UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EXPLORING COMPUTATIONAL INTELLIGENCE TO IMPROVE NETWORK PERFORMANCE

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Yalda Edalat

September 2019

The Dissertation of Yalda Edalat
is approved:

Katia Obraczka, Chair

J. J. Garcia-Luna-Aceves

Jong-Suk Ahn

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Exploring Computational Intelligence to Improve Network Performance

by

Yalda Edalat

Several network protocols, services, and applications adjust their operation dynamically based on current network conditions. Consequently, keeping accurate estimates of network conditions and performance as they fluctuate over time is critical. In this thesis, we explore the use of computational intelligence, in particular machine learning techniques to estimate "near-future" network performance based on past network conditions. We call our approach to network performance estimation SENSE for Smart Experts for Network State Estimation. SENSE is able to respond to network dynamics at different time scales, i.e., long- and medium-term fluctuations as well as short-lived variations.

Then, by applying SENSE, we proposed a novel algorithm to dynamically enable and disable IEEE 802.11 DCF's RTS/CTS handshake. Our algorithm uses current packet size and transmission rate, as well as an estimate of network contention to dynamically decide whether to use RTS/CTS. To the best of our knowledge, the proposed algorithm is the first to enable and disable the RTS/CTS handshake based on a set of current network conditions, and automatically adapt as these conditions change. Simulation results using a variety of WLAN- as well as wireless multi-hop ad-hoc network scenarios, including synthetic and real traffic traces, demonstrate that the proposed approach consistently outperforms current best practices, such as never enabling RTS/CTS or using a pre-specified threshold to decide whether to switch RTS/CTS on or off.

We also propose a modified version of a simple, yet effective machine learning technique called "Fixed-Share" algorithm to optimize IEEE 802.11's backoff algorithm. To the best of our knowledge, this is the first approach that uses machine learning to dynamically set the IEEE 802.11's contention window based on past performance. Through simulations using a variety of network scenarios, we show that our method outperforms IEEE 802.11's original exponential back off algorithm as well as an approach that adapts based on a few recent data transmission events.

This thesis work is dedicated to my husband, Bahador, who has been a constant source of support and encouragement during the challenges of graduate school and life.

Acknowledgments

First and foremost, I have to thank my parents for their love and support throughout my life.

I would like to sincerely thank my thesis adviser, Katia Obraczka, for her guidance and support throughout this study and specially for her confidence in me.

Chapter 1

Introduction

Computer networks have become one of our society's essential commodities and, like power- and water distribution systems, are now considered part of our critical infrastructure. Consequently, it is crucial to keep them operating continuously and delivering adequate performance. This is especially true as networks become increasingly more complex and the services they provide increasingly more sophisticated and demanding.

Like any complex dynamical system, computer networks' performance fluctuates over time influenced by a variety of factors such as traffic load, end system load, communication link conditions (e.g., propagation channel impairments especially in the case of wireless links), etc. In order to adapt to network dynamics, most computer network protocols and algorithms employ a number of operational parameters that constantly estimate current conditions in the network.

Motivated by the need to accurately estimate near-term future network state that may slowly or rapidly change, in our research we have focused on machine-learning techniques, specifically Multiplicative Weight algorithmic family [21] to estimate "near-future" network state based on past network conditions.

Our predictor can be applied to any protocol or application requiring near future network state in order to adapt its performance and parameters. IEEE 802.11 standard is one of the protocols which can benefit from tunning its parameters dynamically based on near future network conditions. Encouraged by that, we focus on 802.11 features and parameters and use our estimator to optimize it.

1.1 Contributions

The thesis main contributions can be summarized as follows:

1. Introduce a computational intelligence method to estimate "near-future" network state based on past network conditions [12] [13].
2. Propose a new approach that uses machine learning techniques to dynamically switch RTS/CTS on and off ahead of data transmission by considering a combination of a set of current network conditions [15].
3. Propose a machine learning technique to dynamically tune the contention window of IEEE 802.11 [14].

1.2 Publications

1. A poster on our algorithm was accepted in ACM SIGCOMM, Chicago 2014.
2. “Network state estimation using smart experts” is published in 11th International Conference in Mobile and Ubiquitous Systems (MOBIQUITOUS), London 2014.
3. “Smart Experts for Network State Estimation.” is published in IEEE Trans. Network and Service Management 13, no. 3 (2016): 622-635.

4. “A machine learning approach for dynamic control of RTS/CTS in WLANs” is published in 15th International Conference in Mobile and Ubiquitous Systems (MOBIQUITOUS), New York 2018.
5. “Dynamically Tuning IEEE 802.11’s Contention Window Using Machine Learning” is published in 22nd ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2019.
6. In preparation: “Smart Adaptive Collision Avoidance for IEEE 802.11”.

1.3 Patents

Patent pending: “Dynamically Tuning IEEE 802.11’s Contention Window Using Machine Learning.”

1.4 Road Map

The remainder of this document is organized as follows. In Chapter 2 we present a complete description of our network state estimator, SENSE. Chapter 3 introduces a machine learning technique to dynamically switch IEEE 802.11’s RTS/CTS on and off based on a set of current network conditions. In Chapter 4 we introduce a machine learning method to dynamically set the contention window size of IEEE 802.11. Finally, Chapter 5 concludes the thesis.

Chapter 2

Smart Experts for Network State Estimation

There are several network protocols and applications adapt their parameters based on the network conditions. Notable examples include the Transmission Control Protocol (TCP) and IEEE 802.11 (Wi-Fi) which adjust the retransmission timeout and contention window size, respectively, according to network congestion and wireless channel state. More specifically, to recover lost packets in a timely manner yet minimizing the number of unnecessary retransmissions, TCP periodically evaluates the degree of network congestion under the assumption that network conditions will stay almost the same until the next evaluation period. It uses the round-trip time (RTT), i.e., the time between sending a segment and receiving confirmation from the other end that the segment was received, as a way to gage network load. TCP adjusts its retransmission timeout, i.e., the interval of time the TCP sender will wait for a segment's acknowledgment from the TCP receiver before retransmitting the segment, based on TCP's current estimate of the RTT. To compute its estimate of the RTT, TCP runs a simple mechanism known as Exponentially Weighted Moving Average (EWMA) with one tunable parameter, which determines the relative weight between the current RTT measurement and the previous RTT estimate.

The IEEE 802.11 responds to congestion buildup in the network by exponentially inflating its back-off window, which stipulates the average amount of time that a node should wait to transmit after a collision has occurred. The rationale for this exponential back off is that collisions are used as congestion indicators; and, after a failed attempt to transmit due to a collision, the transmitter needs to wait longer before trying again. To estimate the "near-future" channel state, IEEE 802.11 counts the number of consecutive collisions that took place during the current estimation time window and exponentially expands the size of the back-off window according to this collision count.

Clearly, the performance of these widely used network protocols heavily relies on how correctly their prediction mechanisms forecast "near-future" network state. Their implicit assumption is that network conditions change smoothly, i.e., that "near-future" state is closely correlated to previous history. As a result, their performance can be negatively affected when their operational parameters are set without accurately accounting for network dynamics. TCP, for instance, statically presets the weight factor in its RTT EWMA equation irrespective of the target network environment and conditions. The fixed weight factor in TCP's RTT EWMA calculation is a relative ratio deciding how much the current RTT measurement and the current RTT estimate should influence the new RTT estimate. The more dynamic the network conditions, the more weight should be placed on the current RTT measurement. Therefore, to achieve better performance, the fixed weight factor should change dynamically depending on network conditions.

IEEE 802.11 rigidly cold-starts and counts collisions at every new frame's transmission without considering previous channel state. This means that considerable resources may be wasted in the process of reaching an adequate congestion window since 802.11's network estimation technique does not keep track of the network state after successful transmissions.

In this chapter, we introduce Smart Experts for Network State Estimation (SENSE) [12] [13] which is a simple, yet efficient machine learning predictor based on the Fixed-Share approach [21] [40] [35] [9]. Unlike conventional network state estimators, SENSE provides a general framework that can incorporate any traditional estimator as an "expert". SENSE can then dynamically select the best experts among the set of all experts being used depending on their performance. It swiftly chooses experts that more faithfully capture network dynamics by penalizing poorly performing experts.

The original Fixed-Share algorithm [21] has four main drawbacks. First, for every dataset, a fix value within the range we are trying to predict is assigned to each expert. Thus, the range of the estimation is required for proper assignment of these values. Second, its accuracy is sensitive to the number of experts and typically, the more experts, the more accurate the prediction since the algorithm basically singles out a few well-behaved experts among the set of competing experts. There is clearly, a "diminishing return" effect after the number of experts gets too high. Third, the "loss function" penalizing experts, relies exclusively on the magnitude of the current error, instead of whether errors have recently increased or decreased. Additionally, all poorly performing experts are equally penalized. Depending on the recent error variation history, the loss function should intensify or alleviate the penalty for each individual expert to accelerate convergence. Finally, to promptly adapt to abrupt changes even when recent measurements become distinctly different from previous ones, the original Fixed-Share algorithm constantly tries to boost the weight given to poorly performing experts while offsetting the weight of well performing ones.

This feature leads to precision degradation as too much emphasis is placed on poorly performing experts, especially when sudden changes rarely happen. To address these problems, SENSE introduces three techniques, namely: (1) smart experts, (2) META-learning, and (3) Level-shift. SENSE's smart experts reduce sensitivity to the number of experts and eliminate the need for a-priori knowledge of the data that we are trying to predict. SENSE employs EWMA equations with different weights as its experts and normalizes errors by the maximum observable output. SENSE's META-learning algorithm expedites convergence by tracing recent past history and adjusting each expert's penalty accordingly. Finally, the Level-shift mechanism [18] employed by SENSE improves its response to sudden data changes by bounding SENSE's learning time window, and upon detecting dissimilar data patterns, SENSE reinitializes its tunable parameters and starts to relearn.

We evaluate SENSE using a variety of datasets including synthetic and real data. In all cases, SENSE outperforms predictors based on pure EWMA as well as Fixed-Share. Furthermore, a key advantage of SENSE is that it automatically adjusts to the data it is trying to predict. As a result, SENSE yields superior performance for all datasets used in our experiments when compared to "pure" Fixed-Share and EWMA. Our results also indicate that the performance of EWMA is quite sensitive to its "smoothing" factor, which determines how much weight will be placed on the "past" versus the "present" when predicting the "future". Another key advantage of SENSE's ability to automatically adjust to the data is that, unlike Fixed-Share, it needs no a-priori information about the dataset and is minimally sensitive to the number of experts. In our experiments, SENSE yields higher prediction accuracy when compared to the Fixed-Share algorithm and EWMA.

The rest of this chapter is organized as follows. Section 2.1 presents some background on history-based prediction algorithms, namely EWMA, Fixed-Share, and its predecessor, Static Experts. Section 2.2 provides an overview of related works. Section 2.3 describes SENSE in detail and Section 2.4 compares the performance of SENSE against EWMA and Fixed-Share algorithm. Finally, Section 2.5 evaluates SENSE's new techniques and parameters.

2.1 Background

SENSE is based on a combination of history-based predictors, more specifically EWMA and Fixed-Share. In this section we review EWMA as well as Fixed-Share and its predecessor, Static Experts, both of which are examples of the Multiplicative Weight algorithmic family.

2.1.1 EWMA

Exponentially Weighted Moving Average (EWMA) based predictors, calculate an exponentially weighted mean of the previous data. Equation 3.1 shows the basic equation of exponential smoothing given by Hunter [23] where x_t and y_t represent, respectively, a sequence of data point that has been observed and a sequence of forecasts given by the predictor. Furthermore, α in 3.1 is the "smoothing factor", a value between 0 and 1 specifying how much relative weight is given to previous estimates (i.e., the "past") versus new samples (the "present").

$$x_t = \alpha \times y_{t-1} + (1 - \alpha) \times x_{t-1} \tag{2.1}$$

The problem of using EWMA based predictors is choosing appropriate α, which should be based on the dataset. Even though there has been no generally accepted statistical technique for choosing α, our observations indicate that α needs to be determined based on the small-lag autocorrelation. Recall that the autocorrelation is a correlation coefficient that instead of measuring the correlation between two different variables, it measures the correlation between two values of the same variable at times t_i and t_{i+k}. The autocorrelation can be used to detect the degree of randomness in data i.e., whether data similar to the present data would appear in the future. The autocorrelation parameter named "time lag" measures how soon the same data pattern will repeat.

Our experiments with a variety of datasets indicate that α should be chosen based on the small-lag autocorrelation. If data is random, the small-lag autocorrelation should be near zero. In this case, low values of are desirable: low has EWMA act as a low-pass filter smoothing out sudden fluctuations occurred in the input data series. In other words, low values of α, favor the "past" over the "present" when computing the current estimate. On the other hand, if data is non-random, then small-lag autocorrelations will be significantly non-zero. In this case, high α acts as a high-pass filter hardly filtering out measurement noise. It means that, with high α, the "present" plays a more important role.

The problem of current EWMA based predictors is that they have to statically set α, for example, by trying to guess what the data will look like in the future. SENSE, however, runs a small number of EWMA experts with different α's and, using the Fixed-Share technique, dynamically picks the best performing EWMA depending on network dynamics.

2.1.2 Multiplicative Weights Method

The Multiplicative Weight algorithmic family has shown to yield performance improvements in a variety of on-line problems [19]. Aiming at minimizing the

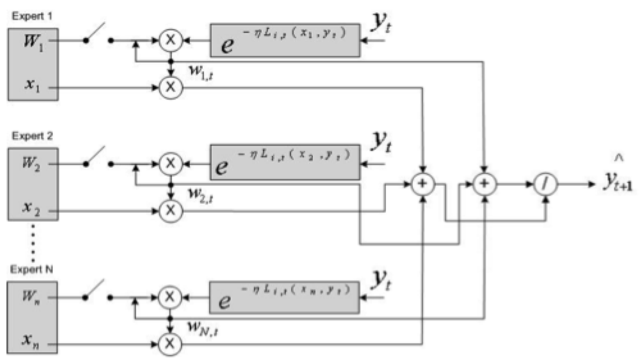


Figure 2.1: Hardware block diagram of Multiplicative Weight algorithm

prediction error, this family of algorithms combines predictions of a set of experts $\{x_1,x_2,...,x_N\}$ to compute the overall prediction denoted by $\hat{y}_t$. To denote the impact of each expert on the overall predictor, it associates each expert with a weight from $\{w_1,w_2,...,w_N\}$. After each trial, the weight of each expert is updated depending on the difference between its prediction and the real data represented by y_t. Weights of "well- performing" experts are not changed, while the weights of experts that are not performing well are reduced. As an illustration, Figure 2.1 shows the implementation of the Multiplicative Weight algorithm with N experts using a hardware block diagram. The shaded boxes on the left- and middle columns correspond, respectively, to the experts denoted as x_i and the penalty function. The process of updating weights and generating the final predictions is represented as a circuit employing the addition, division, and multiplication operators.

Equation 3.2 represents the circuit of Figure 2.1 as a mathematical expression. As shown in 3.2, $\hat{y}_{t+1}$ can be represented by a sum of products of $\alpha_{i,t}$ and $x_{i,t}$ where $\alpha_{i,t}$ is the experts' weights ($0 < \alpha_{i,t} < 1$) which are dynamically and systematically adjusted and $x_{i,t}$ is each expert's prediction. Equation 3.2 confirms that the Fixed-Share algorithm is a selection process, which favors experts whose predictions are closer to the real data by incrementally growing their weights, while reducing other experts' weights.

As highlighted in [21], several schemes have been proposed for updating experts' weights in multiplicative weight algorithms. In the remaining of this section, we discuss two well-known Multiplicative Weight algorithms, namely Static Experts and Fixed-Share.

$$\hat{y}_{t+1} = \frac{\sum_{i=1}^{N} x_{i,t} \times w_{i,t} \times e^{-\eta \times \sum_{i=1}^{N} L_{i,t}(x_i, y_t)}}{\sum_{i=1}^{N} w_{i,t} \times e^{-\eta \times \sum_{i=1}^{N} L_{i,t}(x_i, y_t)}} = \sum_{i=1}^{N} \alpha_{i,t} \times x_{i,t}$$
$$Where \quad \alpha_{i,t} = \frac{w_{i,t} \times e^{-\eta \times \sum_{i=1}^{N} L_{i,t}(x_i, y_t)}}{\sum_{i=1}^{N} w_{i,t} \times e^{-\eta \times \sum_{i=1}^{N} L_{i,t}(x_i, y_t)}} \tag{2.2}$$

Static Experts Algorithm

Static Experts, whose pseudo code is presented in Algorithm 8, is the simplest version of the Multiplicative Weight algorithmic family. Its steps, as described in Algorithm 8, are common to all Multiplicative Weight algorithms with N experts. The Prediction step in Algorithm 8 computes the current prediction by (1) summing, over N experts, the products of the expert multiplied by its current weight and then (2) normalizing the result by the sum of the weights. Using a given "loss function", the Loss function step checks, at each prediction trial, how good of a prediction each expert yields. Then, in the Exponential updates step, the loss computed in the Loss function step is used to adjust the experts' weights, which will be used in the next trial. The Static Experts algorithm has one main drawback: it is not able to adjust to abrupt changes in data fast enough. This is because it takes a relatively long time for the weight of an expert to either shoot up or down when the expert's performance suddenly changes following an abrupt change is the data.

Algorithm 1 **Static Expert Algorithm**

```
Parameters:
    $$\eta > 0, 0 \leq \alpha \leq 1$$
Initialization:
    $$w_{1,1}=...=w_{N,1}=\frac{1}{N}$$
Prediction:
    $$\hat{y}_t=\frac{\sum_1^N w_{i,t}\times x_i}{\sum_1^N w_{i,t}}$$
Loss Function:
    $$L_{i,t}(x_i, y_t) = \begin{cases} (x_i - y_t)^2 & , x_i \geq y_t \\ 2 \times y_t & , x_i < y_t \end{cases}$$
Exponential Update:
    $$\acute{w}_{i,t} = w_{i,t} \times e^{-\eta \times L_{i,t}(x_i,y_t)}$$
```

Fixed-Share Algorithm

To solve the problem of slow convergence of well-performing experts in the Static Experts algorithm, [21] introduced Fixed-Share. The main goal of the Fixed-Share approach is to improve Static Experts' performance, while keeping its simplicity. The basic idea of Fixed-Share is to prevent large differences among experts' weights; to this end, it shares a fixed fraction of the weights of experts that are performing well among the other experts. This additional step, called "Sharing weights" and shown in 2.3, redistributes evenly a certain fixed fraction of pool, which is the sum of a preset portion of each weight.

$$Pool = \sum_{i=1}^{N} \alpha \times \acute{w}_{i,t} \qquad w_{i,t+1} = (1-\alpha) \times \acute{w}_{i,t} + \frac{1}{N} \times Pool \tag{2.3}$$

Although the Fixed-Share algorithm has been shown to perform well when estimating network variables in [40], it exhibits four main weaknesses. First, it must have a priori knowledge of the dataset's range in order to properly set the value of its experts. Likewise, the degree of sharing in the Sharing weights step 2.3 cannot be appropriately determined unless the number of level shifts is known in advance. Level shift [18] is defined as a significant change in the mean of observed data and is discussed in detail in Section 2.3. Second, the accuracy of the algorithm is quite sensitive to the number of experts whose values determine the granularity over the range of values that the variable in question can assume, and ultimately its accuracy. However, as discussed in Section 2.1.2, more experts may also introduce additional errors. Third, since the loss function is predetermined and not changed considering the target environment and application, it is not always able to exhibit adequate convergence. Finally, if experts perform consistently well for long periods of time, sharing their weight with other experts whose performance is not adequate, compromises the algorithm's overall convergence and performance.

2.2 Related Work

Several network protocols and applications make use of heuristics to estimate and adapt to the dynamics of the underlying network. Since the literature on the topic is quite extensive, in this section, we focus on reviewing work that is more closely related to ours. EWMA is a well-known technique adopted by several communication protocols. As previously pointed out, TCP uses EWMA to estimate near-term round-trip time (RTT), which is used to set TCP's retransmission timeout (RTO). Since, depending on the network environment, RTTs may vary considerably in short timescales, a number of mechanisms have been proposed to either replace or augment EWMA. DualPats, a real time TCP throughput prediction service for distributed applications, was introduced in [36]. It utilizes EWMA to make throughput predictions of large transfers augmented with active probing. In [18], EWMA along with other simple linear predictors was employed to show that; in general, history-based methods predict the throughput of TCP transfers more accurately than formula-based techniques, i.e., mathematical models that express TCP performance as a function of network path characteristics. In [17], collision rate is estimated by an EWMA based mechanism to dynamically adjust the contention window parameters in 802.11 MAC protocol.

More recently, a few efforts have used machine learning techniques to estimate near-term network variables. For instance, the work in [40] proposed a TCP RTT predictor based on a simple yet efficient machine learning algorithm called Fixed-Share [21]. The results presented in [40] show that, for a variety of network scenarios and conditions, the proposed Fixed-Share based predictor was able to improve RTT estimation significantly (thus yielding higher throughput) compared to existing approaches. Support Vector Regression (SVR) [48] also introduced a machine learning method, which can accept multiple inputs to generate accurate predictions. This method was used in [37] to predict the end-to-end TCP throughput for arbitrary file sizes.

A variant of the Fixed-Share approach has also been employed in the context of medium-access control (MAC). More specifically, in [43], a collision-free schedule based MAC that uses Fixed-Share to predict offered traffic load was proposed. Simulations as well as testbed results show the benefits of traffic prediction to schedule flows at the MAC layer in terms of delivery delay and delivery ration when compared to contention based MAC protocols. In [32], a method to predict direct and staggered collision probabilities of each node in WLANs has been introduced. Using information from an access point (AP) about network traffic broadcast as well as the AP's local measurements, each node obtains a spatial picture of the network in order to estimate probabilities of collisions locally. Similar techniques to the one used in [32] have been employed in [33] to improve throughput and link adaption in 802.11 networks with hidden terminals. In particular, a link adaption algorithm, in which nodes estimate the channel conditions by comparing the observed loss statistics to the expected loss statistics based on the estimated collision probability, is employed to select the ideal modulation rate under the estimated network conditions.

2.3 SENSE

This section provides a detailed description of our online estimator, SENSE, which employs a combination of Fixed-Share with EWMA. Then, we discuss SENSE's accuracy compared to Multiplicative Weight algorithmic algorithms. More specifically, SENSE, whose pseudo-code is shown in Algorithm 9, is an enhanced version of the Fixed-Share estimator, where, instead of fixed-value experts, EWMA filters are employed as experts. Table 2.1 summarizes SENSE's variables and their descriptions.

2.3.1 SENSE Algorithm

More specifically, in the EWMA experts step of Algorithm 9, the prediction of each expert, $x_{i,t}$, is calculated as a weighted sum of the previously observed data item y_{t-1} and the previous prediction $x_{i,t-1}$ where α represents the relative weight between $x_{i,t-1}$ and y_{t-1}. Initially, each expert is assigned a weight, $w_{i,1} = 1/N$, where N is the total number of experts; each expert is also assigned an α_i value between 0 and 1 which differentiates experts from each other. In SENSE, EWMA experts replace numeric experts used in the Fixed- Share algorithm, which results in making SENSE's accuracy less sensitive to the number of experts used.

Algorithm 2 SENSE

Initialization:

$$\eta_{min} = \eta_{MIN-INIT} \qquad \eta_{max} = \eta_{MAX-INIT} \qquad \beta = \beta_{INIT}$$
$$EL = EL_{USER-DESIRED-ACCURACY} \qquad w_{1,1}=...=w_{N,1}=\frac{1}{N}$$

EWMA Experts:

$$x_{i,t} = \alpha_i \times y_{t-1} + (1 - \alpha_i) \times x_{i,t-1}$$

Prediction:

$$\hat{y}_{i,t}=\frac{\sum_1^N w_{i,t} \times x_{i,t}}{\sum_1^N w_{i,t}}$$

Loss Function:

$NE_{i,t}= \frac{|x_{i,t}-y_t|}{y_{max}}$

$$L(x_i, t)_{i,t} = \begin{cases} NULL & , NE_{i,t} \leq EL \\ NE_{i,t} & , Otherwise \end{cases}$$

META Learning:

$$\eta_{i,t} = \begin{cases} min(\eta_max, (\eta_{i,t-1} \times \beta)) \\ \qquad , NE_{i,t} > NE_{i,t-(j-1)} > NE_{i,t-j} \\ max(\eta_min, (\frac{\eta_{i,t-1}}{\beta})) \\ \qquad , NE_{i,t} < NE_{i,t-(j-1)} < NE_{i,t-j} \\ \eta_{i,t-1} \\ \qquad , Otherwise \end{cases}$$

Weight Update:

$$w_{i,t+1} = w_{i,t} \times e^{-\eta_{i,t} \times L(x_i,t)_{i,t}}$$

Restart Learning:

If Level Shift is detected at nk then,

$$w_{i,t} = w_{i,t} \times e^{\sum_{t=-2T}^{t=-T} \eta_{i,t} \times L(x_i,t)_{i,t}}$$

Table 2.1: SENSE's Variables

Parameter	Description
x_i	Prediction of expert i
y_t	Observed data at time t
$\hat{y}_t$	SENSE's prediction for time t
w_i	Weight of expert i
NE_i	Normalized error of expert i
N	Total number of experts
$L(x_i, t)_{i,t}$	Loss of expert i at time t
y_{max}	Maximum data observed so far
η_i	Determines degree of penalizing expert i
β	Determines how much η_i should be increased or decreased
EL	Error limit (based on user's desired accuracy)
η_{min}, η_{max}	Limit experts' weight
j	Determines the time window to evaluate expert's performance (used to update η_i)

As illustrated in the Prediction step of Algorithm 9, at every trial t, SENSE calculates the current prediction $\hat{y}_t$ by adding the weighted predictions from N experts. After computing $\hat{y}_t$, the loss function step in Algorithm 9 calculates the absolute difference between the actual outcome, y_t, and each expert's forecast $x_{i,t}$; Then it normalizes this error with the maximum outcome y_{max}, which is updated with the largest outcome observed yet. We have experimented with different loss functions and picked the one shown in Algorithm 9 for its efficiency as well as simplicity. Finally, the loss function, $L(x_i, t)_{i,t}$, is set to either the normalized error $NE_{i,t}$ or the $NULL$ function depending on $NE_{i,t}$'s value. If $NE_{i,t}$ lies within the satisfactory boundary EL, SENSE does not penalize experts differently than the original Fixed-Share algorithm, which constantly adjusts the weight until the prediction equals the outcome. Here, EL can be set to any fraction between 0 and 1 according to the accuracy required by the application.

SENSE then runs the META-learning step, which either multiplicatively increases or decreases $\eta_{i,t}$ by β if the normalized error keeps growing or shrinking, respectively, for j consecutive trials. Otherwise, it does not change $\eta_{i,t}$. This META-learning step aims at deciding how to adjust the experts' weights based on their recent-past predictions.Clearly, the less accurate an expert's prediction is, the more severe that expert is penalized. In our experiments, we considered a three-trial observation window (i.e., j = 2) to characterize the recent past but larger observation windows can be used. To prevent each expert's η from becoming too small or too large, $\eta_{i,t}$'s range is specified as $[\eta_{min},\eta_{max}]$. We explore how $\eta_{i,t}$'s range impacts SENSE's behavior in Section 2.5.2. Recall that the goal of META-learning is to speed up convergence of each expert's prediction to the observed outcome. Thus, the Weight update step updates $w_{i,t}$ with what has been learned, i.e., it multiplies $w_{i,t}$ by e to the power of the product of the loss function $L(x_i, t)_{i,t}$ learning factor $\eta_{i,t}$.

Finally, SENSE employs a Level-shift step [18] to detect any significant change in the mean of the observed data. Suppose $\{X_1,X_2,...,X_n\}$ is the sequence of data, where X_1 is the first data after the last detected level shift. The measurement X_k is an increasing (decreasing) level shift if it satisfies the following three conditions:

1. Data $\{X_1,X_2,...,X_{k-1}\}$ are all lower (higher) than the data $\{X_k,...,X_n\}$,
2. The median of $\{X_1,X_2,...,X_{k-1}\}$ is lower (higher) than the median of $\{X_k,...,X_n\}$ by more than a relative difference χ , and
3. $k + 2 \leq n$.

The last condition helps to prevent misinterpreting an outlier as a level shift by making sure that a long enough sequence of data is observed to filter out ephemeral fluctuations. Upon the detection of a level shift, SENSE restarts its experts by only considering data after the level shift occurrence and resetting η for each expert. This means that the weight of each expert is determined only by the accuracy of prediction after the last level shift. In other words, the Level-shift step slides its learning window to consider only data after the last level shift in the experts' weight computation. SENSE's level shift mechanism improves estimation accuracy when compared to Fixed-Share, which keeps weights of poorly performing experts from becoming negligible. It enables SENSE to adapt to persistent conditions as swiftly as Fixed-Share, while allowing poor experts' weight to become as infinitesimal as in Static Experts algorithm.

In summary, SENSE employs three main techniques as follows:

- Smart experts reduce the sensitivity to experts and eliminate the need for a-priori data knowledge. SENSE employs EWMA equations with different weights as its experts and normalizes errors by the maximum observable output.

- META-learning expedites convergence by tracing recent past history and adjusting each expert's penalty accordingly.

- Level-shift improves SENSE's response to sudden data changes by bounding SENSE's learning time window; upon detecting dissimilar data patterns, SENSE reinitializes its tunable parameters and starts to re-learn.

2.3.2 SENSE's Accuracy

According to [21], Static Experts' inaccuracy or "loss" is bounded by the sum of two terms, the total loss of the best expert and the total number of experts involved. This is described in (4), where $L(S, A)$ and $L(S, Expert_i)$ represent the loss of Static Experts algorithm A over the whole dataset S and the loss of the best expert i respectively. Furthermore, c is a constant determined by the type of loss function employed while n is the total number of experts. This upper bound represents how far Static Experts' prediction will deviate from the corresponding real data. Equation 2.4 confirms that the more experts used, the higher the additional loss (i.e., on top of the best expert's loss) incurred by the Static Experts algorithm.

$$L(S, A) \leq L(S, Expert_i) + c \ln n \tag{2.4}$$

This upper bound on Static Experts' loss is only valid under the assumption that a given expert acts as the best expert over the whole dataset. However, when data patterns change so that experts take turns as the best expert, this upper bound needs to be recalculated as follows. At first we need to count all possible scenarios that can happen under dynamic environments. When all samples (trials) l are divided into $k + 1$ segments, for example, the number of ways to place $k + 1$ segments over l trials is $lCk + 1$. Here, a segment refers to a sequence of trials for which a given expert is the best one. Since the number of ways to map n experts to the best expert in each $k + 1$ segments is $n(n - 1)k$, then all possible cases amount to $lCk + 1n(n - 1)k$. If we consider each case as a "partition" expert, a specific partition expert can act as the best one over the whole trial set so that we can adopt 2.4 to predict the accuracy of Static Experts under dynamic environments. Here, the sequence of segments and its associated sequence of best experts are called a partition. Namely under dynamic datasets, the additional loss of Static

Experts due to the number of experts becomes $c[(k+1)\lg n + k\lg l/k + k]$.

To mitigate the dependence of Static Experts' upper bound on l and k, the Fixed-Share was introduced in [21]. Its main goal is to lessen the additional loss of the best partition, not the best expert like Static Experts, by introducing the Share Update operation. In Fixed-Share, the loss consists of three components as shown in 2.5, namely: the loss of the best partition $L(P_k(S))$, the number of experts, and the final loss incurred by the Share update operation $L(sharing)$. However, Fixed-Share has a problem of inappropriately distributing weights of experts. This is because $L(sharing)$ has a term that depends on the number of whole trials l, similarly to Static Experts. Note that Fixed Share tends to cut down a relatively large portion of the best expert's weight in preparation for sudden changes in the dataset, whose occurrence times are unpredictable.

$$L(S, A) \leq L(P_k(S)) + c\ln n + L(sharing) \tag{2.5}$$

Unlike Fixed-Share and Static Experts, SENSE's loss upper bound is independent of the length of whole trials. It is dependent on SENSE's Level-shift mechanism, which restarts experts' weights more rapidly without sacrificing of the best expert.

2.4 Evaluating SENSE's Performance

We evaluate SENSE using a variety of datasets and compare SENSE's performance against that of the original Fixed-Share algorithm and EWMA. In the first set of experiments, we use synthetic data that exhibit different periodic patterns. We use both sine and square wave signals with a range of frequencies. These experiments systematically test how well SENSE can track the variation of input data over a wide spectrum of frequencies when compared to Fixed-Share and

EWMA with different values of its smoothing factor, α. For a thorough comparative study, we also apply SENSE to the RTT dataset used in [40] and compare its predictions against estimates obtained using: (1) the original Fixed-Share algorithm, (2) Jacobson's TCP RTT estimation algorithm [24] (which is a variant of EWMA), and (3) "pure" EWMA with different smoothing factors.

In addition, we run SENSE over real collision rate data collected from a production Wireless LAN environment where access points (APs) periodically collect traffic and load statistics such as the number of retransmissions, total number of frames transmitted, etc. Table 3.2 lists the default values of SENSE's parameters common to all results presented in this section. The performance impact of SENSE's techniques and parameters is evaluated in Section 2.5.2.

Table 2.2: SENSE's Parameters

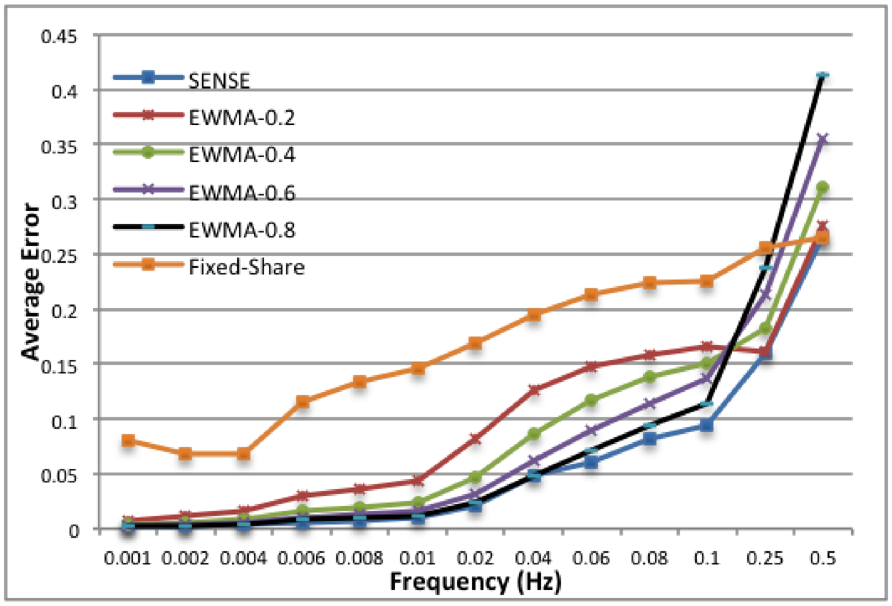
Parameter	Value
β	2
EL	0.01
η_{min}	10
η_{max}	100
N	4
Expert 1's α	0.2
Expert 2's α	0.4
Expert 3's α	0.6
Expert 4's α	0.8
j	2

2.4.1 Datasets with Periodic Patterns

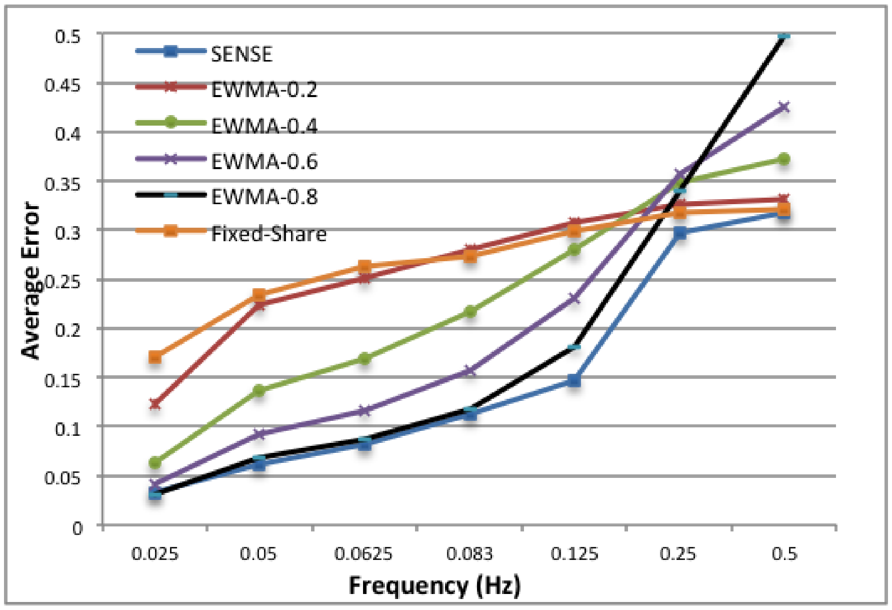
These first sets of experiments compare SENSE's accuracy with EWMA and Fixed-Share when estimating datasets that follow periodic patterns. We use a dataset consisting of 1,000 samples. For the sine wave pattern, these samples create one period for 0.001 Hz and 200 periods for 0.5 Hz. The amplitude of our sine waves fluctuates between 0.25 and 0.75. For the square wave, these samples generate 40 periods for 0.025 Hz and 200 periods for 0.5 Hz. The amplitude of our square waves fluctuates between 0.1 and 0.7.

Choosing the best α value depends on data autocorrelation and is a key factor for EWMA based estimators' performance. Values of α closer to one have less of a smoothing effect and give more weight to recent changes in the data, while values of α closer to zero have a greater smoothing effect and are less responsive to recent changes. Note that, choosing α should be based on how quickly or slowly the dataset change; lower α worsens the accuracy for rapidly changing datasets, while higher α degrades the accuracy when data fluctuations are smoother. We show that SENSE eliminates EWMA's dependency on α.

In these experiments, SENSE uses four EWMA experts with α values evenly spaced between 0 and 1, i.e., 0.2, 0.4, 0.6, and 0.8. We compare SENSE against four EWMA equations with same α values as SENSE, namely: 0.2, 0.4, 0.6, and 0.8 which represent low, medium, and high EWMA smoothing factors. We also compare SENSE with Fixed-Share using 100 experts. In our prior work [40], we used the Fixed-Share algorithm with 100 experts to estimate TCP's RTT and, as expected, observed that beyond 100 experts the resulting improvement in prediction accuracy is not significant given the additional processing cost and convergence time. Each expert was assigned a value in the dataset's range; recall that the expert's value represent its prediction. In the case of sine and square waves, each expert's prediction in the Fixed-Share Expert algorithm is drawn from a uniform distribution from 0 to 1. As for the input data function, we use two patterns: sine-(results plotted in Figure 2.2a) and square waves (results shown in Figure 2.2b).



(a) Sine waves



(b) Square Waves

Figure 2.2: Average error comparison over periodic data

Figure 2.2a plots the accuracy of the three approaches as measured by their average error as a function of the sine wave frequency. Each point in Figure 2.2a is calculated by averaging the absolute error of all 1,000 samples. As expected, at higher frequencies, the input's current value tends to be further apart from the last outcome, which makes it harder to accurately predict. Figure 2.2a confirms that SENSE produces lower average error than any of the four EWMA filters and Fixed-Share over the entire frequency range. As the frequency goes up, errors from EWMA filters rise steeply regardless of the α value. EWMA with higher α tends to exhibit better accuracy over the lower frequency range, while EWMA with lower α performs better for frequencies higher than 0.1 Hz.

The reason for this phenomenon is that at lower frequencies, each sample tends to be similar to its previous one; consequently, tracking the sine wave with higher α by placing more weight on recent trials, yields higher accuracy. At high frequency where recent trials are less correlated to the upcoming trial, however, it is better to stick to previous history that will repeat after a short period of time. Indeed, the higher α causes some constant amount of error at every measurement while the lower alternatively results small and large errors.

Fixed-Share's average error for various frequencies does not change significantly and its graph has a smaller slope. For sine wave's low frequencies, Fixed-Share shows larger error than other methods. The reason is that it takes longer for Fixed-Share to offset its large number of poor experts' weights and boost few well performing experts' weights. In the case of higher frequencies, it does not follow rapid data fluctuations and rather stays around the average value of sine wave due to a large number of experts and few trials for adaptation. In contrast to "pure" EWMA and Fixed-Share, SENSE dynamically adapts according to the frequency by choosing an appropriate EWMA expert for a given frequency range.

As the frequency increase, SENSE shifts its reliance from EWMA with higher α to EWMA with lower α.

Figure 2.2b shows the average error of SENSE, four EWMA filters and Fixed-Share method when driven by square waves. This figure exhibits very similar trend as Figure 2.2a where SENSE outperforms other methods at all frequencies. SENSE's smart experts are able to automatically switch between EWMA with high α value at low frequencies and EWMA with low α over the high frequency range.

2.4.2 Estimating TCP Round-Trip Times (RTT)

We also evaluated SENSE's accuracy when applied to real datasets. TCP, one of the most widely deployed Internet protocols, uses round-trip time (RTT) as an indication of network load. TCP employs its RTT estimates to trigger TCP's core functions such as error- and congestion control. Motivated by how critical accurate RTT estimates are for TCP's performance, we evaluate SENSE's accuracy in estimating RTTs in comparison to the Fixed-Share algorithm employed in [40], as well as TCP's original RTT estimator based on Jacobson's well-known EWMA variant [15] as shown in 2.6, where α is typically set to 0.85.

$$Est_{RTT} = \alpha \times Est_{RTT} + (1 - \alpha) \times RTT \tag{2.6}$$

For these experiments, we use the RTT dataset in [40]. These RTTs were measured when a 16 MB file was transferred over a real network. As shown in Figure 2.3, SENSE is able to keep track of the RTT variations more faithfully than Fixed-Share and Jacobson over the entire observation period.

Table 2.3 summarizes the average normalized error of SENSE, the four different EWMA filters, Fixed-Share and Jacobson's algorithms when applied to the same

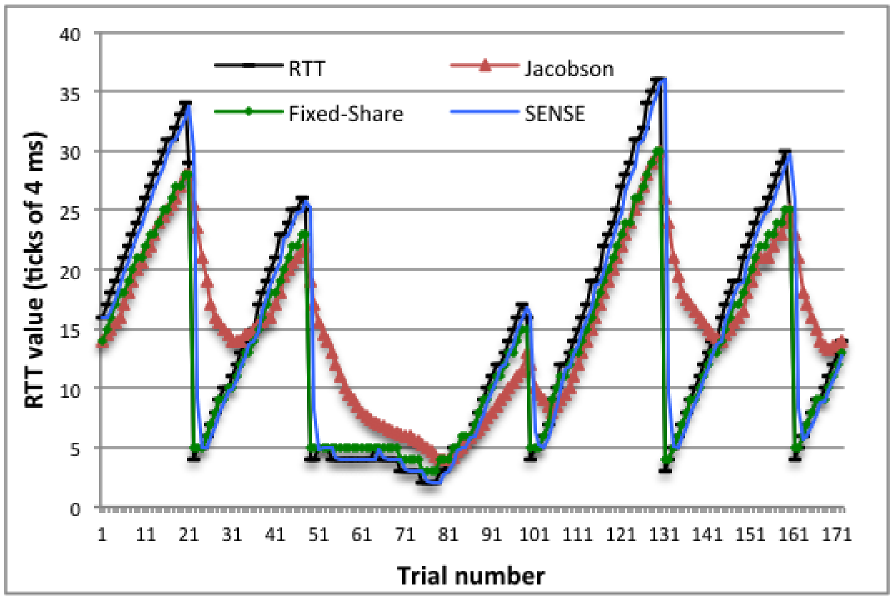


Figure 2.3: RTT prediction by SENSE, Fixed-Share and Jacobson for each data sample (represented by a trial number)

RTT data of Figure 2.3. In order to calculate the average normalized error, we first divide the absolute error of each sample by the real data it is trying to predict; then, we average these normalized errors. To compute the error ratio, we choose SENSE's average normalized error as baseline. Then, we calculate the other methods' relative error compared to SENSE as the difference between their average normalized error divided by SENSE's average normalized error. The resulting error ratio confirms that SENSE's accuracy outperforms both Fixed-Share and EWMA.

2.4.3 Estimating Collision Rates

To further evaluate SENSE's ability to forecast network dynamics in real environments, we applied SENSE to collision rate datasets measured in a production Wireless LAN (WLAN) environment. Collision rates were collected at access points (APs) as they send traffic to a node associated with it while other associated nodes

Table 2.3: Average Normalized Error Comparison for RTT Dataset

	Average Normalized Error	Error Ratio (%)
SENSE	0.26	-
Fixed-Share	0.33	26%
Jacobson	0.79	191%
EWMA-0.2	0.63	142%
EWMA-0.4	0.44	71%
EWMA-0.6	0.34	32%
EWMA-0.8	0.29	11%

concurrently communicate with the AP, as they usually do. Specifically, we transmit 100 Mbps of UDP traffic from the AP to a node for 200 seconds while we simultaneously run different types of traffic between interfering APs and interfering nodes (i.e., located close to the node receiving data from the AP). Collision rates are calculated every second as the ratio of the number of retransmitted packets to the total number of transmitted packets. Since the test AP and the test node are physically close to one another, we assume that retransmitted packets are solely due to collision, and not to noise interference.

Figure 2.4 depicts how SENSE and Fixed-Share track a time series of real collision rates gathered from the test network for 200 seconds. Similarly to the previous experiment, we use 100 experts for Fixed-Share and, based on the collision rate data, Fixed-Share's 100 experts are uniformly distributed between 0 and 0.2. We observe from Figure 2.4 that, initially, the dataset contains considerable "noise" caused by bursty traffic generated by short-lived flows from applications like the Web. After 100 trials (seconds), longer-lived flows resulting from traffic such as wireless video transmission becomes dominant, yielding "smoother" collision rate variations. Figure 2.4 shows that, while SENSE does not exactly follow the sudden jumps in the first half of the time series, its accuracy is significantly higher

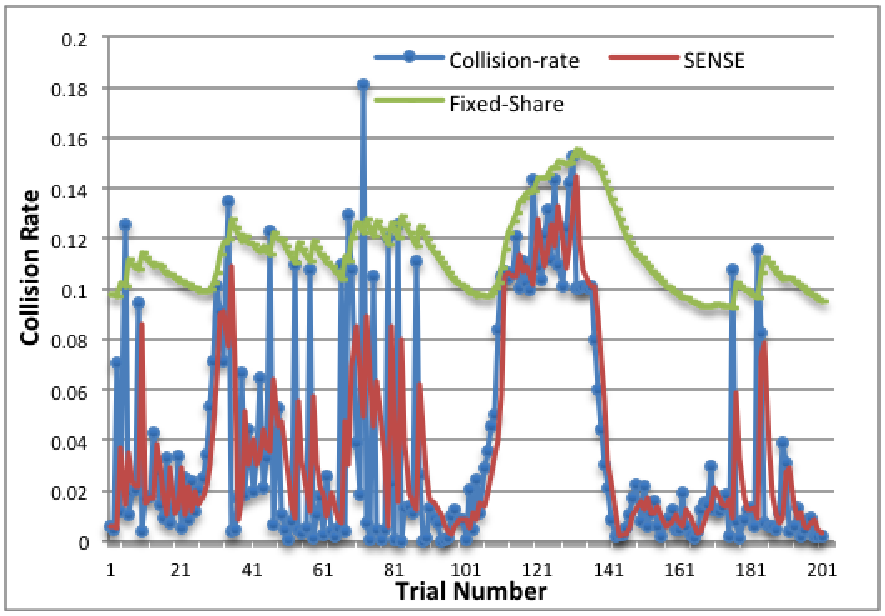


Figure 2.4: Trace of SENSE's collision rate prediction vs. Fixed-Share for each data sample (represented by a trial number)

than Fixed-Share's. And, in the second half of the graph, SENSE is capable of accurately tracking variations in the data.

As shown in Figure 2.4, Fixed-Share does not exhibit adequate accuracy and, according to Table 2.6, results in an unacceptably high error ratio. These results confirmed that Fixed-Share's lack of agility is due to the very short-lived data variations, which do not allow enough time to "train" Fixed- Share's experts.

Figure 2.5 shows a closer view of the behavior of SENSE compared against two EWMA filters over a 25-second interval between 65-90 seconds of Figure 2.4. Note that in this span of time, data fluctuate significantly, which makes it very difficult for any predictor to predict accurately. In this period, SENSE behaves like a low-pass filter, e.g., EWMA with α set to 0.2, while the curve corresponding to EWMA with value α of 0.8 looks like the real data but delayed by a full trial, which results in the highest error. Table 2.4 summarizes the results shown in Figure 2.5 by comparing the average error and error ratio for the first 100 trials of

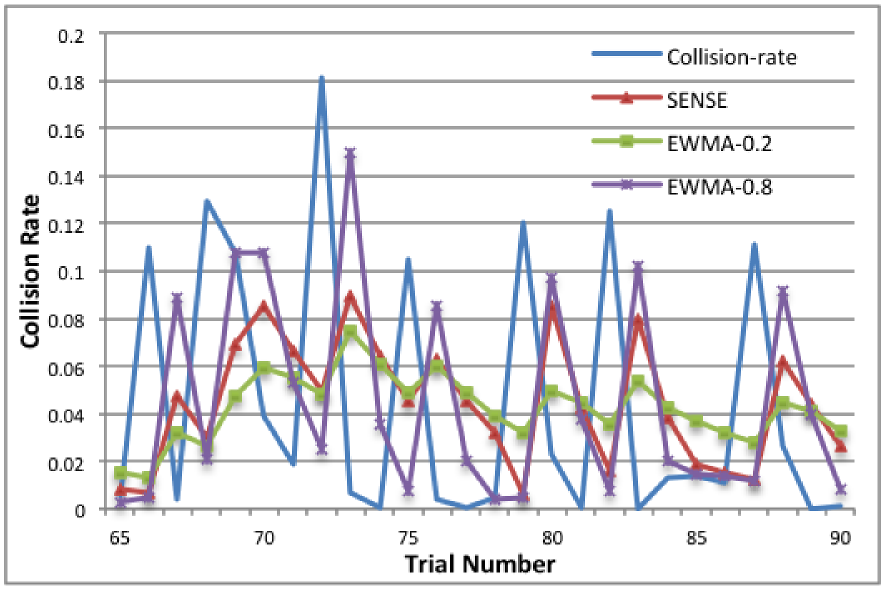


Figure 2.5: SENSE vs. EWMA for highly variant portion of collision rate

the collision rate dataset. It confirms that, for the first half of the dataset, which is quite "noisy", SENSE acts as an EWMA predictor with lower α and yields highest accuracy.

Figure 2.6 zooms in the performance of SENSE and two EWMA filters over the interval of 100-145 seconds in Figure 2.4. As shown in Figure 2.6, SENSE quickly catches up with collision rate changes and behaves similarly to EWMA with $\alpha = 0.8$ (acting as a high-pass filter). In contrast, EWMA with α value of 0.2 lags behind and cannot keep up with the collision rate variation. During this period, EWMA with α value of 0.2 exhibits poor performance comparing to the other methods.

Table 2.5 lists the average error and error ratio for the last 100 trials of Figure 2.4. During this interval where EWMA with $\alpha = 0.8$ is clearly a better choice, SENSE behaves as EWMA predictor with high α but achieves slightly higher accuracy.

Table 2.5 summarizes the average error and error ratio of the five different

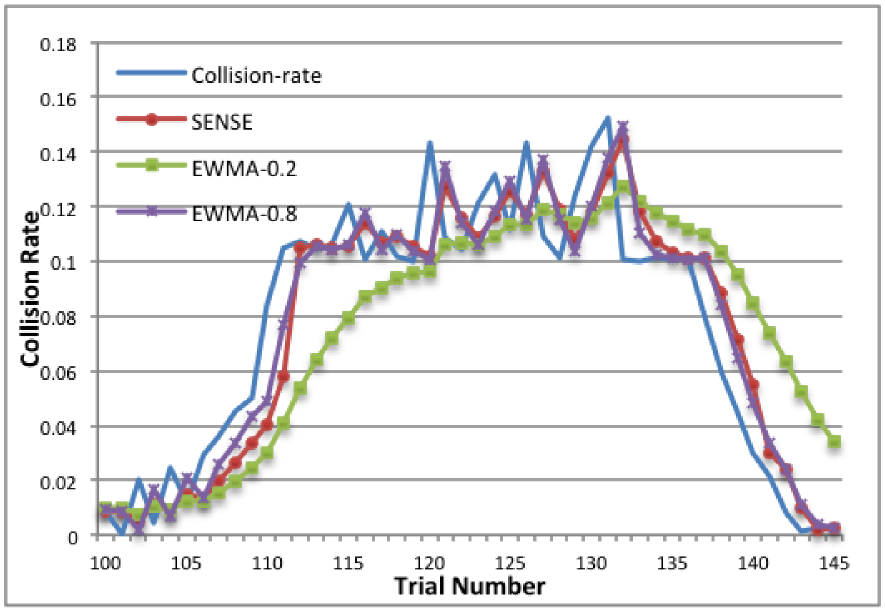


Figure 2.6: SENSE vs. EWMA for smooth portion of collision rate

Table 2.4: Average Error for First 100 Trials of Collision Rate Dataset

	Average Error	Error Ratio (%)
SENSE	0.0323	-
EWMA-0.2	0.033	2.5%
EWMA-0.4	0.0335	4%
EWMA-0.6	0.0346	7%
EWMA-0.8	0.0365	13%

forecast schemes over the whole collision rate dataset depicted in Figure 2.4. It confirms SENSE's ability to automatically adapt its performance based on network dynamics. In the case of uncorrelated behavior, SENSE gives more weight to experts with low α and in the case of correlated data, more weight is given to experts with high α value. Since EWMA does not have this capability, for the first half of the dataset, EWMA with $\alpha = 0.8$ is worse than SENSE by 13% (from Table 2.4) and for the second half of the dataset, EWMA with $\alpha = 0.2$ is significantly worse than SENSE (40% from Table 2.5). Table 2.6 clearly evidences that SENSE

Table 2.5: Average Error for Second 100 Trials of Collision Rate Dataset

	Average Error	Error Ratio (%)
SENSE	0.0138	-
EWMA-0.2	0.0193	40%
EWMA-0.4	0.0158	14%
EWMA-0.6	0.0148	7%
EWMA-0.8	0.0142	3%

yields higher accuracy when compared to all the other four methods by at least 8% for the complete dataset. This comparison confirms SENSE's dynamic behavior to selectively and swiftly chooses the best expert according to the observed network dynamics. During noisy periods in the dataset, SENSE picks an expert with low α while during periods when the data changes more smoothly, SENSE prefers an expert with high α value.

Table 2.6: Average Error for The Whole Collision Rate Dataset

	Average Error	Error Ratio (%)
SENSE	0.013	-
EWMA-0.2	0.0257	9%
EWMA-0.4	0.0244	8%
EWMA-0.6	0.0244	8%
EWMA-0.8	0.025	8.5%
Fixed-Share	0.0774	466%

2.4.4 Prediction Summary

The accuracy of Fixed-Share algorithm depends on prior knowledge on the range of dataset, large number of experts, and proper expert distribution that are not usually available in on-line problem. Even though more experts are essential for better accuracy, this requires a lot of trials for Fixed-Share algorithm to settle down to the best expert, resulting in significant inaccuracy under the dynamic situations.

EWMA prediction accuracy is significantly dependent on the value of α. As it is shown in our results, finding the proper α value requires prior statistical information about data, which is again not available in on-line problems.

SENSE eliminates the requirement for this prior knowledge by using smart experts. SENSE dynamically determines appropriate expert values and weights to minimize prediction error. Our results showed that SENSE agility results in best prediction performance under various scenarios and data set.

2.5 Impact of SENSE's Techniques and Parameters

In this section, we examine the impact of SENSE's parameters and techniques, i.e., Level-shift and META- learning. We start by running the same experiments used in [21] which were designed to show how Fixed-Share algorithm tracks the predictions of the best expert; tracking best expert's predictions has been shown to improve prediction accuracy when compared to the Static Experts for rapidly changing data. Our results show that SENSE's best expert weight adjustment mechanism performs as well as Fixed-Share for rapidly changing data.

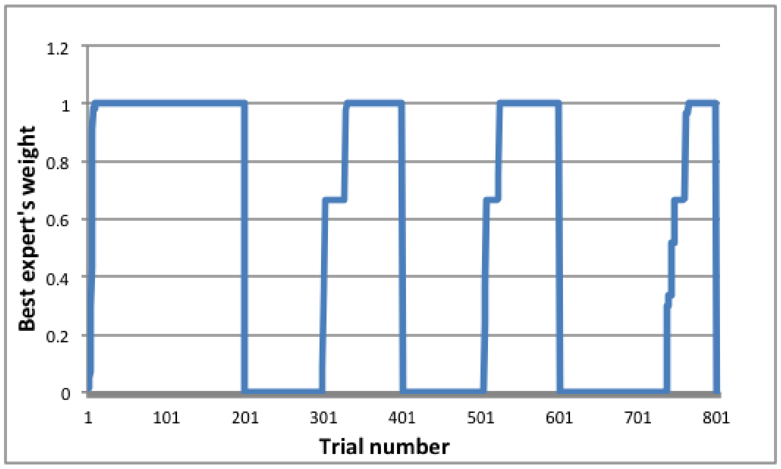
2.5.1 Best Expert's Weight Recovery

As described in Section 2.1.2, Fixed-Share was proposed to overcome the slow weight recovery of the best expert in the Static Expert algorithm. This feature of Fixed-Share was evaluated in [21], which reports on how Fixed-Share tracks the predictions of the best expert compared to the Static Experts algorithm.

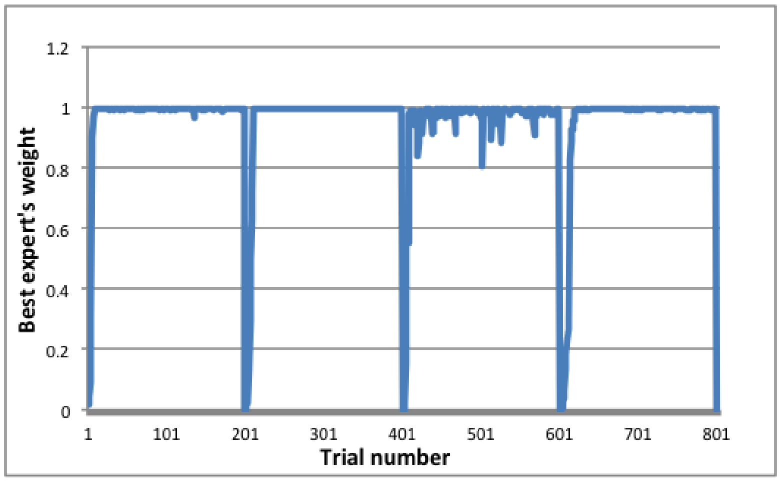
We conducted exactly the same experiments described in [21]; our input data consists of a sequence of 800 trials with three level shifts at trials 201, 401 and 601. For trials in the range of [1,200], [201,400], [401,600], and [601,800], the outcome y_t, is 0, 10, 5, and 15, respectively. The overall prediction was based on 64 experts. Every group of 200 trials has its own best expert, which changes at every shift. Best expert's prediction comes from a uniform random distribution on a smaller range than typical experts. For example, for trial [0,200], since the outcome is always 0, expert's values are chosen from a uniform random distribution between $(0,\frac{1}{2})$ and $(0,\frac{1}{2}\sqrt{0.1})$ for the typical and best experts, respectively. Figure 2.7a, 2.7b, and 2.7c plot the weight change of the best expert in the three algorithms during four 200-trial segments. Confirming the results reported in [21], Figure 2.7a and 2.7b show that in the first 200-trial segment, Static Experts performed comparably to Fixed-Share, whereas in the remaining three segments, it considerably under-performed. For Static Experts, it takes almost 100 trials for the new best expert's weight to approach 1 from almost 0. In contrast, as shown in Figure 2.7b, Fixed-Share can quickly learn the new best expert for the current segment. We should note that, as in [21], the sharing degree parameter (α) is set to its best value considering the number of shifts in data, which is not always the case. Our results as plotted in Figure 2.7c confirm that SENSE can learn as quickly as Fixed-Share. SENSE's quick learning ability is accomplished using Level-shift.

2.5.2 Impact Of Parameters

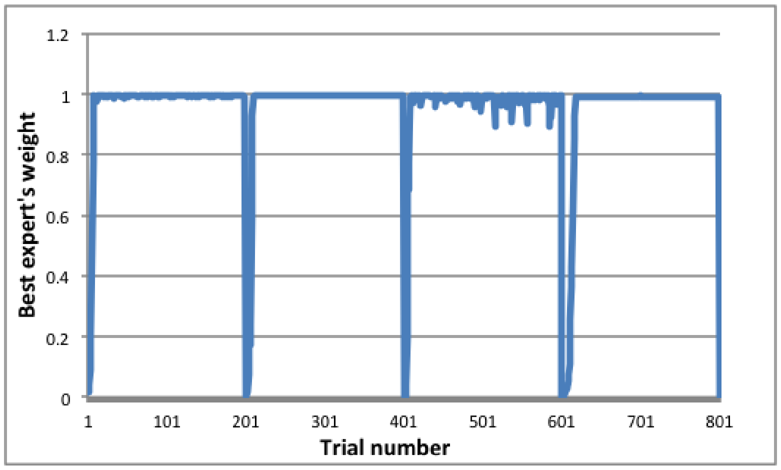
In this section, we evaluate the effect of SENSE's tunable parameters such as number of experts, β, η_{min} and η_{max}. Although results presented in this section are from experiments using datasets following sine wave patterns only, we observed



(a) Best expert's weight in Static Expert Algorithm



(b) Best expert's weight in Fixed-Share algorithm



(c) Best expert's weight in SENSE

Figure 2.7: Best expert's weight recovery test

similar results when we ran these experiments with our other datasets.

Table 2.7: Values for Different Number of Experts

Number of Experts	Values of α
1	0.5
2	0.25, 0.75
4	0.2, 0.4, 0.6, 0.8
8	0.125, 0.25, ..., 0.875, 1
16	0.0625, 0.125, ..., 0.875, 0.9375, 1
32	0.03125, 0.0625, ..., 0.9375, 0.968, 1
100	0.01, 0.02, 0.03, ..., 0.98, 0.99, 1

The experiments whose results are shown in Figure 2.8 evaluate SENSE's sensitivity to the number of experts. We run SENSE with different numbers of experts on a sine wave input over a range of frequencies. The values of α are uniformly distributed between 0 and 1; for instance, in the case of 4 experts, we use α values of 0.2, 0,4, 0.6 and 0.8. Table VII shows α values used for this experiment. As can be observed in Fig. 8, SENSE's performance changes only slightly when the number of experts increases beyond 2. This is consistent with our observations in [40].

Figure 2.9 shows the impact of META-learning's β parameter on SENSE's behavior by plotting the average error-frequency curves for different β values. We observe that the difference in accuracy is almost indistinguishable for different β. This can be explained by the fact that each expert does its best to keep track of the input data. META-learning is invoked only when errors tend to continuously increase or decrease since it is designed to severely penalize static experts that maintain their prediction regardless of current measurements.

We also evaluated the impact of the META-learning parameters, η_{min} and η_{max} on SENSE's performance. Figure 2.10 shows the average error rate for each

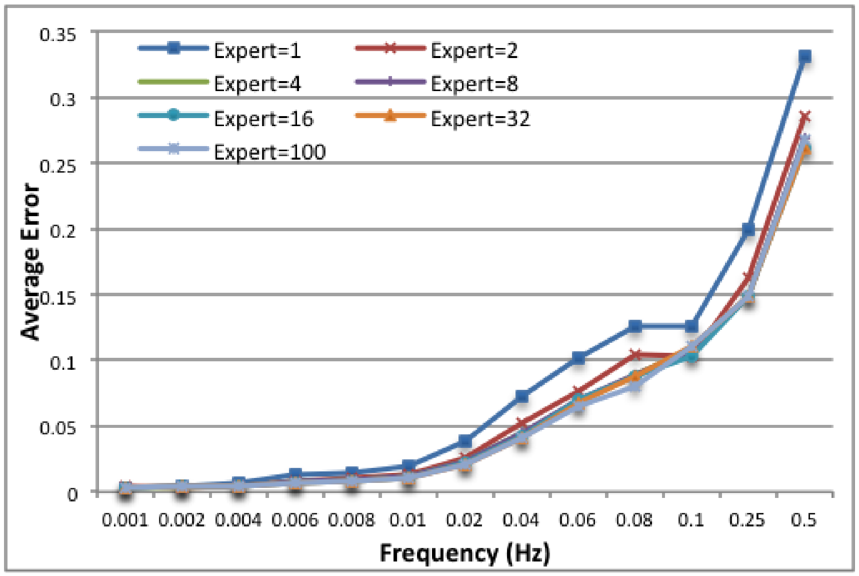


Figure 2.8: SENSE's sensitivity to number of experts over sine waves

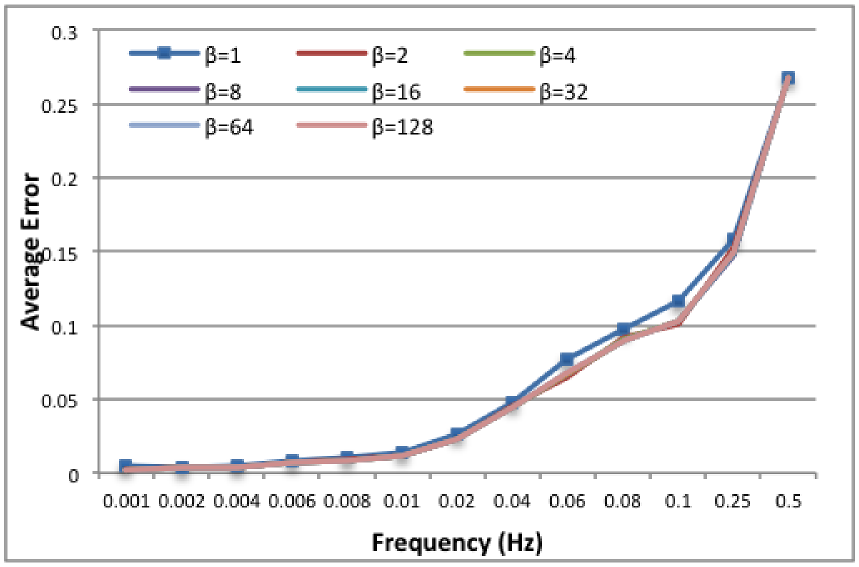


Figure 2.9: SENSE's sensitivity to β over sine waves

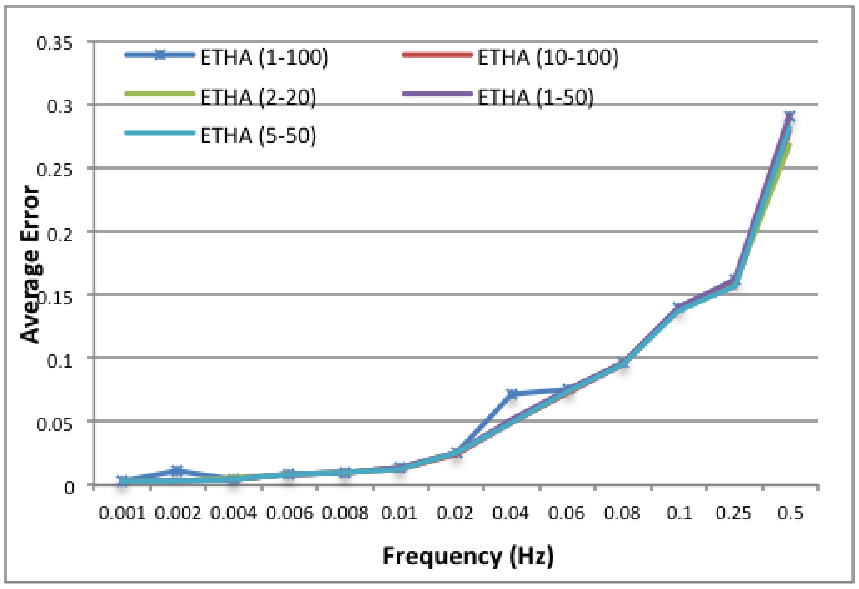


Figure 2.10: SENSE's sensitivity to η_{min} and η_{max} over sine waves

pair of (η_{min},η_{max}) using, as input, to sine waves with different frequencies. We observed that different values of η_{min} and η_{max} do not have significant effect on SENSE's performance. In our experiments, we used $\eta_{min} = 10$ and $\eta_{max} = 100$.

2.5.3 Impact of Level-shift and META-learning

We evaluate the effect of the Level-shift and META- learning methods on SENSE's performance. Figure 2.11 and 2.12 show the increase in accuracy (percentage of average error improvement) when SENSE uses: (1) Level-shift only, (2) META-learning only, and (3) Combined Level-shift and META-learning. Both figures confirm that these techniques improve the performance of SENSE. Note that the improvements resulting from Level-shift on RTT are much higher than on collision rate. The reason is that the RTT data has a larger number of level shifts and SENSE's Level-shift mechanism can detect them and adjust the experts' weights to follow the variations in the data. On the other hand, in the collision rate dataset, data fluctuates significantly and does not trigger the Level-shift

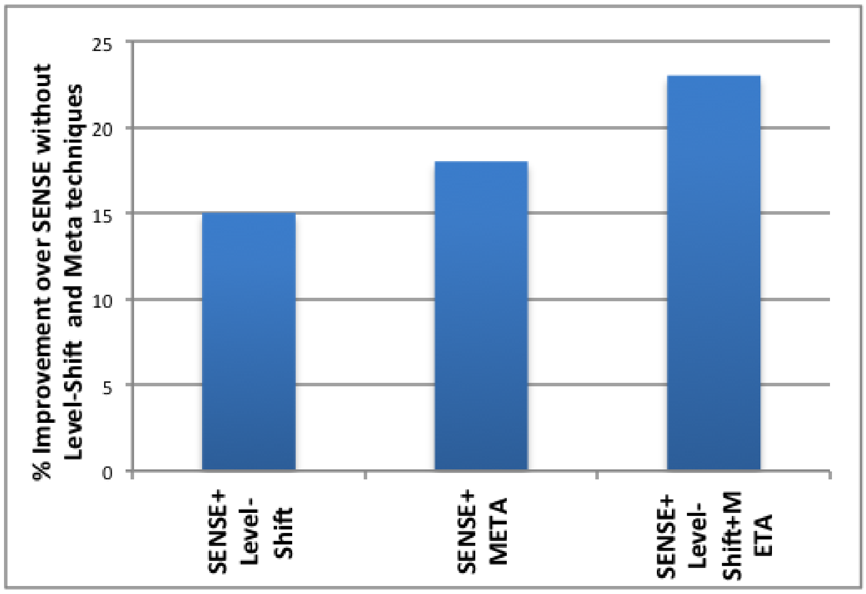


Figure 2.11: Impact of Level-shift and META-learning methods on RTT dataset

mechanism.

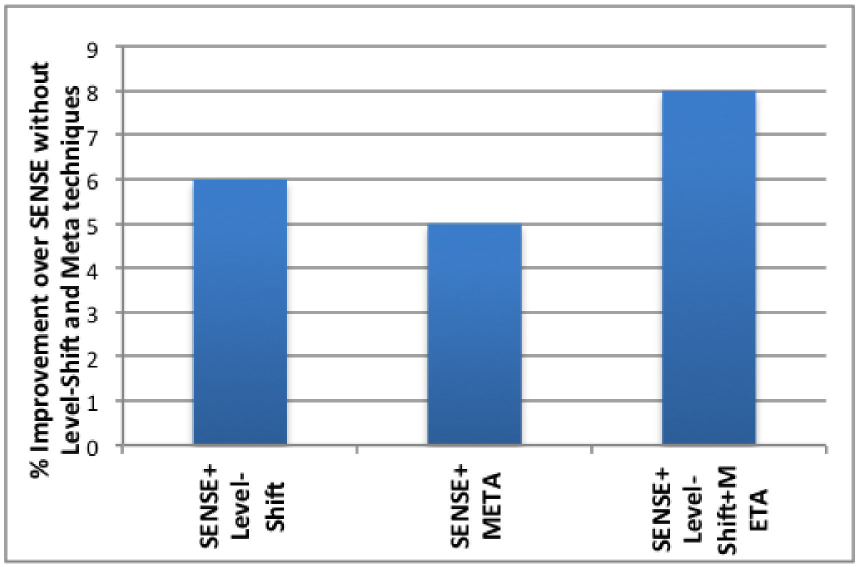


Figure 2.12: Impact of Level-shift and META-learning methods on collision rate dataset

Similarly to Level-shift, META-learning yields larger contribution to SENSE's performance for the RTT dataset than collision rate. And again, the reason is that the RTT dataset exhibits smoother behavior; therefore, META-learning is able to effectively increase the weight of "good" experts and decrease the weight of "bad" experts, which improves SENSE's performance overall. Consequently, the combined improvement of both techniques for the RTT dataset is almost 25% and just below 10% for the collision rate dataset.

Chapter 3

Smart Adaptive Collision Avoidance for IEEE 802.11

The IEEE 802.11 standard, also known as WiFi, specifies both physical layer (PHY) and medium-access control (MAC) protocols for wireless local area network (WLAN) communication [1]. It is considered the de-facto standard for WLANs and, as such, has attracted considerable attention from both networking researchers and practitioners over the years. The first IEEE 802.11 standard was released in 1997, and since then, has grown to a large family of WLAN protocols as new frequency bands and PHY technologies became available. IEEE 802.11 defines two different types of MAC protocols, a mandatory one called Distributed Coordination Function (DCF) and an optional one called Point Coordination Function (PCF), built atop of DCF. IEEE 802.11 DCF, which is by far more widely deployed than its PCF counterpart, arbitrates access to the shared communication medium using a random-access (or contention-based) approach, in particular the Carrier Sense Multiple Access (CSMA) [30] protocol with or without collision avoidance (CA) [27]. In other words, IEEE 802.11 DCF defines a *base mode* which uses physical carrier sensing and a link-layer acknowledgment (ACK) to confirm correct reception of the transmitted data frame. DCF also specifies an optional mode which employs both physical- (i.e., CSMA) as well as virtual (i.e., CA) carrier sensing. As described in more detail in Section 3.1, CSMA/CA [27] was proposed as a way to combat the so-called *hidden terminal* problem. It allows nodes to reserve the channel before engaging in data communication by exchanging short control frames, namely Request to Send (RTS) and Clear to Send (CTS) ahead of transmitting data. RTS/CTS has been part of the IEEE 802.11 standard since its early versions and has been in use since then including more recent variants such as 802.11n and 802.11ac. However, as described in Section 3.1, the RTS/CTS handshake can also negatively impact performance since it introduces additional delay and overhead.

For this reason, IEEE 802.11 has defined a configurable parameter named *RTS Threshold (RT)*, which is used to enable and disable the RTS/CTS exchange. However, the standard does not specify what *RT* value(s) to use. For example, in some implementations, *RT* is set such that for small data frames, DCF's base mode is used. Otherwise, RTS/CTS is used when frame size is large enough. There are also cases where the *RT* value is set to the maximum frame size, so that RTS/CTS is never used. However, 802.11 product manufacturers make recommendations to users that if they are experiencing degraded performance, they should test their network with lower *RT* values.

As described in more details in Section 3.2, a number of studies have explored techniques to dynamically set the value of the *RT* based not only on packet size but also on other characteristics (e.g., transmission rate) and conditions (e.g., packet delivery ratio).

In this chapter, we start by conducting an empirical characterization of RTS/CTS performance as a function of a number of factors including packet size, transmission rate (for both data and control), and network contention. Our experimental RTS/CTS performance characterization study complements existing work that have analytically and/or experimentally studied RTS/CTS performance. It shows that network contention, as well as packet size, and transmission rate must be collectively considered in order to decide whether to enable or disable 802.11's RTS/CTS mechanism. Based on the results of our RTS/CTS performance characterization, we propose a novel approach that uses machine learning to dynamically switch RTS/CTS on and off ahead of data transmission by considering a combination of "air time", i.e. the ratio between the size of data/control information being transmitted and transmission rate, as well as network contention. It is noteworthy that (1) by accounting for network contention, the proposed mechanism is able to automatically adapt to different WLAN environments and dynamics and (2) by considering packet size and transmission rate, it will also accommodate different IEEE 802.11 variants, especially as new versions target new applications and have increasingly larger packet sizes and transmission rates. Additionally, as wireless networks become denser (e.g., in urban scenarios), the importance of avoiding potential interference amongst them grows and thus efficient use of RTS/CTS becomes even more critical to achieve adequate performance.

To the best of our knowledge, our algorithm is the first to allow automatically enabling and disabling the RTS/CTS handshake based on a set of current network conditions, and dynamically adapt when these conditions change. As current and emerging wireless network environments evolve and become increasingly more heterogeneous in terms of the underlying network technologies, connected devices, as well as driving applications, being able to dynamically adapt protocol behavior in response to changing conditions and requirements is critical to achieve adequate performance in a seamless manner. Our experimental results using both synthetic workloads as well as real traces show that our mechanism consistently outperforms current best practices, such as never enabling RTS/CTS or setting the *RTS Threshold* (*RT*), which is used to decide whether to switch RTS/CTS on or off, to a static value.

The rest of this chapter is organized as follows. Section 3.1 provides a brief overview of IEEE 802.11 DCF and discusses RTS/CTS' trade-offs. Related work is presented in Section 3.2 and Section 3.3, conduct our empirical performance evaluation of 802.11's RTS/CTS as a function of different factors. Our method to dynamically enable or disable RTS/CTS is described in Section 3.4. In Section 3.5 and Section 3.6 our experimental methodology and results are presented respectively.

3.1 Background

IEEE 802.11's Distributed Coordination Function (DCF) [1] is based on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol. In DCF's *base mode*, CSMA [30] is used by stations that have data to send to check whether the shared medium is being used. More specifically, a station that wants to transmit a data frame senses the channel to check whether it is idle for a DCF Inter-frame Space (DIFS) interval. If the channel is sensed idle, the station transmits the data. Otherwise, it defers transmission using a random backoff timer. Figure 3.1 illustrates how DCF's base mode works. After transmitting data, the station waits for an acknowledgement (ACK). If the ACK is received, the station considers the data frame delivered. If not, it will assume a collision occurred and uses a slotted Binary Exponential Backoff (BEB) scheme to retransmit the frame at a later time.

IEEE 802.11's DCF can be configured to use, in addition to physical carrier

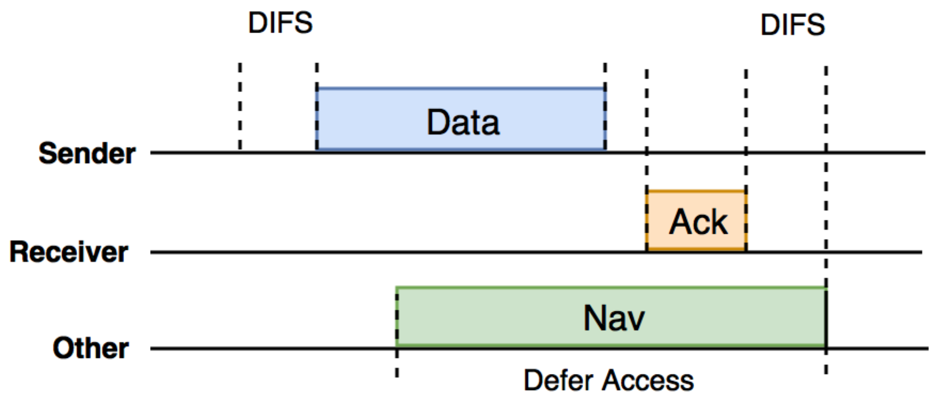


Figure 3.1: IEEE 802.11 Base Mode: CSMA

sensing, virtual carrier sensing, or Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) [27]. CSMA/CA avoids hidden terminal collisions by reserving the channel ahead of data transmission. Channel reservation is achieved via a two-way handshake using small control frames, namely the Request to Send (RTS) and Clear To Send (CTS). The RTS/CTS handshake works as follows: the sender sends the RTS frame to the receiver, and the receiver responds with a CTS. Other stations that overhear either the RTS, CTS, or both mark the channel as busy and set their network allocation vector (NAV) based on the time offered in sender's RTS and/or receiver's CTS. This means that they will defer their transmissions for the interval indicated in their NAV. The sender uses the receipt of the CTS frame from the receiver as the indication that the channel has been reserved for its transmission, and therefore, the sender transmits the data frame. Figure 3.2 shows how the RTS/CTS handshake works.

Although the RTS/CTS handshake has clear benefits, it also introduces overhead. In the remainder of this section, we discuss RTS/CTS' pros and cons.

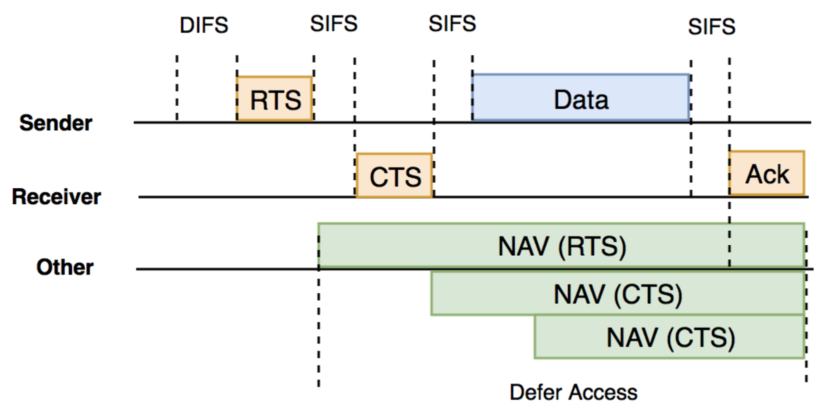


Figure 3.2: IEEE 802.11 CSMA/CA

3.1.1 RTS/CTS Upsides

The RTS/CTS exchange was proposed as a way to combat the hidden node problem as illustrated in Figure 3.3. In the scenario of Figure 3.3, node B is in the transmission range of both nodes A and C. However, because A and C are outside each other's transmission range, they cannot hear each other, and thus are said to be "hidden" from one another. Suppose that DCF's base mode is used and A is sending a packet to B. Suppose that C also has data to send to B. Since C cannot hear A, it senses the channel idle and sends data to B which results in a collision and node B discards both frames.

If RTS/CTS is used, A sends a RTS frame before the actual data. All of A's neighbors mark the channel as busy. Upon reception of the RTS, B sends a CTS back to A if it is not busy. All of B's neighbors, including C, mark the channel as busy after hearing the CTS. As a result, C defers its transmission for the time specified in the CTS frame. To show how effective RTS/CTS can be in avoiding collisions, we ran a basic experiment with four hidden nodes sending to a central node, e.g., and access point. We use four CBR streams each sending at 5.5 Mbps with 1500 byte packets. Comparing the throughput obtained with and without

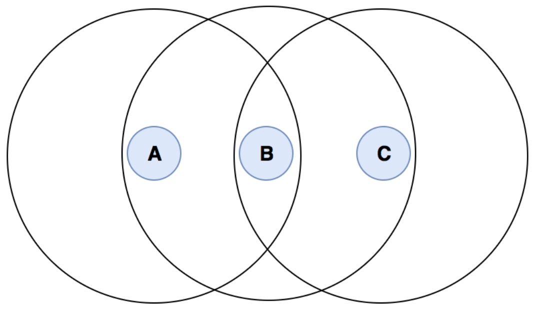


Figure 3.3: Hidden Node Scenario

RTS/CTS (Table 3.1), we observe that RTS/CTS can yield significant performance benefits (in this scenario, more than 70%).

Table 3.1: Throughput Comparison: DCF's Base Mode (CSMA) versus RTS/CTS (CSMA/CA)

	Basic Mode (Mbps)	RTS/CTS Mode (Mbps)
Stream 1	0.109	0.627
Stream 2	0.124	0.595
Stream 3	0.1	0.535
Stream 4	0.134	0.568

3.1.2 RTS/CTS' Downsides

While the RTS/CTS mechanism can mitigate the hidden terminal problem and avoid collisions, it has some drawbacks as described below.

Overhead

One of the main problems is the latency and additional load that RTS and CTS control frames introduce, which, in some scenarios, outweighs RTS/CTS benefits. IEEE 802.11 defines the RTS and CTS frame sizes as 20 and 14 bytes, respectively. So, in the case of short data frames, it may not be worthwhile

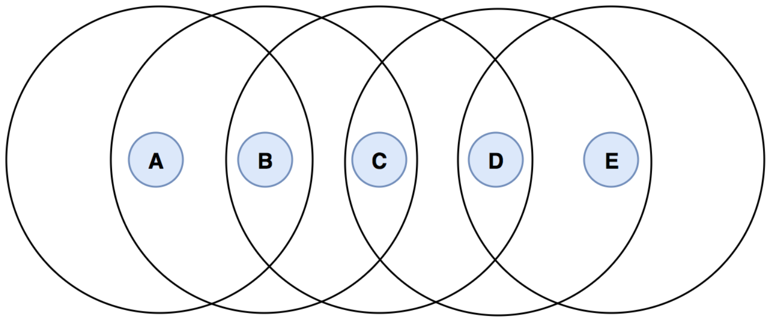


Figure 3.4: 5 Nodes Scenario

incurring the additional delay needed to perform the RTS/CTS exchange. That was the motivation behind proposing the *RTS Threshold*, or RT, which determines the minimum data frame size that will trigger the RTS/CTS handshake to reserve the channel to transmit the data frame. Frames smaller that RT will be sent using DCF's base mode.

Disabling Safe Concurrent Transmission

As illustrated in Figure 3.4, there are scenarios where the RTS/CTS reservation prevents concurrent transmissions that would not result in collisions. For example, suppose that node B is sending data to node A. Assuming that the interference- and the transmission ranges are equal, if node C also wants to send data to D, it would not impair node B's transmission. However, because of B's RTS packet C overhears, C unnecessarily defers its transmission to D for the time defined in B's RTS NAV. This of course negatively impacts the throughput.

Another example scenario that illustrates how RTS/CTS can negatively impact throughput is as follows. Suppose that while node A is transmitting to node B, D has data to send to C and transmits an RTS frame to C. However, C is blocked because of B's CTS sent in response to A's RTS to B and is not able to respond to D with a CTS. In this case, all nodes in the transmission range of D will be

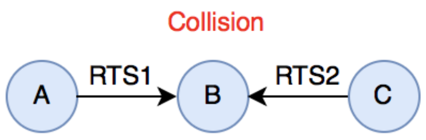


Figure 3.5: RTS collision scenario

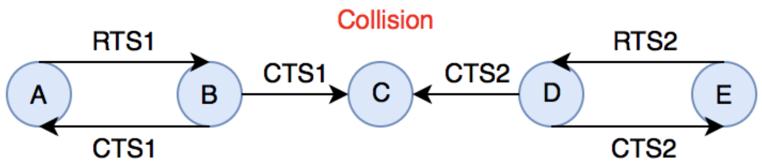


Figure 3.6: CTS collision scenario

blocked for the time specified in D's RTS frame.

It is worth pointing out that Denial of Service attacks could be staged/caused by malicious/malfunctioning nodes who can (un)- intentionally keep sending RTS or CTS frames possibly reserving the channel for long periods of time.

RTS/CTS Collision

In crowded areas, where hidden terminals are prevalent, the RTS/CTS handshake is less effective as a collision avoidance technique [51]. This is because RTS and CTS frames are themselves subject to collision in the same way as data frames. When the traffic load is heavy and the number of hidden terminals is high, the chance of unsuccessful RTS/CTS handshake increases due to higher channel contention and thus higher collision probability. Figure 3.5 and 3.6 shows scenarios where RTS or CTS collision may happen. Besides the delay and overhead incurred by the retransmission of the RTS, the channel would be unusable for nodes who overhear the RTS and CTS for the time specified in the NAV.

3.2 Related Work

Wireless networks has evolved a lot during the past few years and lots of works have been done to improve its performance. Many research papers have been published with focused on studying the performance of IEEE 802.11's DCF optional collision avoidance mechanism which uses the RTS/CTS handshake. We categorize these efforts in two groups. The first group investigates throughput performance when using or not using RTS/CTS and believe that the problems introduced by the RTS/CTS mechanism will tend to counterbalance those benefits. [44], discusses the problems introduced by the RTS/CTS mechanism with focus on the virtual jamming problem which allows a malicious node to effectively jam a large fragment of a wireless network at a minimum expense of power.

In [52], authors show that in some situations, the interference range is much larger than transmission range, where RTS/CTS cannot function well. So, RTS/CTS handshake cannot prevent all interference as we expect in theory. [49] analyzes the maximum throughput of traffic in the IEEE 802.11 DCF network that uses RTS/CTS and believes that as the number of RTS/CTS control frames increases, the collisions between RTS/CTS control frames occur more frequently. As a result, the analysis of networks that use RTS/CTS has been considered difficult because of the complex behaviors of RTS/CTS. The second group proposes ways to dynamically tune the value of RTS Threshold. In a more recent study [47], it was shown that, when the network is under "stress", e.g., high node density, high traffic load, etc, the RTS/CTS threshold RT value can significantly impact performance in terms of end-to-end delay, medium access delay, retransmission attempts, network load, and throughput. In [31] and [50], RTS/CTS performance was evaluated under different data rates and it was concluded that RTS/CTS does not show much benefit at higher data rates.

The other group of researchers explore how to enable/disable RTS/CTS dynamically. In [25], a mechanism in which the sender counts the number of "Waiting for CTS" timeouts is proposed. If this value exceeds a certain threshold, RTS/CTS is disabled. In [45] Authors propose a new adaptive RTS/CTS control method on the basis of existence of hidden terminals.The proposed algorithm in [38] can dynamically adjusts RT based on the frame delivery ratio and shows a significant improvement over existing CSMA/CA and RTS/CTS schemes. In [2], RT is set based on current frame distribution of the network. This value is set to a value such that some specific percentage of frames size's fall below that value. In [31], the RT is adjusted dynamically according to the data rate and number of stations.

As previously discussed, to our knowledge, no prior work has explored the combination of factors we consider in our approach to automatically enable or disable the RTS/CTS handshake in response to changing network- and traffic conditions.

3.3 Characterizing RTS/CTS Performance

In this section, we conduct an empirical characterization of RTS/CTS performance as a function of a number of factors, namely packet size, transmission rate for both data and signaling traffic, as well as network contention. The goal here is three-fold, namely: (1) confirm experimentally some analytical results on the performance of RTS/CTS in the literature; (2) make the case for dynamically adjusting the *RTS Threshold, RT* based on data and control transmission time as well as network contention; and (3) validate our experimental methodology and setup.

To show the effect of each factor on RTS/CTS performance, we run several simulation experiments using the *NS-3* network simulator for both infrastructure-based and infrastructure-less network scenarios. In our experiments, we associate 5, 10, and 20 nodes to an access point (AP) which are either all hidden or not hidden from each other. We use *NS-3*'s *Matrix Propagation Loss Model* and set the propagation loss between each pair of nodes to make them hidden or not hidden from each other. For example, if we set the propagation loss between nodes A and B to a very high value, then A becomes hidden from B, and vice-versa. Nodes send CBR traffic to the AP with packet sizes of 200, 500, 1000, 1500, and 2000 bytes and data rates of 2, 5.5, 11, 24, and 54 Mbps, while signaling transmission rate is kept at 2 Mbps. The graphs shown in Figures 3.7, 3.8, and 3.9 plot *RTS/CTS throughput gain*, T_{Gain} which is defined as follows:

$$T_{Gain} = (T_{RTS/CTS} - T_{Base})/T_{RTS/CTS} \tag{3.1}$$

where $T_{RTS/CTS}$ and T_{Base} are the throughput when using RTS/CTS and the throughput when using DCF's base mode, respectively.

3.3.1 Data Transmission Time

As previously pointed out, data packet transmission time, which is a function of the data packet size and the transmission rate, is an important factor affecting RTS/CTS performance. Data packet transmission time should be long enough to warrant the overhead of the RTS/CTS handshake. When data packet transmission time is comparable to the latency of the RTS/CTS exchange, there is no need to add the extra overhead of RTS/CTS since the cost of data frame collision is comparable to collision of the control frames themselves.

This is illustrated in Figure 3.7 which shows the normalized RTS/CTS throughput gain over IEEE 802.11 DCF's base mode as a function of data packet air time. As expected, for larger data packet sizes, RTS/CTS is more effective (throughput gains above 0). For smaller packets, e.g. 200 bytes, sent at high data rate, e.g. 54 Mbps, the airtime used by data packets is small enough that collision cost is negligible.

One of the current practices to decide when to use RTS/CTS employs a fixed *RTS Threshold, RT* based on packet size. In fact, it is common to set the *RT* to the maximum data packet size which results in never using RTS/CTS, which may be adequate in some scenarios, but not in others as exemplified by the results in Figure 3.7.

A notable practical factor in the performance of RTS/CTS, which is frequently neglected, is that, in multi-rate WLANs, control frames such as ACK, RTS, and CTS are transmitted at a fixed basic rate regardless of the data rate. One of the main reasons is to enable interoperability and to accommodate legacy devices, since all devices in the network must be able to receive these frames. This increases the overhead of the RTS/CTS mechanism in high data rate networks, which is also captured by the results in Figure 3.7 as the signaling rate is kept at 2 Mbps.

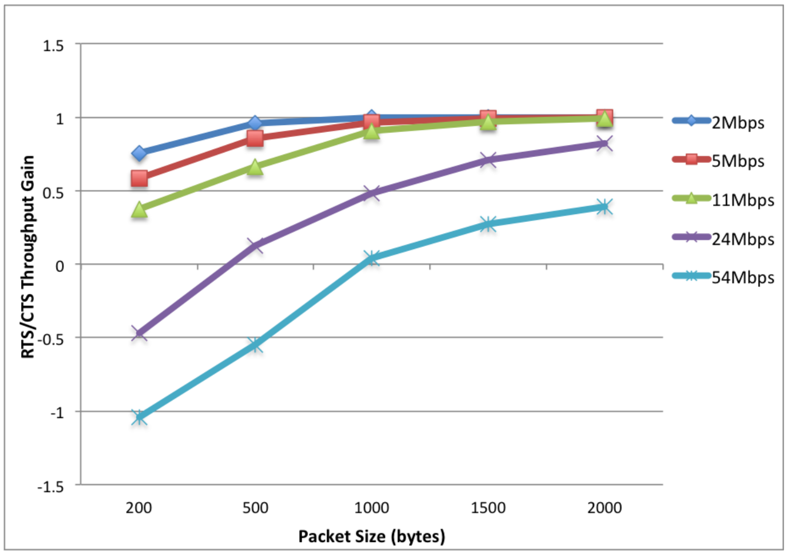


Figure 3.7: RTS/CTS throughput gain as a function of data packet size for different data rates.

3.3.2 Signaling Rate

In this section, we consider network scenarios where signaling can be transmitted at the same rate as data.

Figure 3.8 plots the RTS/CTS throughput gain for different signaling rates (equal to the data rate). It shows positive throughput gains across the board. These results confirm that adjusting *RT* also needs to account for transmission rate of control packets (signaling rate).

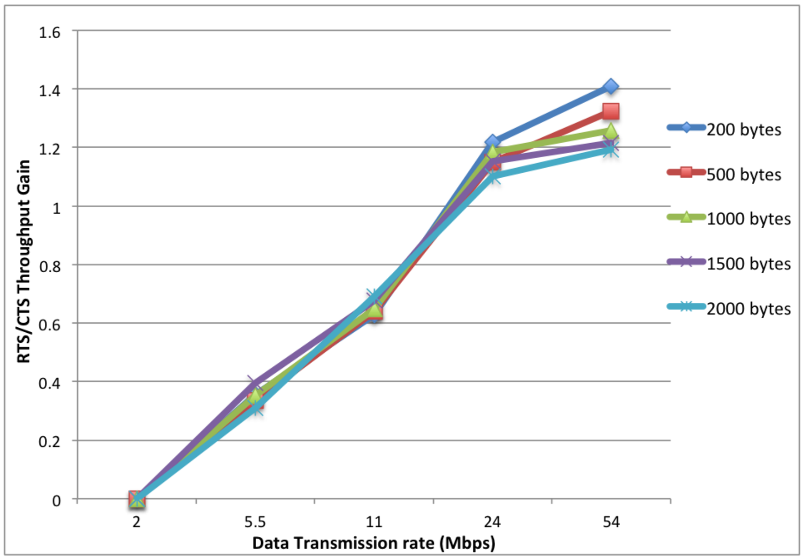


Figure 3.8: RTS/CTS throughput gain as a function of signaling rate for different packet sizes.

3.3.3 Network Contention

Network contention is clearly a critical factor in deciding whether to use RTS/CTS. Recall that the original goal of the RTS/CTS handshake is to avoid collisions, so if collisions are not likely to occur (e.g., low network load), there is no need to use RTS/CTS and incur the additional overhead. In IEEE 802.11 networks, collisions occur either because of the existence of hidden terminals or due

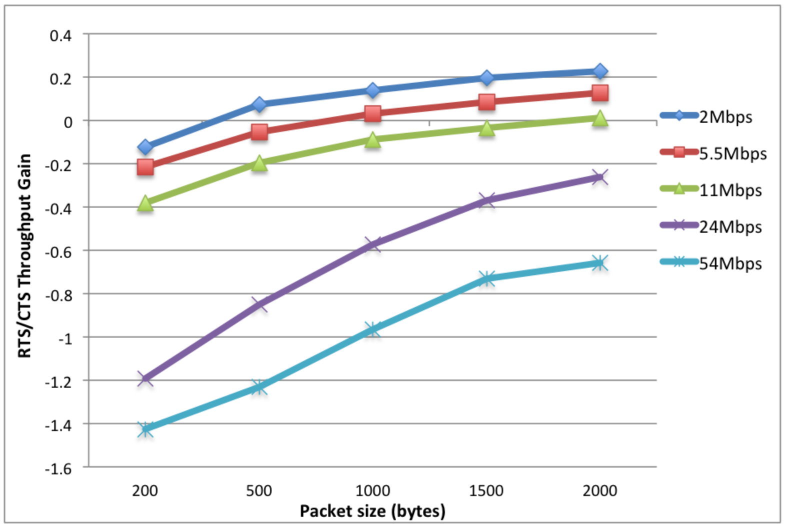


Figure 3.9: RTS/CTS throughput gain as a function of packet size for different data rates when no hidden nodes are present.

to (quasi) simultaneous transmissions (e.g., as a result of back off synchronization).

As described in Section 3.2, there have been a number of proposals to estimate network contention, such as number of hidden terminals [45], mean medium access delay [31], packet delivery ratio [38], number of waiting for CTS [25], to name a few. We measure network contention by calculating the *collision probability*, which we describe in Section 3.4.

We conducted similar sets of experiments as the ones described in Section 3.3.1, except for the fact that we use a topology with no hidden nodes. As shown in Figure 3.9, for most data rates and small- to medium packet sizes, there is no benefit in using RTS/CTS when there is no hidden terminal. These results confirm the importance of accounting for network contention when deciding whether to use RTS/CTS.

3.4 Proposed Approach: Dynamically Switching RTS/CTS On-Off

In this section, we first explain an earlier version of our method which is a simple yet effective algorithm to dynamically enable or disable RTS/CTS [15]. Then, we introduce SACA, Smart Adaptive Collision Avoidance, which is an optimized version of our earlier proposed method.

3.4.1 First Version

Unlike most previous efforts which typically consider individual factors in deciding when to use or not use RTS/CTS, our method considers all factors, i.e., network contention, as well as data and signaling transmission time, also known as *air time*, which is given by Equation 3.2 below.

$$transmission\ time = \frac{data/signaling\ size}{data/signaling\ rate} \tag{3.2}$$

Algorithm 3 describes our approach which essentially evaluates on an ongoing basis the benefit of using RTS/CTS compared to its overhead. If RTS/CTS is deemed beneficial, i.e., it avoids collisions, when *collision cost* is higher than the cost of the RTS/CTS exchange, then RTS/CTS is enabled, otherwise it is disabled. Note that we define *collision cost* as a function of network contention and data transmission time, which we represent by $network\ contention \otimes data\ transmission\ time$ in Algorithm 3. *Data transmission time*, as well as RTS/CTS cost (or *signaling transmission time*) are calculated according to Equation 3.2. Note that data frame size, data transmission rate, signaling transmission rate, and RTS/CTS frame size (which amounts to 34 bytes) are known at the time of transmission. However, current network contention conditions must be estimated on an ongoing basis.

Algorithm 3 Proposal

```
If ($network\ contention \otimes data\ transmission\ time$) $\geq$ $signaling\ transmission\ time$
then enable RTS/CTS
else disable RTS/CTS
```

Network contention depends on the number of competing stations that are simultaneously trying to access the shared communication medium in order to transmit. In this work, we use *collision probability* as an indicator of network contention and measure it by dividing the number of failed receptions at the receiver by the total number of transmissions at the sending node.

As illustrated in Figure 3.10, we divide time into slots, and at the beginning of each slot, there is a short *learning period*, during which RTS/CTS is disabled. *Collision probability* is measured a number of times during the *learning period*. Then, using the SENSE estimator (described in chapter 2) [13], we estimate the collision probability for the remaining of the slot. Based on SENSE's collision probability estimate, our current implementation of Algorithm 3 calculates *network contention* $\otimes$ *data transmission time* as the product between *data transmission time* and collision probability. It then decides whether to turn RTS/CTS on or off for the duration of the slot. Time slot and learning period duration as well as the number of times we calculate collision probability during the learning period are parameters of our approach. In our performance evaluation, we experimented with different values of these parameters and did not observe significant changes in the results.

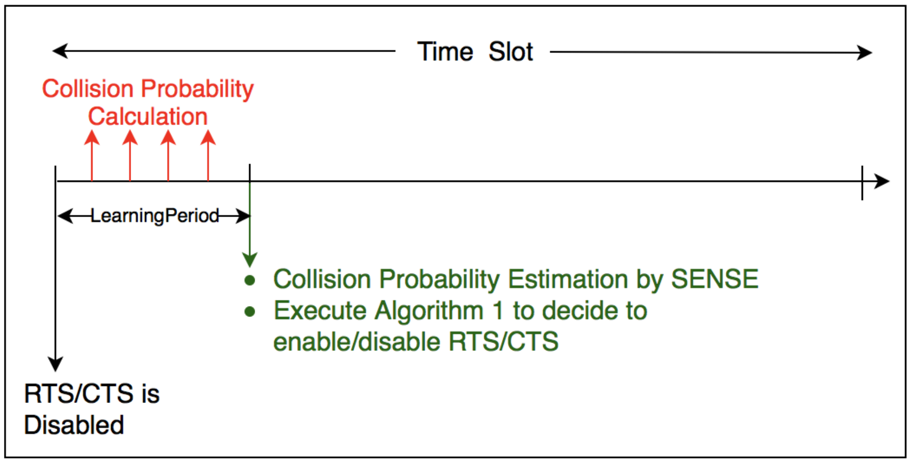

Time Slot
Collision Probability Calculation
LearningPeriod
Collision Probability Estimation by SENSE
Execute Algorithm 1 to decide to enable/disable RTS/CTS
RTS/CTS is Disabled

Figure 3.10: Time slot

3.4.2 SACA

In this section, we describe our Smart Adaptive Collision Avoidance mechanism, or SACA for short, a simple yet efficient technique to dynamically enable or disable IEEE 802.11's collision avoidance in order to automatically adapt to a number of factors such as frame size, transmission rate (for both data and control frames), and network contention. SACA is an optimized version of the algorithm proposed in 3.4.1 and is improved by changing the following items:

- We propose a new approach to evaluate network contention which is carried out at regular periods instead of during a *learning period* at the beginning of a time slot when RTS/CTS is switched off.

- We use a more accurate model to calculate the cost incurred by data- and RTS/CTS collisions. These cost calculations are at the core of SACA since they determine whether RTS/CTS should be turned on or off on a per-frame basis.

- In addition to evaluating SACA in infrastructure-based scenarios, we also conduct experiments to assess SACA's performance in multihop wireless ad-hoc networks, or MANETs.

The main idea behind SACA is to evaluate the cost-performance tradeoff of using RTS/CTS to avoid data collisions. If RTS/CTS is deemed beneficial, i.e., it avoids collisions or reduces collision duration when the cost of a data collision is higher than the cost of the RTS/CTS exchange, then RTS/CTS is enabled; otherwise it is disabled. We discuss how these costs are defined and calculated later in this section. SACA's pseudocode is shown in Algorithm 4.

Algorithm 4 SACA

```
Collision estimation timeout:
    $Collision\_Estimation()$;
Frame transmission:
    If ($Data\_Retransmission\_Cost() \geq RTS/CTS\_Cost()$)
        then  $enable\ RTS/CTS$
        else  $disable\ RTS/CTS$
```

SACA is event-driven and handles two different events, namely: (1) collision estimation timeout and (2) frame transmission. When the collision estimation timer expires, a **collision estimation timeout** event is triggered. The $Collision\_Estimation()$ procedure is then invoked to measure the current collision rate and to compute an estimate of the collision rate for future use.

When a data frame is ready to be transmitted, a **frame transmission** event is triggered and the cost of using and not using RTS/CTS is calculated by $RTS/CTS\_Cost()$ and $Data\_Retransmission\_Cost()$, respectively. Based on how the cost of a data collision compares to the cost of the RTS/CTS exchange, a decision is made to enable or disable RTS/CTS. In the remainder of this section, we describe SACA in more detail.

Adapting to Network Contention

As discussed in Section 3.4.2, network contention can be measured in a variety of ways. In this work, we use *collision rate* as an indicator of network contention and measure it by dividing the number of failed transmissions, i.e., the number of unacknowledged frames at the transmitter by the total number of transmissions as shown in Equation 3.3. Other ways to evaluate network contention in a node's neighborhood include the node's MAC queue length, mean time to access the medium, etc. In this work, because we are using a network simulator, more specifically *ns-3* [8], we use the expression in Equation 3.3 because both its nominator ($number\ of\ failed\ transmissions$) and denominator ($total\ number\ of\ transmissions$) are readily available in *ns-3* and provide fairly accurate collision measurement since losses that are not due to collisions are quite rare in the experimental scenarios we use.

$$collision\ rate = \frac{number\ of\ failed\ transmissions}{total\ number\ of\ transmissions} \tag{3.3}$$

In order to continuously adapt to current network contention conditions, SACA measures collisions regularly. This is accomplished by the $Collision\_Estimation()$ procedure, which is invoked periodically triggered by the **collision estimation timeout** event. $Collision\_Estimation()$, whose pseudo-code is shown in Algorithm 5, measures current data- and RTS/CTS collision rates. Based on current collision measurements, $Collision\_Estimation()$ also estimates near-future collision rates using the SENSE estimator [13] described in Chapter 2.

Measuring Network Contention

The question of how to measure network contention is central to our approach. While we chose to use collision rate as an indicator of network contention, there

Algorithm 5 Collision_Estimation

```
Initialization:
        $Data\ collision\ rate = 0$
        $RTS/CTS\ collision\ rate = 0$
        $Data\_collision\_estimation = 0$
        $RTS\_collision\_estimation = 0$
Collision rate measurment:
        $Compute\ Data\ collision\ rate$
        $Compute\ RTS/CTS\ collision\ rate$
Collision rate estimation:
        $SENSE\ computes\ Data\_collision\_estimation$
        $SENSE\ computes\ RTS/CTS\_collision\_estimation$
```

have been a number of proposals to estimate network contention, such as number of hidden terminals [45], mean medium access delay [31], frame delivery ratio [38], number of RTSs waiting for a CTS [25], to name a few. As part of our future work, we plan to evaluate how different approaches to measuring network contention impact the performance of SACA.

Another question that needs to be addressed is how often collision should be measured. We discuss some alternatives and their pros and cons below.

In SACA's preliminary version (described in Section 3.4.1), time is divided into slots and during a *learning period* at the beginning of each slot, RTS/CTS is disabled and data collision rate is measured. Based on these measurements, SENSE estimates the data collision rate for the rest of the slot. The advantage of using the learning period to measure collision rate is less overhead overall. However, since collision rate is calculated only during learning periods, information about network contention may be out of date by the time collision rate is used to decide whether to switch RTS/CTS on/off. Another problem is that, if contention is high, turning off RTS/CTS for the learning period may result in degraded performance.

Additionally, sudden changes in network contention during the time slot will not be captured until the next learning period. Considering all the pros and cons discussed above, in SACA's current version, which is described in this paper, we decided not to include a learning period and measure the contention every t seconds.

There is a clear trade-off in setting the value of t. Using a smaller t may allow SACA to better capture variations in contention conditions; however, it will incur higher overhead by invoking $Collision\_Estimation()$ more often. Another drawback of setting t too small is to capture short-lived variations in contention conditions. On the other hand, setting t too large results in less computation at the expense of running the risk of not adequately capturing network contention dynamics. t also needs to be set large enough such that sufficient frame transmissions can be observed. Therefore, setting the value of t should also consider the underlying link speed and frame size. In order to set the value of t for our experimental evaluation of SACA, we ran several preliminary experiments with different values of t, namely 0.5, 1, 2, and 3 seconds and did not notice any significant differences in the results obtained. In the results reported in Section 4.4, we use t equal to 1 second.

Switching RTS/CTS On-Off

As shown in Algorithm 4, when a frame is ready for transmission, based on the collision estimation from $Collision\_Estimation()$ and the information from the frame itself, i.e., frame size and transmission rate, the cost of re-transmitting data and the cost of the RTS/CTS handshake are calculated by $Data\_Retransmission\_Cost()$ and $RTS/CTS\_Cost()$, respectively. RTS/CTS is enabled for that specific frame if the data retransmission cost is higher or equal to the RTS/CTS handshake cost; RTS/CTS is disabled otherwise. Costs' calculations are described in the remaining of this section.

Data Collision versus RTS/CTS Handshake

In an ideal scenario when there is no contention, a node can avoid RTS/CTS handshake since there is no risk of collisions. But in congested environments, packets transmitted by different nodes which start to transmit at the same time can collide. Also, frames transmitted by hidden terminals may collide at the receiver even if the sender's transmission does not start exactly at the same time. In these situations, a node can decide to use RTS/CTS to reserve the channle and avoid data frame collisions. However, performing the RTS/CTS handshake ahead of data transmission will incur additional overhead. In order to determine if enabling RTS/CTS is beneficial, SACA compares the cost of the RTS/CTS exchange (calculated by $RTS/CTS\_Cost()$) against the cost of data retransmission in case of collision (computed by $Data\_Retransmission\_Cost()$). Basically, for every data frame transmission, the sender calculates the cost of retransmitting the frame in case of collision and compares it against the RTS/CTS overhead. The idea is to make this decision based on current network conditions as well as frame size and transmission rate.

Cost of data frame collision: The cost of a data frame collision, i.e., the cost to retransmit a frame until successfully received is calculated as:

$$\begin{aligned}&Data\ Retransmission\ Cost = \\ &\qquad overhead\ of\ one\ frame\ retransmission \times \\ &\qquad\qquad average\ number\ of\ retransmissions\end{aligned} \tag{3.4}$$

To calculate the overhead of one frame retransmission, we consider IEEE 802.11's *base mode* which uses only CSMA (no RTS/CTS). As shown in Figure 3.11, which shows the transmission sequence of base mode according to the IEEE 802.11 standard, either the data frame will be transmitted successfully and will not incur any retransmission overhead, or the frame will collide which will require retransmission. Figure 3.11 illustrates a scenario with two hidden terminals, $Sender1$ and $Sender2$, in which Sender 1's first transmission to $Receiver$ is successful, but the second one collides with $Sender2$'s transmission. [1] When a collision happens, the total amount of time spent retransmitting the frame includes: the $DIFS$, backoff time (BO), retransmission time, plus the ACK timeout which is $SIFS$ plus ACK time. Therefore, the overhead of one frame retransmission is:

$$DIFS + BO + Data + SIFS + ACK \tag{3.5}$$

Meanwhile, the average number of retransmissions is calculated by:

$$\sum_{k=0}^{inf} k * (P_{DC})^k * (1 - P_{DC}) = P_{DC}/(1 - P_{DC}) \tag{3.6}$$

where P_{DC} is the probability of data packet collision.

From Equations 3.4, 3.5, and 3.6, we derive Algorithm 6:

Algorithm 6 Data_Retransmission_Cost()

```
$(DIFS + BO + Data + SIFS + Ack) \times (\frac{P_{DC}}{1-P_{DC}})$
```

[1]Note that, as mentioned earlier, there is the possibility of collisions with non-hidden nodes as well which is not demonstrated in this figure.

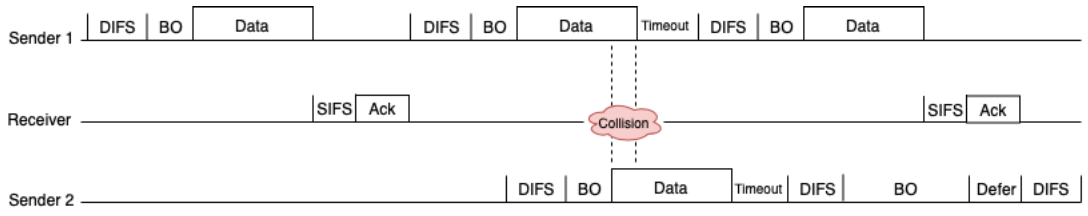


Figure 3.11: Successful and unsuccessful data transmission in IEEE 802.11 base mode

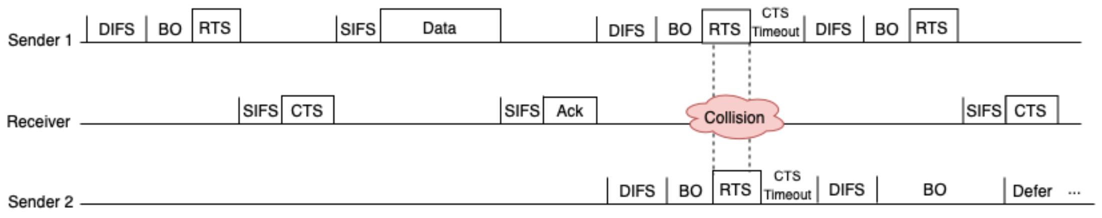


Figure 3.12: Successful and unsuccessful data transmission in IEEE 802.11 collision avoidance mode

Cost of RTS/CTS handshake: To calculate the RTS/CTS overhead cost, we consider IEEE 802.11's operation in congestion avoidance mode as illustrated in Figure 3.12, which shows the transmission sequence of congestion avoidance mode according to the IEEE 802.11 standard. In this mode, either the RTS/CTS will be transmitted successfully and then will be followed by data and ACK frames or there is a possibility that the RTS frame collides with a frame from another node. In this figure, we show the case of RTS collision from two hidden nodes, $Sender1$ and $Sender2$, but RTS collisions can also happen between two non-hidden nodes if they start transmission at the same time. When RTS and CTS frames are successfully transmitted, the overhead incurred includes RTS transmission time (RTS), $SIFS$, CTS transmission time (CTS) and another $SIFS$. However, when an RTS collides, the overhead includes $DIFS$, BO, RTS, and CTS timeout which is $SIFS$ plus CTS. RTS collision cost is much less than data frame collision cost since RTS and CTS frames are typically much smaller than data frames. Similarly to previous work (e.g., [26]), we use the number of CTS timeouts as an indicator of the number of RTS collisions. We denote by P_{RC} the probability of an RTS collision. In the case of RTS/CTS transmission:

$$\begin{aligned}
&RTS/CTS\ Cost = \\
&\qquad RTS/CTS\ successful\ transmission\ overhead + \\
&\qquad overhead\ of\ RTS/CTS\ retransmission \times \\
&\qquad average\ number\ of\ RTS/CTS\ retransmissions
\end{aligned} \tag{3.7}$$

where the RTS/CTS successful transmission overhead is:

$$RTS + CTS + 2 \times SIFS \tag{3.8}$$

The overhead of RTS/CTS retransmission is:

$$DIFS + BO + RTS + SIFS + CTS \tag{3.9}$$

and the average number of RTS/CTS retransmissions is:

$$\sum_{k=0}^{inf} k * (P_{RC})^k * (1 - P_{RC}) = P_{RC}/(1 - P_{RC}) \tag{3.10}$$

From equations 3.8, 3.9 and 3.10, we arrive to Algorithm 7 for overall RTS/CTS cost:

Algorithm 7 RTS/CTS_Cost()

```
$(RTS + CTS + 2 \times SIFS)+$
$[(DIFS + BO + RTS + SIFS + CTS) \times (\frac{P_{RC}}{1-P_{RC}})]$
```

SACA's Implementation and Overhead

Here, we discuss the implementation and deployment of SACA in real networks as well as SACA's overhead.

Simplicity and low overhead were among SACA's main design goals. Additionally, SACA was designed so that each node can run independently of other nodes and only needs information about local conditions, i.e., contention in its neighborhood. In other words, each node evaluates network contention it has been experiencing in recent past and, for each frame to be transmitted, decides to enable or disable RTS/CTS regardless of other nodes.

SACA can be implemented as a separate module that is invoked by 802.11 on a per-frame basis. SACA uses collision ratio as an indicator of network contention in order to dynamically decide whether RTS/CTS should be used or not. To do that, each node keeps track of the total number of successful- and unsuccessful transmissions (see Equation 3.3), both of which are usually readily available in real systems, along with frame size and transmission rate (available from the frame's header).

3.5 Experimental Methodology

We evaluate SACA through extensive simulations using a variety of infrastructure-based as well as ad-hoc scenarios and show that it can automatically adjust to changes in network contention while accounting for airtime to decide whether to enable or disable the RTS/CTS handshake. In this section, we describe our experimental setup and performance metrics. Our results are presented in Section 3.6.

We run experiments using the *ns-3* network simulator [8] and its implementation of the IEEE 802.11n. We use *ns-3*'s *Matrix Propagation Loss Model* and set the propagation loss between each pair of nodes to make them hidden or not hidden from each other. According to *ns-3*'s channel model, if the propagation loss between two nodes is greater than $200dB$, they are considered hidden from one another. For example, if we set the propagation loss between nodes A and B to a very high value, then A becomes hidden from B, and vice-versa. Table 3.2 lists simulation parameters and their values used in all our experiments.

Table 3.2: Simulation parameters

Simulator	*ns-3*
Area	500x500 m^2
Total number of nodes	50
Routing protocol	AODV
Traffic type	CBR
DIFS and SIFS	50 and 10 μs
802.11 version	802.11 n

3.5.1 Traffic Traces

We use three different traffic traces to drive our experiments, namely a synthetic data trace as well as traces collected in real networks. In the synthetic data trace, we vary frame size and the number of senders, and consequently collision rate, every 5 seconds (total simulation time is 50 seconds) to evaluate how closely SACA is able to track network contention fluctuations. Table 3.3 summarizes the synthetic trace we use showing the sequence of frame sizes and number of senders as they vary every 5 seconds. In all cases, sender- and receiver nodes are selected randomly from the set of participating nodes.

We also drive SACA using real traffic traces captured in a public hot spot and a company campus using a wireless sniffer. Data rates and frame sizes provided in the Radiotap Header are used to calculate a frame's airtime. Tables 3.4 and 3.5 summarize the real traffic traces used in our experiments. For the hot spot trace, we captured 10 flows between users and the access point (AP) for 20 minutes and feed the flows to the *ns-3* simulator. In the 10-sender scenario, each of the 10 flows is assigned to a single sender, while in the 30- and 50-sender scenario, each flow is assigned to 3- and 5- senders, respectively. Each flow has a slightly different start time to avoid excessive contention at the start of the experiments. For the company campus trace, 5 individual flows were captured and each flow was assigned to 1, 2, 6, and 10 senders in the 5-, 10-, 30-, and 50-sender scenarios respectively.

Table 3.3: Synthetic traffic trace

Time (s)	frame Size	Number of senders
0 to 5	1500	5
5 to 10	500	8
10 to 15	2000	14
15 to 20	200	20
20 to 25	1000	24
25 to 30	2000	30
30 to 35	500	35
35 to 40	200	38
40 to 45	1500	43
45 to 50	500	45

3.5.2 Performance Metrics

We evaluate SACA by comparing its performance against IEEE 802.11's base mode and congestion avoidance mode with different RT values. As performance metrics, we use average throughput and average throughput improvement, which are calculated as follows: average throughput is the ratio between the number of

Table 3.4: Hot-spot trace

Location	Coffee shop
Time	Around noon
Number of flows	10
Duration	20 minutes
Frame size range	34-2150 bytes
802.11 version	802.11n
Data rate range	6-300 Mbps

Table 3.5: Company campus network trace

Location	Company campus
Number of flows	5
Duration	30 minutes
Frame size range	34-11000 bytes
802.11 version	802.11n
Data rate range	6-1300 Mbps

received packets to the number of transmitted packets, averaged over all nodes; average throughput improvement is calculated as the difference in throughput between SACA and the other approach divided by the other approach's throughput. For infrastructure-based scenarios, we also examine the contention an AP experiences when all nodes are hidden from each other as well as when all nodes are exposed to one another, in order to evaluate how well SACA can estimate and adapt to contention by dynamically enabling/disabling RTS/CTS.

3.6 Results

Results from our performance evaluation study comparing SACA against IEEE 802.11 are presented in two parts, namely: infrastructure-based scenario results and ad-hoc scenario results. For these experiments, 50 nodes are placed randomly in a 500x500 meters area in the transmission range of an Access Point (AP). Table 3.2 summarizes simulation parameters and their values used in both infrastructure-based and ad-hoc experiments.

3.6.1 Infrastructure-Based Scenarios

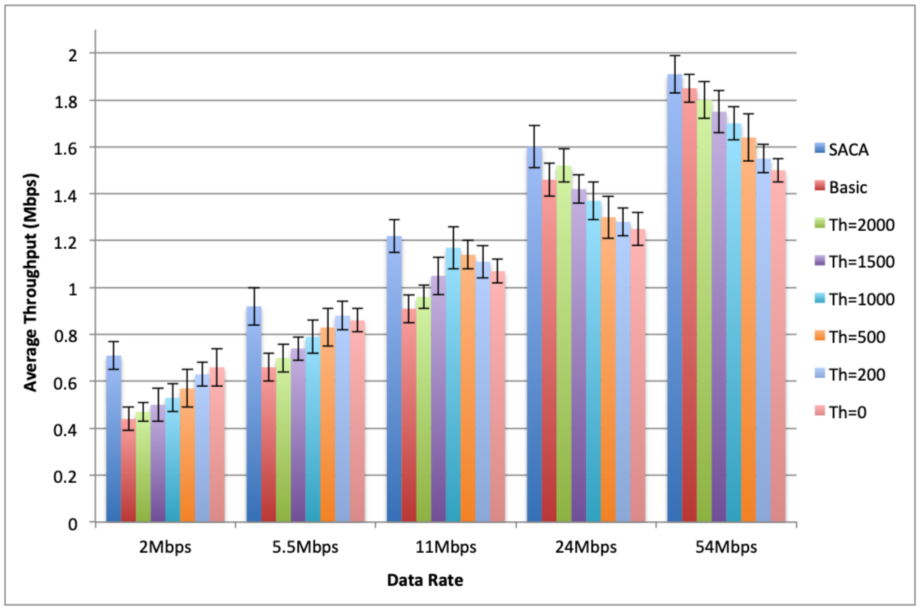


Figure 3.13: Average throughput with the synthetic trace in infrastructure-based scenario. 95% confidence intervals are shown.

Average Throughput

In these experiments, we compare SACA's average throughput against IEEE 802.11's DCF base mode (i.e., no RTS/CTS), IEEE 802.11's DCF congestion

Table 3.6: SACA's slot-by-slot behavior using the synthetic trace with different data rates in infrastructure-based scenario

Time (s)	Frame Size (bytes)	Collision	54Mbps	24Mbps	11Mbps	5.5Mbps	2Mbps
0 to 5	1500	5%	Basic	Basic	Basic	Basic	Basic
5 to 10	500	9%	Basic	Basic	Basic	Basic	RTS/CTS
10 to 15	2000	26%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS
15 to 20	200	38%	Basic	Basic	Basic	Basic	RTS/CTS
20 to 25	1000	47%	Basic	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
25 to 30	2000	59%	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
30 to 35	500	66%	Basic	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
35 to 40	200	71%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS
40 to 45	1500	78%	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
45 to 50	500	83%	Basic	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS

avoidance mode (i.e., RTS/CTS always enabled), and when using statically configured values for the RTS Threshold *RT*, namely 200-, 500-, 1000-, 1500-, and 2000 bytes. In all experiments (unless otherwise specified), half of the nodes are hidden from other nodes. For example, in the 10-sender scenario, 5 senders are hidden, i.e., they can see the AP but they cannot see any other sender. The other senders, which are not hidden, can see the AP as well as the other senders. Each experiment is run for 10 times using different seeds and nodes are selected to be hidden or not hidden randomly.

Synthetic trace: To show how SACA adjusts to network dynamics, we periodically changed the frame size and number of senders (see Table 3.3). Note that by varying the number of simultaneous transmitters, we vary network contention and consequently collision rate. In order to observe the impact of data rates on performance, we ran this experiment with data rates of 54-, 24-, 11-, 5.5-, or 2Mbps, while signaling transmission rate is kept at 2Mbps.

Figure 3.13 shows SACA's average throughput compared against IEEE 802.11 DCF's base mode ("Basic") and IEEE 802.11's DCF congestion avoidance mode with different *RT* values. We observe that SACA outperforms all other approaches

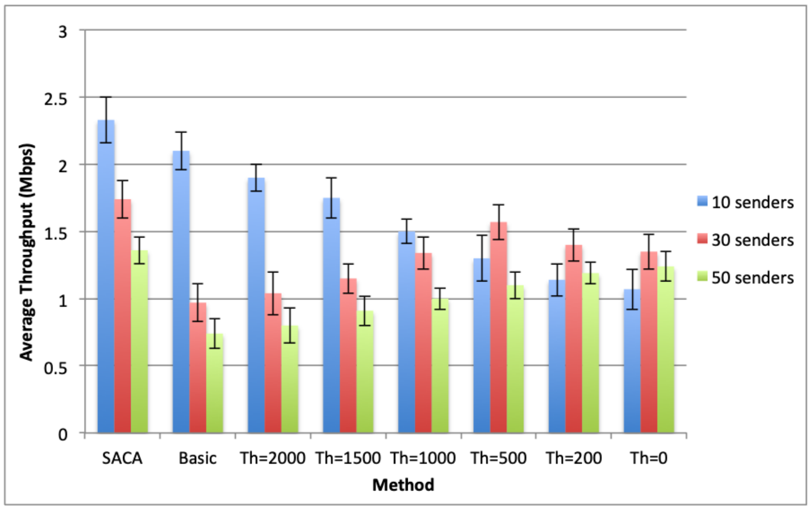


Figure 3.14: Average throughput using hot-spot trace in infrastructure-based scenario. 95% confidence intervals are shown.

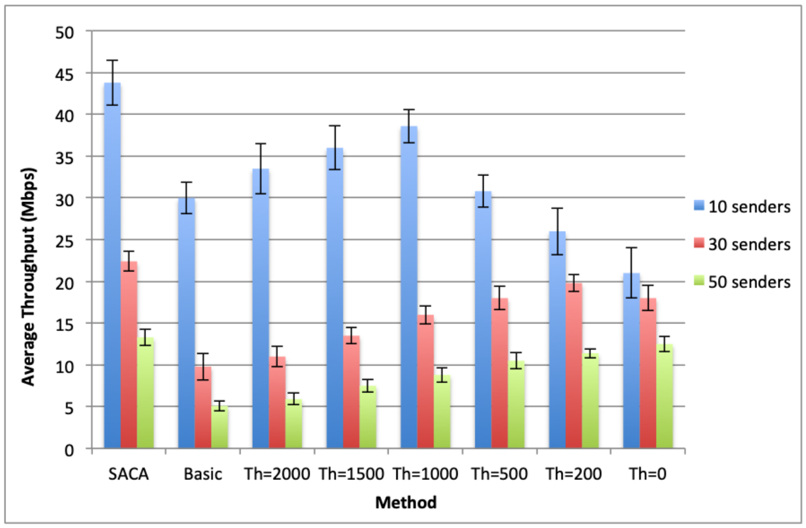


Figure 3.15: Average throughput using company campus network trace in infrastructure-based scenario. 95% confidence intervals are shown.

for all data rates. As expected, for lower data rates, "Basic" has the lowest performance because frame transmission takes longer and therefore collision rate is higher. At lower data rates, RTS/CTS enabled with lower RT values provides better performance. As the data rate increases, performance of $Th = 0$ (i.e., RTS/CTS enabled all the time) improves and eventually outperforms "Basic".

For example, for 24 Mbps, RTS/CTS with mid-range *RT* values provide better performance than both "Basic" and RTS/CTS with larger *RT* values. The reason is that throughput improves when larger frames are "protected" by RTS/CTS at this data rate while, for smaller frames, throughput is higher without RTS/CTS. Our proposed algorithm performs consistently well because it can dynamically decide when to enable or disable RTS/CTS based on current conditions, i.e., frame size, transmission rate, and contention.

Table 3.6 shows how one of the senders uses RTS/CTS during the experiment as contention and frame size change every 5 seconds. These results confirm that, in addition to frame size and network contention, data transmission rate plays an important role in determining whether RTS/CTS should be used or not. For example, in seconds 10 to 15, where frames are 2000 bytes and collision rate is 26%, when data transmission rate is 54- and 24 Mbps, RTS/CTS is disabled. However, at 11-, 5.5-, and 2Mbps, RTS/CTS is used. This is because, for lower data rates, using RTS/CTS is more advantageous.

Hot-spot trace: We ran similar experiments using the hot-spot trace with 10-, 30-, and 50 senders using *ns-3*'s IEEE 802.11n. Since there are 10 individual flows in our trace, each flow is assigned to 1-, 3-, and 5 senders in the 10-, 30-, and 50-sender scenarios, respectively, with each flow starting at slightly different start times. Similarly to the synthetic trace experiments, we compare SACA's average throughput against IEEE 802.11's base mode (no RTS/CTS), as well as statically configured *RT* values of 0 (RTS/CTS always enabled), 200-, 500-, 1000-, 1500-, and 2000 bytes. Figure 3.14 show SACA's average throughput in infrastructure-based scenario with 10-, 30-, and 50 senders.

In the 10-sender scenario, since there is less contention, using IEEE 802.11's base mode or higher *RT* values is more beneficial. With 30 senders, which results in higher contention, using lower thresholds, e.g., 200- and 500 bytes, yields better performance. In the 50-sender scenario, RTS/CTS should be used all the time because of the high contention. In all cases, SACA outperforms all other methods because of its ability to automatically adjust to network contention and airtime. Note that, while in the 10-sender experiment, "Basic" yields similar throughput when compared to SACA, in the 50-sender scenario, "Basic" is the worst performer. In other words, SACA outperforms the best performer among the static methods in all cases.

Company campus trace: Similarly to the hot-spot trace, data rates provided in the Radiotap Header are used to calculate airtime. We ran experiments with 10-, 30-, and 50 senders by assigning each captured flow to 2-, 6- and 10 senders, respectively. Compared to the hot-spot trace, the average frame size and data rate are considerably higher. As shown in Figure 3.15, SACA outperforms all other methods in all scenarios, which is consistent with the results observed in the hot-spot experiments. This is due to SACA's ability to dynamically adjust to frame size, transmission rate, and network contention. For instance, even though contention is not high in the 10-sender scenario, since the ratio of frame sizes to the data rates is larger on average, enabling RTS/CTS yields higher average throughput when compared to Basic. As the number of senders increases, the optimal *RT* value decreases. So in the 50-sender scenario, RTS/CTS should be used all the time.

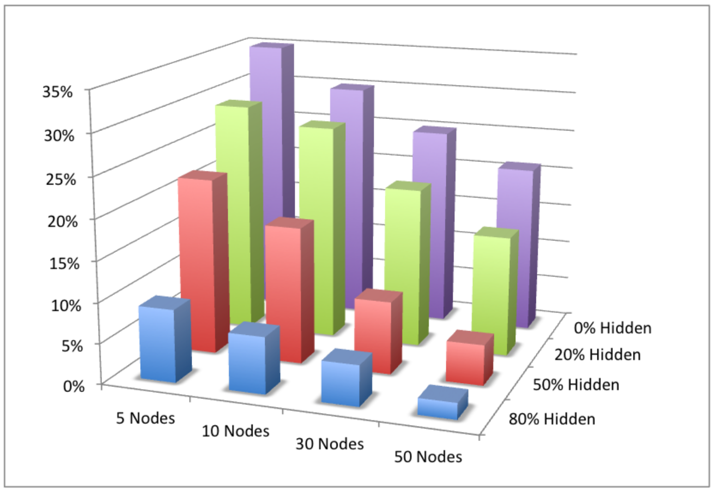
SACA's Throughput Improvement

To further explore SACA's performance, we ran experiments varying not only the number of senders but also the percentage of hidden terminals and measure SACA's throughput improvement.

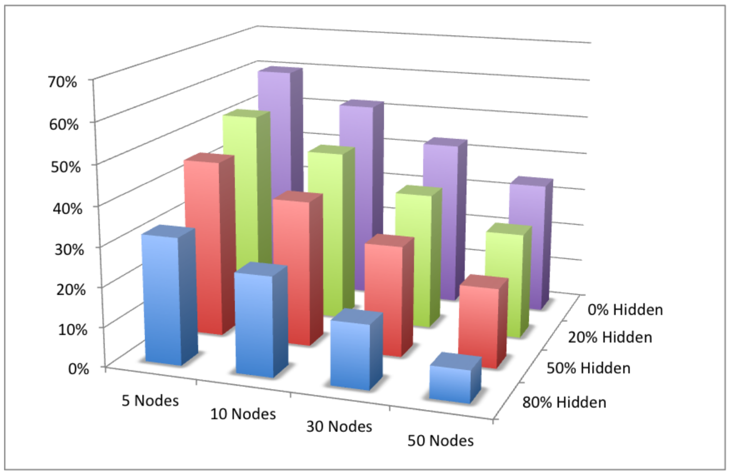
Synthetic trace: The set of experiments driven by the synthetic trace used two different data rates, lower and higher, namely, 2Mbps and 54Mbps. Figure 3.16(a) and Figure 3.16(b) show SACA's throughput improvement over IEEE 802.11 DCF's collision avoidance mode with RTS/CTS always on for 2- and 54Mbps, respectively. As expected, when the number of senders and hidden nodes increase, the benefits of using RTS/CTS all the time increase. However, SACA is still performing better in all cases. For lower data rates (Figure 3.16(a)), SACA's improvement is not as pronounced compared to higher data rate. The reason is that, at higher data rates (Figure 3.16(b)), the cost of using RTS/CTS all the time is relatively more expensive since control frames are sent at lower data rate (i.e., 2Mbps).

We also evaluate SACA's throughput improvement over IEEE 802.11 DCF's base mode (i.e., no RTS/CTS) for data rates of 2- and 54Mbps. is getting more by increasing the number of nodes and hidden nodes. Figure 3.17(a) and Figure 3.17(b) show almost the exact inverse behavior when compared to Figures 3.16(a) and 3.16(b). In other words, SACA's throughput improvement is more accentuated at lower data rates when compared to higher data rates. This is because SACA, in some cases, enables RTS/CTS and RTS/CTS' cost is lower at lower data rates.

Hot-spot trace: We ran similar experiments using the hot-spot trace and evaluated SACA's improvement over IEEE 802.11 DCF's congestion avoidance mode (RTS/CTS on) and IEEE 802.11 DCF's base mode (RTS/CTS off). From Figure 3.18(a), we observe that SACA performs much better than IEEE 802.11 DCF's congestion avoidance mode when chance of collision is low. However, by increasing the number of senders and hidden nodes, SACA's throughput improvement is less pronounced. Figure 3.18(b) shows SACA's throughput gain over IEEE 802.11 DCF's base mode (RTS/CTS off). We observe that as network contention and number of hidden nodes increase, so does SACA's average throughput improvement



(a) 2 Mbps

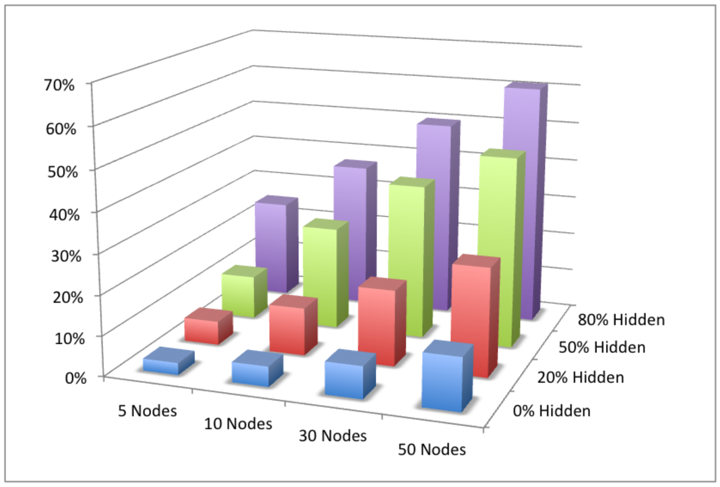


(b) 54 Mbps

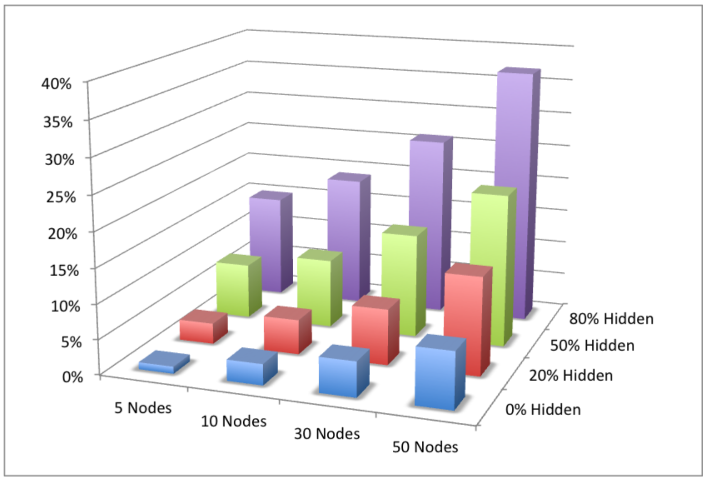
Figure 3.16: SACA's average throughput improvement compared to IEEE 802.11 DCF's congestion avoidance mode (RTS/CTS ON) for synthetic trace

over RTS/CTS off. The reason is that SACA turns on the RTS/CTS dynamically when contention is high, which decreases the collision rate and, as a result, improves overall throughput performance. In all cases, SACA outperforms both IEEE 802.11 DCF's base- and congestion avoidance modes.

Company campus network trace: We observe similar behavior for the company campus network trace. As shown in Figure 3.19(a), when there is less



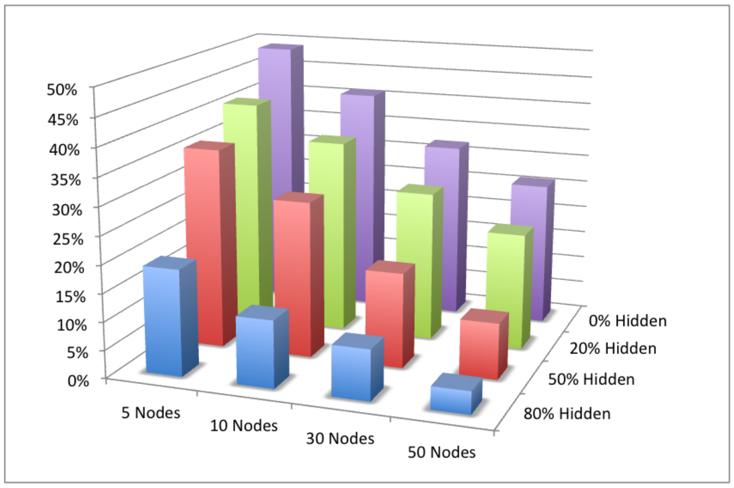
(a) 2 Mbps



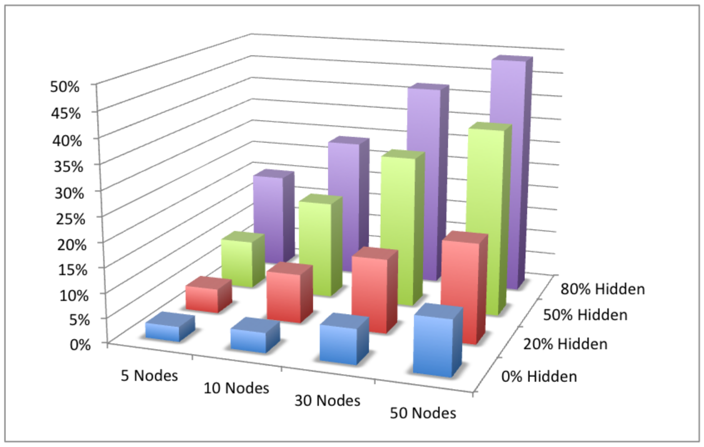
(b) 54 Mbps

Figure 3.17: SACA's average throughput improvement compared to IEEE 802.11 DCF's congestion avoidance mode (RTS/CTS OFF) for synthetic trace

contention and no hidden nodes, there is no need to use RTS/CTS. Therefore, SACA disables RTS/CTS and, as a result, throughput goes up compared to having RTS/CTS on all the time. Although the performance of both techniques are closer in higher contention scenarios, SACA still performs better since it switches RTS/CTS off considering airtime in addition to contention. From Figure 3.19(b), we observe that, since RTS/CTS is always off, in scenarios that exhibit higher contention and number of hidden nodes, SACA's throughput improvement is more



(a) RTS/CTS ON



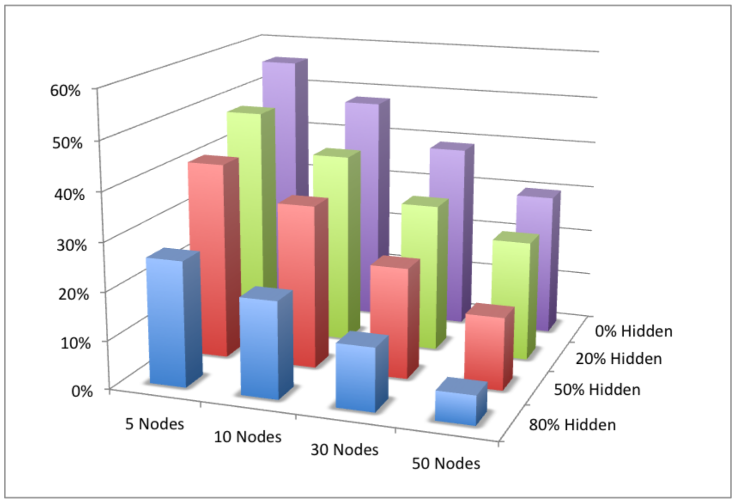
(b) RTS/CTS OFF

Figure 3.18: SACA's average throughput improvement for hot-spot trace

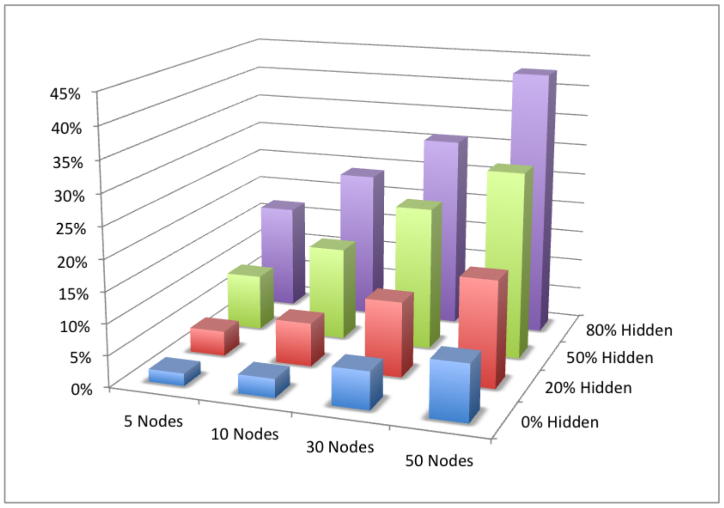
pronounced, while SACA's throughput improvement decreases in scenarios with lower contention and lower number of hidden nodes.

SACA's Network Contention Adaptation

In this section, we examine SACA's ability to adapt to network contention in more detail. More specifically, we investigate how well SACA estimates contention based on its collision rate measurements and then how it uses that information to enable/disable RTS/CTS.



(a) RTS/CTS ON



(b) RTS/CTS OFF

Figure 3.19: SACA's average throughput improvement for company campus trace

To do that, we collect collision rate measurements as well as SACA's collision rate estimates, which are calculated using the SENSE algorithm. We collect data at an AP with 20 senders transmitting according to the public hot-spot and company network datasets. Collision rates were collected with RTS/CTS disabled in order to measure "raw" network contention.

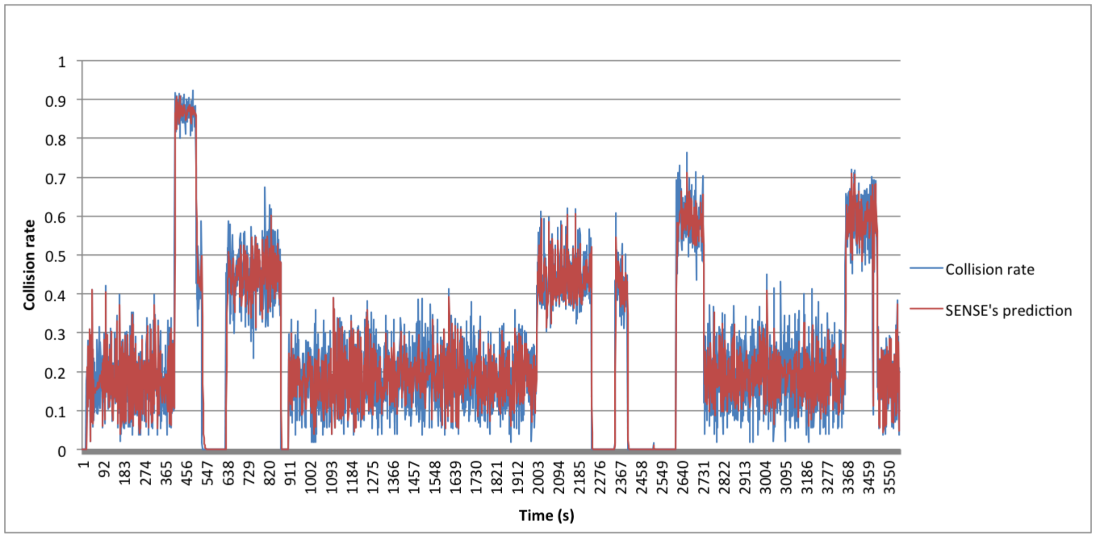
Based on its collision rate estimates, SACA decides whether to enable/disable

RTS/CTS. Then, we also show the AP's RTS/CTS usage time series, where "1" indicates that RTS/CTS is enabled and and "0" that RTS/CTS is disabled. We run experiments for two scenarios, namely when all nodes are hidden from each other when all of them are exposed.

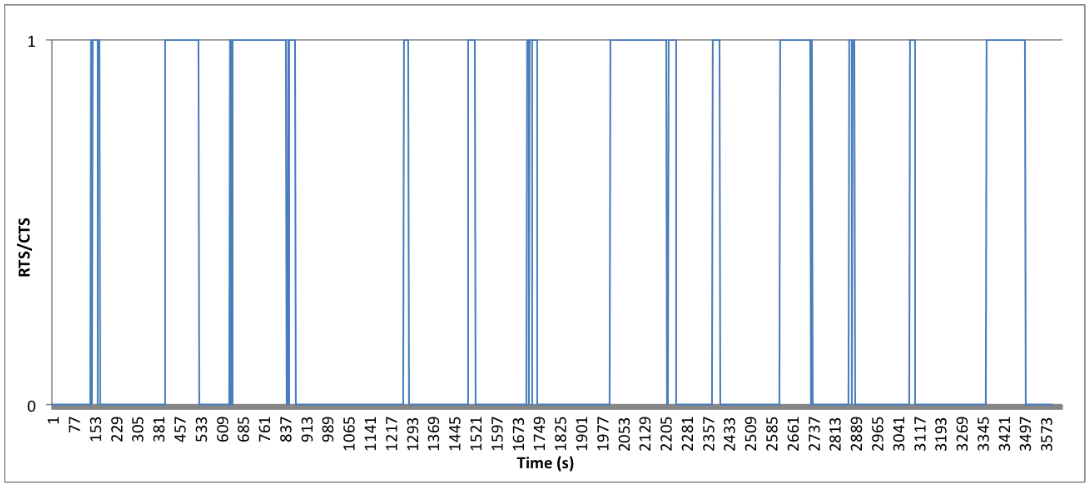
Hot-spot trace: Figure 3.20(a) shows SACA's collision rate measurements and collision rate estimates at the AP when all senders are hidden from each other. As expected, in the presence of hidden nodes, collision rates can be quite high (in this case, as high as 90%). We also observe that SACA can track collision rates closely. Figure 3.20(b) shows the time when RTS/CTS is enabled and disabled. Since in this scenario collision rates are relatively high, RTS/CTS is used frequently (around 30% of the time).

Figure 3.21(a) shows collision rate measurements and SACA's collision rate prediction at the AP when nodes are exposed. As expected, since nodes can see each other, the chance of collision is lower. Therefore, collision rates are considerably lower when compared to the ones plotted in Figure 3.20(a). As a result, Figure 3.21(b) shows that RTS/CTS is rarely enabled (only for around 3% of the time).

Company campus network trace: For the company campus trace, Figure 3.22(a) shows the collision rate and SACA's collision rate estimates at the AP when all nodes are hidden from each other. In this dataset, collision rates are generally higher than in the hot-spot trace. Again, we observe that SACA's estimates are able to follow real collision rate measurements quite closely. Because of the higher average collision rate, as shown in Figure 3.22(b), RTS/CTS usage in the company network trace is around 40% of the time and is higher than in the hot-spot experiment.



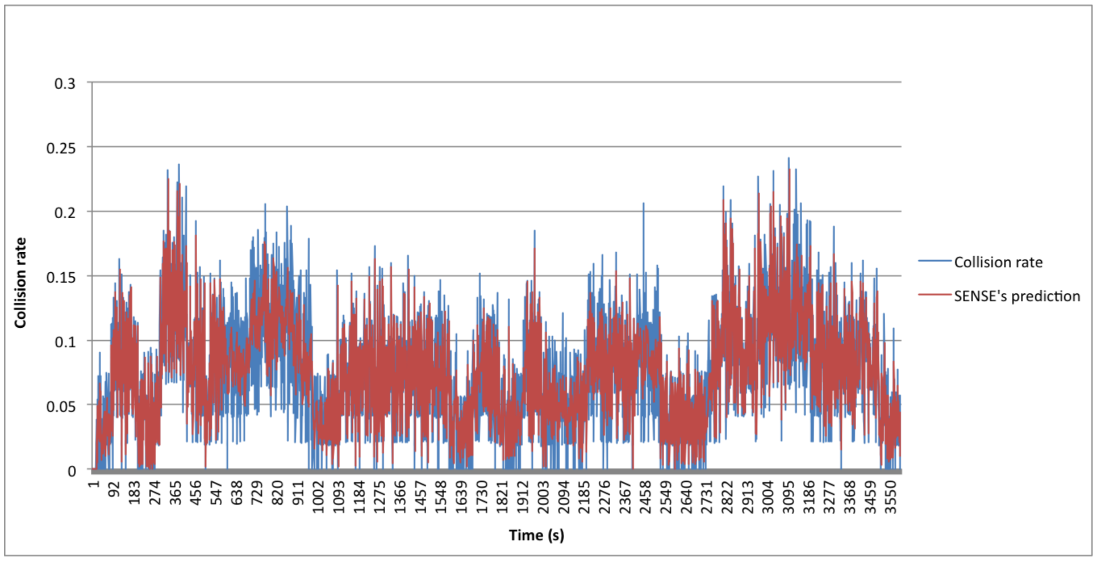
(a) Collision rate and SENSE's prediction



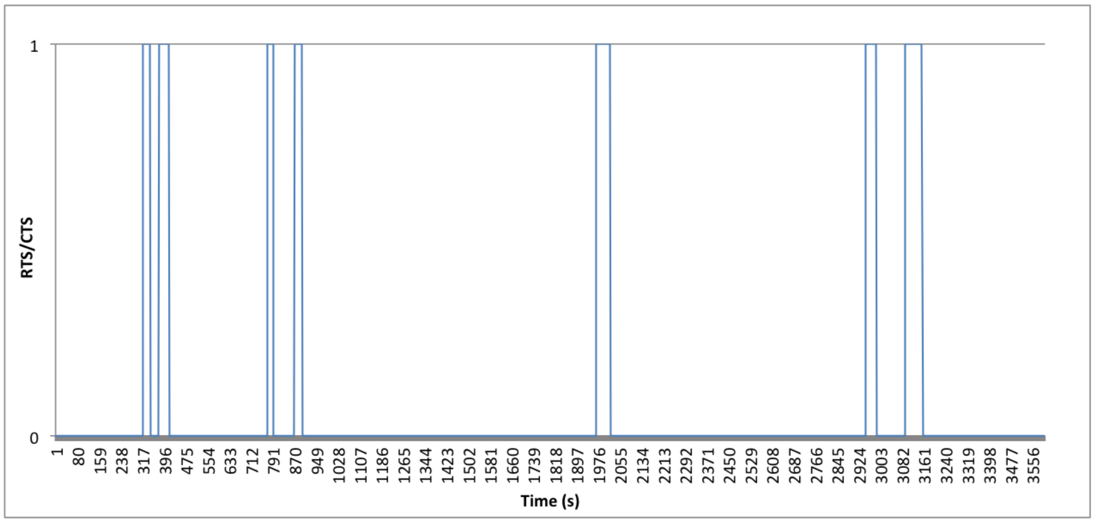
(b) RTS/CTS usage

Figure 3.20: Collision rate, SENSE's prediction and RTS/CTS usage for AP in hot-spot trace when all nodes are hidden from one another

In Figures 3.23(a) and 3.23(b), respectively, we show the collision rate measurements, SACA's collision rate predictions, and RTS/CTS usage for company campus trace where all nodes can see each other. Similarly to what was observed in the hot-spot experiment and as expected, collision rates are considerably lower than the case when nodes are hidden (Figure 3.22(a)) and, as a result, RTS/CTS is used less (around 8% of the time).



(a) Collision rate and SENSE's prediction

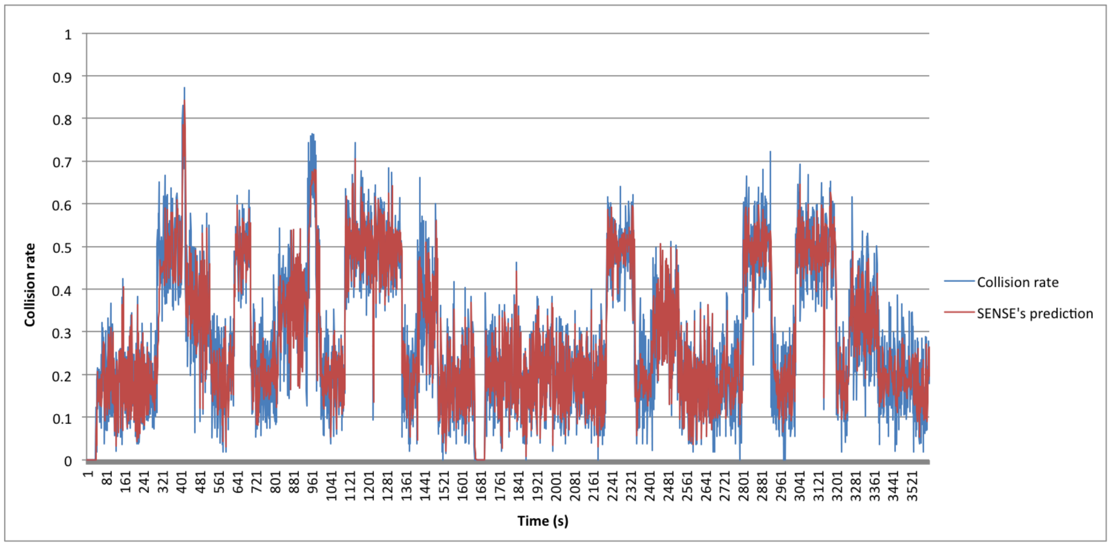


(b) RTS/CTS usage

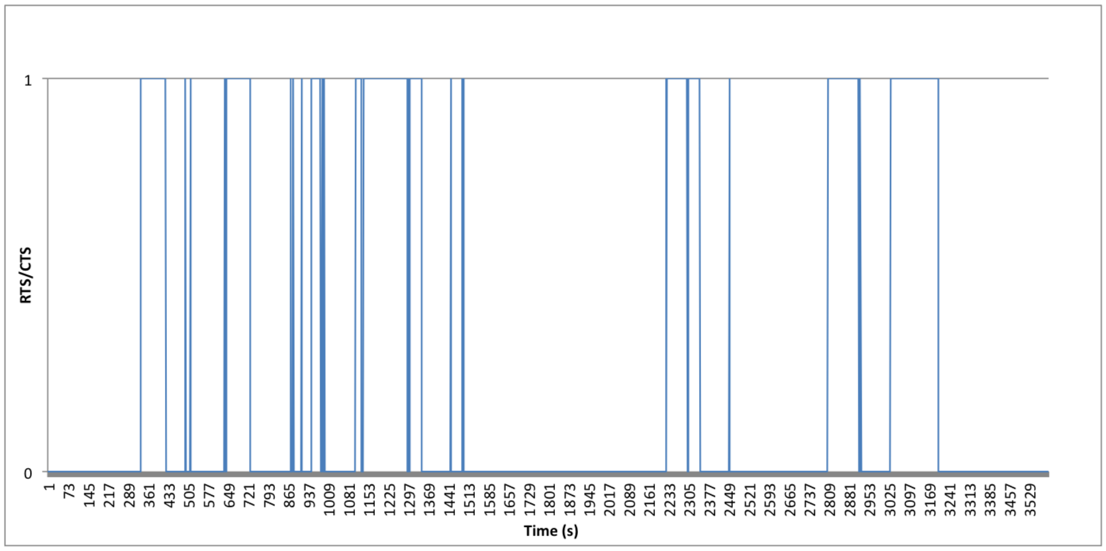
Figure 3.21: Collision rate, SENSE's prediction and RTS/CTS usage for AP in hot-spot trace when all nodes are exposed to one another

3.6.2 Ad-Hoc Scenarios

Experiments similar to the ones conducted for infrastructure-based typologies were run for ad-hoc scenarios using the parameters summarized in Table 3.2. Note that, unlike the infrastructure-based experiments which did not need to use routing, ad-hoc scenarios employed the AODV protocol.



(a) Collision rate and SENSE's prediction

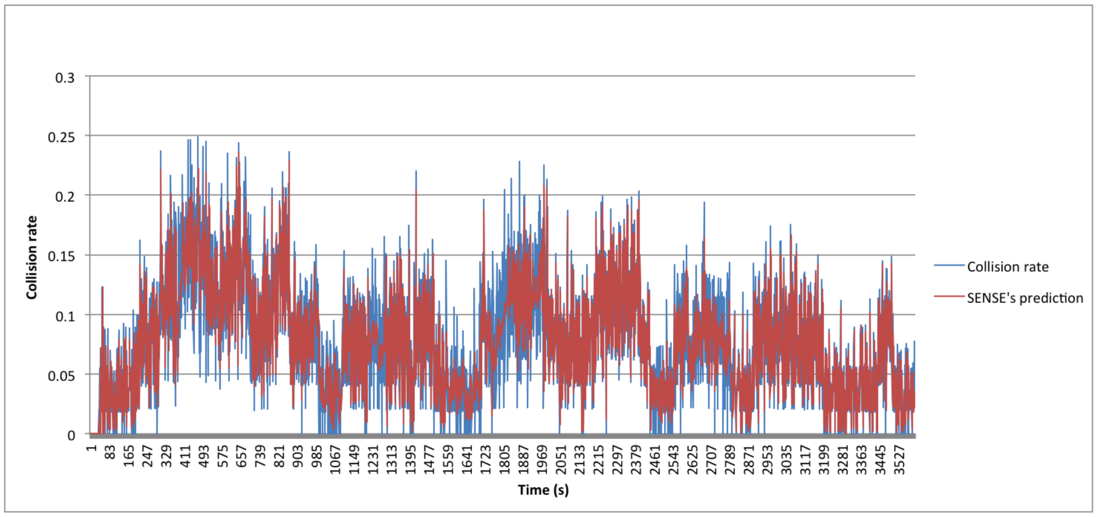


(b) RTS/CTS usage

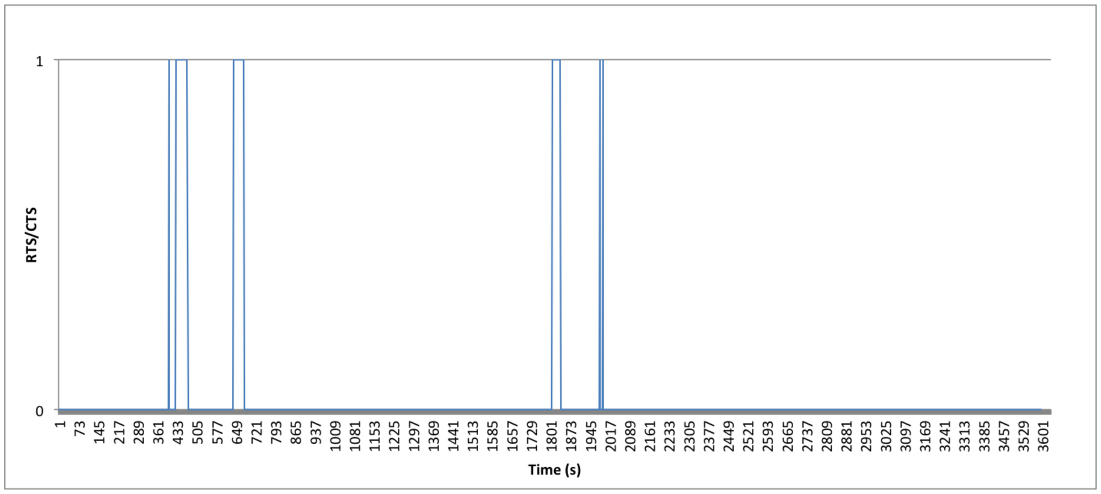
Figure 3.22: Collision rate, SENSE's prediction and RTS/CTS usage for AP in company campus trace when all nodes are hidden from one another

Average Throughput

We compare SACA's average throughput against that of IEEE 802.11's DCF base mode (RTS/CTS always off) and congestion avoidance mode (RTS/CTS always on) in a variety of ad-hoc scenarios. We also show average throughput



(a) Collision rate and SENSE's prediction



(b) RTS/CTS usage

Figure 3.23: Collision rate, SENSE's prediction and RTS/CTS usage for AP in company campus trace when all nodes are exposed to one another

when using statically configured values for the RTS Threshold, RT, namely 200-, 500-, 1000-, 1500-, and 2000 bytes.

Synthetic trace: To show how our algorithm adjusts to network dynamics, we periodically changed frame size and number of nodes transmitting in our synthetic data trace. Note that by varying the number of simultaneous transmitters, we

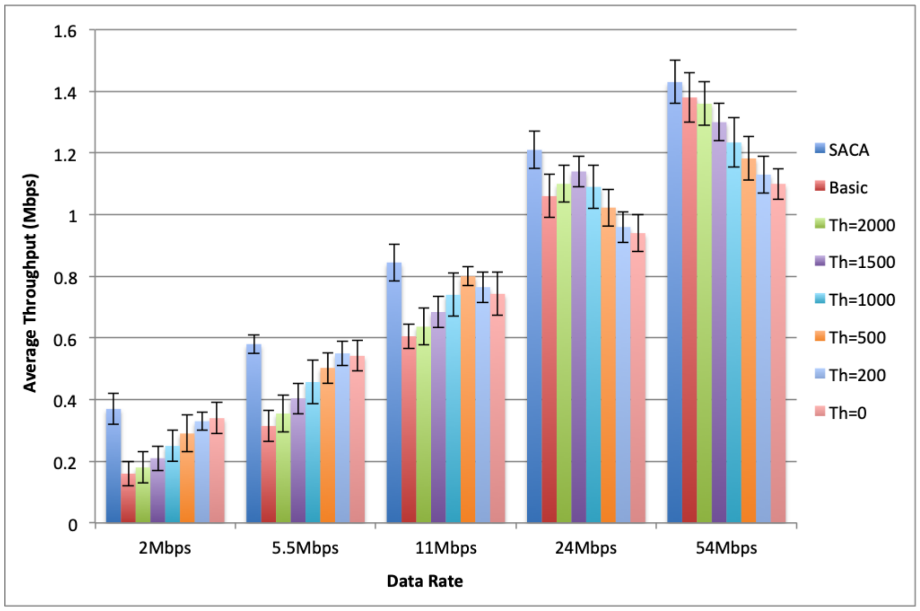


Figure 3.24: Average throughput using the synthetic trace in ad-hoc scenario.95% confidence intervals are shown.

vary network contention and consequently collision rate. We ran each experiment with data transmission rates of 54-, 24-, 11-, 5.5-, and 2Mbps, while signaling transmission rate is kept at 2Mbps. We ran each experiment 10 times randomly selecting senders and their receivers.

Figure 3.24 shows SACA's average throughput as well as that of IEEE 802.11 DCF's base mode ("Basic"), and RTS/CTS enabled based on different RT values. Consistent with results from infrastructure-based scenarios, we observe that our algorithm outperforms all other methods tested for all data rates used. As expected, for lower data rates, "Basic" has the lowest performance because frame transmission takes longer and therefore collision rate is higher. In the low data rate cases, lower RT values provide better performance. As the data rate increases, performance of "Basic" (RTS/CTS disabled all the time) improves and eventually outperforms "Th=0". For example, for 24 Mbps, RTS/CTS with mid-range RT values provide better performance than both "Basic" and RTS/CTS with larger RT values. The reason is that throughput improves when larger frames are "protected" by RTS/CTS at this data rate while, for smaller frames, throughput is higher without RTS/CTS. SACA always performs well because it can dynamically decide when to enable or disable RTS/CTS based on current conditions, i.e., frame size, transmission rate, and contention.

Table 3.7 shows the usage of RTS/CTS at a specific node as contention and frame size changes every 5 seconds. It confirms that, in addition to frame size and network contention, data transmission rate plays an important role as well. For example, in seconds 10 to 15, when frames are 2000 bytes and collision rate is 21%, at 54 and 24 Mbps, RTS/CTS is disabled. However, in 11, 5.5 and 2 Mbps, RTS/CTS is used. This is because, for lower data rates, using RTS/CTS turns out to be more advantageous.

Table 3.7: SACA's slot-by-slot behavior using the synthetic trace with different data rates in ad-hoc scenario

Time (s)	Frame Size (bytes)	Collision	54Mbps	24Mbps	11Mbps	5.5Mbps	2Mbps
0 to 5	1500	2%	Basic	Basic	Basic	Basic	Basic
5 to 10	500	7%	Basic	Basic	Basic	Basic	RTS/CTS
10 to 15	2000	21%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS
15 to 20	200	33%	Basic	Basic	Basic	Basic	RTS/CTS
20 to 25	1000	39%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS
25 to 30	2000	52%	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
30 to 35	500	59%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS
35 to 40	200	63%	Basic	Basic	Basic	RTS/CTS	RTS/CTS
40 to 45	1500	67%	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS	RTS/CTS
45 to 50	500	71%	Basic	Basic	RTS/CTS	RTS/CTS	RTS/CTS

Hot-spot trace: For the hot-spot trace, we employed a similar strategy as in the infrastructure-based scenarios. In order to vary network contention, we used 10-, 30-, and 50 senders and assigned each flow to 1-, 3-, and 5 senders, respectively. Flows have slightly different start times. Similarly to the synthetic trace experiments, we compare the average throughput when using SACA against

IEEE 802.11's base mode (no RTS/CTS), as well as statically configured *RT* values of 0- (RTS/CTS always enabled), 200-, 500-, 1000-, 1500-, and 2000 bytes.

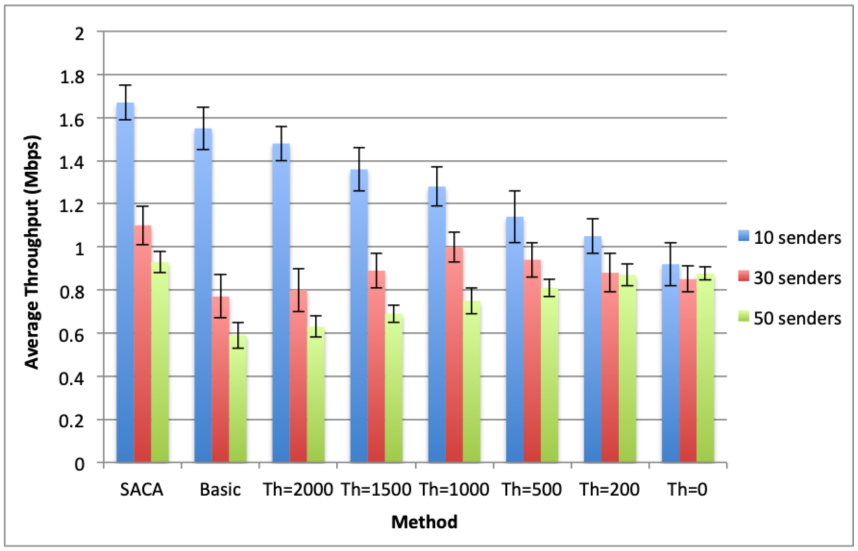


Figure 3.25: Average throughput using hot-spot trace in ad-hoc scenario. 95% confidence intervals are shown.

As shown in Figure 3.25, in the 10-sender scenario, since there is less contention, using the base mode ("Basic) or higher *RT* values is more beneficial. By adding more nodes and, as a result, increasing contention, lower RTS thresholds like 200- and 500 bytes perform better. In the 50-sender scenario, RTS/CTS should be used all the time because of the high contention. In all cases, SACA outperforms all other methods because of its ability to adjust to network contention and airtime. While in the 10-sender experiment, "Basic" yields similar throughput when compared to our approach, in the 50-sender scenario, "Basic" is the worst performer. In other words, SACA can beat the best static method in all cases.

Company campus network trace: Similarly to the hot-spot trace, data rates provided in the Radiotap Header are used to calculate the airtime in the company campus network trace. We ran experiments with 10-, 30-, and 50 senders by assigning each captured flow to 2-, 6- and 10 senders, respectively. Compared to the public hot-spot trace, average frame sizes and data rates are considerably higher in this dataset (see Tables 4.3 and 4.4). As shown in Figure 3.26, in all scenarios, SACA outperforms all other methods since it adjusts dynamically to frame size, transmission rate, and network contention. For instance, even though contention is not high in the 10-node scenario, since frames are larger on average, enabling RTS/CTS yields higher average throughput when compared to "Basic". As the number of nodes increases, the optimal *RT* value decreases. So in the 50-node scenario RTS/CTS should be used all the time.

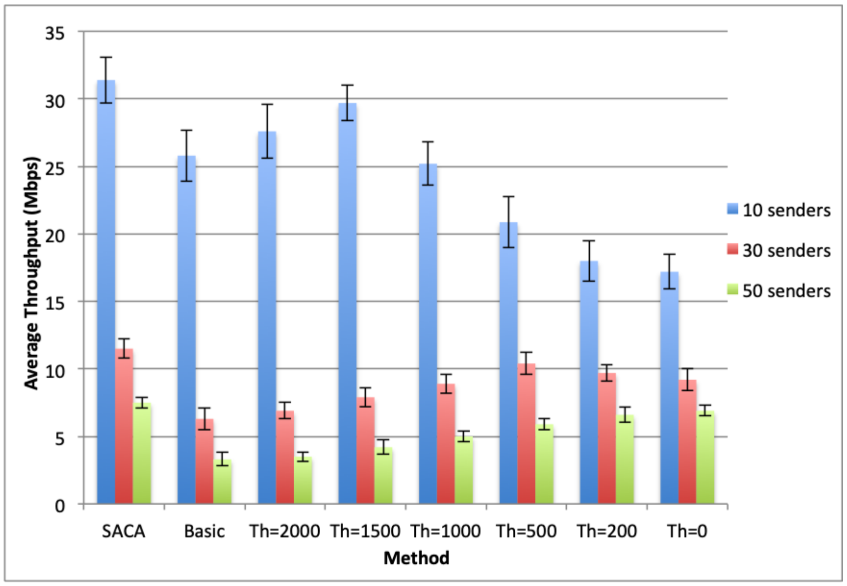


Figure 3.26: Average throughput using company campus trace in adhoc scenario. 95% confidence intervals are shown.

Chapter 4

Dynamically Tuning IEEE 802.11's Contention Window Using Machine Learning

The IEEE 802.11 standard, also known as WiFi, specifies two types of MAC protocols, namely the Distributed Coordination Function (DCF) and the Point Coordination Function (PCF). DCF is IEEE 802.11's most widely used medium access mechanism and uses the Carrier Sensing Multiple Access/Collision Avoidance (CSMA/CA) protocol[1]. CSMA/CA arbitrates access to the shared communication medium using a contention-based, on-demand distributed mechanism. One of the key components of IEEE 802.11's DCF is the Binary Exponential Backoff (BEB) algorithm which was introduced to mitigate channel contention and prevent collisions of packets simultaneously transmitted by multiple stations. It delays the retransmission of a collided packet by a random time, chosen uniformly over n slots ($n > 1$), where n is a parameter called *Contention Window*, or (CW). The BEB

[1]DCF provides two modes of operation: the *Base Mode* which uses CSMA and the *Collision Avoidance Mode*, which uses CSMA/CA.

algorithm works as follows: CW is initially set based on a pre-specified minimum value, (CW_{min}). If a collision happens, the station chooses an exponentially increased CW until it reaches CW's pre-specified maximum value (CW_{max}). As such, CW can significantly impact IEEE 802.11's performance. Choosing small CW values may result in more collisions and backoffs. On the other hand, choosing large CW may result in unnecessary idle airtime and additional delay. In either case, the channel is not used efficiently. Therefore, the value of CW should be adjusted considering the actual level of contention in the channel.

In this chapter, we use a simple, yet effective machine learning technique called *Fixed-Share* to decide the value of CW based on recent past network contention. Unlike the original BEB algorithm which increases or decreases CW based solely on the status of the most recently transmitted packet, our method also accounts for recent network contention conditions in addition to last packet's transmission status. Our experimental results show that our method yields up to 72 % and 50 % throughput improvement over the original IEEE 802.11's binary exponential backoff (BEB) algorithm and History-Based Adaptive Backoff (HBAB) algorithms respectively. End to end delay is improved upto 50 % and 30 % compared to the BEB and HBAB methods respectively.

The rest of this chapter is organized as follows. Section 4.1 provides a brief overview of IEEE 802.11's Binary Exponential Backoff (BEB) algorithm and present related work. Our machine learning based method to dynamically adjust 802.11's contention window is described in Section 4.2. Section 4.3 and Section 4.4 present our experimental methodology and results in infrastructure-based and ad-hoc network scenarios respectively.

4.1 Background and Related Work

IEEE 802.11's Binary Exponential Backoff (BEB) algorithm was introduced to decrease the chance of collision of packets simultaneously (re)transmitted by multiple stations. In the original BEB algorithm, if a node wants to transmit a data frame, it first senses the channel for a *DCF Inter frame Space (DIFS) interval* to check whether the channel is idle. If the channel is sensed idle, the node transmits the data packet immediately. Otherwise, as shown in Equation 4.1, it selects a random backoff time in the range of 0 and CW to avoid collisions. The backoff time is decreased every time thereafter when the node senses the medium idle. When the backoff time reaches zero, the node can then transmit.

$$Backoff\ time = random\ [0, CW] \times slot\ time \tag{4.1}$$

If the transmission is unsuccessful, the CW will be doubled for the next transmission as shown in (4.2). In case of a successful transmission, CW is reset to CW_{min}.

$$CW_{new} = min(CW_{old} \times 2, CW_{max}) \tag{4.2}$$

A number of drawbacks with the original BEB algorithm have been identified. Fairness is one of them; for instance, resetting CW to CW_min after a successful transmission may cause the node who succeeds in transmitting to dominate the channel for an arbitrarily long period of time. As a result, other nodes may suffer from severe short-term unfairness. Additionally, the current state of the network (e.g., load) should be taken into account to select the most appropriate backoff interval.

Motivated by its performance impact, several proposals have focused on optimizing IEEE 802.11's backoff algorithm. In the remainder of this section, we provide a brief overview of prior related work.

We categorize related work on improving IEEE 802.11's BEB performance in two groups. While the first group focuses on how to increase or decrease the size of CW more efficiently, the second group tries to set the values of CW_{min} and CW_{max}, i.e., CW's upper and lower bounds.

Increasing/decreasing CW

In MACAW [6], a new approach to increasing/decreasing CW's size was proposed and works as follows. In the case of unsuccessful transmission, CW is multiplied by a constant factor set to 1.5, instead of doubling it. When transmissions are successful, CW is decreased by 1, instead of resetting it to its minimum value, C_{min}.

The approach proposed in [11] increases CW linearly by adding CW_{min} to the current value of CW in case of unsuccessful transmissions and decreases CW by 1 when transmissions are successful.

An adaptive contention window adjustment algorithm is proposed in [41]. It uses a control parameter called CW_{basic} whose value is fixed. The proposed algorithm works as follows: in case of successful transmissions, if CW is less than CW_{basic}, CW is decremented by 1. Otherwise, i.e., if CW is greater than CW_{basic}, CW is divided by 2. In case of unsuccessful transmissions, CW is doubled and if the new CW is still below CW_{basic}, CW_{basic} is adopted.

In [4], $CW_{threshold}$ is introduced and its value is fixed and set to half of CW_{max}. At the time of transmission, the current value of CW is compared with $CW_{threshold}$. If $CW <= CW_{threshold}$, traffic load is considered to be light. Otherwise traffic load is considered high. If traffic load is light, in case of unsuccessful transmission,

CW is decremented by 1; otherwise, in case transmission is successful, CW is decremented by 2. If traffic load is deemed high, in case transmission is successful, CW is set to CW+2; otherwise, i.e., if a collision occurs, CW is set to 2×CW+2.

The work described in [22] tries to address BEB's unfairness. According to the proposed mechanism, participating stations monitor the state of the channel. Stations use channel state information to keep their CW synchronized so that they all have uniform probability to access the channel access next time they need to transmit.

in [39], collision ratio is used as an indicator of network contention and guides how CW is adjusted. Collision ratio is given by the number of successful transmissions divided by the total number of transmissions. When a node experiences collision, if the collision ratio is lower than a pre-defined threshold, network contention is considered low and CW is incremented by 32. Otherwise, network contention is deemed high, and CW is doubled. After a successful transmission, if the collision ratio is less than the threshold, CW is reset to CW_{min}. Otherwise, i.e., network contention is high, CW is set to $CW_{min} \times 2^{rc} + 32$ where rc is the retransmission count.

The History-Based Adaptive Backoff (HBAB) algorithm [3] updates CW based on whether the current data transmission succeeds or not, as well as on the success or failure of the previous N consecutive data transmissions. We describe HBAB in more detail in Section 4.3 as it is used, along with IEEE 802.11's BEB, in the comparative performance study we conduct to evaluate the proposed contention window adaptation algorithm.

The Inverse Binary Exponential Backoff (iBEB) algorithm [5] tries to improve BEB's short-term fairness by allowing devices that experience more collisions to access the channel with higher probability than devices which experience no

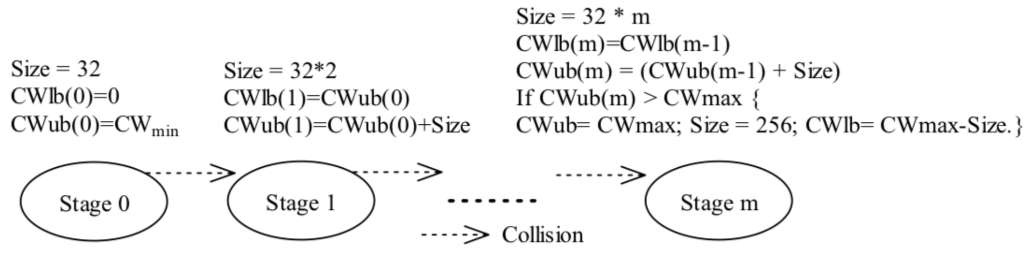


Figure 4.1: Collision resolution stage [34]

collision or fewer. As such, the CW of "collided" nodes is decreased so they can gain higher channel access priority resulting in short-term fairness. Experimental results reported in [5] show that iBEB is able to improve throughput, delay, and collision probability, especially in the case of dense network scenarios.

Other variants of the BEB algorithm have been proposed. For example, the Binary Negative-Exponential Backoff (BNEB) was introduced in [29]. In BNEB, CW of all node is initialized to CW_{max}. If a node's transmission is successful, its CW is halved and if it fails, its CW is reset to CW_{max}. New Binary Exponential Back-off (N-BEB) [46] also tries to mitigate BEB's fairness problem by monitoring and then limiting the number of times each node accesses the channel.

Setting CW's lower and upper bounds

Another group of papers focuses on optimizing the values of CW_{min} and CW_{max}. For instance, as shown in Figure4.1, in [34], the range [CW_{min}, CW_{max}] is split into sub-ranges where each sub-range is assigned to a *contention stage*. Contention stage is initially set to 0 and with each collision, it is increased by 1. With each successful transmission, the contention stage goes back to stage 0.

Contention window parameters have been studied in the IEEE 802.11e which is an amendment to the IEEE 802.11 standard. In IEEE 802.11e, the levels of traffic priority are called Access Categories (AC) where each AC[i] has its own

$CW_{min}[i]$ and $CW_{max}[i]$.

The approach described in [16] proposes a mechanism to set $CW[i]$, i.e., CW for each IEEE 802.11e's Access Category. It uses the collision rate which is measured periodically and smoothed using exponentially weighted moving average. For each transmission, $CW[i]$ is set as a function of the smoothed collision rate, $CW_{min}[i]$, and $CW_{max}[i]$.

Using machine learning to improve network protocol performance

Recently, machine learning techniques have gained huge traction and have been used in a wide range of applications. Specifically in the context on computer networking performance management, in [40], the Fixed-Share algorithm is used to estimate TCP's round-trip time. In our prior work [13], we estimate collision rate using an algorithm called SENSE which employs a combination of Fixed-Share and Exponentially-Weighted Moving Average (EWMA). In [15], SENSE estimates network contention which is then used to enable/disable RTS/CTS in IEEE 802.11 networks.

To the best of our knowledge, our proposed algorithm, which is described in Section 4.2, is the first to use machine learning to automatically adjust IEEE 802.11's CW. Most proposed methods to-date adjust CW based solely on the last transmission, whereas our approach takes into account packet transmission history and adjusts CW accordingly.

4.2 Automatically Adjusting IEEE 802.11's Congestion Window

In this section, we introduce a simple yet effective mechanism based on the *Fixed-Share algorithm* [19] to tune IEEE 802.11's CW. We start with a brief description of Fixed-Share.

4.2.1 Fixed-Share Algorithm

The Fixed-Share algorithm is part of the Multiplicative Weight algorithmic family which has shown to yield performance improvements in a variety of on-line problems [19], [40]. This family of algorithms combines predictions of a set of experts $\{x_1,x_2,...,x_N\}$ to calculate the overall prediction denoted by $\hat{y}_t$. Each expert has a weight $\{w_1,w_2,...,w_N\}$ representing the impact of that expert on the overall predictor. Based on the difference between each expert's prediction and the real data represented by y_t, the weight of each expert is updated [13]. Algorithm 8 shows Fixed-Share Experts' pseudo-code. Each expert is initialized with a value within the range of the quantity to be predicted and the weight of all experts is initialized to $\frac{1}{N}$, where N is the number of experts. At every iteration, based on each expert's current weight and value, the prediction for the next trial is calculated as shown in the **Prediction** step of the algorithm. The **Loss Function** step then checks how good the prediction of each expert was using a loss function $L_{i,t}(x_i, y_t)$. The result of the loss function loss for each expert is used in the **Exponential Update** step to adjust the experts' weights by multiplying the current weight of the $i-th$ expert by $e^{-\eta \times L_{i,t}(x_i,y_t)}$. The *learning rate* η is used to determine how fast the updates will take effect, dictating how rapidly the weights of misleading experts will be reduced. Finally, in the **Sharing Weights** step, a fixed fraction of the weights of experts that are performing well is shared among the other experts. The goal of this step is to prevent large differences among experts' weights [21]. The amount of sharing can be adjusted through the *sharing rate* parameter α.

Algorithm 8 Fixed-Share Algorithm

```
Parameters:
    $\eta > 0, 0 \leq \alpha \leq 1$
Initialization:
    $w_{1,1}=...=w_{N,1}=\frac{1}{N}$
Prediction:
    $\hat{y}_t=\frac{\sum_1^N w_{i,t}\times x_i}{\sum_1^N w_{i,t}}$
Loss Function:
    $L_{i,t}(x_i, y_t) = \begin{cases} (x_i - y_t)^2 & , x_i \geq y_t \\ 2 \times y_t & , x_i < y_t \end{cases}$
Exponential Update:
    $\acute{w}_{i,t} = w_{i,t} \times e^{-\eta \times L_{i,t}(x_i,y_t)}$
Sharing Weights:
    $Pool = \sum_{i=1}^N \alpha \times \acute{w}_{i,t}$    $w_{i,t+1} = (1-\alpha) \times \acute{w}_{i,t} + \frac{1}{N} \times Pool$
```

4.2.2 Proposed Approach

We propose a modified version of the Fixed-Share algorithm to dynamically set IEEE 802.11's CW. More specifically, as illustrated in Algorithm 9, we design loss- and gain functions that account for current network conditions. Our proposed technique works as follows. Similarly to the standard Fixed-Share algorithm (Algorithm 8), in the **Initialization** step in Algorithm 9, the weight of all experts is set to $\frac{1}{N}$, where N is the number of experts. Each expert is assigned a fixed value within the range of [CW_{min}, CW_{max}]. We currently assign the values of 15, 22, 33, 50, 75, 113, 170, 256, 384, 576, 865 and 1023 to 12 experts forming a geometric sequence with ratio of 1.5. We have experimented with 25 experts with ratio of 1.2 and have not seen any significant change in the results.

In the **CW Calculation** step, $\hat{CW}_t$ which is CW's estimate for time t is calculated based on the current value of the experts and their weights. Clearly, experts with more weight will have more influence on the next CW. In the **Loss/Gain Function** step, the performance of all experts is evaluated based on their value, $\hat{CW}_t$, and whether the previous packet transmission was successful or not.

The loss and gain functions are designed to adjust CW to the current conditions of the network. Our loss/gain function works as follows. If a packet is transmitted successfully, the weight of experts higher than $\hat{CW}_t$ will be reduced and the weight of experts lower than $\hat{CW}_t$ will be increased. This will cause the value of the next CW to be lower and, as a result, the next transmission will be scheduled more aggressively. Also, for experts with higher value than $\hat{CW}_t$, the higher the value of the expert, the higher the loss of its weight will be. Similarly, for experts with lower value than $\hat{CW}_t$, the lower the value of the expert, the lower its weight gain will be.

In the case of unsuccessful transmissions, the loss/gain function will increase the weight of experts with values higher than $\hat{CW}_t$ and reduce the weight of experts with values lower than $\hat{CW}_t$. This will result in a higher CW for the next packet transmission and less chance of collision. In this case, for experts with lower value than $\hat{CW}_t$, the lower the expert's value, the higher the weight loss. For experts with higher value than $\hat{CW}_t$, the higher their value, the lower the weight gain.

The overhead that comes with of our method is due to the two factors: extra memory space and additional computations. As for the memory, extra storage space is needed to store expert's values and weights. As for the computations, some operations are needed to determine the next CW, such as if conditional statement, multiplications and summations. These operations are done in real time and as a result, our algorithm is fast enough to catch up with the underlying physical channel.

```
Algorithm 9 Proposed Algorithm
  Initialization:
      $w_{1,1}=...=w_{N,1}=\frac{1}{N}$
      $x_1 = CW_1, x_2 = CW_2, ..., x_N = CW_N,$

  CW Calculation:
      $\hat{CW}_t=\lfloor\frac{\sum_1^N w_{i,t}\times x_i}{\sum_1^N w_{i,t}}\rfloor$

  Loss/Gain Function:
    • If packet received successfully:
      $w_{i,t+1} = \begin{cases}[1-\frac{x_i-\hat{CW}_t}{x_i}]\times w_{i,t} & ,x_i > \hat{CW}_t \\ [1+\frac{x_i}{\hat{CW}_t}]\times w_{i,t} & ,x_i \leq \hat{CW}_t\end{cases}$
    • If packet is not received successfully:
      $w_{i,t+1} = \begin{cases}[1+\frac{\hat{CW}_t}{x_i}]\times w_{i,t} & ,x_i > \hat{CW}_t \\ [1-\frac{\hat{CW}_t-x_i}{\hat{CW}_t}]\times w_{i,t} & ,x_i \leq \hat{CW}_t\end{cases}$

  Sharing Weights:
      $Pool = \sum_{i=1}^N \alpha \times \acute{w}_{i,t}$      $w_{i,t+1} = (1-\alpha)\times \acute{w}_{i,t} + \frac{1}{N}\times Pool$
```

4.3 Experimental Methodology

In this section, we describe our experimental setup including the scenarios, traffic loads, as well as performance metrics used when evaluating the proposed approach. We compare the performance of our technique against both the original IEEE 802.11 contention window adjustment technique as well as the History-Based Adaptive Backoff (HBAB) algorithm [3]. As such, we also provide a brief overview of HBAB.

4.3.1 Experimental Setup

We ran experiments using the *ns-3* [8] network simulator and its implementation of the IEEE 802.11n for both infrastructure-based and ad-hoc network scenarios. In our simulations, we use typologies with 100 nodes randomly placed in a $1000x1000m_2$ area. In order to vary network contention conditions, we vary the number of sender nodes. We explore how dynamically our method is able to adjust the contention window and its effect on network performance. Table 4.1 summarizes the parameters describing our experimental setup and their values. Note that AODV [42] routing was used only in the multi-hop ad-hoc experiments.

Traffic Load: We used synthetic data traces as well as traces collected in real networks to drive our simulations. Table 4.2 summarizes the synthetic data parameters and their values. Our real traffic traces ere collected in two different settings, namely: (1) a public hot spot (Table 4.3), and (2) a company campus network (Table 4.4) using a wireless sniffer. Note that since there are 10 and 5 individual flows in the hot spot and company traces, respectively, we replicate these flows in scenarios with higher number of nodes.

Performance Metrics: We evaluate our contention window adjustment technique by comparing its performance against IEEE 802.11's original mechanism as well as HBAB [3]. As performance metrics, we use average throughput and average end-to-end delay. Average throughput is calculated as the ratio between the number of received packets and the total number of transmitted packets averaged over all nodes. Average end-to-end delay is given by the interval of time between when a packet was received and when it was sent averaged over all received packets.

Channel access fairness is an important issue in MAC protocol design. As such, we also evaluate the proposed approach's fairness by comparing its minimum, maximum, and average throughput against those of IEEE 802.11's BEB and HBAB.

Table 4.1: Simulation setup parameters and their values

Area	1000mx1000m
Number of nodes	100
Traffic	CBR and real traces
IEEE 802.11 Version	802.11n
Number of experts	12
CW_{min}	15
CW_{max}	1023
Routing protocol	AODV

Table 4.2: Synthetic trace

Simulation time	200s
Traffic type	CBR
Frame size	1024 Bytes
Data rate	54 Mbps

4.3.2 History-Based Adaptive Backoff

History-Based Adaptive Backoff (HBAB) [3] increases or decreases the congestion window CW based on the current- as well as past data transmission trials. HBAB defines two parameters α and N; α is a multiplicative factor used to update

Table 4.3: Hot spot trace

Location	Coffee shop
Time	Around noon
Number of flows	10
Duration	20 minutes
Frame size	Varies within 34-2150 byte range
802.11 version	802.11n

Table 4.4: Company campus network trace

Location	Company campus network
Number of flows	5
Duration	30 minutes
Frame size	Varies within 34-11000 byte range
802.11 version	802.11n

CW and N is the number of past transmission trials considered by the algorithm. The outcome of the previous N transmission trials is stored in $ChannelState$; failed transmissions are represented by 0 while successful ones bt 1. For example, if $N = 2$, $ChannelState$=$\{0,1\}$ means that the last transmission succeeded but the previous one failed. Larger values of N mean larger windows into the past but require, albeit relatively small, additional memory.

Algorithm 10 shows HBAB's pseudo-code. Note that we follow HBAB's implementation in [3] and use $\alpha = 1.2$ and $N = 2$, i.e., HBAB examines the status of the two previous and consecutive data transmissions, as well as the current one, to make a decision on how to adjust CW. In case the current transmission is successful, but the two previous transmissions failed, i.e., $ChannelState[0] = 0$ and $ChannelState[1] = 0$, the new value of CW is set to the current CW divided by α. Otherwise, CW is set to CW_{min}. In case the current transmission is unsuccessful, CW is multiplied by α.

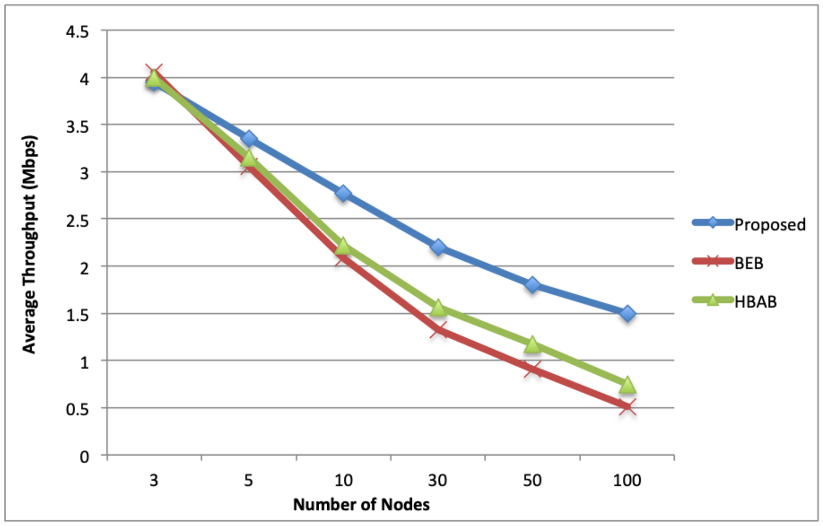
Algorithm 10 HBAB Algorithm

```
Initialization:
    $$CW = CW_{min}, \alpha > 1$$
    $$ChannelState[0] = 1, ChannelState[1] = 1$$

If current transmission succeeds:
    $$CW = \begin{cases} \frac{CW}{\alpha} & , ChannelState[0] = 0 \\ & and\ ChannelState[1] = 0 \\ CW_{min} & , otherwise \end{cases}$$

If transmission failed:
    $$CW = CW \times \alpha$$

ChannelState update:
    $$\begin{cases} ChannelState[0] = ChannelState[1] \\ ChannelState[1] = 0, last\ transmission\ failed \\ ChannelState[1] = 1, last\ transmission\ succeeded \end{cases}$$
```

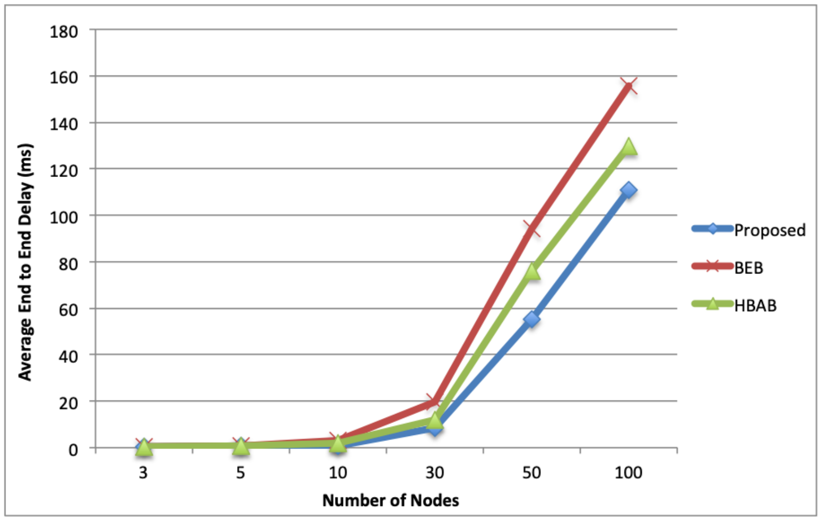
We use HBAB in the performance evaluation of our proposed contention window adjustment mechanism as it represents mechanisms that are similar to ours, i.e., that use transmission status history to set CW.

4.4 Results

As described in Section 4.3, we evaluate our approach using two types of scenarios, namely: infrastructure-based and infrastructure-less (or multi-hop ad-hoc) networks. We start by presenting results obtained for the infrastructure-based scenarios followed by the infrastructure-less scenario results. In all graphs, each data point is calculated by averaging over 10 runs that use different random seeds.



(a) Average throughput

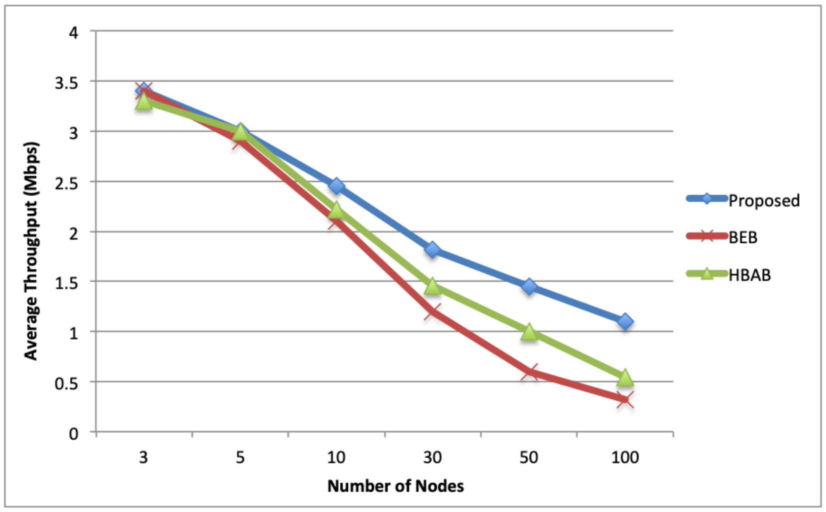


(b) Average delay

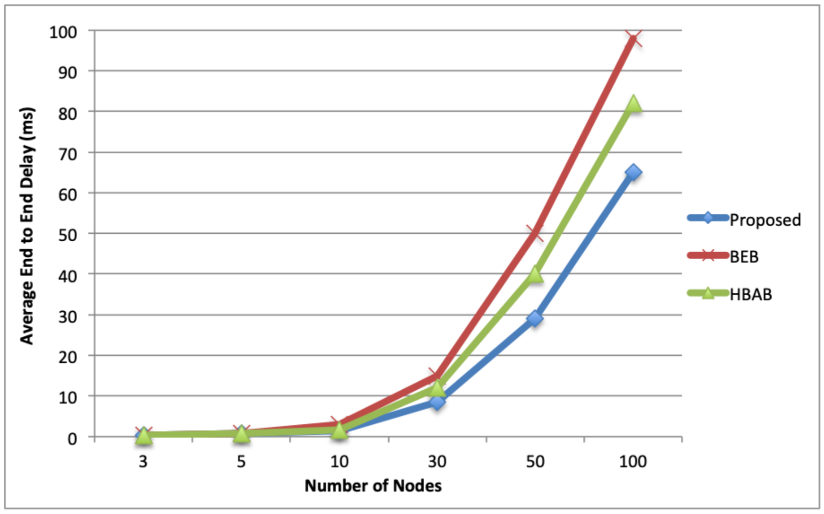
Figure 4.2: Average throughput and delay as a function of number of senders for the synthetic traffic trace in the infrastructure based scenario

4.4.1 Infrastructure-based Scenarios

In the infrastructure-based experiments, randomly selected nodes send traffic to the Access Point (AP) which is placed in the center of the area being simulated. We drive the experiments using the synthetic and real (i.e., hot spot and company campus) traffic traces described in Section 4.3 and vary the number of senders as



(a) Average throughput

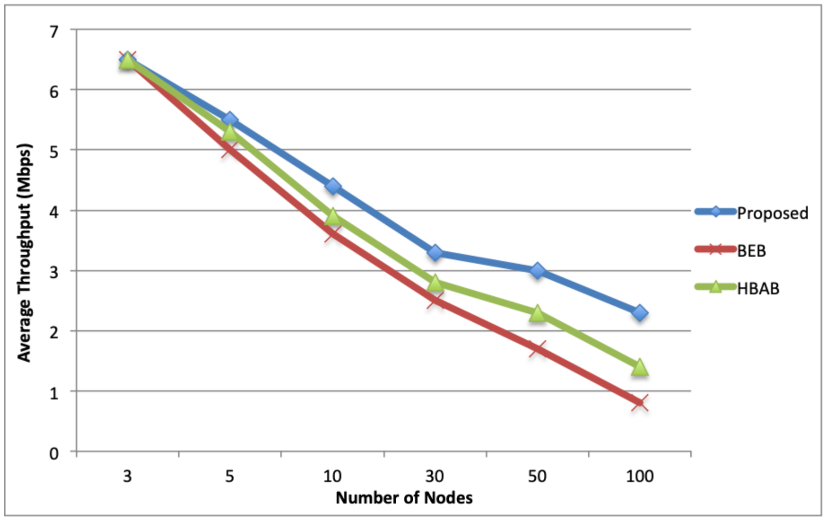


(b) Average delay

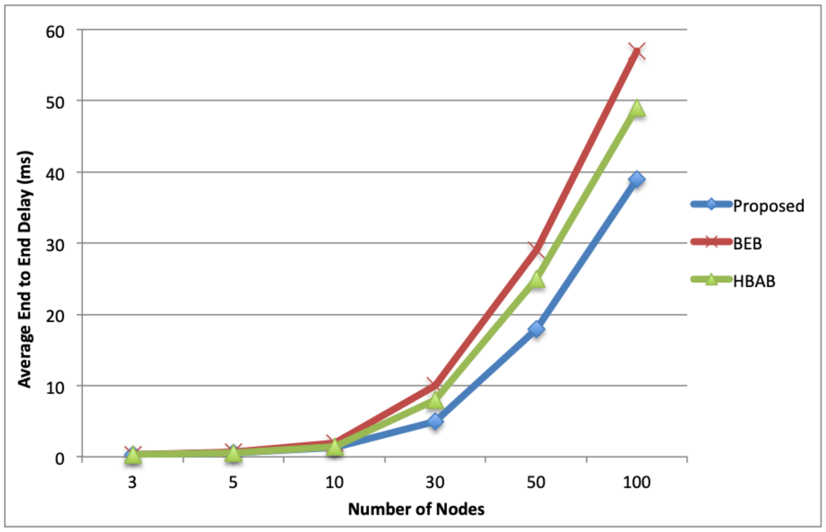
Figure 4.3: Average throughput and delay as a function of number of senders for hot-spot traffic trace in infrastructure based scenario

follows: 3, 5, 10, 30, 50, and 100.

Average Throughput and End-to-end Delay: Figures 4.2, 4.3, and 4.4 compare the average throughput and end-to-end delay of our method against IEEE 802.11's BEB and HBAB for different number of nodes and traffic traces, i.e., synthetic, hot-spot, and company campus data traces. We observe in all



(a) Average throughput



(b) Average delay

Figure 4.4: Average throughput and delay as a function of the number of senders for company campus traffic trace in infrastructure based scenario

three figures similar trends for both average throughput and end-to-end delay. As expected, average throughput decreases and end-to-end delay increases as the number of senders increases. For lower number of senders, e.g., 3 and 5, all three algorithms perform similarly. However, as the number of senders increases resulting in higher network contention, our approach is able to achieve better

Table 4.5: Throughput and delay improvement of proposed congestion window adaptation algorithm compared to IEEE 802.11's BEB and HBAB in infrastructure-based scenario with 100 senders for all traffic traces

	BEB Throughput	HBAB Throughput	BEB Delay	HBAB Delay
Synthetic	180%	90%	28%	12%
Hot-spot	220%	92%	33%	20%
Company	170%	64%	31%	21%

average throughput and end-to-end delay performance when compared to IEEE 802.11's BEB and HBAB for all three traffic traces.

Table 4.5 summarizes the throughput and delay improvement achieved by our congestion window adaptation algorithm when compared to BEB's and HBAB's for 100 senders in the infrastructure-based scenario for all traffic traces. We observe that in such more heavily loaded environments, our approach is able to achieve significant gains both in throughput (up to 220% over BEB and 92% over HBAB) as well as in end-to-end delay (up to 33% over BEB and up to 21% over HBAB).

Fairness: In order to evaluate the ability of our contention window adaptation algorithm to provide a fair share of the channel to participating stations, Table 4.6 shows the minimum, average, and maximum throughput reported by stations when using our algorithm compared against BEB and HBAB for the synthetic data trace in the infrastructure-based scenario with 100 senders. Both the difference between the maximum and minimum throughput as well as the standard deviation (also reported in Table 4.6) show that our approach yields superior fairness performance when compared to both BEB and HBAB. As previously discussed, the main reason for BEB's less fair channel allocation is due to the reset of CW to CW_{min} upon a successful transmission, which gives certain nodes higher chance to seize the

Table 4.6: Minimum, average, and maximum throughput, and standard deviation achieved by our approach, BEB, and HBAB for synthetic data trace in infrastructure-based scenario with 100 senders

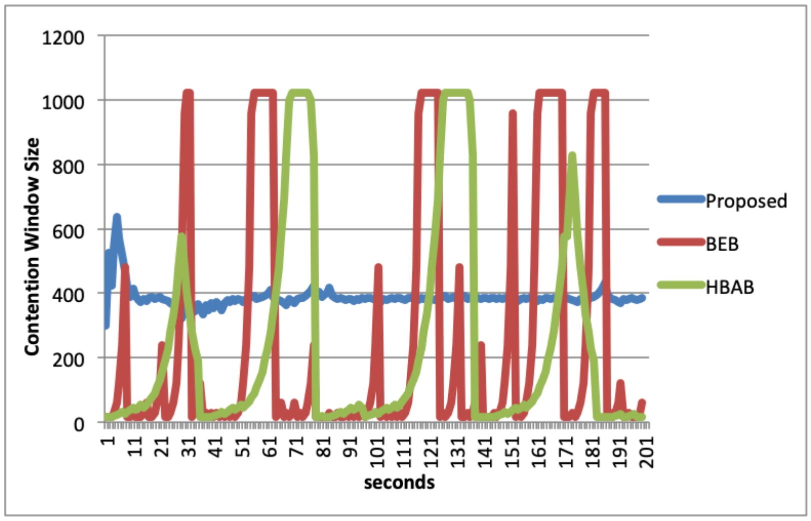
	Minimum (Mbps)	Average (Mbps)	Maximum (Mbps)	Standard Deviation
Proposed	0.64	1.2	1.96	0.48
BEB	0	0.51	2.12	0.89
HBAB	0	0.75	1.56	0.62

channel. HBAB shows improvement over BEB's fairness by avoiding immediate reset of CW to CW_{min} after single successful transmission, but still only considers short term packet transmission history which results in less fair channel allocation when compared to our approach. We should point out that BEB is able to yield the highest maximum throughput which is consistent with its resetting of CW to CW_{min} upon a successful transmission.

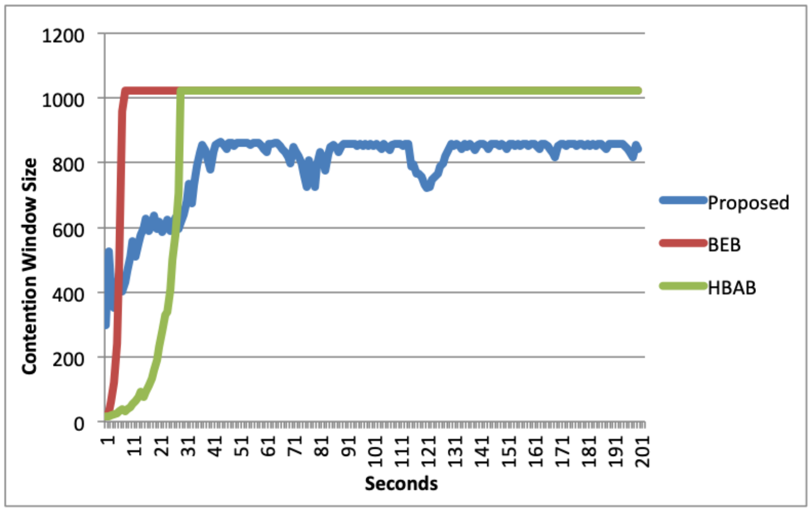
The graphs in Figure 4.5 showing CW variation over time for the nodes with minimum and maximum throughput for the synthetic trace in the infrastructure-based scenario with 100 senders reiterate our observations. We notice from Figure 4.5 that for both BEB and HBAB, CW for the node that reports the minimum throughput stays practically constant at CW_{max} for almost the whole experiment. In the case of the maximum throughput node, its CW varies considerably between CW_{min} and CW_{max}, i.e., 1023, during the whole run under both BEB and HBAB. Under our approach, the maximum throughput node's CW is able to reach steady state quite fast around 400.

CW Variation:

In Figure 4.5a which shows the CW variation for the node with maximum throughput, we observe significant CW oscillation between CW_{min} and CW_{max}



(a) Maximum throughput



(b) Minimum throughput

Figure 4.5: Contention window size variation over time for the nodes with minimum and maximum throughput for synthetic trace in infrastructure-based scenario with 100 senders

under BEB and HBAB. In the case of our approach, CW stays fairly constant throughout the experiment. The reason is that, after each successful transmission, the weight of experts with value higher than the current CW will be reduced and the weight of experts with value lower than CW will be increased. Therefore, for the next transmission, since the CW is calculated as the weighted sum of all experts, its value decreases slowly. Also, in the case of unsuccessful transmission, the weight of experts with values higher than current CW are increased and the weight of experts with values lower than CW are decreased. And again, since the CW is calculated as the weighted sum of all experts, the next CW will be slightly higher for the next transmission. In other words, through the experts and their weights, our approach is able to account for recent past as well as the present.

Figure 4.5b shows the variation of CW over time for the node with the lowest average throughput in the infrastructure-based scenario with 100 senders using the synthetic traffic trace. As the results in Table 4.6 indicate, BEB's and HBAB's minimum throughput is 0 which indicates that there are some nodes in the network that suffer from starvation. We observe that, relatively early in the experiment, CW of the node with the lowest throughput stabilizes at CW_{max} which explains why certain nodes suffer from starvation.

4.4.2 Infrastructure-less Scenarios

In the ad-hoc experiments, randomly selected senders send data traffic to randomly selected receivers according to the three traffic traces described in Section 4.3. Similarly to the infrastructure-based experiments, the number of senders vary as follows: 3, 5, 10, 30, 50, and 100.

Average Throughput and End-to-end Delay: Figures 4.6, 4.7, and 4.8 show the average throughput and end-to-end delay of our method compared with BEB and HBAB for different number of senders and traffic traces in the ad-hoc scenario. Similarly to the trend reported in the infrastructure-based experiments, we observe that, for lower number of senders, all three methods perform similarly. However, when the number of senders increase, which result in higher network contention, our

Table 4.7: Throughput and delay improvement of proposed congestion window adaptation algorithm compared to IEEE 802.11's BEB and HBAB in ad-hoc scenario with 100 senders for all traffic traces

	BEB Throughput	HBAB Throughput	BEB Delay	HBAB Delay
Synthetic	230%	75%	31%	21%
Hot-spot	240%	78%	37%	23%
Company	257%	63%	35%	17%

method is able to achieve higher average throughput and lower average end-to-end delay when compared to both BEB and HBAB.

Table 4.7 summarizes the throughput and delay improvement achieved by our congestion window adaptation algorithm when compared to BEB's and HBAB's for 100 senders in the ad-hoc scenario for all traffic traces. Similarly to what was observed for the infrastructure-based experiment, in high contention networks, our approach yields significant improvement both in average throughput (up to 257% over BEB and 78% over HBAB) and average end-to-end delay (up to 37% over BEB and 23% over HBAB).

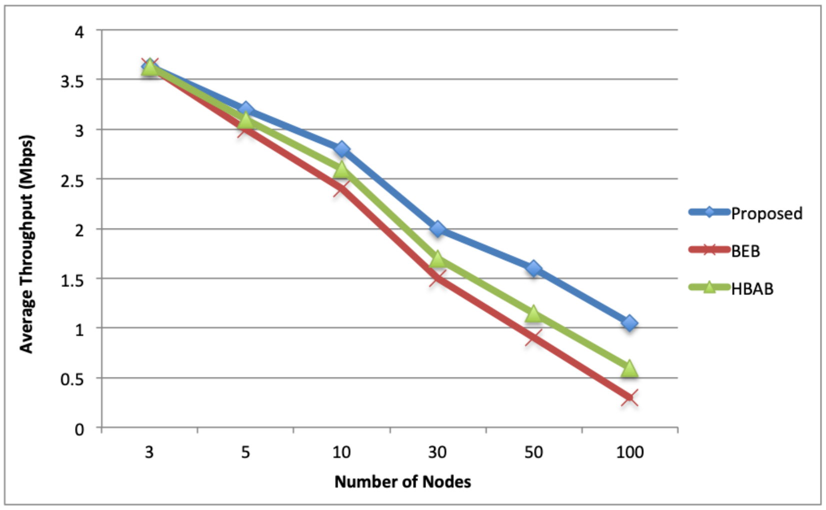
Fairness: To evaluate our algorithm's fairness in ad-hoc scenarios, we show the minimum, average, and maximum throughput for the synthetic traffic trace with 100 senders in Table 4.8. Like the results reported for the infrastructure-based experiments, our approach is able to reduce the gap between the minimum and maximum average throughput with a lower standard deviation, an indication of its ability to deliver improved fairness when compared to BEB and HAB.

CW **Variation:** Figure 4.9 shows *CW* variation over time for both the nodes that yield the maximum and minimum average throughput under our approach

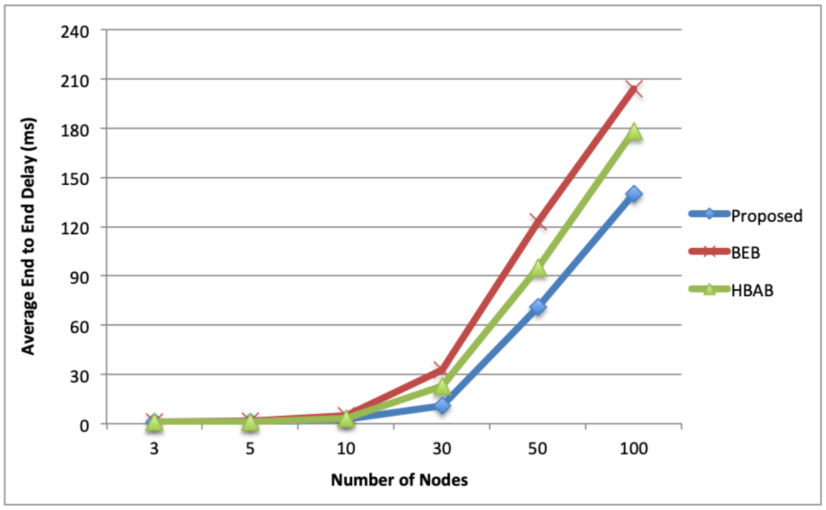
Table 4.8: Minimum, average, and maximum throughput, and standard deviation achieved by our approach, BEB, and HBAB for synthetic data trace in ad-hoc scenario with 100 senders

	Minimum (Mbps)	Average (Mbps)	Maximum (Mbps)	Standard Deviation
Proposed	0.43	1.05	1.6	0.41
BEB	0	0.3	1.8	0.75
HBAB	0	0.6	1.2	0.52

as well as under BEB and HBAB in the ad-hoc scenario with 100 senders using the synthetic traffic trace. Like the trend observed in the infrastructure-based experiments, our approach is able to achieve steady state relatively quickly for both the nodes with maximum- and minimum throughput. The graphs in Figure 4.9 also show that our approach is able to close the gap between the CWs of the highest- and lowest throughput nodes which is another indication of improved fairness.

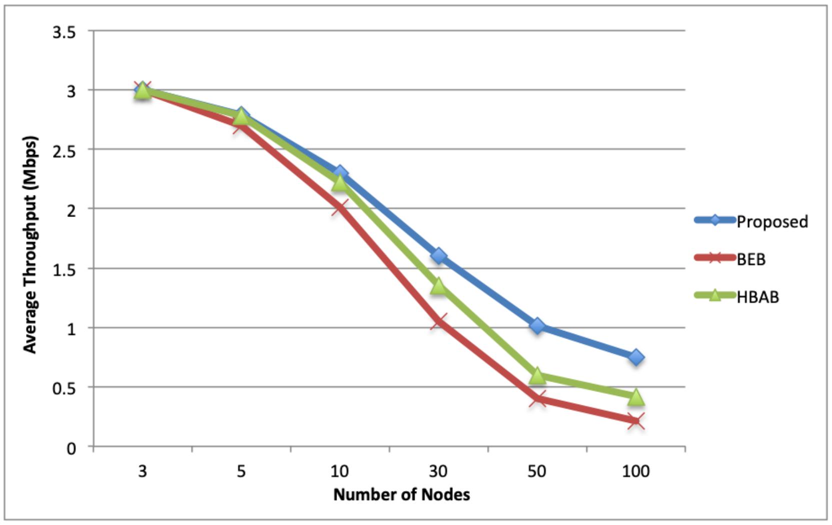


(a) Average throughput

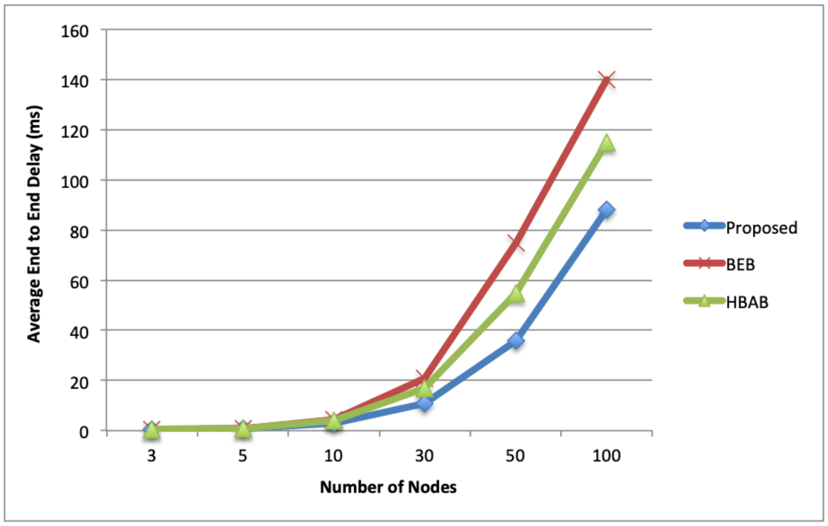


(b) Average delay

Figure 4.6: Average throughput and delay as a function of the number of nodes for synthetic data in ad-hoc scenarios

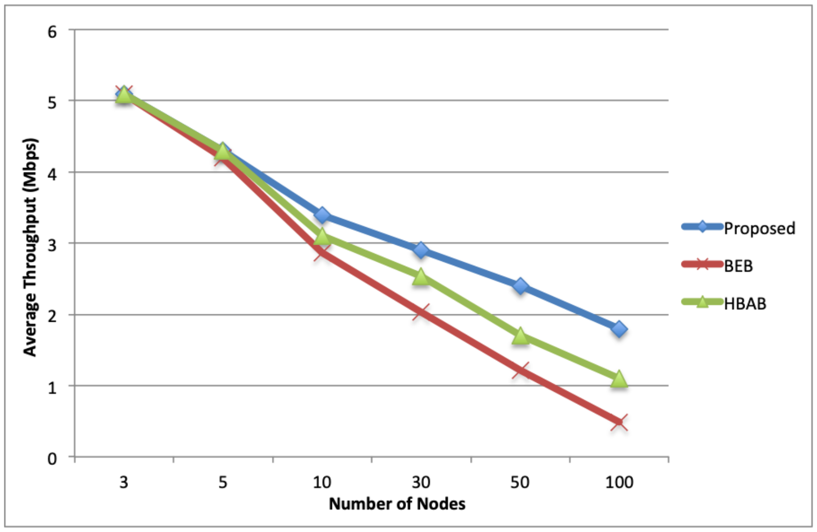


(a) Average throughput

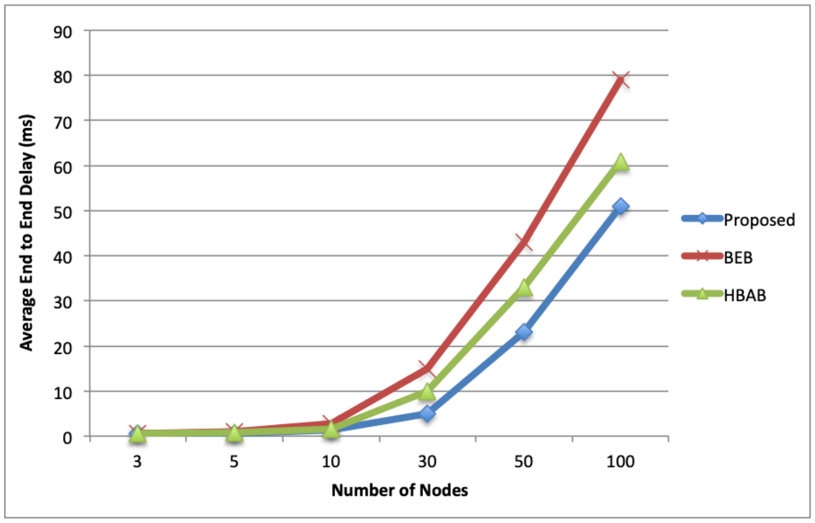


(b) Average delay

Figure 4.7: Average throughput and delay as a function of the number of senders for hot-spot data in ad-hoc scenarios

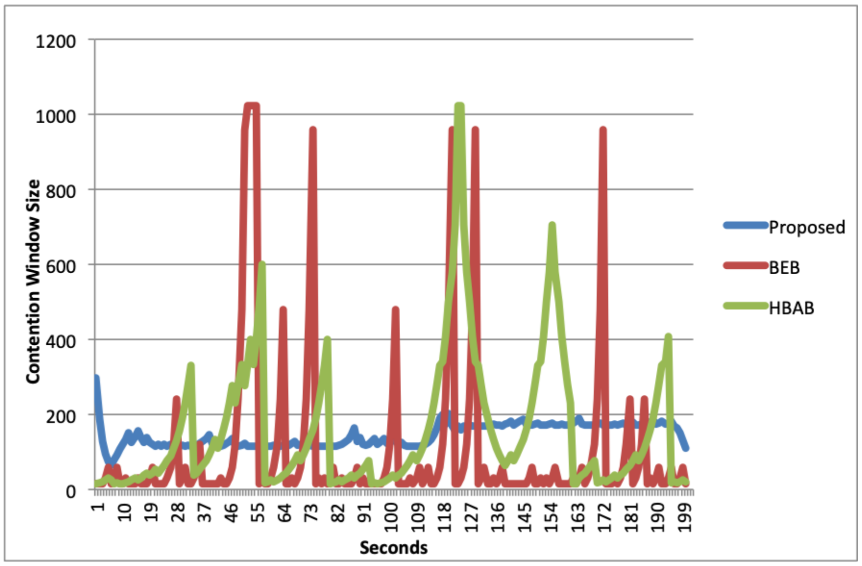


(a) Average throughput

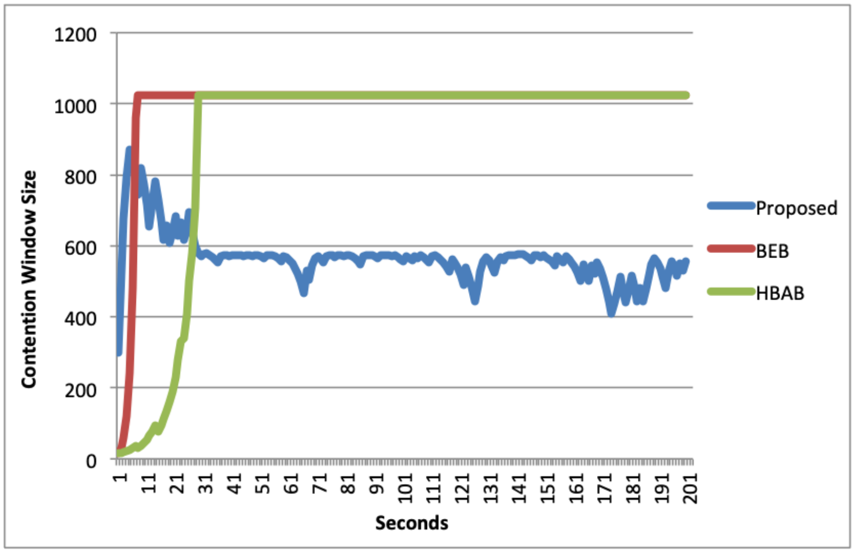


(b) Average delay

Figure 4.8: Average throughput and delay as a function of the number of nodes for company data in ad-hoc scenarios



(a) Maximum throughput



(b) Minimum throughput

Figure 4.9: Contention window size variation over time for the nodes with minimum and maximum throughput for synthetic trace in ad-hoc scenario with 100 senders

Chapter 5

Conclusion

In this thesis, we introduced SENSE (Smart Experts for Network State Estimation) a novel network state predictor based on a simple, yet efficient machine learning technique called Fixed-Share. SENSE improves the Fixed-Share algorithm by employing Exponentially Weighted Moving Average (EWMA)-based "smart" experts, META-learning, and Level-shift techniques. Our experiments on both synthetic and real datasets confirm that SENSE can automatically adapt to fluctuations of different time scales, which sets it apart from "static" techniques such as "pure" EWMA and Fixed-Share.

Then, we conducted an empirical characterization of IEEE 802.11's RTS/CTS performance as a function of packet size, transmission rate, and network contention. Based on our RTS/CTS performance characterization, we proposed a novel algorithm that dynamically decides whether to enable or disable RTS/CTS based on current network conditions and characteristics. Through simulations using a variety of WLAN network scenarios, we showed that the proposed algorithm consistently outperforms current best practice approaches which either do not enable RTS/CTS at all or use a static value of the RTS Threshold (RT) to decide whether to switch RTS/CTS on or off.

At the end, we introduced a modified version of a well-known machine learning technique and apply it to dynamic tuning of the contention window size of IEEE 802.11's DCF. Our method can set the CW wisely based on the history of transmitted packets. Our results confirm that the proposed technique outperforms the BEB algorithm of IEEE 802.11.

Bibliography

[1] IEEE Standard 802.11. Wireless lan medium access control(mac) and physical layer (phy) specifications. November 1999.

[2] SM Rifat Ahsan, Mohammad Saiful Islam, Naeemul Hassan, and Ashikur Rahman. Packet distribution based tuning of rts threshold in ieee 802.11. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1–6. IEEE, 2010.

[3] Maali Albalt and Qassim Nasir. Adaptive backoff algorithm for ieee 802.11 mac protocol. *International Journal of Communications, Network and System Sciences*, 2(04):300, 2009.

[4] Hatm Alkadeki, Xingang Wang, and Michael Odetayo. Improving performance of ieee 802.11 by a dynamic control backoff algorithm under unsaturated traffic loads. *arXiv preprint arXiv:1601.00122*, 2016.

[5] Khaled Hatem Almotairi. Inverse binary exponential backoff: Enhancing short-term fairness for ieee 802.11 networks. In *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*, pages 1–5. VDE, 2013.

[6] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. Macaw: a media access protocol for wireless lan's. *ACM SIGCOMM Computer Communication Review*, 24(4):212–225, 1994.

[7] Giuseppe Bianchi and Ilenia Tinnirello. Kalman filter estimation of the number of competing terminals in an ieee 802.11 network. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 844–852. IEEE, 2003.

[8] Gustavo Carneiro. Ns-3: Network simulator 3. In *UTM Lab Meeting April*, volume 20, pages 4–5, 2010.

[9] Nicolo Cesa-Bianchi, Yoav Freund, David Haussler, David P Helmbold, Robert E Schapire, and Manfred K Warmuth. How to use expert advice. *Journal of the ACM (JACM)*, 44(3):427–485, 1997.

[10] P Chatzimisios, AC Boucouvalas, and V Vitsas. Optimisation of rts/cts handshake in ieee 802.11 wireless lans for maximum performance. In *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, pages 270–275. IEEE, 2004.

[11] Jing Deng, Pramod K Varshney, and Zygmunt J Haas. A new backoff algorithm for the ieee 802.11 distributed coordination function. 2004.

[12] Yalda Edalat, Jong-Suk Ahn, and Katia Obraczka. Network state estimation using smart experts. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 11–19. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[13] Yalda Edalat, Jong Suk Ahn, and Katia Obraczka. Smart experts for network state estimation. *IEEE Trans. Network and Service Management*, 13(3):622–635, 2016.

[14] Yalda Edalat, Katia Obraczka, and Jong-Suk Ahn. Dynamically tuning ieee 802.11's contention window using machine learning. In *Proceedings of the 22nd ACM International Conference on Modeling,Analysis and Simulation of Wireless and Mobile Systems,*. ACM, 2019.

[15] Yalda Edalat, Katia Obraczka, and Bahador Amiri. A machine learning approach for dynamic control of rts/cts in wlans. In *Proceedings of the 15th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2018.

[16] Lassaad Gannoune. A non-linear dynamic tuning of the minimum contention window (cw min) for enhanced service differentiation in ieee 802.11 ad-hoc networks. In *2006 IEEE 63rd Vehicular Technology Conference*, volume 3, pages 1266–1271. IEEE, 2006.

[17] Lassaad Gannoune and Stephan Robert. Dynamic tuning of the contention window minimum (cw/sub min/) for enhanced service differentiation in ieee 802.11 wireless ad-hoc networks. In *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, volume 1, pages 311–317. IEEE, 2004.

[18] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer tcp throughput. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 145–156. ACM, 2005.

[19] David P Helmbold, Darrell DE Long, Tracey L Sconyers, and Bruce Sherrod. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications*, 5(4):285–297, 2000.

[20] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[21] Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.

[22] Yangchao Huang, Yujun Wang, Rui Zhu, Xihao Chen, and Qingwei Meng. Synchronized contention windows-based backoff algorithm in ieee 802.11 wireless networks. In *2016 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5. IEEE, 2016.

[23] J Stuart Hunter. The exponentially weighted moving average. *Journal of quality technology*, 18(4):203–210, 1986.

[24] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 25(1):157–187, 1995.

[25] Laura Huei-jiun Ju and Izhak Rubin. The effect of disengaging rts/cts dialogue in ieee 802.11 mac protocol. In *International Conference on Wireless Networks*, pages 632–638. Citeseer, 2003.

[26] B Kalantari-Sabet, Majlinda Mjeku, Nathan J Gomes, and John E Mitchell. Performance impairments in single-mode radio-over-fiber systems due to mac constraints. *Journal of Lightwave Technology*, 26(15):2540–2548, 2008.

[27] Phil Karn et al. Maca-a new channel access method for packet radio. In *ARRL/CRRL Amateur radio 9th computer networking conference*, volume 140, pages 134–140. London, Canada, 1990.

[28] MA Khan, Tazeem Ahmad Khan, and MT Beg. Rts/cts mechanism of mac layer ieee 802.11 wlan in presence of hidden nodes. *International Journal of Engineering and Innovative Technology (IJEIT) Volume*, 2, 2012.

[29] Hyung Joo Ki, Seung-Hyuk Choi, Min Young Chung, and Tae-Jin Lee. Performance evaluation of binary negative-exponential backoff algorithm in ieee 802.11 wlan. In *International Conference on Mobile Ad-Hoc and Sensor Networks*, pages 294–303. Springer, 2006.

[30] Leonard Kleinrock and Fouad Tobagi. Packet switching in radio channels: Part i–carrier sense multiple-access modes and their throughput-delay characteristics. *IEEE transactions on Communications*, 23(12):1400–1416, 1975.

[31] Zhen-ning Kong, Danny HK Tsang, and Brahim Bensaou. Adaptive rts/cts mechanism for ieee 802.11 wlans to achieve optimal performance. In *Communications, 2004 IEEE International Conference on*, volume 1, pages 185–190. IEEE, 2004.

[32] Michael N Krishnan, Sofie Pollin, and Avideh Zakhor. Local estimation of probabilities of direct and staggered collisions in 802.11 wlans. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–8. IEEE, 2009.

[33] Michael N Krishnan and Avideh Zakhor. Throughput improvement in 802.11 wlans using collision probability estimates in link adaptation. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1–6. IEEE, 2010.

[34] Adlen Ksentini, Abdelhamid Nafaa, Abdelhak Gueroui, and Mohamed Naimi. Determinist contention window algorithm for ieee 802.11. In *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 4, pages 2712–2716. IEEE, 2005.

[35] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[36] Dong Lu, Yi Qiao, Peter A Dinda, and Fabian E Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 414–424. IEEE, 2005.

[37] Mariyam Mirza, Joel Sommers, Paul Barford, and Xiaojin Zhu. A machine learning approach to tcp throughput prediction. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 97–108. ACM, 2007.

[38] Mostafa Mjidi, Debasish Chakraborty, Naoki Nakamura, Kazuhide Koide, Atushi Takeda, and Norio Shiratori. A new dynamic scheme for efficient rts threshold handling in wireless networks. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 734–740. IEEE, 2008.

[39] B Nithya, A Justin Gopinath, Venkatesh Kameswaran, and P Yogesh. Optimized tuning of contention window for ieee 802.11 wlan. *International Journal of Engineering, Science and Technology*, 9(2):15–25, 2017.

[40] Bruno Astuto Arouche Nunes, Kerry Veenstra, William Ballenthin, Stephanie Lukin, and Katia Obraczka. A machine learning framework for tcp round-trip time estimation. *EURASIP Journal on Wireless Communications and Networking*, 2014(1):47, 2014.

[41] Qixiang Pang, Soung Chang Liew, Jack YB Lee, and S-HG Chan. A tcp-like adaptive contention window for wlan. In *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*, volume 6, pages 3723–3727. IEEE, 2004.

[42] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (aodv) routing. Technical report, 2003.

[43] Vladislav Petkov and Katia Obraczka. Collision-free medium access based on traffic forecasting. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–9. IEEE, 2012.

[44] Ashikur Rahman and Pawel Gburzynski. Hidden problems with the hidden node problem. In *Communications, 2006 23rd Biennial Symposium on*, pages 270–273. IEEE, 2006.

[45] Tetsuya Shigeyasu, Makoto Akimoto, and Hiroshi Matsuno. Throughput improvement of ieee802. 11dcf with adaptive rts/cts control on the basis of existence of hidden terminals. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 46–52. IEEE, 2011.

[46] Mohammad Shurman, Bilal Al-Shua'b, Mohammad Alsaedeen, Mamoun F Al-Mistarihi, and Khalid A Darabkh. N-beb: New backoff algorithm for ieee 802.11 mac protocol. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 540–544. IEEE, 2014.

[47] Harsukhpreet Singh, Amandeep Kaur, Anurag Sharma, and Vishal Sharma. Performance optimization of dcf-mac standard using enhanced rts threshold under impact of ieee 802.11 n wlan. In *Advanced Computing & Communication Technologies (ACCT), 2015 Fifth International Conference on*, pages 421–424. IEEE, 2015.

[48] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[49] Takuya Sugimoto, Nobuyoshi Komuro, Hiroo Sekiya, Shiro Sakata, and Kengo Yagyu. Maximum throughput analysis for rts/cts-used ieee 802.11 dcf in wireless multi-hop networks. In *International Conference on Computer and Communication Engineering (ICCCE'10)*, pages 1–6. IEEE, 2010.

[50] Ilenia Tinnirello, Sunghyun Choi, and Youngsoo Kim. Revisit of rts/cts exchange in high-speed ieee 802.11 networks. In *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a*, pages 240–248. IEEE, 2005.

[51] Yu Wang and Jose Joaquin Garcia-Luna-Aceves. Collision avoidance in multi-hop ad hoc networks. In *Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 145–154. IEEE, 2002.

[52] Kaixin Xu, Mario Gerla, and Sang Bae. How effective is the ieee 802.11 rts/cts handshake in ad hoc networks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 1, pages 72–76. IEEE, 2002.