FUZPAG: A Fuzzy-Controlled Packet Aggregation Scheme for Wireless Mesh Networks

Peter Dely, Andreas Kessler  
Computer Science Department  
Karlstad University  
651 88 Karlstad, Sweden

Nico Bayer, Hans-Joachim Einsiedler, Dmitry Sivchenko  
Deutsche Telekom Laboratories  
Deutsche Telekom AG  
10587 Berlin, Germany

Abstract—Wireless mesh networks (WMNs) are wireless multi-hop backbone networks in which mesh routers relay traffic on behalf of clients or other routers. Due to large MAC layer overhead, applications such as Voice over IP, which send many small packets, show poor performance in WMNs. Packet aggregation increases the capacity of IEEE 802.11-based WMNs by aggregating small packets into larger ones and thereby reducing overhead. In order to have enough packets to aggregate, packets need to be delayed and buffered. Current aggregation mechanisms use fixed buffer delays or do not take into account the delay characteristics of the saturated IEEE 802.11 MAC layer. In this work, we present FUZPAG, a novel packet aggregation architecture for IEEE 802.11-based wireless mesh networks. It uses fuzzy control to determine the optimum aggregation buffer delay under the current channel utilization. By cooperation among neighboring nodes FUZPAG distributes the buffer delay in a fair way. We implemented and evaluated the system in a wireless mesh testbed. For different network topologies we show that FUZPAG outperforms standard aggregation in terms of end-to-end latency under a wide range of traffic patterns.

Index Terms—IEEE 802.11, Wireless Mesh Network, Performance Evaluation, Aggregation, Fuzzy Logic

I. INTRODUCTION

Wireless mesh networks (WMNs) are considered to be a promising technology for cost-efficient proliferation of wireless Internet access in both sparse rural areas as well as in densely populated urban areas. In a WMN, mesh routers relay traffic on behalf of clients or other routers and by this form a wireless multi-hop network. Most WMNs are based on IEEE 802.11 commodity hardware. However, the performance of such WMNs can be low. One reason is the inefficient handling of small packets by the IEEE 802.11 MAC layer. The transmission time of a 100 byte packet sent at 54 Mbit/s consists to 95% of overhead created by the IEEE 802.11 MAC layer. Furthermore, as reported by [1] about 50% of the packets on the Internet are smaller than 700 bytes. For WMNs it is consequently important to transmit small packets in an efficient way. One possibility to increase the efficiency is packet aggregation.

Consider the example in Figure 1, in which three packets are transmitted towards the common next hop without and with aggregation. A normal IEEE 802.11 MAC layer data transmission consists of backoff, DIFS, the data transmission, SIFS and ACK. Using packet aggregation, the packets are concatenated and prepended with an aggregation header that informs the receiver how to deaggregate the packets. Instead of three DIFS, SIFS, ACKs and backoff periods, only one is thus needed, significantly reducing overhead. To have enough packets to aggregate, they need to be buffered first which adds extra delay. This can lead to quality degradation in latency-sensitive applications such as Voice over IP (VoIP) or video conferencing. Thus, it is important to find the best aggregation delay. If this delay is too low, the reduction in overhead is low and the total system throughput is low as fewer aggregation opportunities exist. If the buffer delay is too large, quality degradations in delay-sensitive applications are inevitable.

Existing solutions tackle the problem of finding the buffer delay in different ways. Recent IEEE 802.11 standards, such as IEEE 802.11e [2] or IEEE 802.11n [3] provide static aggregation mechanisms such as block-ACKs or MAC-frame aggregation. In IEEE 802.11e aggregation is only performed if there is more than one frame available in the sender queue, without artificially delaying frames. [3] shows that frame aggregation in 802.11n can lead to great performance improvements, however leaves unanswered how to set the aggregation parameters.

In [4] the ingress mesh router probes the path to the destination to determine the end-to-end latency. The aggregation delay is set so that end-to-end latency plus the buffer delay does not exceed a pre-configured threshold. Intermediate nodes are not allowed to artificially delay packets further, but can aggregate additional packets whenever available. Since [4] does not take into account the actual network load, it delays packets even if not necessary.

[5] uses the expected transmission count (ETX) to estimate the packet loss probability due to collisions and due to bit errors. Furthermore [5] utilizes the processing intensive RMON mode of the wireless card to estimate the channel load. For a given ETX and channel load, the frame size which maximizes the saturation throughput is derived analytically.

The key problem for providing an efficient aggregation
mechanism is to find the best aggregation delay. As the main contribution of this paper we present a novel mechanism to derive this value based on a fuzzy control system. As improvement to existing solutions, it estimates the channel load from the clear channel assessment (CCA) data of IEEE 802.11 and through cooperation distributes the buffer delay among nodes in a fair way. This allows both low aggregation delay under low load and high efficiency. The rest of this paper is outlined as follows: In Section 2 we describe the system model. In Section 3 we derive a control strategy for the buffer delay. Section 4 presents the implementation and evaluates the performance of the proposal. Finally, Section 5 concludes the paper.

II. SYSTEM DESCRIPTION

The aim of the aggregation mechanism is to increase network efficiency and scalability by tuning the network to a state, where it provides a high throughput while keeping the latency low. We use Malone’s model [6] of the IEEE 802.11 distributed coordination function for non-saturated networks to study throughput and MAC layer service delay. The system throughput is defined as the ratio of time spent on payload transmissions and time spent in an idle channel, by successful transmissions or collisions. The MAC layer service delay is the time span from when the packet arrives as head-of-line at the MAC layer and is successfully sent and is estimated according to [7]. Furthermore, we incorporate a new metric, the channel busy fraction (CBF). The CBF is the ratio of the periods in which the wireless channel is sensed busy and all channel states (busy or idle). The CBF can be measured directly by the network card [8].

A. Performance Analysis

Using the model above we study the system throughput, the channel busy fraction and the MAC layer service delay. Figure 2a depicts normalized system throughput for 4, 8 and 12 sender with IEEE 802.11a and 6 Mbit/s PHY rate. For packet sizes of 200, 800 and 2000 bytes the throughput increases almost linearly up to 3, 4.5 and 5 Mbit/s total offered load. If more traffic is inserted into the network, more and more collisions occur and the throughput grows slower or even decreases. For larger networks (12 nodes) the saturation throughput is smaller than the maximum throughput. Due to the lower overhead and fewer collision opportunities, larger packets (2000 bytes) allow a higher throughput (approx. 0.85) than smaller packets (approx. 0.6 for 200 bytes).

In Figure 2b we consider the MAC layer service delay for different traffic injection rates. For low traffic rates, the collision probability of two packets is low. Consequently, the MAC layer service delay is low as well. When the network goes from a non-saturated into a saturated state, the collision probability and delay increase sharply. For example, for 2000 byte packets, the MAC layer service delay starts to increase at about 5.1 Mbit/s.

Figure 2c shows the channel busy fraction for increasing traffic. Due to higher overhead, the channel busy fraction increases faster with small packets than with large ones. For example, at 3 Mbit/s injection rate, 200 byte packets cause a more than 80% channel busy fraction, while 2000 byte packets cause less than 60%.

Finally, in Figure 2d the MAC layer service delay is displayed as a function of the channel busy fraction. Almost independent of packet size or network size, up to 90% channel busy fraction the MAC layer service delay is low and then increases sharply.

Figure 2: Analysis of IEEE 802.11 DCF Non-Saturated

B. Impact of Packet Aggregation

Figure 2d shows that channel busy fractions greater than 90% will cause high MAC layer service delay. Furthermore, Figure 2c shows that for the same traffic injection rate different packet sizes cause different MAC layer service delays. Due to the different channel utilization, if one sends larger packets instead of smaller ones (but with the same aggregate injection rate), one can reduce the channel busy fraction and the MAC layer service delay. Packet aggregation exploits this relation by sending larger packets and reducing thereby the MAC layer service delay. By artificially delaying and then aggregating packets, it is possible to reduce the CBF, the MAC layer service delay and thus reducing the end-to-end latency. Since the packet arrival pattern and packet size distribution is not known in general, one cannot know in beforehand how much buffer delay is necessary to obtain a desired aggregation rate. However, a control system can measure the CBF and infer the network load. If the load is too high, the buffer delay is increased, which results in higher aggregation rates and lowers the network load. If the load is low, no aggregation is needed and the buffer delay should be low accordingly.

III. FUZZY CONTROLLED PACKET AGGREGATION

The structure of the problem (making decisions about the buffer delay using vague and fuzzy information) motivates
the use of fuzzy logic control. We transform the knowledge from the previous section into a fuzzy control system, which controls $delay_{MAX}$, the time a packet is artificially delayed at maximum. As soon as enough packets are available (MTU size) or $delay_{MAX}$ expires, all packets are aggregated and sent as one MAC layer transmission.

A. Input Variables

a) busyfraction: We use the channel busy fraction as indicator for the system throughput and MAC layer service delay of the network. In Figures 2c and 2d we observe three operating regions for the busyfraction: low, medium and high. With a high busyfraction the MAC layer service delay is high. With a medium busyfraction the throughput is high and the delay is low. With a low busyfraction throughput and delay are low. Figure 3a depicts the membership functions.

b) $\Delta$busyfraction: denotes the change in busyfraction between current and previous execution of the fuzzy controller. The universe of discourse ranges from -100% to 100% and is split into a negative, a neutral and a positive region (Figure 3b). Values outside this range are mapped to the border of the universe by the input pre-processing.

c) ratiodifference: represents fairness. Nodes that insert more traffic than average into the network should be punished by using a higher $delay_{MAX}$. A node estimates the busyfraction it causes by the packets (ownbusyfraction) it sends. It then calculates $busyratio_{OWN} = \frac{ownbusyfraction}{delay_{MAX}}$. Furthermore, it computes the average busyratio of its two-hop neighbors $busyratio_{NB}$. Each node computes the difference in his own busyratio and the average neighbors’ ratio as $ratiodifference = (1 - \frac{busyratio_{OWN}}{busyratio_{NB}})$.

B. Output

$\Delta delay_{MAX}$ is the requested change of $delay_{MAX}$. By altering the aggregation delay, the number of aggregated packets, the overhead and channel busy fraction change. The output is divided into fuzzy sets for large and small positive and negative changes, as well as one fuzzy set for changes around zero. The crisp output value of $\Delta delay_{MAX}$ is calculated using center of gravity de-fuzzification [9].

C. Fuzzy Rules

As outlined in Section II-B increasing $delay_{MAX}$ reduces busyfraction, decreasing $delay_{MAX}$ increases busyfraction. At the system setpoint (busyfraction is medium, centered at 0.7) the throughput is high and the MAC layer service delay is low.

The dynamic system behavior can be characterized by nine zones, as displayed in Figure 4. To achieve convergence, a node considers the following rules: If busyfraction is low and not changing towards medium (zones 7/8), a node decreases $delay_{MAX}$. If busyfraction is high and not changing towards medium (zones 3/4), a node increases $\Delta delay_{MAX}$. If busyfraction is medium but increasing or decreasing (zones 2/6), a node decreases or increases $\Delta delay_{MAX}$ accordingly. Only if busyfraction is changing towards medium (zones 1/5) or busyfraction is already medium and not changing (zone 9), $delay_{MAX}$ remains unchanged. Those rules only apply if all nodes aggregate similarly much (ratiodifference is neutral). If ratiodifference is negative, i.e. the node aggregates little compared to its neighbors, $\Delta delay_{MAX}$ increases a little despite of the busyfraction or $\Delta delay_{MAX}$. On the opposite, if a node aggregates more than the average, it reduces $\Delta delay_{MAX}$ a bit.

Table I lists the complete rule-set. It consists of 27 rules, one for each combination of the three input variables and the associated three fuzzy sets. The rules have the form “IF $ratiodifference$ IS input1 AND busyfraction IS input2 AND $\Delta delay_{MAX}$ IS input3 THEN $\Delta delay_{MAX}$ IS output”. For example, the output of “$ratiodifference$ IS Neutral AND busyfraction IS Medium AND $\Delta delay_{MAX}$ IS Neutral” is “Neutral”.

IV. IMPLEMENTATION AND EVALUATION

A. Implementation

We have implemented FUZPAG system, which is shown in Figure 5, on Linux 2.6.22. It consists of user-space components (fuzzy controller, olsrd extension[10]) and kernel-space components (aggregation, deaggregation, medium sensing).
Table I: Fuzzy Rules

<table>
<thead>
<tr>
<th>Input:</th>
<th>Input:</th>
<th>Input: Δbusy(\text{fraction} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio(\text{difference} )</td>
<td>busy(\text{fraction} )</td>
<td>Negative</td>
</tr>
<tr>
<td>Negative</td>
<td>Low</td>
<td>Little Positive</td>
</tr>
<tr>
<td>Negative</td>
<td>Medium</td>
<td>Little Positive</td>
</tr>
<tr>
<td>Negative</td>
<td>High</td>
<td>Little Positive</td>
</tr>
<tr>
<td>Neutral</td>
<td>Low</td>
<td>Negative</td>
</tr>
<tr>
<td>Neutral</td>
<td>Medium</td>
<td>Negative</td>
</tr>
<tr>
<td>Neutral</td>
<td>High</td>
<td>Neutral</td>
</tr>
<tr>
<td>Positive</td>
<td>Low</td>
<td>Little Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>Medium</td>
<td>Little Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>High</td>
<td>Little Negative</td>
</tr>
</tbody>
</table>

Figure 5: FUZPAG Architecture

The aggregation module is integrated into the Linux traffic control subsystem. It concatenates IP-packets with the same next-hop and prepends an extra IP header indicating it is an aggregated packet. On the receiving node, aggregated incoming packets are identified by the extra IP header, de-aggregated in a netfilter-module and inserted into the normal Linux IP-stack. \(\text{delay}_{\text{MAX}}\), the maximum aggregation delay, is set from the fuzzy controller via a netlink interface. The fuzzy controller is implemented using the Free Fuzzy Logic Library[11]. \(\text{busy}\text{fraction} \) is obtained as the fraction of the Atheros chipset registers 0x80f4 (PROFCNT_RXCLR) and 0x80f8 (PROFCNT_CYCLE). To calculate \(\text{ratio}\text{difference} \), information about the current \(\text{delay}_{\text{MAX}}\) and \(\text{ownbusy}\text{fraction} \) is exchanged with a new olsrd [10] FUZPAG-message type among 2-hop neighbors.

B. Evaluation Environment

We evaluated FUZPAG on the KAUMesh testbed, a testbed consisting of 20 Cambria GW2358-4 network computers installed in ceiling of the engineering building of Karlstad University (partly depicted in Figure 6). The nodes are equipped with Atheros 5212-based wireless cards. For the single-hop tests the PHY data rate was fixed to 6 Mbit/s. For the multi-hop tests the PHY rate was fixed to 18 Mbit. Time synchronization for the latency measurements nodes was done with wired Ethernet and NTP. The traffic was generated with mgen [12].

C. Controller Stability and Settling Time

The controller is executed every \(T_{\text{exec}}\) seconds. After the controller changes \(\text{delay}_{\text{MAX}}\) it waits for \(T_{\text{exec}} + \text{rand}[0, T_{\text{exec}}]\), to avoid a ping-pong effect and allow the controller to stabilize. If all nodes change \(\text{delay}_{\text{MAX}}\) at the same point of time, they might add too much delay at first, then too little and so on. Also, if \(T_{\text{exec}}\) is too small, the changes might not be reflected in the controller input variables such as the busyfraction yet. We sample the busyfraction every 250 ms. To avoid distortions by short traffic bursts, we filter \(\text{busy}\text{fraction} \) using an exponential moving average (\(\alpha = 0.1\)). For sudden changes in the busyfraction it takes several seconds until filtered value converges towards the real busyfraction. Thus \(T_{\text{exec}}\) needs to be large enough to allow busyfraction to settle. Since the controller needs to adapt to changes in the network traffic, a too large \(T_{\text{exec}}\) would also render the controller useless.

In order to obtain a reasonable value for \(T_{\text{exec}}\), we conducted a series of experiments. Nodes 7, 10, 13, 22 and 23 each send 480 UDP datagrams/s à 200 bytes to n21. For different values of \(T_{\text{exec}}\), we checked whether the controller converges. The minimum value of \(T_{\text{exec}}\) to have a reproducible and stable behavior was 5 seconds.

For \(T_{\text{exec}} = 5\text{s}\), Figure 7a displays the settling behavior. The upper part shows busyfraction (measured by all nodes), the lower part the aggregation delay during the first 35 seconds of the experiment. First the nodes detect a high busyfraction and increase \(\text{delay}_{\text{MAX}}\). With some time lag, the busyfraction decays. Finally, nodes tune \(\text{delay}_{\text{MAX}}\) to around 2000 µs. Due to the fairness property of FUZPAG and all nodes inserting the same amount of traffic, all nodes have the same \(\text{delay}_{\text{MAX}}\) (with a small error).

D. Single-Hop Scenario

Next, we studied the impact of the aggregation on end-to-end latency. Again, nodes 7, 10, 13, 22 and 23 sent 200 byte datagrams to node 21. We used a combined traffic injection rate of 2, 3.2, 4.4 and 5.6 Mbit/s. We ran each experiment for 60 seconds and repeated it 5 times. The fuzzy controller got in addition 40 seconds to settle before the measurement of the delay started. Figure 7b displays the average values and the standard deviation for no aggregation, aggregation with a fixed delay of 500 to 3000 µs and the fuzzy controlled aggregation.

With 2 Mbit/s input rate the network is lightly loaded and aggregation is not required. Here the latency is lower with no aggregation than with static aggregation values. The fuzzy...
controlled aggregation tunes the aggregation delay to 0 in this situation and therefore is as good as no aggregation. As other aggregation schemes use fixed delays, the average packet delay is 2-3 ms higher. With 3.2 Mbit/s injection rate and no aggregation the network is close to saturation. The delay is higher than before and varies a lot. All aggregation modes are about equally good. With 4.4 Mbit/s injection rate, 500-1500 µs aggregation delay is not enough. Too few packets are aggregated, the efficiency is low and thus the network is too highly utilized. The latency is between 220 and 300 ms\(^1\). With 5.6 Mbit/s injection rate, also 2000 µs aggregation delay is not sufficient. For all injection rates, the fuzzy controlled aggregation has lowest or close to lowest end-to-end latency.

E. Multi-Hop Scenario

The next scenario resembles an WMN connected to the Internet. Node 15 is the gateway and nodes 21, 7 and 11 communicate with the gateway. Nodes 7 and 21 send and receive their traffic to/from node 7 via 22 and 13, node 11 via node 13. The routes were set-up using the OLSR protocol and fixed during the experiments. In this experiment we evaluate the impact of aggregation on VoIP traffic. One VoIP call is emulated by a constant stream of 50 UDP datagrams per second (200 bytes length) from and to the source.

Figure 7c depicts the average end-to-end latency for different aggregation strategies. For low network loads (12 and 15 calls) it is better not to use aggregation. Static aggregation adds delay unnecessarily. When the network load gets higher, aggregation should be used. For 18 and 21 concurrent calls using fuzzy controlled aggregation reduces the end-to-end delay by 45 ms and 65 ms compared to not using aggregation. Fuzzy controlled aggregation outperforms static aggregation delay settings by about 5-10 ms. Since in this scenario the load is not uniformly distributed over the network, the nodes should use different aggregation delays. For example, using the fuzzy controlled aggregation and 18 concurrent calls, the highly loaded node 13 uses an average aggregation delay of 955 µs, while the lighter loaded nodes 21 and 7 have 346 µs and 343 µs respectively.

\(^{1}\)We cut Y-axis of the the diagram at 40 ms for scaling reasons. The cut-off values are all > 200 ms.

V. CONCLUSIONS

In this paper we have demonstrated that, by using fuzzy control, it is possible to tune the aggregation buffer delay \(d_{\text{MAX}}\) so that extra delay is only added when necessary. FUZPAG outperforms a static configuration of \(d_{\text{MAX}}\) in a wide range of scenarios. Depending on the network topology and traffic, the reduction in end-to-end latency by using FUZPAG is up to 65 ms in multi-hop and several hundred milliseconds in single-hop scenarios. As future work we plan to dynamically tune the membership functions using a genetic model reference adaptive controller.

REFERENCES