Performance Modeling of IEEE 802.11 Wireless LANs with Stochastic Petri Nets

Armin Heindl and Reinhard German

Technische Universität Berlin, Prozeßdatenverarbeitung und Robotik, Franklinstr. 28/29, 10587 Berlin, Germany, e-mail: {heindl,rge}@cs.tu-berlin.de

Abstract

In 1997, IEEE standardized the physical layers and the medium access for wireless local area networks. This paper presents a performance study of the Distributed Coordination Function, the fundamental contention based access mechanism. Most performance studies adopt unchecked simplifying assumptions or do not reveal all details of the simulation model. We develop a stochastic Petri net model, which captures all relevant system aspects in a concise way. Simulation allows to quantify the influence of many mandatory features of the standard on performance, especially the backoff procedure, extended interframe spaces, and the timing synchronization function. We identify conditions when simplifying assumptions commonly used in analytical modeling are justified. Applying these conditions, we derive a more compact and analytically tractable model from the detailed model.

Key words: Wireless Local Area Networks, IEEE 802.11 Standard, Stochastic Petri Nets, Simulation and Analytical Studies.

1 Introduction

In the last decade, the market for wireless communications has explosively expanded worldwide in fields such as *cellular telephony*, *satellite communications*, and *wireless local area networks* (WLANs). Wireless LANs are becoming increasingly popular for data communication over small areas, where wiring for conventional networking is difficult or not economic. To enable interoperability among WLAN systems of different vendors, IEEE has released the final version of an international standard for WLANs in 1997 (see [6] for the latest edition). This 802.11 standard specifies in detail the *Medium Access Control* (MAC) and the *Physical* (PHY) layer for WLANs operating in the 2.4-2.5 GHz band at data rates of 1 or 2 Mbps. Work is also in progress for WLANs with higher bit rates [7].

The primary MAC scheme of the 802.11 protocol, referred to as *Distributed Coordination Function* (DCF), is a variant of *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) [12]. All stations of a cell contend for the same channel. A station wishing to transmit a data frame first senses the channel and transmits only, if the channel has been idle for a certain period. Otherwise, it defers the transmission according to a probabilistic backoff scheme in order to avoid collisions. The receiver of the data frame sends back an acknowledgment. Besides this two-way handshake, also a four-way handshake with two more frames is defined. In both cases, collisions can occur, if two or more stations start to transmit the initial frames of their handshakes at almost the same time. In general, the wireless medium exhibits much higher bit error rates than wired links resulting in increased requirements for the receiver. Efficient protocols have to take into account that usually a sender cannot listen to its own transmissions. Thus, collisions cannot be detected immediately as it is common in wired LANs (using CSMA/CD, CD = collision detection). Instead, the lack of a positive acknowledgment from the receiver indicates a failed transmission to the source station.

In this paper, we assume perfect channel sensing and ideal channel conditions, i.e., we will not deal with *signal fading* (short-term fluctuations of channel quality), the *near/far effect* (which allows for *capture*, the possibly successful reception of simultaneous transmissions), and *hidden terminals* (some stations cannot communicate directly with each other). Each station hears any other station. In this scenario, collisions only occur due to unfortunate contention resolution between stations with a transmission request. The critical parameter in this respect is the so-called *vulnerable period*, during which an ongoing transmission may be corrupted by other stations, because their MAC layers are unable to recognize that transmission due to their channel assessment time, the time to switch from receiving to transmitting and the air propagation time.

In the literature, the performance of the 802.11 protocol has mainly been evaluated by means of simulation [5,18]. In [3], also the problem of hidden terminals is addressed. Most proposed analytical models [2,1,3] adopt quite strict assumptions like simplified backoff rules [2,10] or saturation conditions [1]. In [1], the saturation throughput of the WLAN in DCF mode is analyzed approximately on the level of the stochastic process via an embedded Markov chain. Other features mandated by the standard like *Extended Interframe Spaces* (EIFS) and the *Timing Synchronization Function* (TSF) are ignored by all analytical approaches, while for most simulation studies their specific implementation – if carried out at all – is not outlined in publications. This paper extends a previously published model [10] by incorporating EIFS and TSF. By means of simulation, the impact of these mandatory features on the performance of the DCF medium access control is investigated for different system configurations (including the influence of various PHY parameters). We are

especially interested in the following questions: How much does system performance suffer from the overhead of the TSF? Do EIFS really improve system performance? And in which way is the impact of EIFS affected by the backoff procedure? The performance indices throughput and mean waiting time are determined. The obtained results also deliver conditions under which specific simplifications are justified or seem unreasonable. We apply these simplifications to obtain a more compact and analytically tractable model from the detailed one. Following an approach similar to the one in [16], this is achieved by folding the station subnets.

Our general modeling approach is based on *stochastic Petri nets* (SPNs, [14]), for describing the protocol. More specifically, we use the *stochastic Petri net language* (SPNL, [8]) with its ability to structure complex models into modules. So it is possible to capture all relevant details in a yet concise model. In general, SPNs are well suited to represent system aspects like concurrency and synchronization and are therefore practicable in in performance and dependability evaluation. Moreover, non-exponential timing as provided by *deterministic and stochastic Petri nets* (DSPNs, [15]) and *Markov regenerative stochastic Petri nets* (MRSPNs, [4]) allow to build more realistic models, while at the same time different tools support their efficient evaluation. In this paper, we use TimeNET [19] for the simulations and SPNica [9] for the analysis. In the past, SPNs have already been applied to contention-based protocols, like CSMA/CD in [16].

The paper is organized as follows. In Sect. 2, the IEEE 802.11 standard is briefly reviewed with emphasis on the DCF including the TSF. After the description of the detailed SPN model in Sect. 3, numerical experiments based on this model are presented in Sect. 4. They compare different MAC mechanisms and investigate the influence of system parameters. Based on the simulation data, the analytically tractable SPN model is derived from the detailed model in Sect. 5, with a comparison of corresponding results in Sect. 6. Concluding remarks are given in Sect. 7.

2 The IEEE 802.11 DCF

In this section, we will briefly point out the features of the IEEE 802.11 standard mandatory for a so-called *Independent Basic Service Set* (IBSS) with emphasis on the *Distributed Coordination Function* (DCF). An IBSS as the basic type of an IEEE 802.11 wireless LAN is an *ad-hoc network*, in which a finite number of stations – typically only a few – communicate directly in a peer-to-peer manner within the coverage area. Without an *access point* functioning as a hub, the centralized MAC protocol supporting contention free and time bounded services, the optional *Point Coordination Function* (PCF), can-

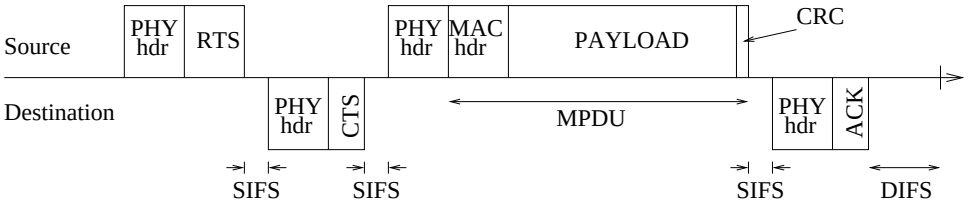


Fig. 1. A successful RTS/CTS handshake

not be employed. Thus, the only way to access the medium is the fundamental DCF mechanism.

Two versions of the DCF are defined: *Basic Access* (BA) based on two-way handshaking and *Request-To-Send/Clear-To-Send* (RTS/CTS) based on four-way handshaking. A station can use RTS/CTS for data frames exceeding a configurable threshold. In both cases only the first packet has to contend for the medium and the access is based on three time periods: the *DCF Interframe Space* (DIFS) and *Short Interframe Space* (SIFS) with SIFS < DIFS, as well as the much larger *Extended Interframe Space* (EIFS). Basic Access functions as follows:

(1) A station wishing to transmit a directed data frame senses the channel.
(2) If the channel has been idle for longer than a DIFS, it transmits the frame and waits for a positive acknowledgment (ACK).
(3) The station goes into backoff if
    (a) the channel has not been free for a period of a DIFS (or EIFS after a failed previous reception).
    (b) no ACK has arrived in time (corresponding to a collision).
    (c) the frame is consecutive to a previous transmission of the same station (to prevent channel seizure).

After the successful reception of the data frame, the receiving station waits for a period equal to SIFS and sends the ACK.

In RTS/CTS, two more packets are exchanged: the RTS packet contains information on the length of the upcoming data frame, the CTS packet is an acknowledgment for the RTS packet and contains the same information. The gaps between successive transmissions are equal to SIFS. Figure 1 shows a successful RTS/CTS handshake. From the information contained in both the RTS and CTS packets, the other stations can derive how long the medium will be busy. The intentions of using the more complex RTS/CTS mechanism are twofold. First, since only the short RTS contends for the medium, collisions (indicated by the lack of the CTS response) waste less bandwidth. RTS/CTS may therefore be viewed as a *virtual collision detection* mechanism. Second, additional collisions caused by hidden terminals are avoided with the RTS/CTS exchange.

In both handshakes, there are two ways for a station to discover that its transmission has failed: either it does not receive the expected response frame (ACK or CTS, respectively) within a specified timeout, or it recognizes the transmission of a different packet on the channel after the completion of its own transmission. In any case, the notice of such a collision invokes the backoff procedure.

A slotted binary exponential backoff scheme is applied. As soon as the channel is monitored idle for a DIFS (after a successful reception) or for an EIFS (after a failed reception), the station selects a backoff time composed of a random number (*backoff counter*) of slot times. With no medium activity indicated for the duration of such a slot of size $aSlotTime$, the backoff counter is decremented. It is temporarily suspended or "frozen" for periods when a transmission is detected on the channel. Before the backoff procedure is resumed, the channel must have been sensed idle for the duration of a DIFS or EIFS period, as appropriate. Whenever the backoff counter reaches zero, the transmission commences. The backoff time is chosen as follows:

$$BackoffTime = BackoffCounter \times aSlotTime,$$

where $BackoffCounter$ is a uniformly distributed integer in $[0, CW]$. The contention window CW is an integer equal to $(aCWmin + 1) \cdot 2^{bc} - 1$, where $aCWmin$ is the initial value and bc is a variable which is initialized with zero and incremented before a repeated backoff procedure for a pending frame. The quantity bc can grow up to a maximum value $bcmax$ corresponding to $aCWmax$; afterwards it remains unchanged until it is reset to zero by the next successful transmission. In practice, a frame is only retransmitted for a limited number of times ($aShortRetryLimit$ for BA, $aLongRetryLimit$ for RTS/CTS) before being cancelled. In this paper, the retry limits will be set to infinity.

Figure 2, adapted from [1], illustrates the backoff procedure. Two stations A and B among others share the same wireless channel. After the end of a previous transmission, station B waits for a DIFS and then chooses a backoff counter equal to 6 before sending the consecutive packet. Meanwhile, a new frame is generated at station A. Station A senses the channel idle for a DIFS and transmits the frame. The transmission of station A occurs in the middle of the slot corresponding to a backoff counter equal to 3 for station B. During the transmission, station B stops decrementing its backoff counter. It is decremented again when the channel has been sensed idle for a DIFS.

A critical parameter neglected in Fig. 2 is the *vulnerable period*, during which an ongoing transmission may be corrupted by other stations, because their MAC layers can only recognize that transmission with some delay. It consists of the air propagation time ($aAirPropagationTime$), the time the receiver needs to assess the medium ($aCCATime$) and to deliver its state to the MAC, and the time required to change from the receiving to the trans-

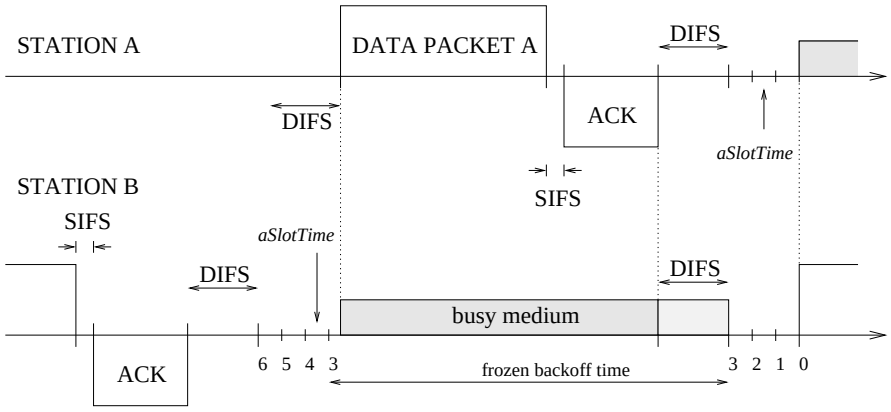


Fig. 2. The backoff procedure with BA

Table 1

PHY-dependent parameters	**DSSS**	**FHSS**
$aSlotTime$	20 μs	50 μs
$aCCATime$	$\leq$ 15 μs	27 μs
$aRxTxTurnaroundTime$	$\leq$ 5 μs	20 μs
$SIFS$	10 μs	28 μs
$DIFS$	50 μs	128 μs
$EIFS$	1148 μs	1180 μs
$aCWmin$	31	15
$aCWmax$	1023	1023
$PHYHeader$	192 bits	128 bits
$BEACON$	808 bits	840 bits
$MaxFrameBody$	8157 bytes	4061 bytes

mitting state ($aRxTxTurnaroundTime$). Generally, $aSlotTime$ should be at least as large as the vulnerable period. Since cells of a few hundreds of meters are considered, the propagation time is small compared with the other times. They depend on the physical layer, just like all interframe spaces (SIFS, DIFS, EIFS). The standard specifies three different physical layers (PHYs): the *Direct Sequence Spread Spectrum* (DSSS), *Frequency Hopping Spread Spectrum* (FHSS), and *Infrared* (IR). For DSSS and FHSS, the PHY specific values derived from the standard are given in Table 1, where *PHYHeader* comprises the PLCP preamble and header, which are added to the *MAC protocol data unit* (MPDU) to aid in demodulation and delivery at the receiver. For the *Timing Synchronization Function* to be explained below, the management frame BEACON with the given number of bits (excluding the *PHYHeader*) is used, while *MaxFrameBody* denotes the maximum payload of a data packet.

The following parameters are independent of the physical layer: $aAirPropagationTime$=1 μs (by assuming a maximum distance of 200 m), $MACHeaderCRC$=272 bits, ACK=112 bits, RTS=160 bits, CTS=112 bits, $TimeOut$=300 μs, $aBeaconPeriod$=100000 μs, minimum bit rate 1 Mbps, possible bit rates B=1 and B=2 Mbps. The PLCP preamble and header are always transmitted with the minimum bit rate and the MPDU with one of the possible bit rates. The quantity $MACHeaderCRC$ contains both the MAC header and the frame check sequence CRC, which flank the payload of a data packet, as shown in Fig. 1. The values of $TimeOut$ and $aBeaconPeriod$ are not prescribed by the standard. $TimeOut$ might be different for ACK and CTS frames, but it should not be too close to the minimum response time of the destination station. Draft versions of the standard suggested the given value of $aBeaconPeriod$ as a default, which defines the period between contiguous target beacon transmission times (see below).

Besides RTS, CTS, ACK, and data frames, implementations compliant with the IEEE 802.11 protocol must support the transmission and reception of many other MAC frames – especially management frames, like probe requests, probe responses, etc. However, in the considered scenario, such frames actually never occur on the channel and thus do not affect system performance. As an exception, special frames called *beacons* are periodically transmitted in order to keep the local timers for all stations in the WLAN synchronized. The resulting reduction in available bandwith is generally assumed to be small, but has not yet been quantified to the authors' knowledge. In an IBSS, all stations participate in a distributed algorithm called the *Timing Synchronization Function* (TSF), which regulates beacon generation and transmission. Each station maintains its own timer, which is updated according to the information conveyed in the received beacons. A series of target beacon transmission times (TBTTs) is defined by instants exactly $aBeaconPeriod$ time units apart starting from zero. At each TBTT, a station shall ([6], page 124)

(1) suspend the decrementing of the backoff counter for any pending non-beacon transmission,
(2) choose a random delay uniformly distributed in the discrete range between zero and $2 \cdot aCWmin \cdot aSlotTime$ with a step size $aSlotTime$,
(3) wait for the period of the sampled delay, decrementing the corresponding delay counter using the same algorithm as for backoff,
(4) cancel the remaining random delay and the pending beacon transmission, *if* a beacon arrives before the random delay has expired, *or*
(5) send the beacon, *if* the random delay has expired and no beacon has arrived during the delay.

Thus, when a beacon has been either sent or successfully received, the MAC control returns to the suspended non-beacon transmission or to the idle state

awaiting other transmission requests or reception indications. A beacon may also collide. As a broadcast message, a beacon is not acknowledged so that in ideal channel conditions its sender could only detect its collision with a data frame (!) whose transmission lasts beyond the end of the beacon transmission. Although the invocation of the beacon generation function may be delayed due to ongoing transmissions at the TBTT, subsequent beacons shall be scheduled at the next TBTT, which is fixed a priori.

In the model presented in the following section, we will ignore most of the optional features of the standard (like power management and fragmentation of large MSDU). But we point out that the model completely describes the performance aspects of an IBSS which meets the minimum requirements to be called compliant with the IEEE 802.11 standard. In ideal channel conditions (and without the optional fragmentation of data packets), the virtual carrier sense mechanism defined in the IEEE 802.11 standard, referred to as the *Network Allocation Vector* (NAV), does not affect system behavior.

3 A detailed SPN model

A detailed SPNL model of the dynamics of the DCF including the TSF is developed. In SPNL, common elements, like *places* (represented as circles), input, output and inhibitor *arcs*, indistinguishable *tokens* (dots or parameters in places), timed and immediate *transitions*, are used as in ordinary SPNs. The firing time distributions may either be exponential (exponential transitions, represented as empty rectangles), deterministic (deterministic transitions, black rectangles), or general (general transitions, dashed rectangles). Transitions, which are never preempted, are referred to as *persistent*. For non-exponential non-persistent transitions, a firing policy has to be defined. In this paper, we adopt the two policies *preemptive resume* (prs) and *preemptive repeat different* (prd) for different transitions [17]. These terms mean that, when such a transition is disabled without having fired, its already performed work is maintained or lost, respectively. We also use *guards* and marking-dependent *arc multiplicities*. Such marking-dependent expressions, also called *rate rewards*, are formulated with respect to the number of tokens in places (`#P` denotes the number of tokens in place `P`). In SPNL, arc multiplicities as well as all identifiers are located in the proximity of the corresponding objects, while all other expressions, like transition attributes (guard, policy, distribution), are listed below the graphical representation of the net. SPNL allows to structure SPNs hierarchically, similar to the module concept of programming languages like Modula-2 or Ada. The building blocks are *processes* or *processtypes*. *Processtypes* may be instantiated and parameterized. *Rewards* and *ports* declared in the public part (between the boldfaced keywords **process/processtype** and **private**) serve to connect the separate pieces of the

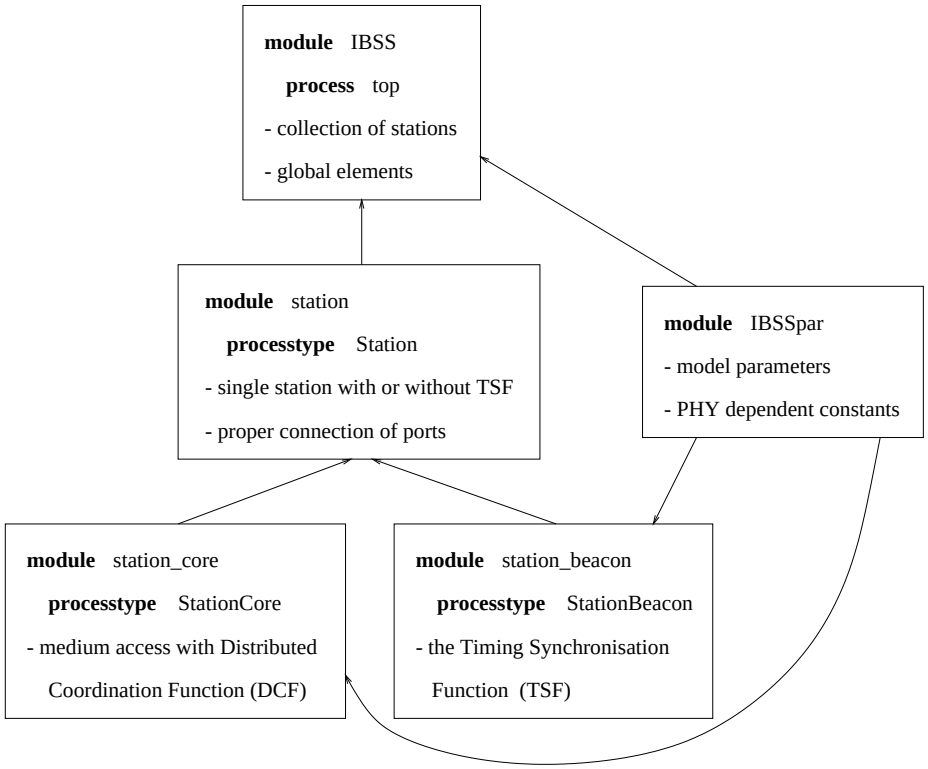


Fig. 3. Hierarchal structure of SPNL modules

model. *Ports*, shown as small filled squares in the graphical area, merely are the intersection points of cut-through arcs.

The complete SPNL model of the IBSS consists of five modules. Their hierarchal structure is depicted in Fig. 3, where an arc from one module to another means that the declarations of the source module may be used in the target module (see keyword **use** in the SPNL model). In the boxes, which represent the modules, the names of the enclosed **process** or **processtype**s (if any) are given together with a short functional description.

Module *IBSSpar* in Fig. 4 simply contains the list of parameters used in the definitions of the other modules (based on Table 1 and PHY-independent constants). The three modules *IBSS* (Fig. 5), *station_core* (Fig. 7), and *station_beacon* (Fig. 8) can be considered as the key parts of the model, forming the top (*IBSS*) and bottom (*station_core*, *station_beacon*) level in the SPNL hierarchy. These parts will be described in more detail in the following paragraphs. Module *station* in Fig. 6 represents the instance of a single station, but actually merely serves as an intermediate hierarchy level. The enclosed processtype *Station* properly connects the two instances of *StationCore* and *StationBeacon* (see Figs. 7 and 8) with each other and with process *top* (see Fig. 5), while passing rate rewards down to the two process instances and measures up to the process *top*.

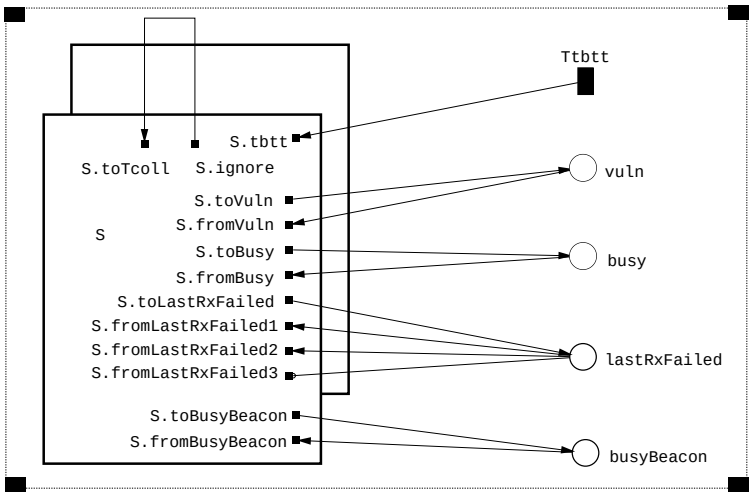
```
module IBSSpar;

(* all times in µs *)

parameter B = 2; (* BitRate in Mb/s, possible values: 1 or 2*)
          MinBitRate = 1;

(* model parameters *)
parameter V=1.0; (* virtual load *)
          K=2;   (* max number of packets in MAC layer *)
          N=10;  (* number of stations *)

          MaxFrameBody=8157*8; (* bits, specified by DSSS, for FHSS: 4061*8 *)
          L = MaxFrameBody/2;  (* mean length of FrameBody, here: uniform distribution *)
          λ = V * B / (N * L);

(* physical medium dependent: DSSS *)
parameter aSlotTime=20;          (* FHSS: 50 *)
          aCCATime=14;           (* FHSS: 27 *)
          aRxTxTurnaroundTime=4; (* FHSS: 20 *)
          SIFS=10;               (* FHSS: 28 *)
          DIFS=50;               (* FHSS: 128 *)
          EIFS=1148;             (* FHSS: 1180 *)
          aCWmin=31;             (* FHSS: 15 *)
          aCWmax=1023; bcmax=5;  (* aCWmax=2^bcmax*(aCWmin+1)-1*)
                                 (* FHSS: bcmax=6 *)
          PHYHeader=192;         (* bits, FHSS: 128 *)
          BEACON = 808;          (* bits, FHSS: 840 *)

(*physical medium independent*)
parameter aAirPropagationTime = 1;
          MACHeaderCRC = 272; (* bits, MAC header + CRC*)
          ACK = 112;          (* bits *)
          RTS = 160;          (* bits *)
          CTS = 112;          (* bits *)
          TimeOut = 300;
          aBeaconPeriod = 100000;

(*--- transmission times ---*)
parameter PHYHeaderTransTime = PHYHeader/MinBitRate;
             (* PHYHeader is always transmitted with minimum bit rate*)
          MACHeaderCRCTransTime = MACHeaderCRC/B;
          ACKTransTime = ACK/B;
          RTSTransTime = RTS/B;
          CTSTransTime = CTS/B;
          BeaconTransTime = BEACON/B;

(* transmission times for Basic Access in case of success/collision *)
parameter vulnerablePeriod = aAirPropagationTime+aCCATime+aRxTxTurnaroundTime;
          collConstTransTime = PHYHeaderTransTime+MACHeaderCRCTransTime+DIFS-aCCATime;
             (* for RTS/CTS: = PHYHeaderTransTime+    RTSTransTime     +DIFS-aCCATime *)
          collMaxBodyTransTime = MaxFrameBody/B;
             (* for RTS/CTS:   = 0 *)
          succConstTransTime = 2*PHYHeaderTransTime+MACHeaderCRCTransTime+SIFS
                               +aAirPropagationTime+ACKTransTime+DIFS-aCCATime;
             (* for RTS/CTS: = 4*PHYHeaderTransTime+RTSTransTime+CTSTransTime
                               +MACHeaderCRCTransTime+ACKTransTime+3*SIFS+
                               +3*aAirPropagationTime+DIFS-aCCATime *)
          succMaxBodyTransTime = MaxFrameBody/B;

end IBSSpar.
```

Fig. 4. SPNL module for the parameters of the WLAN (DSSS, BA)

In Fig. 5, which shows the top hierarchy level of the IBSS, the shadowed rectangle stands for N instances of the processtype *Station* corresponding to the N stations in the IBSS [1]. Each station is connected with the deterministic transition `Ttbtt` and the four places below in an identical manner via ports. In addition, the arc from port `S.ignore` to port `S.toTcoll` establishes a connection from each station to all other stations including the source station itself. The DCF dynamics of the stations is also influenced by means of rate rewards defined at the bottom of Fig. 5 (following the keyword **rreward**). Each station imports them as input parameters. The parameters K and λ are set in *IBSSpar* and denote the maximum number of frames a station can hold

[1] Their spatial distribution may be neglected, since the propagation delay is short compared with other parts of the vulnerable period.

```
module IBSS; (* N stations with beacon modeling *)
  use IBSSpar, station;
process top;
  smeasure throughput, waitingTime;
private
  rreward Nvuln, Nbusy, NlastRxFailed, NbusyBeacon;
  processinstance S: array [1..N] of Station(K,λ,Nvuln,Nbusy,NlastRxFailed,NbusyBeacon);
```




```
  transatt Ttbtt: dist = det(aBeaconPeriod);
  rreward  Nvuln = #vuln;
           Nbusy = #busy;
           NlastRxFailed = #lastRxFailed;
           NbusyBeacon = #busyBeacon;
  smeasure throughput  = ArithmeticMean(S.throughput) * N; (* = sum of throughputs *)
           waitingTime =  ArithmeticMean(S.waitingTime);
  end top;
end IBSS.
```

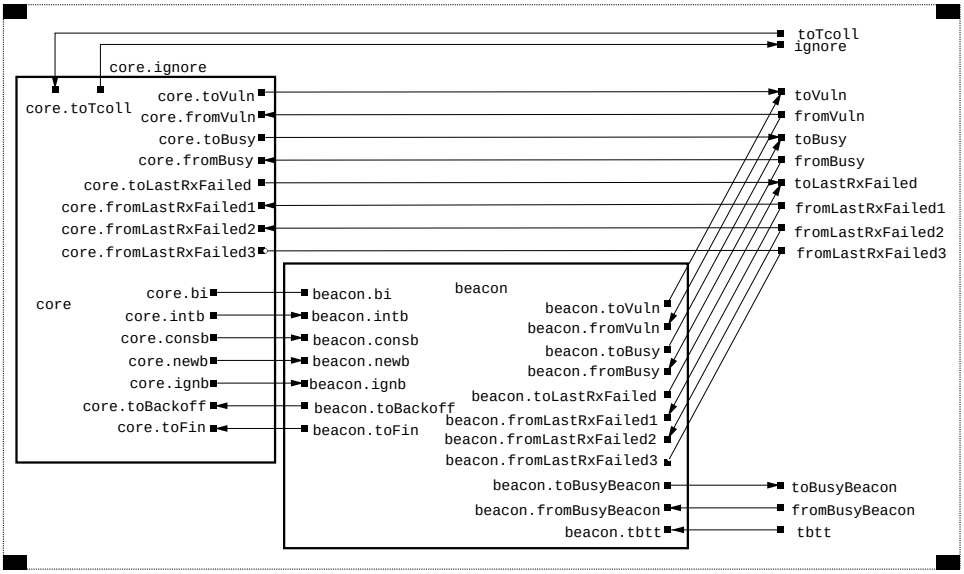
Fig. 5. The top layer of the WLAN model

in its MAC layer and the frame arrival rate at a single station, respectively. The marking of places `vuln`, `busy`, and `busyBeacon` can be interpreted as the state of the channel, while tokens in place `lastRxFailed` indicate a potential EIFS period to the stations. Transition `Ttbtt` with the deterministic firing time *aBeaconPeriod* is always enabled and fires at every target beacon transmission time. For ideal local timers, these TBTTs coincide for all stations. Of course, this assumption actually turns synchronization obsolete. But we are interested in how the overhead of sent beacons affects the system performance.

Now, we will describe the lowest level in the SPNL hierarchy of the model depicted in Figs. 7 and 8. The meanings of places and transitions in Figs. 5 and 7 are briefly summarized in Table 2. Processtype *StationCore* in Fig. 7 models the DCF without TSF and is instantiated once for every station (in module *station*). The ports on the right-hand side provide the connection to the places in process *top* (via processtype *Station* in module *station* of Fig. 6).

Place `free` models the free buffer places for frames in the MAC layer of the station. Initially, the complete buffer of size K is available. Exponential transition `gen` models the arrival of data units from higher protocol levels and place `wait` models buffered frames waiting for transmission. The remaining part of the SPN models the channel access and transmission of frames and may thus be viewed as the service unit of this queue.

```
module station; (* station with beacon modeling *)
  use station_core, station_beacon;

  processtype Station(param K, λ:    rreward  Nvuln, Nbusy, NlastRxFailed, NbusyBeacon);
  port toVuln, toBusy, toLastRxFailed, toBusyBeacon:  t->p;
       fromVuln, fromBusy, fromBusyBeacon, fromLastRxFailed1, fromLastRxFailed2, toTcoll:   t<-p;
       fromLastRxFailed3:  to-p;
       tbtt:  p<-t;
       ignore: p->t;
  smeasure throughput, timeInSystem;
private
  processinstance core: StationCore(K,λ, Nvuln, Nbusy, NlastRxFailed);
  processinstance beacon: StationBeacon(Nvuln, Nbusy, NbusyBeacon, NlastRxFailed, core.NignE);
```




```
  smeasure throughput = core.throughput;
           timeInSystem = core.timeInSystem;

  end Station;
end station.
```

Fig. 6. SPNL module for a single station with TSF

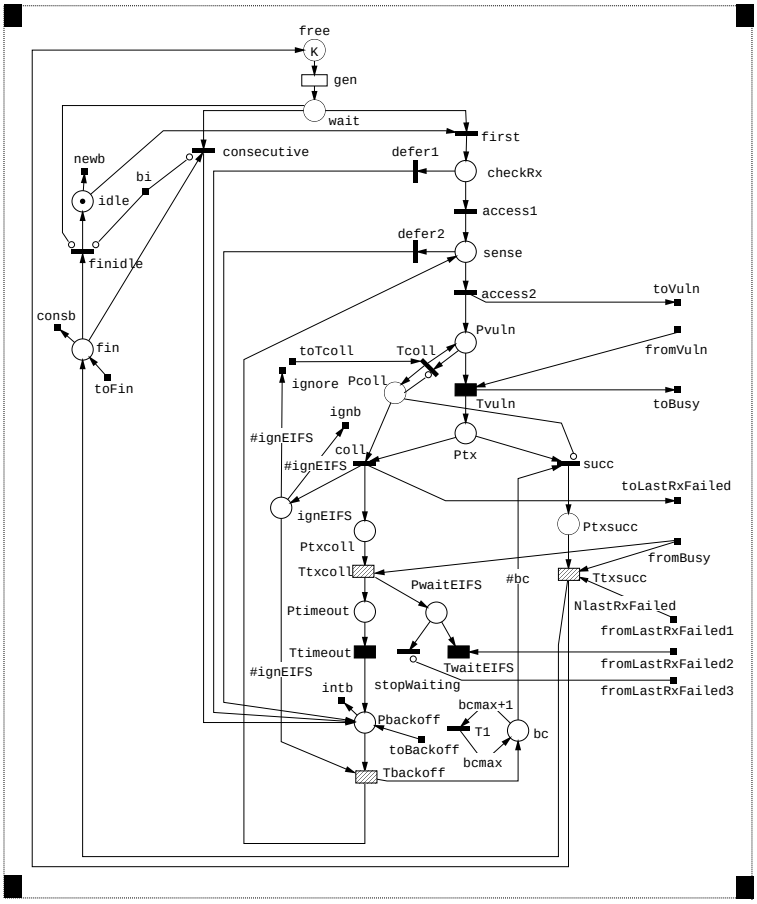
Tokens in `wait` may enter the lower subnet, when a token is in place `idle` or `fin`. If the station has not accessed the channel for a while and does not perform the beacon generation function, a token is in `idle`. Immediate transition `first` can fire and puts a token in place `checkRx`. Otherwise immediate transition `consecutive` fires and puts a token in place `Pbackoff` (start of backoff procedure). In `checkRx`, the station checks whether within the past period of an EIFS the end of a failed reception was indicated to its MAC. As will become clear later, this condition holds, if there is a token in place `lastRxFailed` of Fig. 5. According to the assigned guards (see **transatt** in Fig. 7), one of the two immediate transitions `defer1` and `access1` is enabled. The former puts a token in place `Pbackoff` (start of backoff procedure) and the latter in place `sense` (sensing of channel). Similarly, in place `sense`, a decision – now between `defer2` and `access2` according to guards based on place `busy` of Fig. 5 – may lead to the start of the backoff procedure (place `Pbackoff`). Tokens in place `busy` represent the number of perceptible ongoing transmissions. In case of an idle medium, transition `access2` puts a token into place `Pvuln` (start of transmission in its vulnerable period). If in this period any other station accesses the medium, a collision occurs. The number of transmissions in the vulnerable period is represented by tokens in place `vuln` of Fig. 5. A collision

Table 2

SPN element(s) in Figs. 5 and 7	**interpretation**
`Ttbtt`	timer for *aBeaconPeriod*
`busy`	number of stations transmitting perceptibly for others
`vuln`	number of stations in vulnerable period
`lastRxFailed`	EIFS period after collided frame has not yet elapsed
`busyBeacon`	number of stations transmitting a beacon perceptibly
`free`	free buffer places in a station
`gen`	arrival of payload to MAC layer
`wait`	buffered frames
`consecutive`, `first`	consecutive or first transmission
`checkRx`	check last received frame
`access1`	proceed to access medium
`sense`	sensing the channel
`access2`	start to access medium
`defer1`, `defer2`	defer access, go into backoff
`Tvuln`, `Pvuln`	vulnerable period
`Tcoll`, `Pcoll`	frame collision
`Ptx`	station transmits perceptibly for others
`coll`, `succ`	transmission will collide or succeed
`Ttxcoll`, `Ttxsucc`	durations of collided and successful transmissions
`Ptimeout`, `Ttimeout`	ACK or CTS timeout
`Pbackoff`, `Tbackoff`	backoff procedure
`bc`, `T1`	backoff counter with upper limit
`ignEIFS`	this station was involved in last collision
`fin`	end of medium access after successful frame exchange
`finidle`	no consecutive transmission
`idle`	station has no pending frame to transmit

corresponds to more than one token in `vuln`. In this case immediate transition `Tcoll` fires and puts a token in place `Pcoll`, which serves as a memory that a

```
module station_core; (* core part of station, here Basic Access *)
  use IBSSpar;
  processtype StationCore( param K, λ:   rreward  Nvuln, Nbusy, NlastRxFailed);
  (* ports to top model *)
  port toVuln, toBusy, toLastRxFailed:  t->p;
       fromVuln, fromBusy, fromLastRxFailed1, fromLastRxFailed2, toTcoll:   t<-p;
       fromLastRxFailed3:  to-p;
       ignore: p->t;
  (* ports to StationBeacon *)
  port bi: to-p; (* beacon indicate *)
       intb, consb, newb:  p->t;
       ignb:  p->t;
       toBackoff, toFin:  p<-t;
  smeasure throughput, waitingTime;
  rreward NignE;
private
```




```
  smeasure throughput = ( E{#gen}*L)/B;
           waitingTime = (K -  E{#free})/ E{#gen};
  rreward  NignE = #ignEIFS;
  transatt gen:       dist = exp(λ);
           Tvuln:     dist = det(vulnerablePeriod);
           Ttxcoll:   dist = uniform(collConstTransTime, collConstTransTime+collMaxBodyTransTime);
           Ttxsucc:   dist = uniform(succConstTransTime, succConstTransTime+succMaxBodyTransTime);
           Ttimeout:  dist = det(TimeOut-DIFS);
           TwaitEIFS: dist = det(EIFS-DIFS),
                      policy = prd;
           Tbackoff:  dist = discreteuniform(0,((2^#bc)*(aCWmin+1)-1)*aSlotTime,aSlotTime),
                      guard = (#ignEIFS=0 and NlastRxFailed=0 and Nbusy=0) or (#ignEIFS>0 and Nbusy=0),
                      policy = prs;
           defer1:    guard = NlastRxFailed > 0;
           access1:   guard = NlastRxFailed = 0;
           defer2:    guard = Nbusy > 0;
           access2:   guard = Nbusy = 0;
           Tcoll:     guard = Nvuln > 1;
  end StationCore;
end station_core.
```

Fig. 7. The core of the DCF

collision has occurred. Note that this collision detection and collision memory is a "model artifact"; in the real network, the station does not know that a collision has occurred. The duration of the vulnerable period is represented by the deterministic transition `Tvuln`.

Depending on whether a token is in `Pcoll` or not, either the immediate transition `coll` or `succ` fires, which enables either general transition `Ttxcoll` or `Ttxsucc`. The firing time of transition `Ttxsucc` reflects the duration of the successful frame exchange sequence of the employed technique (either BA or RTS/CTS), including the required interframe spaces: SIFS between frames and DIFS after the concluding ACK. The firing time of transition `Ttxcoll` takes into account only the frame, which contended for the medium (data packet or RTS, respectively) plus DIFS. In both cases $aCCATime$, which is a component of the vulnerable period, must be subtracted from the firing times of `Ttxcoll` and `Ttxsucc`. Note that appending DIFS at the end of the transmission time simplifies the modeling of channel sensing: otherwise transition `access2` would have to be timed. The firing of `Ttxsucc` puts a token back to places `free` and `fin`, and the firing of `Ttxcoll` puts a token into places `Ptimeout` and `PwaitEIFS`.

Deterministic transition `Ttimeout` models the ACK or CTS timeout, as appropriate, and forwards the token to place `Pbackoff`. The general transition `Tbackoff` together with its complex guard models the backoff procedure. Transition `Tbackoff` is a non-persistent transition and its firing policy is set to prs, in order to correctly model the temporarily "frozen" backoff timer. Before sampling from its discrete uniform distribution, the guard must evaluate to *true*. The guard involves three places: "station" place `ignEIFS` and the two "system" places `busy` and `lastRxFailed`. A token in `ignEIFS` simply tells `Tbackoff` that it should ignore any EIFS period indicated by tokens in `lastRxfailed`, since this station did not receive any corrupted frames. On the contrary, the station itself was sending one of the collided frames. In this case (`#ignEIFS` > 0), the firing time of `Tbackoff` is sampled or decremented, as soon as the medium has been sensed idle for a DIFS (`#busy` $= 0$). In the other case (`#ignEIFS` $= 0$), the station must additionally check that an EIFS period has passed (`#lastRxFailed` $= 0$).

After firing, a token is put back to place `sense` and the backoff counter represented by place `bc` is updated. The marking-dependent multiplicity of the arc from `ignEIFS` to `Tbackoff` empties place `ignEIFS` in any case. The token in `ignEIFS` for ignoring the EIFS period is also removed by the firing of transition `Tcoll` in *any* station (see ports `ignore` and `toTcoll`, which are connected in Fig. 5). Correspondingly, the considered station in the backoff procedure receives a corrupted frame, after which the EIFS must be observed. The self-loop from place `bc` via `T1` with the depicted arc multiplicities guarantees that `bc` never contains more than $bcmax$ tokens (in a tangible marking). For DSSS WLANs, $bcmax$=5, as opposed to $bcmax$=6 for FHSS WLANs. The multiplicity of the arc from `bc` to `succ` is again marking-dependent: the firing of `succ` resets the backoff counter.

Whenever the transmission of a station will collide, transition `coll` puts a

token into place `lastRxFailed` (via port `toLastRxFailed`). Any station receiving a corrupted frame on the medium – i.e., in ideal channel conditions, any station not involved in the collision – delays its transmission requests until `#lastRxFailed` $= 0$. A successful transmission may establish this condition. In this case, the sending station "flushes" all tokens in place `lastRxFailed` by the firing of transition `Ttxsucc` via a marking-dependent arc. Alternatively, the construct consisting of place `PwaitEIFS` and transition `TwaitEIFS` with connected arcs guarantees that the token put into `lastRxFailed` by the station with a failed transmission is only removed, when the period of EIFS (starting at the end of the transmission) has elapsed. If a subsequent transmission succeeds earlier, transition `TwaitEIFS` with firing policy prd is preempted by the firing of `stopWaiting`.

Note that, for a repeatedly transmitted frame with BA, the transmission time is resampled. This approximation is very close to Kleinrock's independence assumption [13].

The firing times of transitions `gen`, `Tvuln`, `Ttimeout`, `TwaitEIFS`, and `Tbackoff` are independent of the chosen access mechanism and are given after the keyword **transatt** in the lower textual part of Fig. 7. Note that *vulnerablePeriod* is the sum of $aCCATime$, $aRxTxTurnaroundTime$, and $aAirPropagationTime$. The terms *exp*, *det*, and *discreteuniform* denote the exponential, deterministic, and discrete uniform distribution, respectively, with corresponding rate, delay or discrete range. The distributions of `Ttxsucc` and `Ttxcoll` depend on the access mechanism. Their firing times are defined as follows.

- `Ttxcoll`: in case of BA

  $$PHYHeader/1Mbps + (MACHeaderCRC + FrameBody)/B \\ +DIFS - aCCATime$$

  in case of RTS/CTS

  $$PHYHeader/1Mbps + RTS/B + DIFS - aCCATime$$

- `Ttxsucc`: in case of BA

  $$2 \cdot PHYHeader/1Mbps + (MACHeaderCRC + FrameBody + ACK)/B \\ +SIFS + aAirPropagationTime + DIFS - aCCATime$$

  in case of RTS/CTS

  $$4 \cdot PHYHeader/1Mbps + (RTS + CTS)/B \\ +(MACHeaderCRC + FrameBody + ACK)/B \\ +3 \cdot SIFS + 3 \cdot aAirPropagationTime + DIFS - aCCATime$$

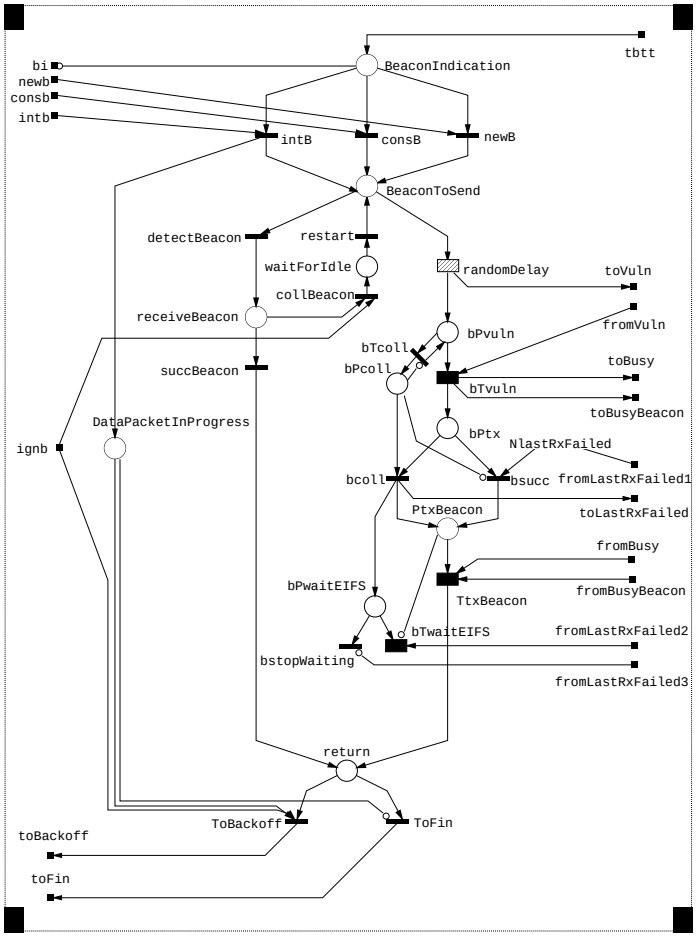
Table 3

Firing times of transitions (in ms)	`Tvuln`	`Ttxcoll`	`Ttxsucc`
DSSS, BA	0.019	[0.614, 33.242]	[0.623, 33.251]
DSSS, RTS/CTS	0.019	0.558	[1.163, 33.791]
FHSS, BA	0.048	[0.537, 16.781]	[0.578, 16.822]
FHSS, RTS/CTS	0.048	0.481	[1.026, 17.270]

For example, Fig. 1 illustrates the composition of the firing time of `Ttxsucc` in case of the RTS/CTS mechanism. As one *aAirPropagationDelay* for the first frame has already been considered in `Tvuln`, only three more air propagation delays have to be included in `Ttxsucc`. The other terms for the durations of `Ttxsucc` and `Ttxcoll` are composed analogously; only fewer frames are involved. A DIFS was added to these firing times to prevent other stations to access the medium too early. Since, however, in case of a collision, the sending station starts its timeout counter right after the end of its transmission, this DIFS has to be subtracted again from *TimeOut* in the delay of `Ttimeout`, and similarly for transition `TwaitEIFS`. The payload $FrameBody$ may be arbitrarily distributed over $[0, MaxFrameBody]$ with a mean length equal to L. Obvious choices for the distribution are a determininistic time corresponding to a particular data length, a uniform distribution over the whole interval, or a discrete distribution (e.g., a three point distribution with small, medium, and large data lengths). Table 3 gives possible firing times for the transitions `Tvuln`, `Ttxcoll`, and `Ttxsucc` for DSSS and FHSS, Basic Access (BA) and RTS/CTS, as they result from the parameter set of *IBSSpar* in Fig. 4.

A separate module, called *station_beacon* and shown in Fig. 8, is dedicated to the Timing Synchronization Function. Note that the performance of a WLAN without TSF can be evaluated without this module. We will do this for comparative experiments in Sect. 4. Then, the process instance *beacon* (with corresponding arcs) in Fig. 6 is removed. The "open" ports in Fig. 7 (`newb`, `bi`, `consb`, `toFin`, `ignb`, `intb`, `toBackoff`) are simply ignored in the SPNL compilation process (see Sect. 4). If the process instance *beacon* is included (to model an IBSS with the TSF), these ports are connected with the corresponding ports on the left-hand side of Fig. 8 (via processtype *Station* in Fig. 6). Processtype *StationBeacon*, an instance of which pertains to each station, models the beacon generation function. Here, we relate parts of the net to the items enumerated at the end of Sect. 2. The upper part above place `BeaconToSend` directs the MAC control to the beacon generation function at every TBTT and corresponds to (1). Item (2) translates to transition `randomDelay` with its discrete uniform distribution. The backoff procedure as prescribed in (3) is implemented by the guard of `randomDelay` (identical to the one of `Tbackoff` in Fig. 7) and the flushing of place `ignEIFS` of Fig. 7 by transition `collBeacon`.

```
module station_beacon;
  use IBSSpar;
  processtype StationBeacon( rreward Nvuln, Nbusy, NbusyBeacon, NlastRxFailed, NignE);
  (* ports to StationCore *)
  port bi: p-ot;
       intb, consb, newb:  t<-p;
       ignb:  t<-p;
       toBackoff, toFin:  t->p;
  (* port to global timing *)
  port tbtt: p<-t;
  (* ports to global channel states *)
  port toVuln, toBusy, toBusyBeacon, toLastRxFailed:  t->p;
       fromVuln, fromBusy, fromBusyBeacon, fromLastRxFailed1, fromLastRxFailed2: t<-p;
       fromLastRxFailed3:  to-p;
private
```




```
  transatt detectBeacon: guard = NbusyBeacon >  0;
           collBeacon:   guard = Nbusy > 1;
           restart:      guard = Nbusy = 0;
           succBeacon:   guard = Nbusy = 0;
           bTcoll:       guard = Nvuln > 1;
           randomDelay:  dist = discreteuniform(0, 2*aCWmin*aSlotTime, aSlotTime),
                         policy = prd,
                         guard = (NignE = 0  and Nbusy = 0  and NlastRxFailed = 0)
                                 or (NignE > 0and Nbusy = 0);
           bTvuln:       dist = det(vulnerablePeriod);
           TtxBeacon:    dist = det(PHYheaderTransTime+BeaconTransTime+DIFS-aCCATime);
           bTwaitEIFS:   dist = det(EIFS-DIFS),
                         policy = prd;
  end StationBeacon;
end station_beacon.
```

Fig. 8. Extension for synchronization (TSF)

On the contrary, we chose prd as the preemption policy of `randomDelay`. This resampling deviates from the standard, if the station receives a corrupted beacon frame (path via transitions `collBeacon` and `restart`), but is correct for every invocation of the beacon generation function at TBTTs. The left path from place `BeaconToSend` to place `return` via transitions `detectBeacon` and `succBeacon` corresponds to item (4), while the parallel and more complex path

on the right models the transmission of the beacon (item (5)). Note that the guard of `detectBeacon` examines place `busyBeacon` of Fig. 5. Tokens in place `busyBeacon` represent the number of perceptible ongoing beacon transmissions. The subnet for beacon transmission resembles that of the corresponding one in Fig. 7 with identifiers only slightly modified. The main differences stem from the fact that beacons are broadcast frames of a fixed size. Therefore, transition `TtxBeacon` is deterministic with a delay

$$PHYHeader/1Mbps + BEACON/B + DIFS - aCCATime$$

in any case and is not followed by a timeout and backoff procedure for retransmission. If a (pending) non-beacon transmission had had to be suspended (one token in place `DataPacketInProgress`), the MAC control returns to place `Pbackoff` via transition `ToBackoff`, otherwise to `fin` via `ToFin` (see Fig. 7).

Stationary performance measures for all detailed models can easily be expressed in terms of rate and impulse rewards. $\mathrm{E}\{\#P\}$ gives the expected number of tokens in place `P` and $\mathrm{E}\{\#T\}$ the mean number of firings per unit time of transition `T`. The measures throughput and mean waiting time are defined for each station in processtype *StationCore* right below the graphical area (keyword **smeasure** in Fig. 7). The throughput is normalized to bit rate B. The mean waiting time accounts for the time from data generation until the end of transmission and is obtained by Little's law. For system indices, (station) throughputs are added and mean waiting times are averaged. The model parameter λ (see transition `gen`) is determined from the input parameter V, the "virtual load", by $V = N\lambda L/B$. V is the load, if the buffer had infinite capacity $K = \infty$ (i.e., the arrival process is not interrupted), normalized to bit rate B.

The model demonstrates that the SPN formalism is well suited to model the dynamics of the distributed medium access mechanism in a concise form. Unlike typical simulation studies, all behavioral details could be described succinctly. It can be evaluated by the publicly available tool `TimeNET` [19] using discrete event simulation. Numerical analysis is not possible, since non-exponential transitions are concurrently enabled and due to the large state space.

Various extensions of the model are possible. Bursty traffic sources can be incorporated (e.g., by using *Markov modulated Poisson processes*), fading can similarly be represented by a modulating Markov chain (e.g., by using the *Gilbert channel model* [11]), and fragment bursts can be included by simply changing the firing time of `Ttxsucc`. However, aspects like hidden terminals, capture, and mobility of stations would require a complete model redesign.

4 Simulation results

In this section, we present numerical results obtained from the model introduced above. All results were computed by means of the SPNL simulation component of the software tool `TimeNET` [19] with a confidence level of 99% and a maximum relative error margin of 1%. This yielded confidence intervals too small for a reasonable representation in the figures. A compiler transforms the different modules of the model into a (conventional) flat Petri net without SPNL specific elements like processes, processtypes and ports. If not stated otherwise, the parameters of the model are set as described in Sect. 2 (for $aCCATime$ and $aRxTxTurnaroundTime$ in case of DSSS the values 14 and 4 are chosen, respectively). The bit rate B equals 2 Mbps. $FrameBody$ is assumed to be uniformly distributed over $[0, MaxFrameBody]$. The packet generation rate is determined from the virtual load by $\lambda = (V/N) \cdot (2B/MaxFrameBody)$. For Poisson loads, different configurations are compared. Furthermore, we assume that all stations in the IBSS uniformly adopt either the BA or the RTS/CTS mechanism for the exchange of their data packets, which are always directed and consequently have to be acknowledged by the destination station upon successful reception (as opposed to the broadcast beacon frames).

In a first set of experiments, we consider an IBSS with three stations (N=3), which is a small, but nevertheless realistic configuration for WLANs. Our interest in the quantified impact of EIFS and beacons on system performance causes us to investigate two further modified variants of the presented model. For a WLAN without the beacon generation function, simply the part of the model depicted in Fig. 8 together with its instantiation in Fig. 6 and the "system" place `busyBeacon` are not included. If also EIFS are ignored, all arcs originating or ending in place `lastRxFailed` of Fig. 5 must be removed in addition. For each of the three variants (without EIFS and beacons; with EIFS, but without beacons; with EIFS and beacons), a simulation experiment yields two curves: (a) the throughput S versus the virtual load V (in a log-linear plot) and (b) the mean waiting time W versus the throughput S (i.e., the delay-throughput characteristics).

In Figs. 9 and 10, we show results for the access mechanisms BA (with optional physical layer DSSS) and RTS/CTS (with FHSS). The mean waiting time W is given in ms. Furthermore, the increase in throughput due to an additional buffer place (K=2 instead of K=1) can be observed. While the extra buffer place reduces the blocking/loss probability, it accounts for much longer waiting times at high throughputs as illustrated by the upward shift of the delay-throughput characteristics. The main conclusion to be drawn from Figs. 9 and 10, however, is that for an IBSS with three stations neither EIFS nor beacons have a noticable impact on system performance. For a specific

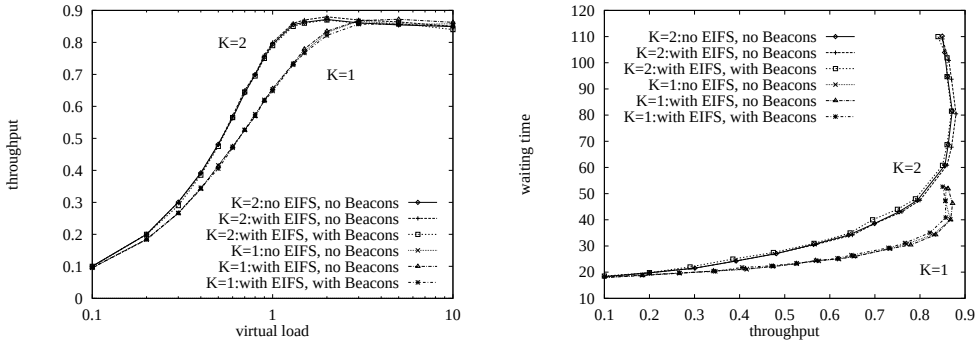


Fig. 9. DSSS, BA, N=3: S vs. V and W vs. S

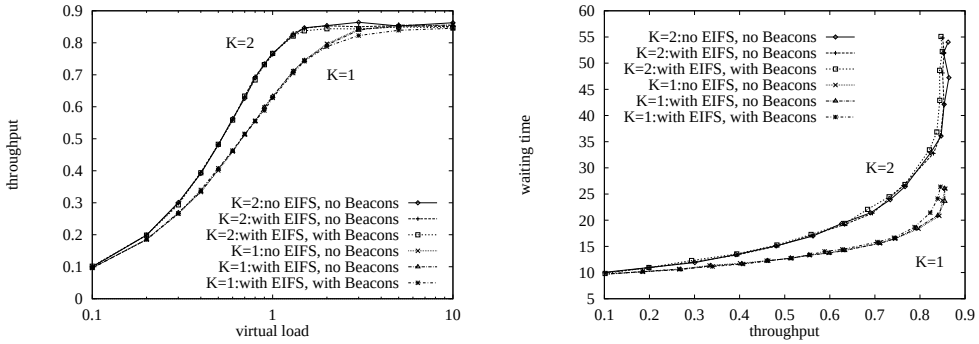


Fig. 10. FHSS, RTS/CTS, N=3: S vs. V and W vs. S

configuration, the curves for the three variants more or less coincide with a maximum deviation of 2%. From the high throughputs even at high virtual loads for the BA scheme (where for a larger number of stations, typically a throughput deterioration is observed), one deduces that too few collisions occur for the EIFS periods to play a significant role, while at the same time the backoff mechanism provides enough idle periods for the beacons to be conveniently accommodated. Our studies also revealed that the presented curves are nearly insensitive to the maximum value of the backoff counter (even for $bcmax = 0$, which corresponds to the simplified backoff procedure employed in the analytical study of [10]). Thus for N=3, abstracting from EIFS and beacons and simplifying the backoff procedure appear to be justified.

The left-hand side of Fig. 11 shows that this cannot be taken for granted and very much depends on the interaction between backoff procedure, EIFS, and beacons. For an IBSS with ten stations (N=10), we show throughput S versus virtual load V for BA (with PHY DSSS) – again for the three variants. However, this time we adopt a simplified backoff procedure ($bcmax$=0 instead of 5), which, of course, is not conform to the standard. But it allows to demonstrate that EIFS can make a significant difference. Clearly, we observe the throughput deterioration of BA for high virtual loads in all cases. It is significantly mitigated by the introduction of EIFS leading to throughputs more than 20% higher in saturation conditions (V>5). The achievable maximum throughput is not only shifted to higher loads, but also increased from 0.70 (at V=1) to

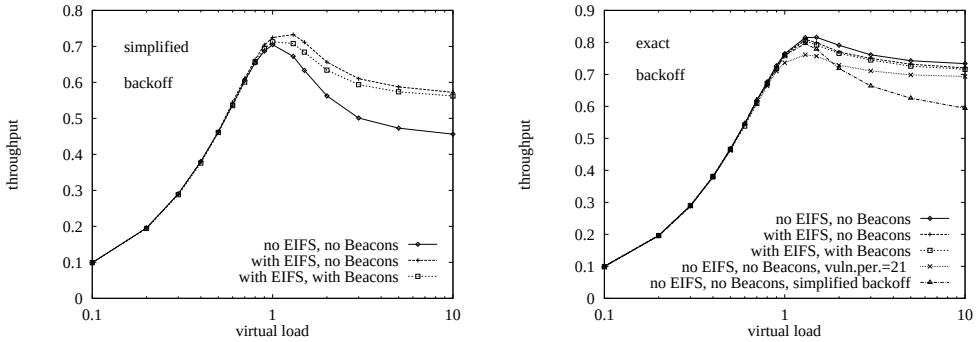


Fig. 11. DSSS, BA, N=10, K=1: simplified and exact backoff

0.73 (at V=1.3). What concerns the beacons, the bandwidth reduction due to the TSF is most pronounced in these operating points with respect to all our experiments. Nevertheless, with less than 4%, it still remains quite negligible.

For the simulation runs of Fig. 11 (left-hand side), we set *aCCATime* and *aRxTxTurnaroundTime* to their maximum values 15 and 5, respectively, as tolerated by the standard. This choice leads to a vulnerable period (21 μs) larger than the predefined *aSlotTime* (20 μs) resulting in unnecessary collisions. The aggravating effect on throughputs could easily be prevented by an inverse relationship of these periods. The right-hand side of Fig. 11 verifies the last statement, where corresponding plots are shown for the same IBSS with *aCCATime* and *aRxTxTurnaroundTime* being set again to the original values of 14 and 4, respectively (except for one case marked by vuln.per.=21). The only curve based on the simplified backoff procedure reveals the beneficial effect the standardized backoff procedure has on system throughput for high loads. However, EIFS no longer yield an improved throughput (rather the opposite), while beacons hardly make a difference at all. As a consequence, simplifying the backoff procedure of the IEEE 802.11 protocol in analytical performance studies should be conducted with great care (see Sect. 5).

In the last experiment, we again consider a WLAN with ten stations based on DSSS. As before, *aCCATime* = 14, *aRxTxTurnaroundTime* = 4, and the standardized backoff procedure is employed, whereas K equals 2. In Fig. 12, curves are shown for both BA and RTS/CTS. Obviously, RTS/CTS is superior over BA in heavy load conditions. Surprisingly, EIFS (and beacons) only marginally affect the performance measures. In contrast to our expectations, but as indicated by Fig. 12, EIFS even slightly lowered throughputs for all virtual loads with waiting times being prolonged. This exactly contradicts the original intention of EIFS, which privilege collided frames over other pending and newly generated transmission requests during channel access in an EIFS period in order to shorten their waiting times. Our studies suggest that under heavy loads the backoff intervals soon become too large compared to the EIFS period, so they counteract the beneficial potential of EIFS (see Fig. 11). What remains is the deferral for an EIFS of the majority of stations having received

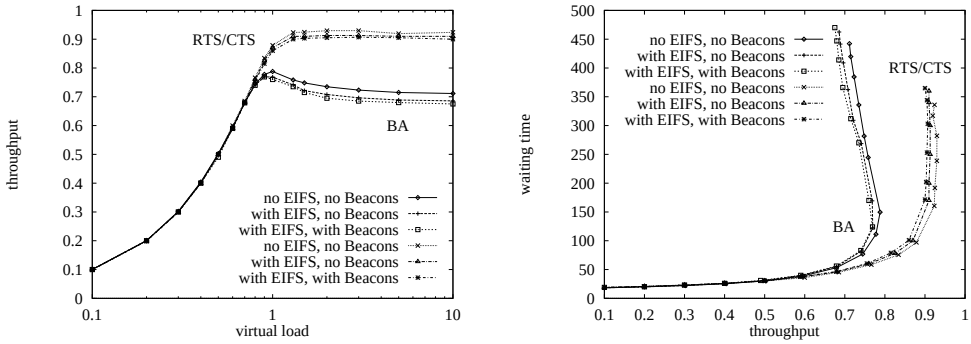


Fig. 12. DSSS, N=10, K=2: S vs. V and W vs. S

a corrupted frame resulting in lower throughput and larger waiting times.

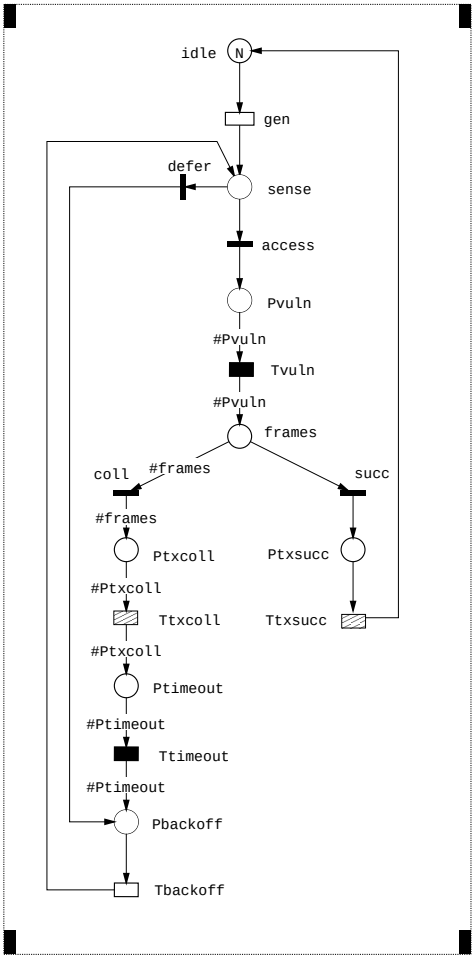
Obviously, the backoff procedure itself as defined by the standard avoids collisions in a sufficient way, at least for up to around ten stations in an IBSS. Within the parameter set tolerated by the IEEE 802.11 standard, it seems that the additional implementation effort for EIFS rarely pays off. Extensive simulation experiments confirm that especially the performance measures of the RTS/CTS mechanism for both DSSS and FHSS are almost insensitive to the incorporation of EIFS as well as to beacons.

5 The folded model

The sequential simulation of the detailed model with 99% confidence interval and maximum relative error of 1% typically computed curves with one varying parameter in a time period of several hours to one day on an UltraSPARC workstation with 300 MHz (less time is needed for throughputs only and in case of less rigid requirements for the simulations). A presumably faster numerical analysis of the model in its current state is not possible. In this section, we obtain a more compact and analytically tractable representation of the detailed model. Based on the simulation results of the previous section, we will ignore EIFS and beacons. This alleviates to fold the station subnets and apply lumping. The model developed in this section is an improved version of a similar one in [10].

The buffer size K of the detailed model is set to 1 and the traffic sources are superposed. As a consequence, the model now represents just the medium access for a single data packet, and no longer the whole MAC buffer. Furthermore, the dynamics during the transmission are slightly approximated by exclusively enabled non-exponential transitions. The backoff procedure is altered, but matched to capture the impact of the standardized backoff. Figure 13 shows the folded SPNL model. This main module imports the parameters in module *IBSSpar* of Fig. 4. Most identifiers were carried over from the detailed

```
module folded_IBSS; (* N stations without EIFS and beacons *)
  use IBSSpar;
  parameter coll_idx = 0.8 +sqrt(vulnerablePeriod)/3.5 *
                          (1- e^(-0.01*V*vulnerablePeriod));
process folded_net;
  smeasure throughput, waitingTime;
private
```




```
smeasure throughput = ( E{#gen}*L)/B;
         waitingTime = (N -  E{#idle})/ E{#gen};
transatt gen:      dist = exp( λ ),
                   concurrency = is;
         Tvuln:    dist = det(vulnerablePeriod),
                   guard = #Pvuln > 0;
         Ttxcoll:  dist = uniform(collConstTransTime,
                                  collConstTransTime+collMaxBodyTransTime),
                   guard = #Ptxcoll > 0;
         Ttxsucc:  dist = uniform(succConstTransTime,
                                  succConstTransTime+succMaxBodyTransTime);
         Ttimeout: dist = det(TimeOut-DIFS),
                   guard = #Ptimeout > 0;
         Tbackoff: dist = if coll_idx<1
                             then exp(2/(aCWmin*aSlotTime))
                             else exp((1/coll_idx)*(2/(aCWmin*aSlotTime))),
                   guard = #Ptxcoll = 0 and #Ptxsucc = 0,
                   concurrency = is;
         defer:    guard = #Ptxcoll > 0 or #Ptxsucc > 0;
         access:   guard = #Ptxcoll = 0 and #Ptxsucc = 0;
         coll:     guard = #frames > 0;
         succ:     guard = #frames = 1;
  end folded_net;
end folded_IBSS.
```

Fig. 13. Folded SPNL model

model. Their meaning can be looked up in Table 2. In order to keep the model structure simple, repeated use is made of guards and marking-dependent arc multiplicities.

Tokens in place `idle` represent stations with no pending packet to transmit. Exponential transition `gen` still represents the generation of data packets. Due to the superposition of arrival processes it has the same rate λ but now an infinite server semantics, i.e., its firing rate is marking-dependent and equal to $\#idle \cdot \lambda$. The number of transmissions in the vulnerable period is now represented by tokens in place `Pvuln` directly and the number of perceptible transmissions by tokens in `Ptxcoll` and `Ptxsucc`. After generation of a data packet, the corresponding token is put either in place `Pbackoff` or `Pvuln`. More than one token in `Pvuln` corresponds to a collision. The firing of `Tvuln` flushes all tokens into place `frames`. Depending on the number of tokens in `frames` either immediate transition `coll` or `succ` are enabled. In case of more than one token (i.e., a collision has occurred), all tokens are flushed by the transitions `coll`, `Ttxcoll`, and `Ttimeout` to place `Pbackoff`. The backoff procedure is approximated by an exponential transition with infinite server semantics: its maximal rate is $2 \cdot \#Pbackoff/(aCWmin \cdot aSlotLength)$, the reciprocal of the mean delay of the first backoff interval. In the absence of collisions, this rate sufficiently reflects the delay due to the backoff procedure. As delays become longer with an increasing number of collisions, the parameter *coll_idx* reduces the rate to capture this effect. The larger the critical parameter *vulnerable period* and the virtual load V, the more collisions occur and the higher the value of *coll_idx*.

Note that the flushing of tokens from `Pvuln` to `Pbackoff` means a synchronization of the collided frames, as if their transmissions had begun at the same instant, while assuming identical lengths of the collided frames at the same time (which is true for the RTS/CTS mechanism).

Comparing the compact model with the detailed one, four simplifications can be identified. First of all, synchronizing the collided frames by the firing of transition `Tvuln` slightly shortens the overall time the medium is busy due to collisions. In case of Basic Access, this synchronization has two more consequences: on the one hand, the distribution of `Ttxcoll` should actually follow the one of the maximum data packet involved in the collision[2]; on the other hand, due to these different transmission durations, the stations do not invoke their backoff procedures at the same time (as it is modeled by the marking-dependent arc from `Ttimeout` to `Pbackoff`). Numerical experiments with similar models (see [10]) suggest that these three approximations only lead to minor differences of the compact and detailed model. The fourth approximation concerns the backoff procedure itself. The general distribution of `Tbackoff` in the detailed model has been replaced by an exponential one with the rate as defined in Fig. 13. Compared with the predecessor model in [10], this approximation has been refined, since the results in [10] and the simulation

[2] The maximum of two uniform distributions is a special triangular distribution which could be modeled as an expolynomial distribution in TimeNET.

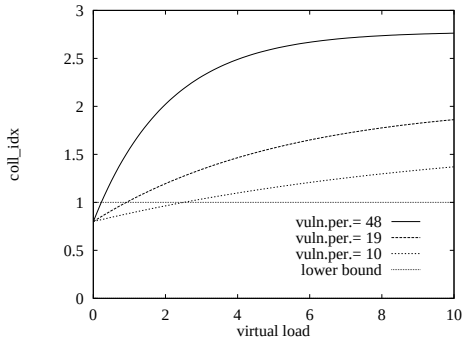


Fig. 14. Parameter *coll_idx* as a function of V

experiments of Sect. 4 urged us to reconsider the Petri net implementation of the backoff procedure. As a consequence, we introduced the parameter *coll_idx*:

$$coll\_idx = 0.8 + \frac{\sqrt{vulnerable\ period}}{3.5} \cdot \left(1 - \exp^{-0.01 \cdot V \cdot vulnerable\ period}\right) \ .$$

Whenever *coll_idx* is greater than unity, the maximal rate of `Tbackoff` is reduced by a factor $1/coll\_idx$. Figure 14 depicts the curves of *coll_idx* as a function of V for different values of the vulnerable period. At first sight, the backoff counter appears to have been dropped. However, the function for *coll_idx* was matched to several simulation data so that the rate of `Tbackoff` results in a behavior almost equivalent to the one of an IBSS, in which the stations employ the slotted binary exponential backoff. Our numerical results reveal that the complex rate covers any standardized configuration with BA or RTS/CTS arbitrarily combined with DSSS or FHSS. The alternative approach to include an explicit backoff counter in the folded model would lead to more intricate models with much larger state spaces.

With these simplifications the model becomes analytically tractable. The number of states is given by the distribution of N tokens over the given places (where `frames` and `sense` are empty in all tangible markings and `Ptxsucc` holds at most one token). Therefore, the state space has a moderate size and can be handled with common SPN tools. Moreover, all non-exponential transitions are persistent and mutually exclusive, such that recently developed analytical techniques based on Markov renewal theory or supplementary variables can be employed [4,9,19]. We use SPNica, a Mathematica package for the analysis of SPNs [9]. Curves with one varying parameter can be computed in several minutes (on a PC with 233 MHz and 64 MB). As the numerical results of the next section illustrate, the model can predict main performance characteristics of the MAC protocol with different physical layers. At the expense of an increased state space, it would also be possible to include a MAC buffer ($K > 1$) in the folded model as well.

In Fig. 13, the (stationary) performance measures of the compact model are again defined below the graphical area. The definitions of the measures

throughput and *waitingTime* differ only slighty from the ones of Fig. 7, as they represent system indices:

- throughput $S = E\{\#gen\}L/B$, normalized to bit rate B and
- (mean) waiting time $W = (N - E\{\#idle\})/E\{\#gen\}$.

The virtual load, defined in module *IBSSpar*, is the same as in Sect. 3. The measures are analogous to the added throughputs or averaged mean waiting times of the detailed model, respectively, and can be compared with them.

6 Analytical results

In this section, we compare the analytical results obtained from the folded model of the previous section with simulations of the detailed model. The detailed model used for the experiments in this section also ignores the TSF and EIFS, but – in contrast to the analytical model – employs the standardized backoff procedure. Simulations were performed with the SPNL-component of `TimeNET` [19], again with 99 % confidence interval and a maximum relative error of 1 %; all analyses were performed with SPNica [9]. If not stated otherwise, the parameters of the model are set as described in the first paragraph of Sect. 4, e.g., in case of DSSS *aCCATime*=14 and *aRxTxTurnaroundTime*=4. In the following experiments, we consider a WLAN with ten stations (N=10, each having a single buffer place K=1)) operating with different physical layers (DSSS or FHSS) and different medium access schemes (BA or RTS/CTS). In the figures, we give both the simulation (label "sim") and the analytical results (label "ana"). Generally, the corresponding curves show a very good coincidence, with the tendency that the analytical results are slightly pessimistic. The largest deviation occurs for DSSS and BA at V=2.0 (with a vulnerable period of 19 μs) and amounts to -4.9% for the throughputs and $+8.6\%$ for the mean waiting times.

Figures 15 and 16 show the throughput versus the virtual load and the mean waiting time versus the throughput for the two physical layers, respectively. Again, the throughput deterioration observed with BA is almost compensated in the RTS/CTS mechanism. Indeed, the virtual sense mechanism makes the protocol very similar to non-persistent CSMA/CD. It can also be seen that FHSS achieves a lower throughput than DSSS due to the longer vulnerable period. In Fig. 15, we also plotted the throughput versus the virtual load and the delay-throughput-characteristic for BA in DSSS with a different vulnerable period (label "vuln.per.=10" standing for a vulnerable period of 10 μs). Assuming advanced technology with *aCCATime*=7 and *aRxTxTurnaround-Time*=2, i.e., half of the previously used values for these periods (in μs), the performance of the IBSS can be considerably improved. The small deviations

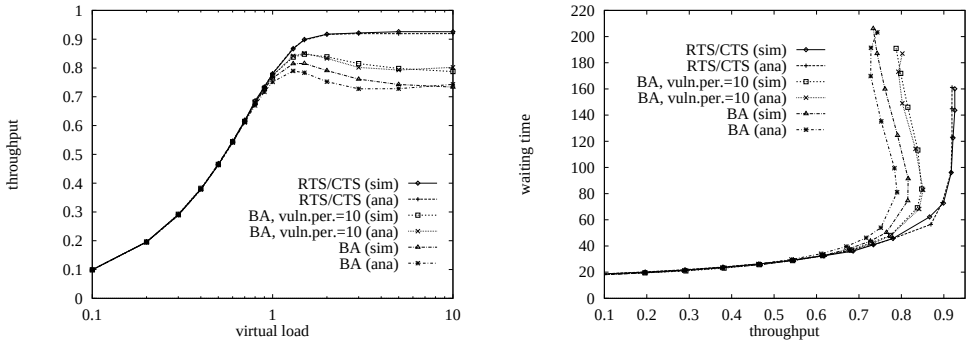


Fig. 15. DSSS, N=10, K=1: S vs. V and W vs. S

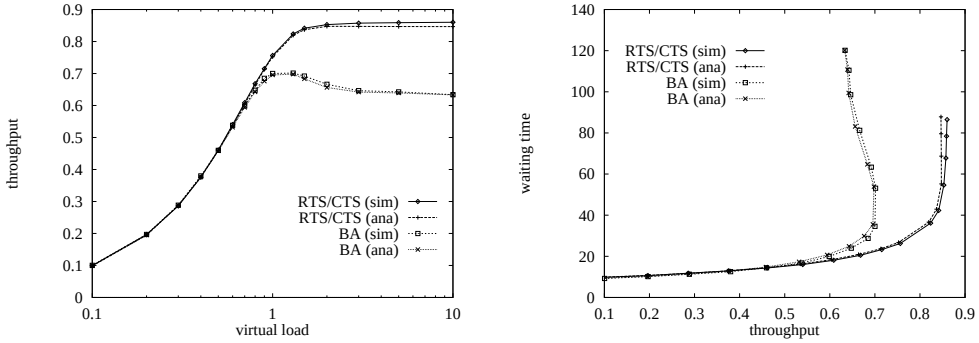


Fig. 16. FHSS, N=10, K=1: S vs. V and W vs. S

of the analytical curve from the simulation data underline the validity of the rate function for the "equivalent" backoff transition in the complete range of the standardized configurations.

7 Conclusions

A simulation study of the performance apects of a fully compliant IEEE 802.11 wireless LAN has been presented. Besides the distributed coordination function (including EIFS), the timing synchronization function (TSF) has been incorporated into a detailed Stochastic Petri Net (SPN) model. Results have also been obtained from a compact analytical model which was derived from the detailed model based on the insight due to the simulation data. Performance differences of physical layer options and the influence of system parameters have been demonstrated. SPNs have proven to be a flexible modeling formalism which allows to capture the relevant system dynamics in a concise way. This provides a solid basis for comparing the results of different studies.

For different configurations, simulation results allow to quantify the bandwith reduction due to the synchronization overhead (beacon frames) and the effect of EIFS on system performance. Despite their potential to improve system performance under heavy loads, the study suggests that for standardized sys-

tem configurations EIFS rather tend to slightly reduce system throughputs and prolong mean waiting times – at least for WLANs consisting of up to around ten stations. As a main reason for this, the slotted binary exponential backoff prescribed by the standard has been identified. Employing this backoff procedure, the impact of EIFS and beacons amounted to less than 10% in our experiments. In conclusion, widespread (but not yet verified) assumptions applied in analytical modelling (i.e., ignoring EIFS and beacons) could be confirmed for the considered WLANs. Applying these simplifications to the detailed model, we arrived at an analytically tractable model.

The following extensions of this work are nearby. Bursty traffic and unreliable channel conditions can be added to the model. The presented model can also be extended for the evaluation of high-speed wireless LANs which are currently in the standardization process.

References

[1] G. Bianchi. Throughput evaluation of the IEEE 802.11 distributed coordination function. In *Proc. 5th Int. Wshp. on Mobile Multimedia Communications*, pages 307–318, Berlin, 1998.

[2] F. Cali, M. Conti, and E. Gregori. IEEE 802.11 Wireless LAN: Capacity analysis and protocol enhancement. In *Proc. of INFOCOM'98*, San Francisco, USA, 1998.

[3] H. S. Chaya and S. Gupta. Performance modeling of the asynchronous data transfer methods of IEEE 802.11 MAC protocol. *Wireless Networks*, 3:217–234, 1997.

[4] H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov regenerative stochastic Petri nets. *Perf. Eval.*, 20:337–357, 1994.

[5] B. P. Crow. Performance evaluation of the IEEE 802.11 wireless local area network protocol. Master's thesis, Dept. of Electrical and Computer Engineering, University of Arizona, 1996.

[6] I. S. Department. *802.11: IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999 Edition.

[7] J. Geier. *Wireless LANs*. Macmillan, 1998.

[8] R. German. SPNL: Processes as language oriented building blocks of stochastic petri nets. In *Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 1469, pages 123–134. Springer, 1997.

[9] R. German. Markov regenerative stochastic Petri nets with general execution policies: Supplementary variable analysis and a prototype tool. *Perf. Eval.*, 39:165–188, 2000.

[10] R. German and A. Heindl. Performance evaluation of IEEE 802.11 Wireless LANs with stochastic Petri nets. In *Proc. 8th Int. Workshop on Petri Nets and Performance Models*, pages 44–53, Zaragoza, Spain, 1999.

[11] E. N. Gilbert. Capacity of a burst-noise channel. *Bell System Tech. J.*, pages 1253–1266, 1960.

[12] J. Hammond and P. O'Reilly. *Performance Analysis of Local Computer Networks*, volume I. Addison-Wesley Publishing Company, 1986.

[13] L. Kleinrock. *Queueing Systems*, volume II: Computer Applications. John Wiley & Sons, 1976.

[14] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[15] M. A. Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1986*, LNCS 266, pages 132–145. Springer, 1987.

[16] M. A. Marsan, G. Chiola, and A. Fumagalli. An accurate performance model of CSMA/CD bus LAN. In *Advances in Petri Nets 1987*, pages 146–161. Springer-Verlag, 1987.

[17] M. Telek, A. Bobbio, and A. Puliafito. Steady state solution of MRSPNs with mixed preemption policies. In *Proc. IEEE Int. Performance and Dependability Symp.*, pages 106–115, Urbana-Champaign, Illinois, USA, 1996.

[18] J. Weinmiller, M. Schlager, A. Festag, and A. Wolisz. Performance study of access control in wireless LANs IEEE 802.11 DFWMAC and ETSI RES 10 HIPERLAN. *Mobile Networks and Applications*, 2:55–67, 1997.

[19] A. Zimmermann, J. Freiheit, R. German, and G. Hommel. Petri net modelling and performability evaluation with TimeNET 3.0. In *Proc. 11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Chicago, USA, 2000.