European Collaborative Innovation Centres for Broadband Media Service

One way delay estimation in Multiradio Mesh Networks with Channel Switching

Case Study Report

EUROPEAN UNION
European Regional
Development Fund

Investing in the future by working together for a sustainable and competitive region

The Interreg IVB North Sea Region Programme

One way delay estimation in Multiradio Mesh Networks with Channel Switching

Mats Persson, Rickard Karlsson, Peter Dely, Andreas Kassler
Computer Science Department
Karlstad University

Abstract
Wireless Mesh Networks (WMNs) are an interesting option for extending Internet access to areas where the deployment of wireline Internet access is either impossible, more costly, or simply more cumbersome and less comfortable. WMNs establish access to the Internet over multiple wireless hops. A mesh client connects to a mesh router and the mesh routers form a mesh backbone consisting of wireless links which may operate on different channels. Some mesh routers are attached to the Internet via wire-line connections and are hence called mesh portals or mesh gateways. This case-study focuses on the estimation of the one-way delay of IP packets which we define as the time between entering the mesh backbone and the time when leaving it again at the mesh gateway or the last mesh router in the case of intra-mesh traffic. One obvious way to profit from one-way delay measurements would be to utilize it as a routing metric. Furthermore, one-way delay measurements could be used for selecting the optimal gateway for real-time applications or simply for monitoring the WMN performance. To solve this problem this thesis presents TOM++, an improved implementation of the Tool for One-way delay Measurement (TOM). The improvements take queuing time and channel switching into consideration when performing one-way delay measurements in a multi-radio, multi-channel wireless mesh network based on Net-X. This has been accomplished by approximation of the sending time of packets in the transmission queue at each intermediate node. TOM++ has been tested and evaluated on the KAUMesh LivingLab testbed, and this resulted in an increase of approximately 5% in the one-way delay measurement accuracy.

1 Introduction

A wireless mesh network is a network that is structured like a mesh with radio links as data bearers. The mesh is constructed of mesh routers which connect wirelessly with each other. For Internet connection every mesh network needs to have dedicated mesh routers with gateway functionality relay packets between the networks. Communication between nodes in a wireless mesh network is broken down into a series of hops. This allows data to be sent over a large distance. Traffic in wireless mesh networks is often forwarded to and from a wired network through a gateway. There are three generations of wireless mesh networks; Single-Radio, Dual-Radio, and Multi-Radio. Single-Radio uses only one radio channel for both clients and backhaul, therefore providing merely low performance. In a Dual-Radio Mesh the mesh routers are equipped with two radios; one for clients and one for the backhaul. The radios can operate at different frequencies so that sending and receiving can be done in parallel. Although the performance is improved from Single-Radio Mesh the backhaul is still operating at the same frequency on all mesh routers. Multi-Radio Mesh uses different channels for the links in the backhaul that does not interfere with adjacent links and one channel for servicing clients on a different interface. This increases the performance significantly since each link can be used in parallel.

2 Hybrid Multiradio Architecture

By increasing the number of interfaces, one can also increase the upper limit of the capacity in the network. More channels will be needed to be able to have more interfaces sending at the same time. Depending on the country there can be up to 14 channels available in 802.11b/g and up to 13 channels in 802.11a. In a simple approach, channels are assigned to interfaces in a static way. While this makes channel assignment simple, it may lead to longer routes and network disconnections. In the hybrid multiradio architecture, a node uses multiple interfaces for sending and receiving packets. One of these interfaces is called the Fixed Interface, as it is allocated at a fixed channel over rather long periods of time compared to the switchable interface. The fixed channel is also selected by a fixed channel selection protocol and can automatically adapt in accordance with for example the interface situation and the traffic load. The other interface on each node is called the Switchable Interface. It can switch dynamically between any of the remaining channels according to traffic demand, and the maximum and minimum channel switch interval can be configured. All the data that is sent to a given node must be addressed to the fixed channel of the receiving interface since that is the only channel the node always listens on. A node can send data through the fixed interface if the receiver shares the same fixed channel; or else the switchable interface of the

sender will change to the receivers fixed channel, allowing the data to be sent through the switchable interface instead. Because of this, the mechanisms do not require common control channels.
In order to inform neighbors about their fixed interface channel, each node periodically broadcasts a "Hello"-message on all channels. The message carries the fixed channel of all 1-hop nodes. Through this a node can learn the fixed channels of all 2-hop nodes and then use the information to balance the assignment of the fixed interface. Once a node changes its fixed channel, it sends out this information including the learned information from one hop neighbors. As the time to switch between channels is in the order of a few milliseconds, switching on a per packet basis is too high overhead. Instead, once tuned to a given channel, the switchable interface stays there for some minimum amount of time to reduce the overhead. However, once the channel is switched, packets may arrive that need to be forwarded on a different channel. Therefore, per channel packet buffers are used which hold all those packets to be sent out on channels which are currently not served. A simple round-robin channel scheduler serves different channel queues over time.

3 Delay Monitoring

Monitoring packet delay within a network is important as it allows to infer different characteristics. There are in principle two approaches. Active monitoring approaches typically generate probing traffic. This results in an overhead and lowers the capacity of the network. However, such active monitoring is flexible as it gives control over the generating of monitoring granularity. There are two different ideas for active monitoring: Round Trip Time(RTT) and Packet Pairing(PP). The PP idea is if two packets are sent back-to-back they should arrive also close by within router queues. At the probing receiver the time when the first packet has been received is then compared to the time when the second packet starts to be received. This is called the dispersion or transmission time. If it is possible to add a timestamp just before the transmission is made, it will result in a very accurate estimation of the transmission time. The downside of this idea is that the clocks at the nodes have to be synchronized, since hardware clocks in general are not coordinated and not even equally fast, synchronization is difficult. In the RTT idea, no synchronized clocks are necessary, it only needs to record the time until a packet returns from the receiver using e.g. ping command. In a wireless environment the downside of this idea is that it assumes that the receiver can answer very fast and that the links are symmetric. Due to different switching patterns in hybrid wireless mesh networks, such approach would not give an exact time of the one-way delay. Another disadvantage of this idea is that additional traffic needs to be generated to update the measurements. Information gathered this way will only be available at the node that triggered it. Further, the probing packets are very small, and the consequence is that the measurements may underestimate larger packets.
In passive monitoring, characteristic of sniffed packets are used to infer the delay. The basic idea is to log an estimate of the experienced packet delay within the packet header. Each node then can exactly calculate the intra node queuing delay and estimate the one hop forwarding delay along the multihop path. Those measured and estimated delay components will then be used to update the experienced packet delay. TOM has been implemented as a passive monitoring tool that is used to estimate the one-way delay of packets in a wireless mesh network. TOM uses two main ideas. The first one is an estimation of the one-way delay as the sum of the exactly measurable intra-node queuing and processing delays and the estimated inter-node transmission delays. The second idea is that each packet carries delay information in its IP header option field to make it immediately available. As the transmission delay can only be approximated due to random backoffs and retransmissions, accuracy of TOM can be improved by estimating the error.

4 System Design

The goal with this case study is to develop an one-way delay estimation tool for a wireless multiradio mesh using channel switching, based on the TOM approach. This requires to measure the time spent in a mesh node due to node internal packet processing and queueing, referred to as the measurement. In addition The second requirement is to estimate the time it takes to send a packet over the medium between two mesh nodes to also approximate the transmission queue waiting time, this is referred to as the approximation. In addition due to channel switching operation, an estimation of the unpredictable time which includes packet resends, backoffs and channel switches is required. This requirement is referred to as the estimation process. The fourth requirement is to transfer the delay estimation values between the nodes so as to update them in order to infer the total delay. This is achieved by extending the IP header using the options field with accumulated per hop delay estimates. The last requirement is that the software in clients connecting to the network is to remain unmodified. This means that the one-way delay measurement will only refer to the wireless mesh network backhaul. Therefore the transmission time between a client and the first mesh router will not be included. Also, the time from the last mesh router to the destination will not be included. The IP options field will be added by the first mesh router and removed by the last mesh router. The difference between TOM and TOM++ is the approximation, estimation and channel switches.

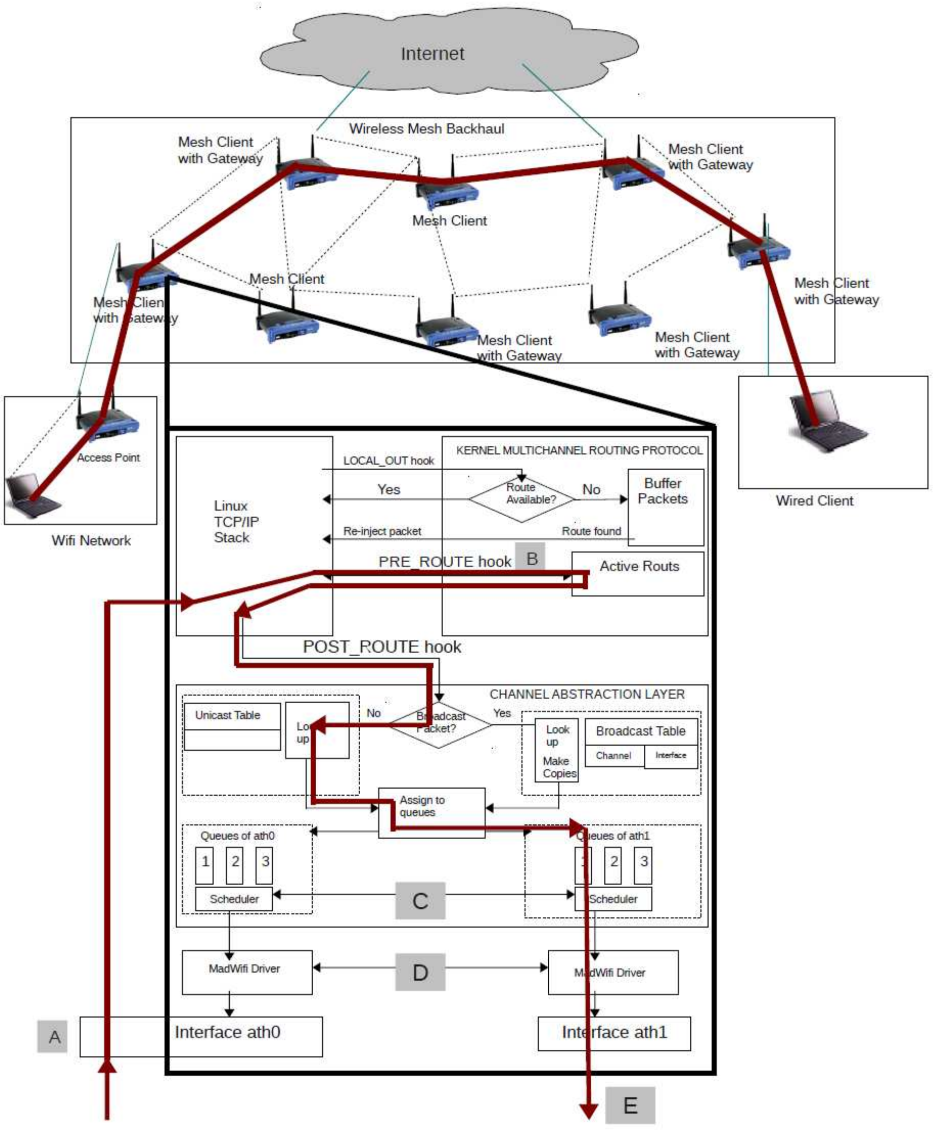

Figure 1: Architecture

4.1 Measurement Process

The one-way delay measurement will be updated at every node within the wireless mesh backhaul. The measurement process is from when a packet arrives at one node until it is transferred to the hardware transmission queue. Once a packet is in the hardware transmission queue, it cannot be updated anymore with delay estimation. Therefore, it is required to estimate the time it takes to flush the hardware queue out, which depends on the size of the queue. A small size may lead to better estimation but low forwarding performance. A larger size will lead to good forwarding performance but higher delay estimation error. As shown in Figure 1 the following points within the mesh node architecture are important for measurement/estimation:

- At point A, a packet arrives at a mesh router, in this example a mesh gateway, and the packet is marked with a timestamp by the hardware.
- At point B the IP header option field is checked for the delay information. If it is present the delay fields are updated with the timestamp from point A. Otherwise the delay information is added to the option field and initialized. The measurement is started.
- At point C the packet is put into the correct channel queue where it resides until the scheduler switches channels to the channel associated to the queue, then transfers the packet to the MadWiFi driver. Once a channel switch happens, delay estimates are reset. This is a key feature that distinguishes TOM from the adopted approach in this paper.
- At point D the sending time t_n, where n is the position of the packet in the queue, is

approximated by dividing the packet size s of this packet and the possible packets in the transmission queue with the transmission rate r in $t_n = \frac{\sum_{i=0,..,n} s_i}{r}$. Then the measurement is ended by updating the delay fields TOM_i resp. $TOM{+}{+}_i$ where i is the current one-way delay and i-1 is the previous one-way delay. This is done with the current time currTime, the approximation a and the respective estimation e. The packet is then transfered to the hardware transmission queue.

$$TOM_i = TOM_{i-1} - currTime + eTOM$$
$$TOM{+}{+}_i = TOM{+}{+}_{i-1} - currTime + a + eTOM{+}{+}$$

- At point E the estimation is updated as described below.

4.2 Estimation Process

The estimation process includes the backoff, possible packet resends and acknowledgment, which can be seen at point E in the Figure 2. The backoff is a random time that precedes any transmission. This time is determined by the hardware and the range of values that the time can obtain increases in case of collisions in the medium. It is used to resolve contention. A resend can be caused by for example packet collisions and interference in the medium, thus it is impossible to predict. The acknowledgment is used to determine when to end the estimation and thus its sending time is included in the estimation. Hence the estimation of a packet will be used on the next packet to be sent. In TOM++ the estimation is smoothed using an Exponential Weighted Moving Average (EWMA) filter that uses weighting factors which decrease exponentially, giving more importance to recent measurements while not discarding the old measurements completely.

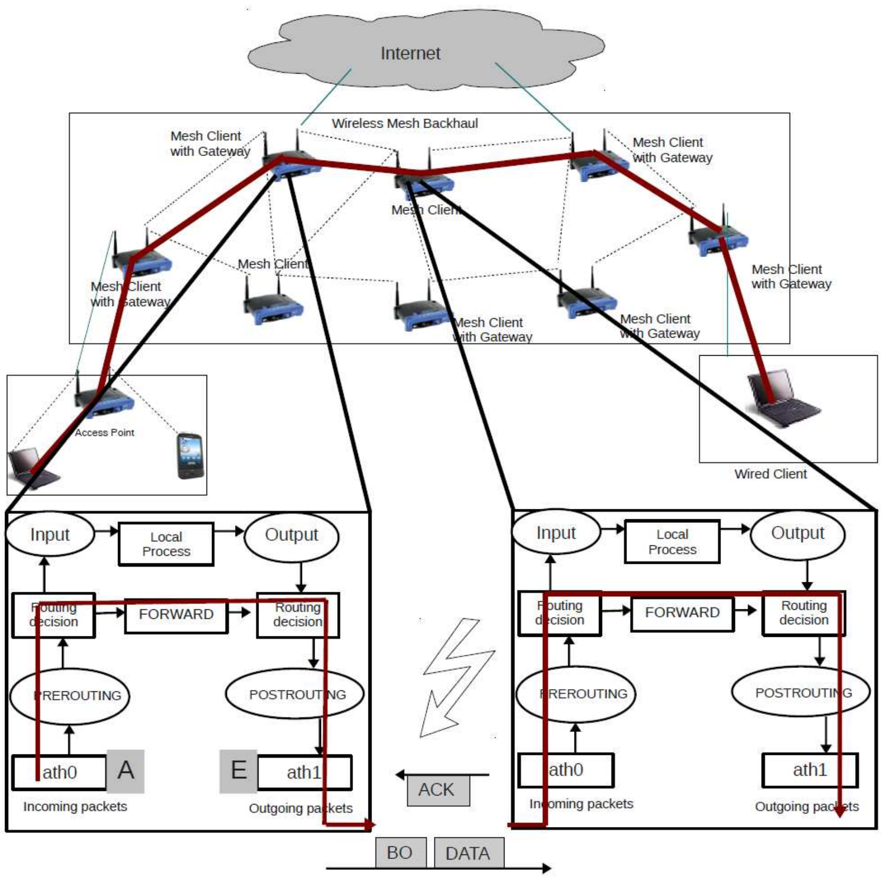

Figure 2: Estimation Process

5 Implementation

As described roughly before, the one-way delay measurement includes several steps, measuring the time a packet spends in the node until it is put into the hardware queue, approximating the sending time for the packet and the hardware transmission queue, and estimating the approximation error, which includes the backoff and retransmission time, and the time for the acknowledgment to return. When a packet arrives at a node it goes through the Linux Netfilter, and if the packet arrives from outside the wireless mesh network the three delay fields are added to the IP header, initialized and the measurement is started, otherwise the fields are updated to start the measurement. Then in the MadWifi driver the measurement is completed, the approximation is calculated, delay fields and estimations are updated. The TOM algorithm, however, has no approximation and therefore only uses an estimation. Estimations for TOM++ are different in that TOM++ uses a simple averaging function. All variables referring to time are in nanoseconds.

In order to estimate the transmission time of the queue in the MAdWifi driver, the following changes have been done. The changes to the transmit queue handling revolve around the variable axq_totaltime which has been added to the struct ath_txq. This variable contains the approximated total time it will take to send the queue. Each time a packet is added to or removed from the transmit queue the approximated time it will take to send that packet, explained below, is added to or subtracted from axq_totaltime respectively. At the end of the function ath_tx start() the packet is added to the hardware queue. From there on no changes to the packet can be made. Thus, near the end of that function, the IP header delay fields are updated accordingly, taking into account the approximated time needed to send the current packets plus all other packets in front of the current packet. This is different to TOM, which does not include this estimation.

A function from the MadWifi driver is used to approximate the sending time of a packet. The parameters to the function includes channel configuration (sc), node state (an) and the packet in a socket buffer (skb) for which this approximation should be calculated. The function takes the packet length and adds the sizes of the 802.11 encapsulation and cyclic redundancy check, encryption overhead, fast-frame header and padding, and tunneling overheads. It then uses the hardware abstraction layer (HAL) to calculate the time it will take to send that amount of data with the current state and configuration. The value returned by ath_approx_txtime() is in microseconds so it is converted to nanoseconds. This required changes to the kernel to add support for floating point operations.

Packets arriving to an empty queue got low delay values. To compensate for this the approximation for a packet arriving to an empty queue is set to β* calculatedApproximation and for other packets it is calculatedApproximation + axq_totaltime. In this project β is set to 1.5. The approximation and estimation are stored in the driver packet buffer (struct ath_buf) and subtracted from the TOM++ IP header delay field. Then the measurement is calculated by subtracting the current time from the delay fields, since the time when the packet arrived at the node where added to the delay fields as described above. A timestamp, bf->tomppTimestamp, is also stored in the driver packet buffer which is used for updating the estimation for both TOM++. This timestamp starts the measurement for the estimation. The bf->timestamp is used by TOM.

The estimation is updated in the function ath_tx_processq() which is called by an interrupt when an ACK is received. TOM++ estimations are measured from when a packet is head of the transmission queue until an ACK for that packet is received. To measure the estimation from when a packet becomes head of the transmission queue the timestamp for the next packet is set to the current time. TOM++ estimations are initiated to zero when the driver is loaded into the Linux kernel. TOM++ uses an EWMA filter to update the estimation on a per packet basis, which is located in the function tomppUpdateDelayEstimation(). The estimation algorithms are reset when a channel switch is made in the function ath_chan_set(). TOM++ are reset to zero, because the state of the new channel is unknown, since no measurements are made for the estimation on the idle channels.

6 Evaluation

The proposed approach has been implemented and evaluated in the KAUMesh livinglab testbed in a multi-channel multi-radio environment, where traffic is forwarded over multiple hops. All traffic in the tests are generated by using MGEN, which is an open source traffic generator. It generates UDP datagrams of configurable size and rate. When MGEN generates a packet it marks it with the local system time. Furthermore, when MGEN receives a packet, it writes the sending timestamp and the receiving time into a log file. To have an accurate measurement of a one-way delay to compare with, the clocks at the sender and receiver needs to be synchronized. For this we can use the wired monitoring connection of the mesh nodes. To synchronize the clocks, the start node (where the traffic is generated) continuously does NTP updates against the end node via the wired interface. The mesh routers are configured to send at a rate of 6Mbps and the packet size for all packets are set to 1000 bytes during all tests.

For a better understanding of the graph descriptions the terms offset, spike and shift are explained here. When the term offset is used, it means the difference between the delay measured by MGEN and the delay measured by the algorithm. The term spike is used for delays noticeably higher than the average delay. If the delay curve of the modules shows the same course as the delay curve of MGEN but both curves are out of sync, the packet difference is referred to as shift. TOM++ has been evaluated in the KAUMesh testbed to determine the accuracy of the improved delay measurement, compared to the TOM implementation. This is done on a stream of packets using a normal percent error formula. Note though that the actual delay time is smaller than that reported by MGEN and thus these values are not completely accurate but can be seen as a guideline. This is due to that MGEN executes in user space and thus an amount of time is taken to transfer the packet to the kernel.

We now describe the test setup for the two hop case. In this setup, node 25 is sending traffc in one stream through node 19 to node 17. All nodes use two interfaces ath0 and ath1. They are set up to listen on a fixed channel on ath0 and to send on two switchable channels on ath1. The specific channels for each node are set up as follows. Node 25 uses channel 36 as fixed channel and can switch among channels 44 and 64. Node 19 uses channel 44 as its fixed channel and switches among channels 36 and 64. Finally, node 25 uses channel 64 as fixed channel and switches among channels 36 and 44. This set-up is to see how the delay estimations react to the channel switches that hybrid multi-radio platforms experience. Channel switches are forced by sending broadcast messages on node 19. The minimum time to stay on a channel after a channel switch is set to 25ms and the maximum time is set to 60ms. As we are sending only over two hops one traffic stream, channel switching is limited. However, the periodic broadcast of control information (like channel neighborhood information) requires the nodes to periodically switch channels as broadcast messages will be broadcasted on all available channels.

Two tests were performed with this setup. In the first test, the stream was sending at a rate of 50 packets per second, and then 100 packets per second in the second test. The three spikes in the figure below occur when there is a channel switch. The last of these spikes represents a channel switch on node 19. Only packet 3270 arrives at node 19 while it is sending on the alternate channel during this channel switch, so this packet is put into the channel queue until the node switches back to sending on the other channel. The time spent in the channel queue counts as a part of the measurement described above, thus the time reported by both TOM and TOM++ will be near accurate for that packet. Packet 3271 arrives when node 19 has started to transmit on the test channel again.
The delay measurements for this packet is not nearly as accurate. The reported MGEN time is about 6.5ms greater than the TOM++ and almost 9ms greater than the TOM delay. This might be because of delays in the hardware after the channel switch since this would not be covered by the measured or the approximated part. The figure also suggests that packet 3271 waits in the hardware transmit queue for a brief time since the TOM++ delay is 3.1ms higher than the TOM delay and the increase in delay to packet 3272 is greater in TOM than in TOM++. The reason the TOM delay increase is higher might be because the TOM estimation includes the hardware transmit queue time while the TOM++ estimation only includes the time from when the packet becomes head of the queue until the acknowledgment that it has been sent is returned. At packet 3274 the channel has started stabilizing after the channel switch.

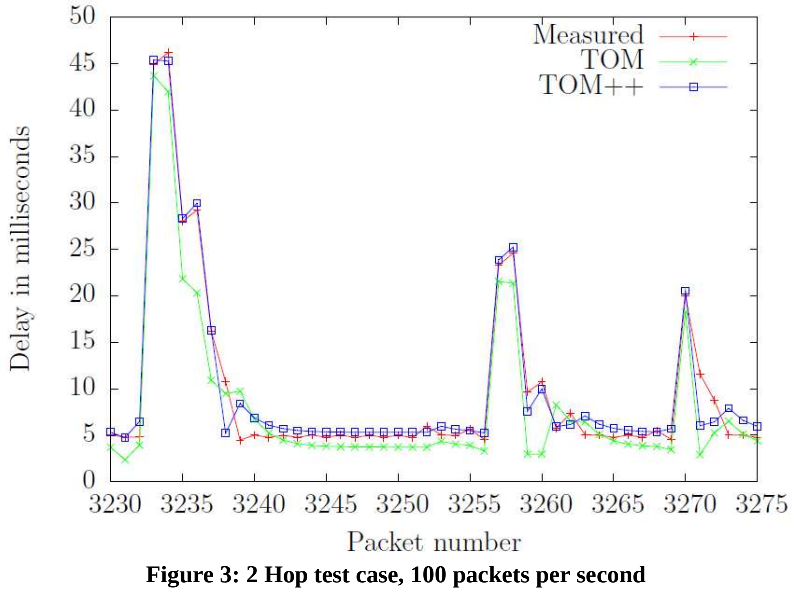

Figure 3: 2 Hop test case, 100 packets per second

The second spike is when a channel switch at node 25 occurs. Here, two packets, 3257 and 3258, arrive while node 25 is sending on the alternate channel. Packet 3259 arrives when node 25 has switched to the test channel again and therefore it is put directly into the transmit queue with one or two packets ahead of it in the queue as shown by the slight increase of delay relative to the normal delay of about 5ms. At the first spike both node 25 and node 19 switches channel before packet 3233 resulting in a total delay of almost 45ms at packet 3233. Both TOM and TOM++ keep up with this rapid change since the packets mainly reside in the channel queue and can thus be measured exactly. Node 25 is sending on the alternate channel for two packets thus at packet 3235 the delay drops. However, node 19 is still transmitting on the alternate channel until packet 3237 has arrived. As seen in the area between the spikes, the TOM++ delay is greater than the reported MGEN time. This is because there is no queue buildup and thus the approximation for the packets are 50% higher which is explained in detail in section above. The estimation for TOM++ is reset at each channel switch thus the packets in the spikes do not get an estimation. This proves to be good as the TOM++ delay gets close to the reported MGEN time. However, packets 3238, 3259 and 3271 after each spike have a lower value compared to the reported MGEN time which can be because the TOM++ estimation part was reset to zero during the channel switch.

Overall, the TOM++ estimation is 9.3% and 13.2% better than TOM in the 50pkt/s and 100pkt/s cases respectively. Note, that due to higher traffic load, larger queues build up. Due to the estimation of the queue transmission time, TOM++ estimations are more accurate.

7 Conclusion

This report presented a way to improve the TOM algorithm to be more accurate with little added system load in a hybrid multi-radio setting involving channel switching. The improvements from the TOM algorithm to the TOM++ algorithm is the calculated approximation of the transmission queue and packet sending times and that the estimated approximation error is reset at each channel switch. This was done because the TOM algorithm was too inaccurate and estimated the above mentioned sending times. Therefore, by calculating the sending time it is only the backoffs and packet resends left to estimate which leads to less inaccuracy in the one-way delay measurement. This improvement resulted in an increased accuracy. The diffculties in doing this have been mainly the hardware abstraction layer. The HAL prevents changes to the packets after transferring them to the hardware transmission queue. This results in larger unpredictable delays, which might add up over multiple hops and with increasing delay.