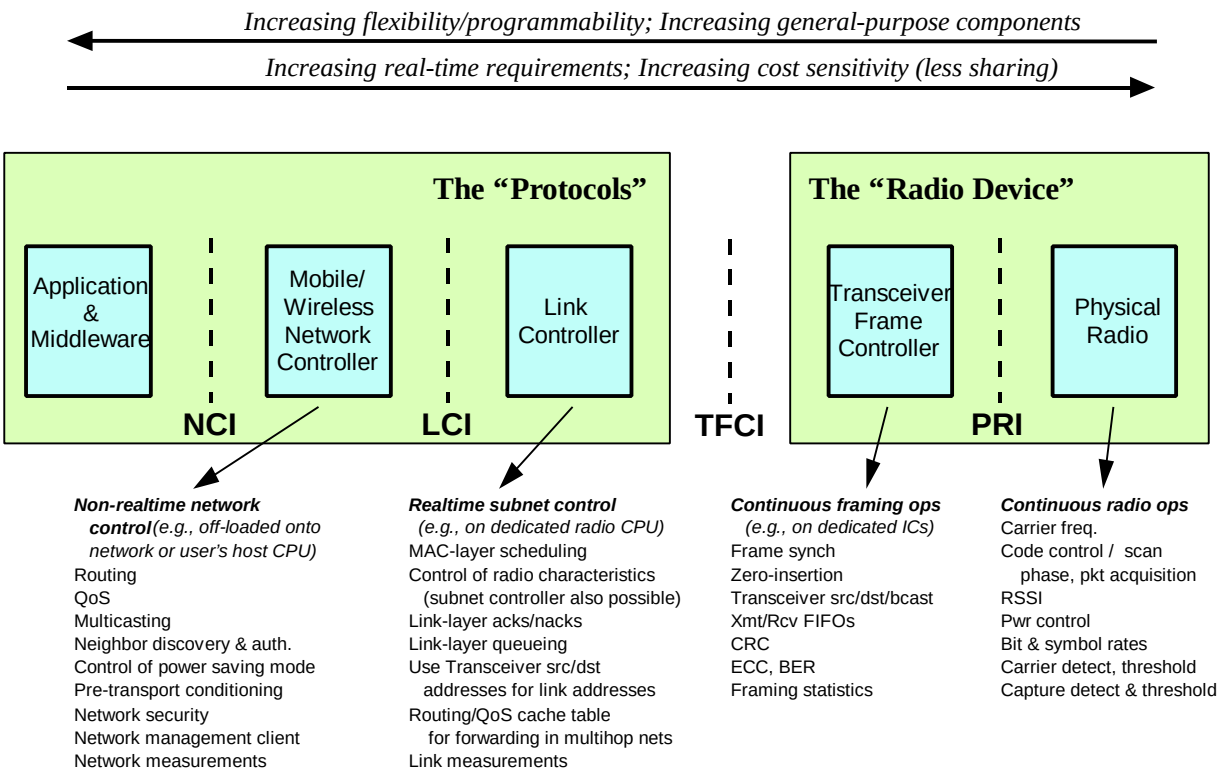

Figure 1: Position of the Radio Device API

- Baseband spreading (direct sequence and / or fast frequency hopping),
- Preamble generation, detection, and synchronization,
- Framing (start / stop flags, zero-bit insertion, ...),
- Error detection and / or correction (CRC computation, interleaving, error control coding),
- ***Real-time transmit/receive hardware control,***
- ***Time-of-Arrival (TOA) measurement, preamble correlation quality measurement, and***
- ***TDMA slot clocking (time slot boundary synchronization and timing).***

The TDMA Radio Device API assumes that the following operations are performed by the software protocols (i.e., "above" the Radio Device API), again with changes to the core specification highlighted by italics:

- Media Access Control (MAC) protocols ***that provide direction to the radio device*** for the scheduling of packets on the channel (for collision avoidance and fair, efficient use of the channel),
- Link-layer protocols (reliable delivery of packets between neighbors or of local broadcast packets, fair sharing of link resources among neighbors, discovery and authentication of new neighbors),
- Network-layer protocols (efficient routing free of persistent loops despite mobility and dynamics, routing and queuing according to the service and priority requirements of the traffic, efficient multicasting, security of network control traffic), and
- Internetwork-layer protocols (wireless-to-wired Internet routing issues, network management, Internet-compatible interfaces).

3 Logical Functionality

3.1 Primitives

Examples of new variable primitives include: the TDMA schedule and the current TDMA synchronization state (e.g., Search, Passive, Active, Silent). Examples of new asynchronous signals include cryptographic statusing signals.

3.2 Data Packet Handling

To support the additional per-packet control and status requirements of TDMA radios, the core specification's packet information structure must be extended to add the following:

- Slot number that the radio device should use for packet transmissions, or to report slot number of packet receptions.
- Offset into slot (in usecs) to use for packet transmissions or to report packet receptions.
- The duration, in slots, during which reception should be attempted for a receive packet buffer.

"Continuous transmission bursts" can still be used to specify that packets without specified transmit times are to be transmitted end-to-end in a continuous transmission with respect to the protocol software's view. However, these transmissions may actually be divided by the radio over many slots in the currently active TDMA schedule, with the decision on whether or not to saddle packets across slot boundaries dependent on the radio implementation.

More often with a TDMA radio, the protocol software will instead specify the transmit time (slot and offset in the current schedule) for each packet, rather than use the "burst" mode.

3.3 TDMA Schedules

3.3.1 Schedule Types

TDMA schedules are defined by a limited set of *slots*, numbered from 0 to the number of slots in the schedule minus 1. For a *repeating* schedule, the sum of the durations of all of the slots in the schedule is called the schedule's *cycle-period*. The TDMA schedules are defined as one of the following two types:

- *Packet Timing Schedule* which provides the means for the protocols and radio device to communicate precisely-timed packet transmission and reception opportunities, and
- *Radio Characteristics Schedule* which specifies the radio's transmit and receive waveform characteristics for each slot in the schedule (unless overridden by radio waveform characteristics specified in a transmit or receive packet information structure, see below).

Multiple schedules of each type can be communicated across the API, with each schedule being identified by its *schedNum*, which must be unique for all current schedules (regardless of their types). At most, one schedule of each type may be active at any time. For the protocols to specify the TDMA timing of packet transmissions or receptions, a Packet Timing Schedule must be active. The schedule(s) are typically determined by the protocols, and are written to the radio device using the TRadioCmdSched primitive.

Packet transmissions can be scheduled to start in a certain slot, and optionally also at a certain offset (in usecs) from the start of the slot. As explained above, radio or waveform characteristics that accompany the RadioCmdXmtPkt operation in the packet information structure will override those specified by the

RadioVars and the schedule. In addition, for radio devices that support it, the protocol software may use RadioCmdRcvPkt operations to attempt packet receptions at a particular time and with particular radio or waveform characteristics. This may be done instead of, or in addition to, simply posting "unspecified" packet receive buffers to be filled with any receive packets, as is the only method for the core Radio Device API.

3.3.2 Schedule Timing

All TDMA schedule timing is based on the periodic *Logical Synchronization Strobe* (or simply, the *strobe*) which resides in, and is maintained by, the radio device.

The process of synchronizing the strobe may be accomplished in a number of ways (see Figure <tdma1>) such as:

- Embedded synchronization hardware and/or software running transparently in the radio device, perhaps using a "beacon" signal,
- Reception of timing pulses from an attached GPS device,
- Reception of timing pulses broadcast onto a cable tapped by a number of co-located radios, or
- Synchronization protocols run by the software above the API, possibly with control packets between nodes in the network, and then initializing and maintaining the strobe in the radio device using TRadioVarStrobe and TRadioVarStrobeAdj primitives.

Timing information about the strobe (the strobe period, the current strobe count, the offset into the current strobe period) is available through the TRadioVarStrobe primitive.

The following two rules defines how the strobes and schedules must relate:

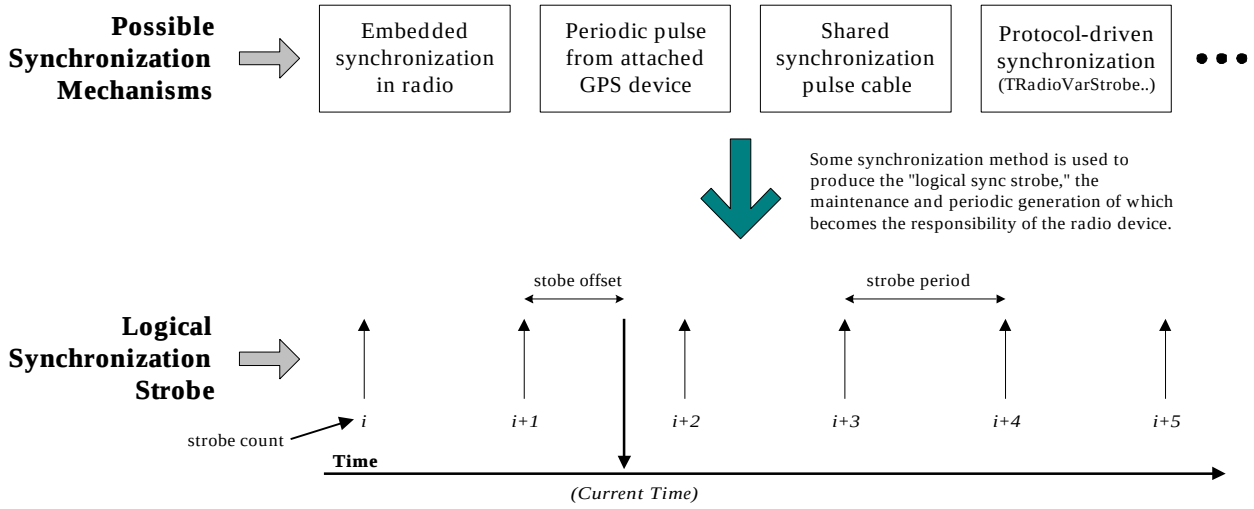

Figure <tdma1>: Logical Synchronization Strobe

1. **All schedules must be started on a Logical Synchronization Strobe.**
2. **All schedule-cycle lengths must be an integral multiple of the Logical Synchronization Strobe period.** [2]

[2] *For ease of implementation, we may want to modify rule 2. to be one of the following: "All schedule slot durations must be an integral multiple of the Logical Synchronization Strobe period." (As in Jason's original draft.), and/or, "If the last slot of a*

Therefore, the start and end boundaries of all schedules will coincide with the Logical Synchronization Strobes.

In a simple case, all of the slots in the schedule are defined with a duration equal to the strobe period. Because there are an integral number of slots in a schedule, the schedule cycle boundaries will coincide with strobes (see Figure <tdma2>). In another simple case, a schedule is defined with a cycle period exactly equal to the strobe period (see Figure <tdma3>).

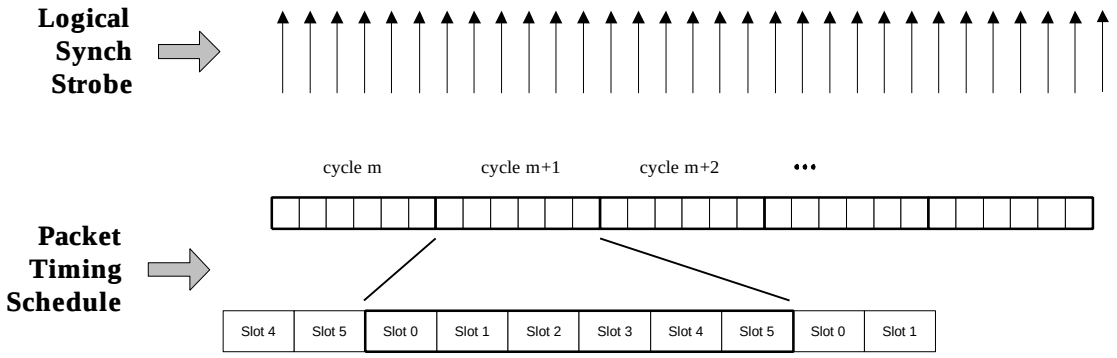

Figure <tdma2>: Example of Simple TDMA Scheduling

(For TDMA packet timing with slot boundaries coincident with strobes)

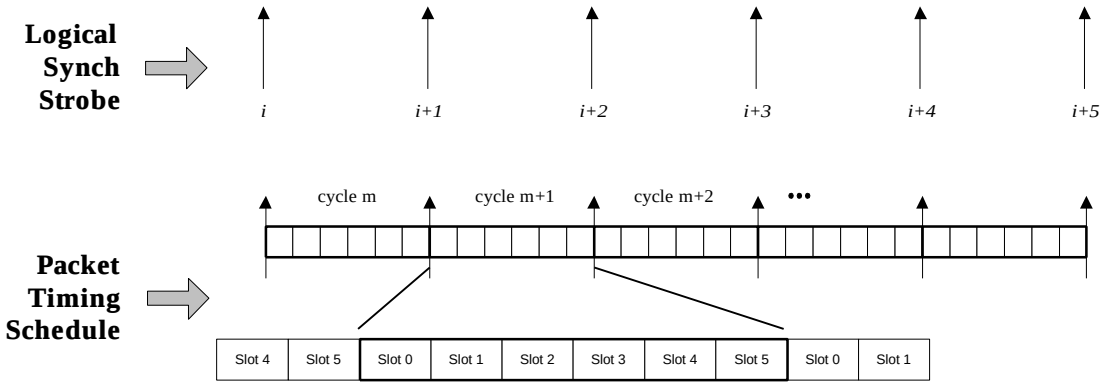

Figure <tdma3>: Second example of Simple TDMA Scheduling

(For TDMA packet timing with one schedule-cycle per strobe)

The strobe is "logical" in the sense that it may not actually refer to the physical channel directly. Figure <tdma4> presents an example where 40% of the physical channel is actually being used for "traditional broadcast voice" and embedded control traffic, by the radio device, or other process(es) independently and transparently to the protocols above the Radio Device API.

---

repeating schedule does not coincide (within some round-off error margin) with a Logical Synchronization Strobe boundary, then this last slot will be automatically extended to the next Strobe, when the schedule will be restarted at slot 0."

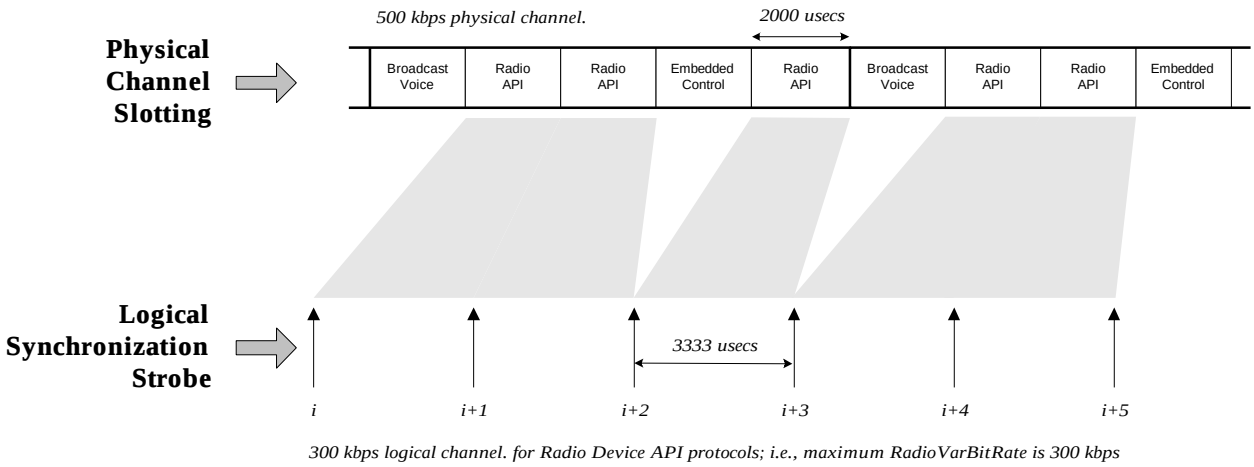

Figure <tdma4>: Example Mapping from Physical Channel to Logical Sync Strobe

(where 40% of channel is reserved for "traditional broadcast voice" & embedded control)

In addition, the capability to run both the packet timing and radio characteristics schedules simultaneously allows for quite sophisticated radio schedules to be specified by the protocols. For example, the slots of one schedule may be defined to exactly coincide with the entire schedule-cycle of the other schedule; i.e., one schedule may be "embedded" in the other. Figure <tdma5> presents an example where the packet timing schedule is embedded within the radio characteristics schedule.

3.4 Precedence for Radio Waveform Characteristics

The following precedence is used by the radio device to determine the current settings for the radio's waveform characteristics:

Highest precedence	Characteristics specified in a packet information structure for an active packet transmission or reception attempt.
Middle precedence	Characteristics specified in the current slot of an active Radio Characteristics schedule.
Lowest precedence	Characteristics specified by the persistent radio state variables (RadioVar... primitives).

Also, as in the core Radio Device API, packet transmissions have precedence over receptions.

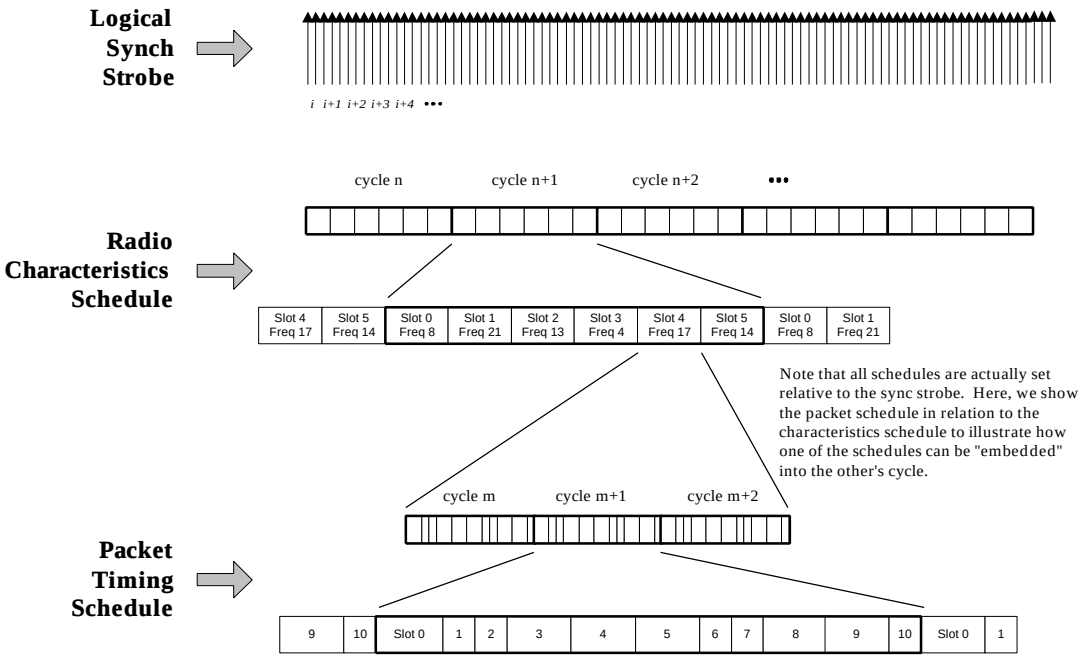

Figure <tdma5>: Example of Sophisticated TDMA Scheduling

(Independent characteristics and packet schedules, with one embedded in the other)

4 Primitive Descriptions

4.1 Commands

This section lists the new and extended or modified synchronous commands that can be issued by the protocols to the radio.

RadioCmdReset Command

Requirement: Mandatory
Qualifiers:
Data:
Description: A command used to reset the radio. Any receive or transmit packet buffers should be returned to the protocols using the RadioSigRcvPkt and RadioSigXmtPkt signals with an error qualifier indicating that no packet data was actually received or transmitted. If performed through a function call, the function should "block" until the reset operation has completed. ***If the radio is providing channel synchronization functionality, it should revert back to the 'unsynchronized' state.***

RadioCmdRcvPkt Command

Requirement: Mandatory
Qualifiers: *chNum*
Data: A packet buffer and its associated protocol buffer handle, ***Slot #, offset, duration, Radio WF mode & characteristics.***
Description: Command to pass a buffer to the radio to be used for received packet data, ***and optionally to specify reception timing. Slot # could be used to specify the 'time' in an active packet timing schedule at which reception should be initiated. Duration specifies the time over which reception should be attempted (in slots). If no packet is received, RadioSigRcvPkt is used to return the packet buffer to the protocols with a RadioRetPktRcvFail error code. WF mode and characteristics are used to specify the set of receiver parameters to be used. Section A.1.1.***

RadioCmdXmtPkt Command

Requirement: Mandatory
Qualifiers: *chNum*
Data: A packet buffer and its associated protocol buffer handle, ***Slot #, offset, Radio WF mode & characteristics.***
Description: Command to transmit a packet, ***and specify transmit timing***. ***Slot # and offset specify the 'time' in an active packet timing schedule at which transmission should be initiated. WF mode and characteristics is used to specify the set of transmitter parameters to be used. See Section A.1.1.***

TRadioCmdSched Command

Requirement: Mandatory
Qualifiers: *chNum*
Data: schedNum, schedType, slot duration(s), num of slots in sched, radio WF characteristics for each slot, periodic or one-time flag.
Description: Command used to communicate a TDMA schedule between the protocols and the radio. schedType identifies the type of schedule being communicated, as one of: T_RADIO_SCHED_PKT (for packet timing schedules); T_RADIO_SCHED_CHAR (for radio characteristics scheds); or T_RADIO_SCHED_BOTH (for a schedule to be used for both purposes). These schedule(s) are used to define the RF channel as a discrete set of time slots that are provide opportunities to xmt or rcv packets under tight timing restrictions, and/or to determine the radio xmt and rcv characteristics during each slot. Refer to the Section A.1.2 for more information.

TRadioCmdSchedRelease Command

Requirement: Highly desirable
Qualifiers: *chNum*
Data: schedNum
Description: Commands the radio device to "release access" of the memory containing the specified schedule number (e.g., clear any pointers that refer to this memory, but ***do not*** perform any operating system "free()" operations on any pointers received from the protocols!)

TRadioCmdSchedStart Command

Requirement: Mandatory
Qualifiers: *chNum*
Data: schedNum, strobeCount, immediateFlag, slotNum, slotOffset, inheritFlag
Description: Start (or restart) a schedule at the specified slot and at the occurrence of the strobeCount sync strobe. However, if the immediateFlag is set, the schedule start immediately at either the specified slot and offset, or by inheriting those of the currently active schedule (if any), depending on inheritFlag.

TRadioCmdSchedStop Command

Requirement: Mandatory
Qualifiers: *chNum*
Data:
Description: Stop the currently active TDMA schedule.

4.2 Variables

This section lists the new and extended or modified radio state variables.[3]

RadioVarRcvPreCorr Variable[4]

Requirement:	Highly Desirable
Qualifiers:	*get, chNum*
Data:	Preamble Correlation Quality
Description:	This value represents the correlation quality of a received DS SS signal, in terms of units or a table index as defined within the radio-specific header file.

RadioVarWFMode Variable <<still needs to be defined better, as part of ASPEN/WNR, then may move to core API>>

Requirement:	Optional
Qualifiers:	*get/set, chNum*
Data:	Waveform mode (coded value)
Description:	The radio-specific header file will contain a standard definition of its custom RF capabilities. One approach is to have a standard set of waveform modes that combine modulation, spreading, etc. attributes into a list of available waveform modes. A single variable can be used to represent a MODEM configuration that provides the desired performance.

TRadioVarSyncState Variable

Requirement:	Mandatory
Qualifiers:	*get, chNum*
Data:	TDMA synchronization status
Description:	A read-only variable that provides the current TDMA strobe synchronization state (e.g. Search, Passive, Active, Silent).

TRadioVarStrobe Variable

Requirement:	Highly Desirable
Qualifiers:	*set/get, chNum*
Data:	Current strobe count, strobe offset, strobe period.
Description:	Set or get the current timing of the Logical Synchronization Strobe. If all control of the strobe is owned by the radio device, attempts to set this primitive will result in RadioRetInvQual being returned.

[3] *TRadioVarClk and TRadioVarClkAdj were replaced by TRadioVarStrobe and TRadioVarStrobeAdj. The system clock is handled elsewhere, though may still want to add primitives to adjust timing of strobe relative to the system clock??*

[4] *If RadioVarPreCorr variable is only used in conjunction with a received packet (in the DevPktInfo struct), then this should be stated here. I.e., it be illegal to query this variable as a persistent variable "get" operation.*

TRadioVarStrobeAdj Variable

Requirement: Highly Desirable

Qualifiers: *inc, chNum*

Data: Positive or negative adjustment value for the strobe offset in usecs.

Description: This command can be used to adjust the radio's strobe timing. If the adjustment crosses a strobe boundary, the strobe count is adjusted as well. This capability is mandatory if the radio does not have embedded TDMA synchronization logic. If all control of the strobe is owned by the radio device, attempts to inc this will result in RadioRetInvQual being returned.

4.3 Asynchronous Signals

This section lists the new and extended or modified asynchronous signals that can be generated by the TDMA radio device.[5]

RadioSigRcvPkt Signal

Requirement: Mandatory

Qualifiers: *isr, chNum*

Data: Rcv'd pkt buffer, associated protocol buffer handle***, and associated Rx Slot # & offset.***

Description: A signal generated when a packet has been received. The protocol buffer handle is equal to that used in the corresponding RadioCmdRcvPkt command. This signal is also used, with the RadioRetPktRcvFail return code, to return a receive packet buffer to the protocols before it has been filled in with received packet data (due, for example, to a RadioCmdReset).

RadioSigXmtPkt Signal

Requirement: Mandatory

Qualifiers: *isr, chNum*

Data: Xmt'd pkt buffer, associated protocol buffer handle, ***and associated Tx Slot # & offset.***

Description: A signal generated when a packet has completed transmission. The protocol buffer handle is equal to that used in the corresponding RadioCmdXmtPkt command. This signal is also used, with the RadioRetPktXmtFail return code, to return a transmit packet buffer to the protocols before the data has been actually transmitted (due, for example, to a RadioCmdReset).

[5] *"Cryptographic Statusing" signal was mentioned in intro, but no signal was listed here. Should there be?*

TRadioSigSyncEvent	Signal
Requirement:	Highly Desirable (for radios that control synchronization)
Qualifiers:	*isr, chNum*
Data:	New TDMA synchronization status
Description:	A signal that indicates an asynchronous change in the radio's TDMA synchronization state.

4.4 Return Codes

<<no change>>

4.5 Summary of Primitives

<<no change>>

5 Radio- Specific Characteristics

<<no change>>

6 Acknowledgments

The development of this document was supported by … xxxxx.… . by the Defense Advanced Research Projects Agency (DARPA) through the Global Mobile (GloMo) program's Wireless Internet Gateways (WINGS) project (contract no. DAAB07-95-C-D157) … .

A. Generic Device Driver Implementation of Radio Device API

A.1 Commands, Variables and Signals

A.1.1 Packet Transmission and Reception

For the Generic Device Driver implementation, every packet Xmt and Rcv primitive for TDMA radios uses the following TRadioDevPktInfo structure:[6]

```
typedef struct tRadioDevPktInfo {
        RadioDevPktInfo     radioPktInfo;   / / must be first

        uint32              signature;      / / Identifies this type for debugging

        uint32              slotNum;        / / xmt or rcv slot number of pkt sched (or
                                            / / T_RADIO_SLOT_NUM_UNUSED)

        uint32              slotOffset;     / / offset into slot in usecs (or
                                            / / T_RADIO_SLOT_OFFSET_UNUSED)

        uint32              rcvDuration;    / / The duration, in slots, during which
                                            / / reception should be attempted (or
                                            / / T_RADIO_SLOT_RCV_DURATION_UNUSED)
} TRadioDevPktInfo;
```

A.1.2 Specification of TDMA schedule

The TRadioCmdSched command primitive uses the structure described below to allow the protocols to specify a TDMA schedule to the radio device. By using the "schedNum" field, the protocols can specify more than one schedule (such as a schedule for each of a series of "slow" frequency hopping patterns). The protocols would then be able to dynamically switch among the various schedules. The memory used for these structures must remain accessible by the radio device and shouldn't be modified by the protocols until it is released with a TRadioCmdSchedRelease command for this schedule number.

The TRadioCmdSched command uses the following structures to convey a new TDMA schedule to the radio device:

[6] *Deleted "logical channel number" in favor of "charGroup" number and "WFMode" settings. E.g., slow frequency hopping is handled above the API, or by setting a radio characteristics schedule which is then run below the API.*

```
typedef struct tRadioSched {
        uint32          signature;      / / Identifies this type for debugging

        uint32          schedType;      / / One of: T_RADIO_SCHED_PKT,
                                        / / T_RADIO_SCHED_CHARS, or
                                        / / T_RADIO_SCHED_BOTH

        uint32          numSlots;       / / The number of slots in the schedule

        uint32          oneTime;        / / If TRUE, the schedule is to be only run
                                        / / to the end (through the last slot), before
                                        / / stopping and converting to the persistent
                                        / / radio characteristic variables.  Otherwise
                                        / / the schedule is to be repeated
                                        / / continuously until commanded by the
                                        / / protocols to stop or switch schedules.

        uint32          slotDuration;   / / The duration of each slot in the schedule
                                        / / in usecs, unless this field is 0, in which case
                                        / / the slot durations may be variable, and will
                                        / / be specified in the durations[] array below.

        TRadioSlotInfo  slotInfo[numSlots]; / / Information for each slot
} TRadioSched;


typedef struct tRadioSlotInfo {
        uint32          signature;      / / Identifies this type for debugging

        uint32          duration;       / / slot duration in usecs (used if default slot
                                        / / duration (in TRadioSched) is 0, else ignored)

        uint32          fwdSlotNum;     / / slot number to xmt any packets rcv'd in this
                                        / / slot or T_RADIO_SLOT_NUM_UNUSED.
                                        / / If there is also a RadioCmdPktRcv pkt buffer
                                        / / posted for this slot, then the pkt will both be
                                        / / forwarded, and passed up to the protocols.

        uint32          fwdSlotOffset;  / / fowarding slot offset or
                                        / / T_RADIO_SLOT_OFFSET_UNUSED

        uint32          fwdMacAdr;      / / Dest MAC address to use for forwarded
                                        / / packekts.

        uint32          charGroup;      / / The groupNum, set previously using the
                                        / / RadioVarGroup primitive variable, that defines
                                        / / the radio characteristics to use for this slot,
                                        / / or RADIO_CHAR_GROUP_UNUSED.

        uint32          charModsNum;    / / Number of characteristics modifiers
                                        / / in charMods[] array

        DevChar         charMods[RADIO_SLOT_CHAR_MODS_MAX];
                                        / / List of radio characteristics used for this
                                        / / slot.  These modifiers have precedence
                                        / / over the charGroup specified, if any.
} TRadioSlotInfo;
```