TDMA for Long Distance Wireless Networks

Sam Leffler, Errno Consulting

sam@errno.com

Background

In 2006 the UC Berkeley TIER Project studied the need to deploy network infrastructure to emerging regions. Their goal was to "address the challenges in bringing the Information Technology revolution to the masses of the developing regions of the world" [1]. Out of that work came plans for WiFi Based Long Distance Networks (WiLDNet) that leveraged commodity 802.11 wireless parts to preserve cost savings [2]. WiLDNets were specifically intended for deployments where station separation far exceeded normal 802.11 use (e.g. 30-100 kilometer station separation). A key component of the WilDNet design was the use of a Time Division Multiplex Access (TDMA) MAC [3] in place of the standard DCF MAC that is part of the 802.11 specification [7].

About the same time the Intel Research Laboratory in Berkeley, CA developed the Rural Connectivity Platform (RCP). TIER was a joint project between UC Berkeley and Intel and the RCP was an offshoot focused more on easy setup, low maintenance, and production use. For TDMA the RCP project developed a unique scheme to leverage the capabilities of Atheros wireless devices to implement TDMA almost entirely in hardware [4]. The scheme has proven to work very well and has been incorporated in commercial products. In 2008 the RCP software base was moved from Linux to FreeBSD and in 2009 Intel released the FreeBSD implementation of the TDMA software as open source. This paper describes the protocols and algorithms as they appear in FreeBSD 8.0.

TDMA Background

Most 802.11 networks use the Distributed Coordination Function (DCF) to arbitrate access to the shared wireless medium. With DCF each station first tests if the medium is busy using a Channel Control Access (CCA) mechanism. If the medium is deemed "free" then the station attempts to transmit a frame. Transmit attempts may still collide in which case each station waits a small random amount of time before trying again to use the medium. Once the frame transmit completes, the sending station optionally listens for an Acknowledgement frame from the receiving station. If no Ack frame is received within a timeout then the frame is transmit again.

DCF is designed for networks where stations are near each other. With highly distributed networks stations may not hear each other resulting in collisions and excessive retransmits. This issue is termed the "hidden node problem". Also, for very long distances the time required to wait for each frame to be acknowledged can dramatically affect performance. All these factors motivate the use of a different scheme than DCF.

Time Division Multiplexed Access (TDMA) eliminates mediated access by assigning each station in the network a *time slice* during which it may transmit. No arbitration is required once a slot is assigned to a station. TDMA is a good way to handle media access in long distance networks but it does

potentially increase packet latency and waste bandwidth because stations can transmit only during their time slot. To mitigate the latency problem one can use short time slots. It is possible to reduce unused bandwidth by dynamically assigning slots and/or changing their size but this can significantly increase complexity and overhead. In general the balance between latency and throughput means the time slot configuration in a TDMA network should be chosen according to the needs of applications (e.g. latency must be low enough for packetized voice data).

TDMA requires a slot assignment protocol and a synchronization algorithm to ensure time slots do not collide as the clocks drift. Implementing TDMA purely in software is feasible but can use significant resources to guarantee the real-time constraints. Also, software-based TDMA implementations usually perform worse than those with hardware assistance.

In the sections below we describe a TDMA network configuration in terms of:

- The number of time slots in the network ("*N*").
- The length of each time slot ("*slot-length*").
- The time between each time slot needed to safeguard packets transmit during that slot ("*guard-interval*").

The time between each station's time slot is termed the *superframe* and is defined as:

$$superframe = N * (slot - length + guard - interval)$$

Figure 1 shows a typical TDMA configuration with two slots (A and B), slot-length 2.5 milliseconds, and a superframe of 5 milliseconds. The selection and/or calculation of TDMA configuration parameters is discussed below.

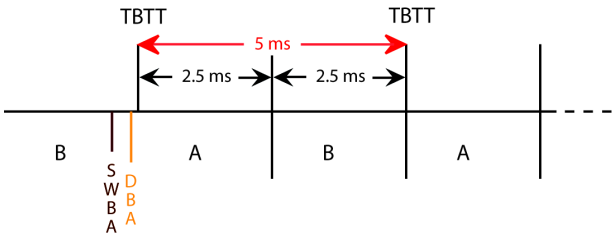

Figure 1: Example TDMA configuration.

TDMA for wireless networks is not a new idea. There are several wireless products that use custom hardware to do TDMA. Also IEEE 802.16 (aka WiMAX) includes TDD support that can do dynamic slot allocation. The work described here is novel in that it leverages commodity 802.11 hardware to implement TDMA.

FreeBSD TDMA Support

The TDMA support in FreeBSD is made up of two components: an 802.11 protocol module that fits into FreeBSD's net80211 network layer, and device driver support for the Atheros hardware. The 802.11 protocol component is device-independent and designed to support multiple stations in a TDMA BSS. The protocol support uses vendor extensions to the 802.11 protocol so that it can coexist with other 802.11 networks[1]. The device driver support is responsible for implementing the TDMA scheduling mechanism by which frames are sent over the air. For the Atheros driver this code is small as the hardware is well-suited to doing TDMA.

In our TDMA design one station is defined to be the "*master*". The master is required to broadcast

1 One would not normally setup a TDMA BSS on the same channel as another BSS because performance would suffer.

beacon frames when it becomes operational (e.g. locates a channel to operate on). These beacon frames include a special TDMA Information Element (IE) that identifies the station as operating in TDMA mode and includes information such as whether there are free slots in the network. Slave stations search for a master using a pre-assigned SSID.

Non-TDMA stations do not try to join a TDMA BSS because the beacon frames lack the necessary capabilities (BSS for Infrastructure mode and IBSS for adhoc mode). This is the same technique used by the forthcoming 802.11s specification to distinguish Mesh stations from Infrastructure/IBSS stations [8].

Once a slave finds a master that has a free slot it adopts the TDMA configuration of the master and begins transmitting it's own beacon frames. These frames include a TDMA IE that identifies it's selected TDMA slot. The master sees the slave's beacon frames and records the now occupied slot at which time the slave is free to start operating. If multiple stations try to join a BSS simultaneously the master chooses the station with the oldest TSF as the “winner”. The station that did not get a slot either chooses another slot (if available) or continues searching for a BSS with an open slot.

Beacon frames are used by slaves to synchronize their time slot. With the current scheme (described more below) the master operates without adjusting its timer and the slaves adjust their timers to match. Slave stations never transmit a beacon frame except in response to receiving a frame from the master (in a two station network).

Aside from time slot synchronization beacon frames also serve to identify when a station leaves the BSS. If a slave stops hearing beacon frames from the master it may roam to find it. Likewise if a master stops hearing a slave it will mark the time slot in the BSS available. Coordinated channel changes are also done using the Channel Switch Announcement (CSA) mechanism that is part of 802.11h [9].

The master station in a TDMA BSS is responsible for propagating TDMA configuration parameters to other stations in the BSS. For example, the slot length (in microseconds) is transmitted by the master and slaves adopt it when they join the BSS. This mechanism is used to auto-configure stations when they join and also to dynamically update parameters.

TDMA slots are numbered from 0 with the master assigned slot 0 and the slave slot 1 (when more than two slots are in configured slave slots are numbered 1 to N-1). While the TDMA protocol is designed to support multi-slot networks, in practice it has been optimized for two stations; i.e. point-to-point links.

Net80211 TDMA Support

FreeBSD 8.0's wireless networking is based on a virtual radio architecture that uses device cloning to present network services [5]. A cloned device is termed a “vap”. TDMA fits into this model; users create a TDMA vap that is cloned just like other 802.11 devices[2].

```
ifconfig wlan create wlandev ath0 wlanmode tdma
```

2 TDMA vaps are actually implemented using “adhoc-demo mode” vaps. These are a general-purpose vap that provides direct station-to-station communication and neighbor discovery a la IBSS operation.

The TDMA master and slave roles described above are configured by setting the TDMA slot for a vap. By default a vap is assigned slot 1 and it will not transmit beacon frames. To configure master operation one just overrides this slot assignment:

```
ifconfig wlan0 tdmaslot 0
```

Similarly the other TDMA parameters can be configured with ifconfig:

```
ifconfig wlan0 ssid freebsd-ap tdmaslotcnt 2 tdmaslotlen 2500 tdmaslot 0
```

Recall that a slave adopts TDMA configuration from the master so the only parameter that must be set is the SSID. Parameters set for a vap operating as a slave are not used.

Scanning happens automatically when a TDMA vap is marked up. Likewise beacon miss handling may trigger scanning. Both operations are managed using the state machine framework that is a standard component of net80211. If 802.11h support is enabled for the vap then coordinated channel changes are supported using CSA.

A TDMA vap is designed for building a point-to-point bridge using the standard Layer 2 bridging mechanism. Setting up the master side of a TDMA bridge is as simple as:

```
ifconfig wlan create wlandev ath0 ssid freebsd-tdma tdmaslot 0 up
ifconfig bridge create addm wlan0 up
```

A fixed transmit rate is used for sending data frames because there are no Ack frames to provide feedback to a rate control algorithm. The default rate is chosen as the highest available rate that also optimizes transmit power. For example, when operating on an 802.11a channel 24Mb/s is used. This is configured on the command line with:

```
ifconfig wlan0 ucastrate 24
```

FreeBSD 8.0 requires all stations in a TDMA BSS manually set the transmit rate. Forthcoming changes include this in the TDMA configuration parameters sent in each beacon frame.

Atheros Driver Transmit Support

The driver support for TDMA transmit is straightforward. Atheros devices support multiple hardware transmit queues that are programmed to dispatch frames to the Protocol Control Unit (PCU) according to one of several scheduling algorithms (see Figure 2). Typically data frames use an "ASAP" scheduling algorithm that dispatches frames immediately as they appear in the queue. For access point operation the hardware also supports a frame

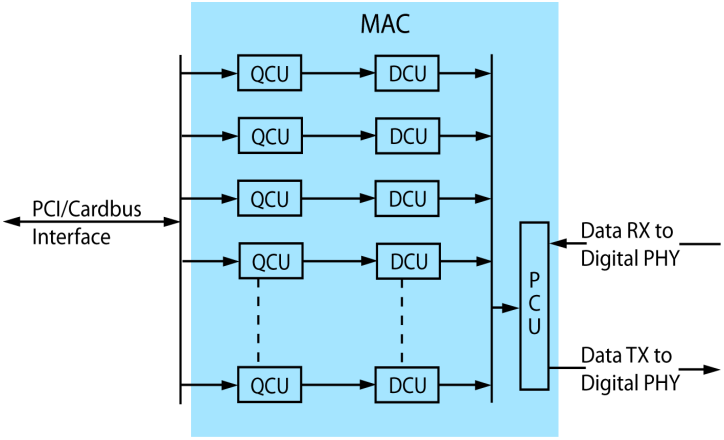

Figure 2: Atheros AR5212 MAC

scheduling algorithm where frames are dispatched according to the beacon timer for sending beacon frames. By applying this frame scheduling algorithm to the normal data frames the hardware can be used to schedule data frames according to the beacon schedule.

The only remaining requirement for the frame scheduling is to limit traffic so frames are transmit only within the assigned time slot. This turns out to be easily handled by setting the maximum duration for bursting frames according to the TDMA slot length. Aside from a few other minor details that is everything required to have the hardware schedule and send frames according to a periodic schedule defined by the hardware beacon timer(s).

One complication that arises with using the hardware directly is that the Transmit DMA engine that walks the linked list of transmit descriptors runs asynchronously to the driver. Normally the driver triggers a descriptor list walk so it knows when it is safe to reclaim transmit descriptors for operations that have completed. But with the asynchronous operation the “next descriptor” field in the last descriptor in the chain may be re-read at any time by the hardware so one must reclaim the descriptor carefully.

There is a related issue caused by asynchronous operation of the DMA engine. The driver tells the hardware new descriptors are present by writing a register that holds the address of the first entry in the descriptor list. But if the hardware is actively traversing this list any write is ignored. The driver handles this by checking if the DMA engine is active before doing these writes and if it is then it records the need for a deferred write. Deferred writes are processed when another packet is submitted for transmit or a beacon frame is prepared.

Note the above changes fit within the existing driver/hardware framework. This means, for example, that priority queueing of QoS frames is still done so even while frames are transmit according to a TDMA schedule WME/WMME service is preserved. Also the mechanisms described here work with all Atheros parts (though some parts cannot be used for reasons described below).

802.11 ACK Frames

With the current TDMA implementation there are no 802.11 ACK frames. This is done by encapsulating all traffic with a QoS header and setting the ACK Policy field to “No ACK required”. No driver changes are required for this. Net80211 marks neighbor stations as using QoS so that frames are suitably encapsulated and user configuration forces the ACK Policy.

TDMA Clock Setup

The hardware transmit machinery is driven by the hardware beacon timers and the packet bursting controls of each hardware transmit queue. Each station's time slot is scheduled by setting the beacon timer interval according to the TDMA BSS' superframe. The time slot length is controlled by the burst duration assigned to each hardware transmit queue.

The beacon timer in the AR5212 MAC has a 1 TU (1024 microseconds) granularity which is very course-grained. To compensate for this limitation the driver must round up the calculated superframe to a multiple of 1 TU. The packet bursting duration has a one microsecond granularity which is perfect for our needs.

A more important factor in calculations is that the hardware transmit machinery does not "*kill a frame in flight*" that might exceed the burst length. That is, the hardware might start transmitting a frame just before the end of the slot and overflow into the next slot. To handle this we calculate a "guard time" that covers this possibility. When a fixed transmit rate is used to send data frames the guard time includes the time to send a maximal-sized frame at the fixed rate. Using a variable transmit rate requires this calculation to be done with the lowest possible rate.

TDMA Clock Synchronization

Once the TDMA transmit scheduling is in place the other key requirement is to synchronize scheduling of each station so frames do not collide. The obvious way is to implement a consistent clock across all stations in the BSS. There are existing algorithms for doing this when there are three or more stations and/or when there is an external reference clock (e.g. from GPS). But in a point-to-point network there are only two stations and we cannot assume a reference clock source will be available. Instead a novel algorithm was used that leverages information obtained from the wireless hardware to work with only two stations and without a reference clock. Not only does this algorithm synchronize the slot scheduling of each station in the network but it also works independent of the distance between stations. This turns out to be very useful both for fixed and mobile applications.

Recall that the master station sends beacon frames periodic in the superframe. This process is driven by the hardware beacon timers. The slave station is responsible for synchronizing with the master. This is done using the master's beacon frames. The wireless hardware provides a snapshot of the *local TSF* in the receive descriptor of the beacon frame. This value is copied from the TSF at the point where the DMA engine pushes the received frame to the host and can be adjusted by the air time to calculate the value at the point where the start of the frame is received. Additionally the snapshot is only 15-bits (on the MACs currently supported) and must be extended to 64 bits for calculations. Given a 64-bit microsecond time for when a beacon frame is received it is simple to calculate the next time at which the receiving station should transmit—i.e. the start of the next TDMA slot. This value is then compared to the hardware timers to calculate any adjustment required. No knowledge of the propagation delay is needed for the calculations because the slave clock is synchronized using only the local time source.

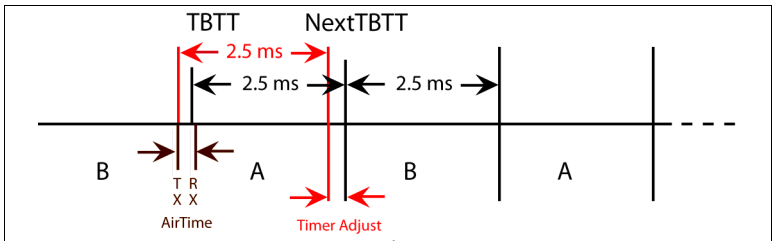

Figure 3: NextTBTT Timer Adjustment

The key detail in the above algorithm is that we need to effectively do sub-TU adjustments of the hardware beacon timers while they are running. The beacon timer registers do not have sufficient granularity but one can write the TSF directly if done with care. In particular the hardware does not handle setting timers *backwards*. However because we do not synchronize clocks but only the difference between clocks the actual value of each station's TSF does not matter. This means we can implement a negative timer adjustment *T* by setting the hardware timer forward (*superframe-T*). Together with some other hard-learned tricks it is possible to make the necessary adjustments without having the hardware lockup. One caveat is that the techniques we employ only work for a class of Atheros AR5212 chips; the driver enforces this by enabling TDMA support only for capable devices.

Using the above scheme the time slots are typically synchronized within 3 microseconds. There may be wider swings if beacon frames are not received. Transmit/receive conflicts caused by loss of synchronization show up as Transmit Override Receive (TOR) PHY errors recorded by the radio. The driver also maintains detailed statistics about clock synchronization. And each beacon frame returns the 64-bit time stamp of the other station's last beacon beacon frame so stations can track “round trip time”.

Note that timers are synchronized only from beacon frames. The default beacon interval schedule is chosen to be similar to a normal Infrastructure BSS; e.g. for a two slot BSS with a 2.5 millisecond time slot beacon frames are sent every 20 superframes which is comparable to the 100 TU value often used by access points. The frequency of beacons is a tradeoff between the overhead to send and receive the frames and how often it is necessary to calibrate the clocks (it also constrains how quickly TDMA parameters can be changed). The Atheros radios seem to very have stable clocks and require minimal adjustment. If issues are noticed however one can use more beacon frames and adjust more often. One can also send beacon frames at intermediate superframe boundaries as needed without any consequences.

Performance

The performance of the TDMA implementation was initially evaluated using a testbed where the stations reside in RF isolation chambers and are cabled together. To simulate the propagation delay of long distance separation a Spirent SR5500 Wireless Channel Emulator was used [10]. Each station was built-up from a Gateworks 2348 board [11], high transmit power Atheros wireless card (Wistron DCMA-82), and FreeBSD software. Attenuators were used to avoid overloading the RF components in the system.

Basic network performance was measured using tools such as netperf [12] and iperf [13] while monitoring throughput and error statistics collected by the hardware and software. To verify the slot timing synchronization concurrent unidirectional UDP “blast tests” were run to saturate the channel in each direction. Different TDMA network configurations were tested while varying the channel delay to simulate inter-station separation between 0 and 200 miles. The Spirent device uses digital signal reproduction so signal attenuation due to distance did not occur (allowing tests at “impossible” distances).

Test results were consistent with expected results calculated from the TDMA parameters. Packet loss is zero. Cumulative channel bandwidth is 70% or more of the fixed transmit rate. There is a drop as the slot length is reduced toward 1 millisecond and the host overhead required by additional interrupts for each beacon interval becomes noticeable and affects packet processing.

Tests were repeated without the link emulator hardware by just using inline attenuators and results were identical. Table 1 shows results for this configuration with 80 dB of pad (RSSI 14-15 dBm) operating in 802.11a on channel 36. Longer slot lengths have slightly better bandwidth as one would expect. One anomaly is that transmit rates above 24Mb/s show poorer than expected performance (especially with larger slot lengths). At 54Mb/s this was accompanied by CRC errors that are (as yet) unexplained.

Finally outdoor tests were run under varying conditions with consistent results. In all situations it is critical to verify the hardware setup for noise that might be introduced by cables or noise sources such as fans, rotating media, and power supplies. Because there are no link layer acknowledgements throughput is sensitive to packet loss. Typically one can identify the cause from error statistics but sometimes it cannot be done without an independent device to monitor the channel. For this reason a third station was always present to do passive monitoring.

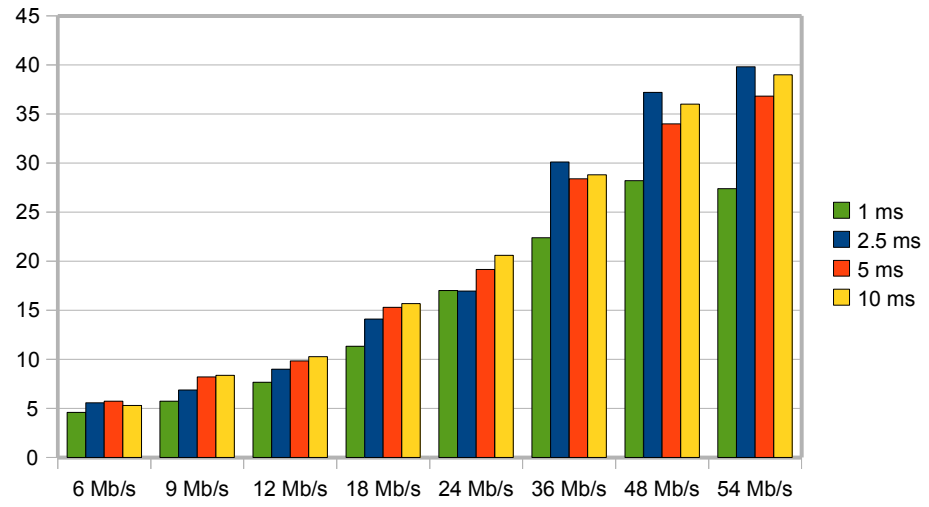

Table 1: Cumulative bandwidth with 80 dB fixed attenuation on 5180/20.

Frame rates and detailed packet traces were also collected to verify inter-packet gaps and media access overhead were at a minimum. Initially it was believed we would need to disable CCA for optimal performance but experiments indicate otherwise. This is important as disabling CCA is not allowed in most locales as it violates local regulatory requirements.

Time slot jitter was also measured to evaluate the synchronization algorithm. As reported previously TSF adjustment on the slave was typically less than 3 microseconds. This value was measured using a rolling average over a twenty sample window. Wider swings seem to occur with lost beacon frames though this is dependent on the frequency with which beacons are sent. One interesting observation was that jitter was noticeably worse when operating on a 2.4GHz channel than on a 5GHz channel. This may be due to the conversion of the MAC's internal clock for some base-band circuits. These anomalous results were not observed when the clock was varied for narrower (5MHz and 10MHz) or wider channel widths (40MHz for Turbo mode). However even with the increased jitter operation on 2.4GHz channels was not affected (e.g. no packet loss was observed due to time slots colliding).

Future Work

The FreeBSD 8.0 TDMA protocol does not include all network configuration parameters in the beacon frames. Adding the fixed transmit rate will enable full auto-configuration of slave stations. It is also minimal effort to add a per-station “bandwidth allocation” mechanism that can be used to vary the slot length at each station. Dynamic changes to these parameters can be done at each beacon interval to improve throughput by varying “upstream/downstream bandwidth” as proposed for systems such as WiMAX [6].

Frame encapsulation is done using a 3-address 802.11 header format. This only works for a network with two stations (as the peer association is implied). It also makes encrypting traffic awkward.

Switching to a 4-address format header is required to support more than two stations. This change also simplifies the handling of keys as they can be maintained for each pair of stations in the BSS using existing facilities.

The current scheme provides no link-layer acknowledgements. Packet loss is normally low so this has not been important. But when there is significant packet loss it affects high level protocols such as TCP. There are extensions to the normal TCP algorithms that try to address lossy links such as one might encounter in a wireless environment. Another option is to extend the TDMA support to include explicit acknowledgement and packet retransmit. This can readily be done with a "Bulk ACK" mechanism tied to the slot scheduling. But without sufficiently short slot lengths retransmissions done at the link layer do not help as delays in TCP ACK frames are still noticed by the TCP protocol causing throughput to drop.

The 802.11 TDMA protocol support is designed to support a BSS with up to eight stations but the time synchronization algorithm has only been field-tested with two-station networks. Once there are more than two stations participating many other algorithms are know to work well, but none of these algorithms handle arbitrary station separation as the current algorithm does. It may be possible to extend the current slot synchronization algorithm to more stations (experimental results are encouraging for up to four stations).

The Atheros AR5212 hardware limits the effectiveness of the TDMA support. The 1 TU beacon timer granularity and the lack of PCU packet kill support result in significant underutilization of the medium. Newer Atheros hardware addresses both these problems: starting with the AR5416 the beacon timer has a one microsecond granularity and the hardware kills frames queued to the PCU when the maximum burst duration is reached. Newer hardware would also enable higher throughput using 802.11n transmit facilities .

Conclusions

The TDMA wireless support in FreeBSD has proven to be a valuable facility for deploying networks. It has been used for everything from remote monitoring to building network infrastructure where it is impractical to use normal facilities [14]. The ability to automatically configure the link layer parameters without regard for station separation is useful for fixed point applications and, in the future, is expected to enable new mobile applications. The software support is simple to understand, robust, and enables the construction of systems for very low cost. Features such as bulk acknowledgements and dynamic bandwidth assignment can be done entirely in software to improve service. Support for newer 802.11n hardware devices will improve network utilization and enable greater bandwidth.

The original TDMA work was done for the Intel Research Laboratory in Berkeley, CA who graciously released it to the general community. Kevin Fall and Alan Mainwaring of Intel were key contributors. Recent work was supported by Carlson Wireless Technologies and Hobnob, Inc.

References

[1] "Rethinking Wireless in the Developing World", Lakshminarayanan Subramanian, Sonesh Surana, Rabin Patra, Sergiu Nedevschi, Melissa Ho, Eric Brewer and Anmol Sheth. Hot Topics in Networks (HotNets-V), November 2006.

[2] "WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks", Rabin Patra, Sergiu Nedevschi, Sonesh Surana, Anmol Sheth, Lakshminarayanan Subramanian, Eric Brewer; USENIX NSDI, April 2007.
[3] "Packet Loss Characterization in WiFi-based Long Distance Networks", Anmol Sheth, Sergiu Nedevschi, Rabin Patra, Sonesh Surana, Lakshminarayanan Subramanian, Eric Brewer IEEE INFOCOM, 2007.
[4] "Long Distance Wireless (for Emerging Regions)", Sam Leffler, Errno Consulting; EuroBSDCon, September, 2007.
[5] **net80211(9)** manual page in FreeBSD 8.0.
[6] "Adaptive Downlink and Uplink Channel Split Ratio Determination for TCP-Based Best Effort Traffic in TDD-Based WiMAX Networks", Chih-He Chiang, Wanjiun Liao, Tehuang Liu, Iam Kin Chan, Hsi-Lu Chao; IEEE Journal on Selected Areas in Communications, Vol. 27, No. 2, February 2009.
[7] *IEEE Standard for Local and Metropolitan Area Networks Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*, IEEE Std 802.11, 1999.
[8] "IEEE P802.11s/D3.0", March 2009.
[9] *IEEE Standard for Local and Metropolitan Area Networks: Amendment 5: Spectrum and Transmit Power Management Extensions in the 5 GHz band in Europe*, IEEE Std 802.11h, 2003.
[10] Spirent Comunications. http://www.spirentcom.com
[11] Gateworks Corp. http://www.gateworks.com/products/avila
[12] netperf. http://www.netperf.org/netperf
[13] iperf. http://iperf.sourceforge.net
[14] "HPWREN collaborates South of the Equator", James Hale, David Rabinowitz, Sam Leffler; http://hpwren.ucsd.edu/news/20090516, May, 2009.