Christoph Biesinger
Projektmodul
Institut für Informatik der Universität Augsburg
Kommunikationssysteme

Analyse und Anwendung des Entwicklungsrepository ns-3-dev-TSCH

Zusammenfassung

Durch die Erweiterung *Amendment A: MAC sublayer* auf die Version IEEE 802.15.4e wurden Low-Rate Wireless Personal Area Networks LR-WPANs um mehrere Funktionen erweitert. Die am meisten diskutierte Erweiterung beschreibt dabei den *Timeslotted Channel Hopping Mode TSCH*, welcher ein Kanalsprungverfahren implementiert, wodurch 802.15.4-Netzwerke robuster und zuverlässiger im industriellen Umfeld werden sollen. Daher beschäftigt sich diese Arbeit mit dem Entwicklungsrepository, welcher das bestehende LR-WPAN Modul im Netzwerksimulator ns-3 um die Funktionalitäten des TSCH Modes erweitert. Dabei soll der grundsätzliche Entwicklungsstand analysiert werden, sowie anhand von Beispielen die mögliche Anwendung innerhalb eines vordefinierten Anwendungsfalles, dem Network Formation Process im TSCH Mode, erforscht werden. Aufgrund dieser Vorgabe behandelt diese Arbeit in nachfolgender Reihenfolge die Grundlagen der Theorie und Implementierung des TSCH Mode, den theoretischen Ablauf des Network Formation Process, wichtige Kernmethoden und Funktionalitäten des LR-WPAN TSCH Modul in ns-3, sowie abschliessend verschiedene experimentelle Anwendungen mit dem Entwicklungsrepository. Damit dient diese Arbeit als theoretische Vorlage für künftige Entwicklungsarbeiten am Kommunikationssysteme Lehrstuhl der Universität Augsburg.

Inhaltsverzeichnis

1 Grundlagen

1.1 IEEE 802.15.4 Low-Rate Wireless Personal Area Networks

Der Standard IEEE 802.15.4 auch Low-Rate Wireless Personal Area Networks LR-WPANs, beschreibt drahtlose Kommunikation für Sensornetzwerke mit günstiger, hauptsächlich batteriebetriebener Kommunikationshardware welche geringe Datenraten ermöglicht. Der Standard mitsamt seinem beschriebenen Protokoll bildet die technologische Basis für Anwendungen im Internet der Dinge IoT.
Die Grundversion offenbarte einige technische Lücken und Nachteile im Vergleich zu anderen Vergleichbaren Technologien (WLAN), weshalb mit dem *Amendment A: MAC sublayer* zusätzliche Eigenschaften und Funktionen hinzugefügt wurden. Einige der wichtigsten Probleme waren dabei,

- keine garantierte Bandbreite ausserhalb von Guaranteed Time Slot GTS.
- keine Latenzbeschränkung.
- kein Schutz gegen Interferenzen und Fading.
- keine Technologie um die Frequenz des Kanals zu wechseln.
- einen unzuverlässigen MAC Sublayer durch die Verwendung von CSMA/CA Die Wahrscheinlichkeit von Kollisionen über den Kanal steigt mit der Anzahl der Knoten durch erhöhte Contention Time.

1.2 IEEE 802.15.4e Amendment A: MAC sublayer

Um die Probleme von 802.15.4 zu beheben wurde die 802.15.4e Working Group geschaffen, welche das existierende MAC Protokoll verbessern sollte. Dadurch wurde der 802.15.4 Standard um zwei Kategorien an MAC Verbesserungen erweitert. Einmal den MAC Behavior Modes, welche spezifische Anwendungsbereiche verbessern, sowie General Functional Enhancements für generelle Verbesserungen. Bei der Entwicklung wurden dabei viele Ideen aus bereits existierenden Standards wie WirelessHART und ISA 100.11.a übernommen.

General Functional Enhancements

Information Elements IE sind bereits seit der Grundversion implementiert, übernehmen aber zusätzliche Aufgaben im Amendment A.

Enhanced Beacons EB bilden eine Erweiterung zu Beacon Frames mit grösserer Flexibiliät. Werden als anwendungsbezogene Beacons via IEs im TSCH Mode verwendet.

Low Energy LE Austausch der Eigenschaften niedrige Latenz durch niedriger Energieverbrauch, wodurch das Gerät immer erreichbar erscheint

Behavior Modes

Timeslotted Channel Hopping TSCH für die Verbesserung von automatisierten Anwendungsbereichen.

Deterministic and Synchronous Multi-Channel Extension DSME zur Unterstützung von industriellen und kommerziellen Anwendungen, welche strikte Vorgaben an Zeit und Verfügbarkeit stellen.

1.3 Networksimulator ns-3

Im Laufe der Projekts wird der Netzwerksimulator ns-3 verwendet, dabei kann die Grobbeschreibung direkt von dem Internetauftritt unter http://www.nsam.org entnommen werden.

```
ns-3 is a discrete-event network simulator for Internet systems,
targeted primarily for research and educational use.
ns-3 is free software, licensed under the GNU GPLv2 license,
and is publicly available for research, development, and use.
```

ns-3 ist konzipiert als eine Sammlung von Software-Bibliotheken, welche zusammen ein System bilden. Diese einzelnen Bibliotheken werden als Module bezeichnet und implementieren jeweils einen eigenen Standard, welche auf den Kernmodulen aufbauen. Dadurch können Benutzerprogramme durch Verwendung dieser Module aufgebaut werden. Wir gehen im weiteren davon aus, dass der Leser mit ns-3 vertraut ist, falls nicht, kann das mit Hilfe des Tutorial nachgeholt werden.

1.3.1 LR-WPAN Modul

Im Verlauf dieser Arbeit wurde hauptsächlich das LR-WPAN Modul angewendet, welches in der offiziellen ns-3 Version (NS-3.24) den IEEE 802.15.4 Standard mitsamt dessen Funktionalitäten und Hilfsanwendungen implementiert.
Der Aufbau des Modules erfolgt dabei nach ns-3 Standard und entspricht dabei der Ordnerhierarchie *models* mit den direkten Implementierungen des Standards, *helper* wichtigen und nützlichen Helperklassen und Methoden, zur einfachen Nutzung des Standards sowie den selbsterklärenden Ordnern *examples* und *test*.
Für eine genaue Beschreibung und Informationen zur Entwicklung des LR-WPAN Modules, kann auf die Arbeit von Nikica Krezic-Luger[**bachelorarbeit**] verwiesen werden. Zusätzlich zum offiziellen LR-WPAN Modul, wird nun das Entwicklerrepository ns-3-dev-tsch der Entwicklungsgruppe *RICH* von EIT Digital betrachtet. Dieses Repository

besteht aus einem vollständigen ns-3 Build mitsamt den Erweiterungen im LR-WPAN Modul um die TSCH Mode Funktionalitäten. Dieser Ansatz zur Erweiterung von ns-3 wurde im Januar 2015 zur Codereview freigegeben und kann in Github unter ns-3-dev-TSCH gefunden werden.
Zusätzlich zum Code hat die Entwicklergruppe eine Verlautbarung zum Commitstand abgegeben, die folgende Funktionen im Detail beeinhaltet:

- Wechsel von Standard 802.15.4 auf den TSCH Mode, während der Kommunikation
- das Energy Model
- multi-path fading modeling (FadingBias)

2 Timeslotted Channel Hopping TSCH

In diesem Kapitel werden nun die Funktionen und Eigenschaften des TSCH Mode in der Theorie und dessen Implementierungsstand innerhalb von NS3 erläutert.
Der Timeslotted Channel Hopping TSCH Mode ist hauptsächlich zur Unterstützung von Anwendungen bei industrieller Automatisierung sowie Prozessmonitoring entwickelt werden und charakterisiert sich dabei durch folgende Eigenschaften:

Time Slotted Access erhöht den möglichen Datendurchsatz durch das Verhindern von Kollisionen zwischen in Wettbewerb stehenden Knoten und sorgt damit gleichzeitig für eine vorhersagbare Latenz.

Multi-Channel Kommunikation durch die Verwendung von mehreren “Ubertragungskanälen können mehrere Knoten zur gleichen Zeit (innerhalb des gleichen Zeitslots) senden, wodurch wiederum die Datendurchsatzkapazität gesteigert wird.

Channel Hopping (auch Frequency Hopping) durch die verfügbaren “Ubertragunskanäle reduziert die Auswirkungen der Interferenz und Fading, wodurch die Kommunikation zuverlässiger wird.

Daher wird eine erhöhte Netzwerkauslastung und Zuverlässigkeit garantiert mit vorhersehbarer Latenz bei hoher Energieeffizienz, was zu einer Unabhängigkeit von Herstellern fürht und besser für Multi-Hop Netzwerke eignet.

2.1 Time-slotted Access

2.1.1 Time Slots

Timeslots sind klar definierte Zeitabschnitte, in denen ein Knoten agieren kann. Dieser Zeitraum entspricht dabei dem im nachfolgenden Bild dargestellten Vorgang, dem senden eines Pakets mit maximal möglicher Grösse (127 Byte) und dem Erhalt der Antwort mittels ACK (1 ms) und dem *packet processing* zur Verarbeitung der Kommunikation (5 ms), wodurch ein Timeslot ca. 10ms lang ist.

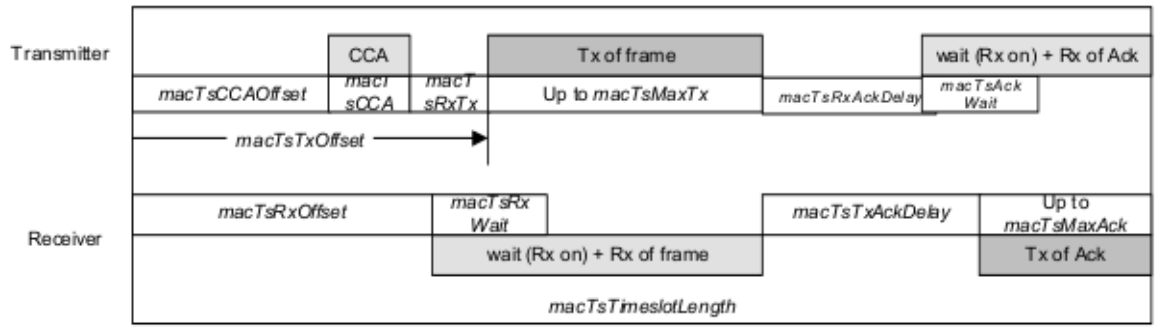

Abbildung 2.1: Timeslot [4]

Im aktuellen Entwicklungszustand, sind die Timeslots vollständig implementiert, nachstehend die Datenstruktur für die MacTimeslots in *lr-wpan-tsch-mac.h* in Zeile 146, welche der Abbildung 2.1 entsprechen.

```
typedef struct {
  //Table 52e TSCH-MAC PIB attributes for macTimeslotTemplate
  uint8_t m_macTimeslotTemplateId;
  uint16_t m_macTsCCAOffset;
  uint16_t m_macTsCCA;
  uint16_t m_macTsTxOffset;
  uint16_t m_macTsRxOffset;
  uint16_t m_macTsRxAckDelay;
  uint16_t m_macTsTxAckDelay;
  uint16_t m_macTsRxWait;
  uint16_t m_macTsAckWait;
  uint16_t m_macTsRxTx;
  uint16_t m_macTsMaxAck;
  uint16_t m_macTsMaxTx;
  uint16_t m_macTsTimeslotLength;
}LrWpanMacTimeslotTemplate;
```

2.1.2 Slot Frames

Slot Frames entstehen durch die Kombination mehreren Timeslots, welche sich zyklisch wiederholen, wodurch sich die Knoten innerhalb des LR-WPANs miteinander synchronisieren. Die Anzahl der Timeslots pro Slotframe Zyklus ist dabei variabel, im nachfolgenden Beispiel sieht man ein Slotframe mit 4 Timeslots.

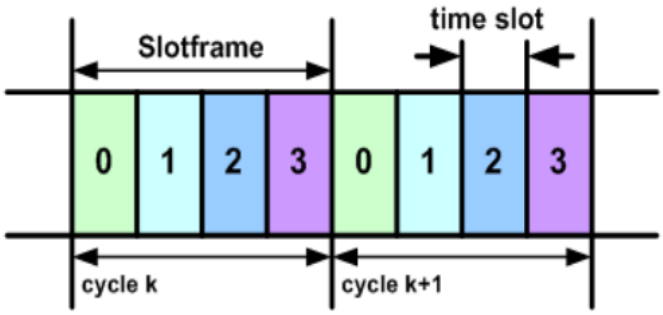

Abbildung 2.2: Slot Frames [1]

Auch die Slotframes sind vollständig implementiert. Als Beispiel die Datenstruktur für die Slotframe Operationen aus *lr-wpan-tsch-mac.h* in Zeile 202. Diese gibt die möglichen Operationen an, welche die MLME Schicht (MAC Sublayer management entity) durchführen kann. Namentlich, das hinzufügen und löschen einer Anzahl Slotframes bzw. das gezielte Modifizieren der Slotframegrösse.

```
typedef enum {
  MlmeSlotframeOperation_ADD = 1,
  MlmeSlotframeOperation_DELETE = 2,
  MlmeSlotframeOperation_MODIFY = 3
}LrWpanMlmeSlotframeOperation;
```

2.2 Multi-Channel communication

2.2.1 Node TSCH Schedule

Unter dem TSCH Schedule eines Knoten wird die Frage beantwortet, *Was macht der Knoten innerhalb eines Timeslots?* Ein Knoten kann dabei drei Aufgaben einnehmen, er kann als *Transmit Slot* ein Datenpaket senden und auf ein ACK warten, als *Receive Slot* auf ein Datenpaket warten und anschliessend ein ACK senden oder im Schlafmodus sein und keine Aktion durchführen.
In der Implementierung werden alle drei Aktionen innerhalb der Datei *lr-wpan-tsch-mac.cc* ab Zeile 1458 durchgeführt in der Methode:

```
void
LrWpanTschMac::ScheduleTimeslot(uint8_t handle, uint16_t size)
{
...
}
```

1. Transmit Slot
   Ist für einen Knoten ein Timeslot als Transmit Slot festgelegt, prüft dieser ob ein Paket im Buffer wartet, welches an den für diesen Timeslot festgelegten Nachbarknoten vorgesehen ist. Sollte dies zutreffen wird das Paket gesendet, andernfalls gewartet bis der Nachbarknoten innerhalb eines Timeslot erreichbar ist.

   In der Implementierung kann das in *lr-wpan-tsch-mac.cc* ab Zeile 1520 nachverfolgt werden.

```
    if(m_macCCAEnabled)
      {
        Time time2wait = MicroSeconds(def_MacTimeslotTemplate.
    m_macTsCCAOffset);
        Simulator::Schedule(time2wait, &LrWpanTschMac::
    SetLrWpanMacState,this,TSCH_MAC_CCA);
        m_lrWpanMacStatePending = TSCH_MAC_CCA;
        Simulator::ScheduleNow(&LrWpanTschMac::SetLrWpanMacState,this,
    TSCH_MAC_IDLE);
      }
    else
      {
        Time time2wait = MicroSeconds(def_MacTimeslotTemplate.
    m_macTsTxOffset);
        Simulator::Schedule (time2wait,&LrWpanTschMac::
    SetLrWpanMacState,this,TSCH_MAC_SENDING);
        m_lrWpanMacStatePending = TSCH_MAC_SENDING;
        Simulator::ScheduleNow(&LrWpanTschMac::SetLrWpanMacState,this,
    TSCH_MAC_IDLE);
      }

```

2. Receive Slot
   Befindet sich ein Knoten im während eines Timeslots im Receive Modus, wird dieser wie in Abbildung 2.1 gezeigt für einen definierten Zeitraum auf ankommende Frames und deren “Ubermittelungszeit warten und nach einem erfolgreichen Erhalt des Frames ein passendes ACK-Frame zurücksenden, andernfalls wird der Knoten

einfach bis zum Ende des Timeslots verweilen und nichts unternehmen. In der Implementierung ist das in *lr-wpan-tsch-mac.cc* ab Zeile 1540 sichtbar.

```
    else if (it->macLinkOptions[1]) {
      //receive
      NS_LOG_DEBUG("Start timeslot receiving procedure");
      Time time2wait = MicroSeconds(def_MacTimeslotTemplate.
m_macTsRxOffset);
      Simulator::Schedule (time2wait,&LrWpanTschMac::SetLrWpanMacState
,this,TSCH_MAC_RX);
      m_lrWpanMacStatePending = TSCH_MAC_RX;
      Simulator::ScheduleNow(&LrWpanTschMac::SetLrWpanMacState,this,
TSCH_MAC_IDLE);
      }
```

Damit ergibt sich ein Muster wie die Kommunikation zwischen den Knoten stattfindet, was im Bild 2.3 exemplarisch dargestellt wird.

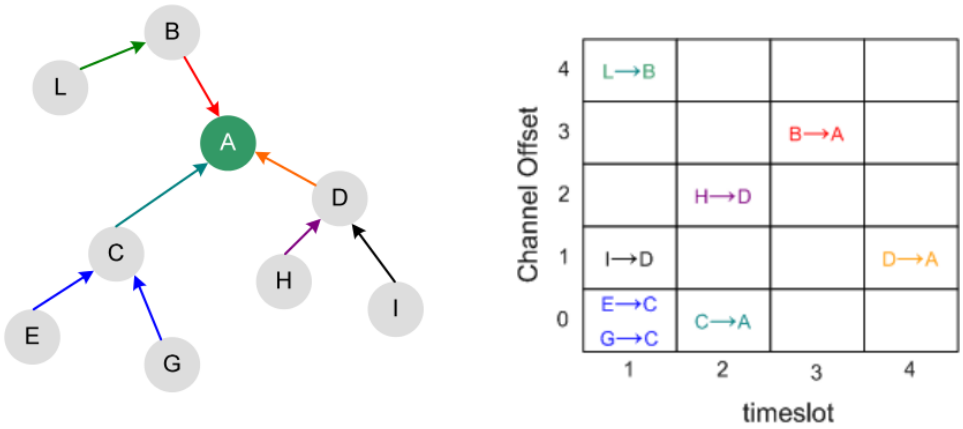

Abbildung 2.3: TSCH Link [2]

2.3 Frequency Hopping

2.3.1 Absolute Slot Number ASN

Die Absolute Slot Number ASN, gibt an wie viele Timeslots seit dem Beginn des Netzwerkes durchlaufen wurden. Damit können sich die Knoten untereinander synchronisieren und neue neue Knoten dem Netzwerk beitreten (näheres dazu im Kapitel Network Formation Process). Alle Knoten innerhalb des Netzwerkes kennen zu jedem Zeitpunkt die

global eindeutige ASN, welche deshalb für zeitkritische Anwendungen verwendet wird und zur Berechnung des Frequenzhoppings. Zum Beginn des Netzwerkes wird die ASN auf den Wert 0 instanziiert und anschliessend nach jedem Timeslot inkrementiert. Dies kann auch in der Implementierung in *lr-wpan-tsch-mac.cc* ab Zeile 1408 erkannt werden.

```
void
LrWpanTschMac::IncAsn()
{
  NS_LOG_FUNCTION (this);
  m_newSlot = 1;
  m_macTschPIBAttributes.m_macASN++;
  Simulator::Schedule (MicroSeconds(def_MacTimeslotTemplate.
    m_macTsTimeslotLength),&LrWpanTschMac::IncAsn,this);
  ...
}
```

Während des laufenden Netzwerkes kann die ASN mit folgenden Formel nachberechent werden, wobei k der fortlaufende Slotframezyklus ist, S die Slotframegrösse angibt und t den SlotOffset beschreibt.

$$ASN = (k * S + t) \tag{2.1}$$

2.3.2 Channel Hopping

Die ASN wird insbesondere für das Channel Hopping angewendet, wodurch ein Knoten am Anfang bzw. Ende eines Timeslots seinen physikalischen Kommunikationskanal wechselt. Da dieser Vorgang dauerhaft wiederholt wird und nur eine begrenzte Anzahl an verfügbaren Kanälen zur Verf"gung stehen *springt* der Knoten zwischen den Kanälen hin und her. Die Intention hinter dem Channel Hopping liegt darin begründet, dass in jedem Slotframe ein anderer Kanal im jeweilig gleichen Timeslot angewendet wird.
Die Umsetzung dieses Kanalsprungverfahren wird durch folgende Formel bewerkstelligt,

$$Frequenz = F[(ASN + channelOffset) \bmod nFreq] \tag{2.2}$$

wobei F eine Lookup Tabelle entspricht mit allen verfügbaren Frequenzen und $nFreq$ die Anzahl der Frequenzen ist (laut Standard maximal 16 Frequenzen).
Durch die sich ständig “andernde ASN wird sichergestellt, dass nach jedem Slotframe ein anderer Kanal berechnet wird und somit bildhaft durch die Frequenzen gesprungen wird.
In der Abbildung 2.4 wird gezeigt wie für den gleichen Timeslot (3) jeweils ein anderer Kanal berechnet wird. Das Channel Hopping ist auch vollständig in ns3 implementiert, wie im nachfolgenden Codesnippet aus *lr-wpan-tsch-mac.cc* in der Methode *LrWpanTschMac::ScheduleTimeslot* in Zeile 1458 deutlich wird.

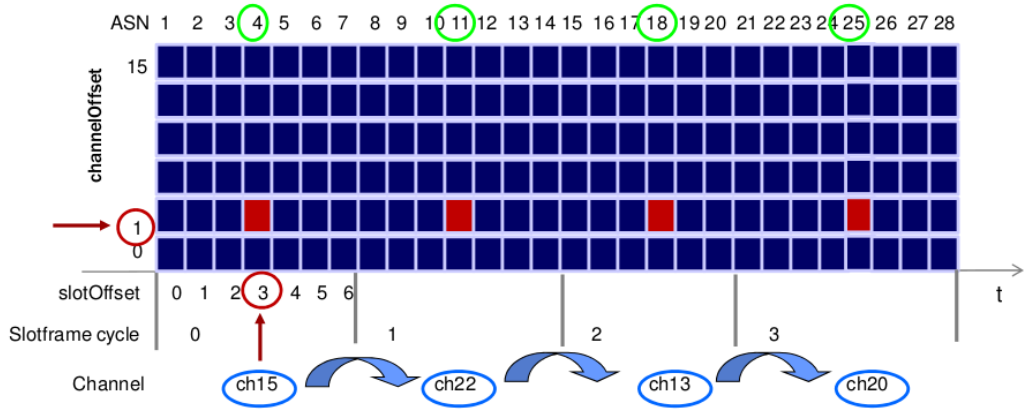

Abbildung 2.4: Channel Hopping [3]

```
if (m_macHoppingEnabled)
  {
    //Get next channel
    m_currentChannel = def_MacChannelHopping.m_macHoppingSequenceList[
      (m_macTschPIBAttributes.m_macASN+it->macChannelOffset) %
    def_MacChannelHopping.m_macHoppingSequenceLength
      ];
}
```

2.4 Time Synchronisation

Die Kommunikation innerhalb von TSCH PAN verläuft immer innerhalb von Timeslots, weshalb eine Synchronisation zwischen den beteiligten Knoten umbedingt benötigt wird. Um diese Synchronisation zu gewährleisten müssen dabei alle Knoten den Beginn das Ende der Timeslots kennen. Um dies Umzusetzen muss sich jedes Gerät mit mindestens einem anderen Gerät innerhalb des PAN synchronisieren. Diese Sychronisation erfolgt dabei im Regelfall vom PAN-Coordinator ausgehend durch das gesamte Netzwerk und wird in der Abbildung 2.5 exemplarisch dargestellt. Dabei bedeutet ein Pfeil, dass der Zielknoten sich am ausgehenden Knoten sychronisiert. Im Beispiel synchronisiert sich Device 2 mit Gerät 1 und dem PAN-Coordinator.
Die nötigen Timing Informationen werden dabei in den normalen Daten und ACK Paketen gesendet, welche innerhalb eines Time Correction Information Element enthalten sind (mehr Informationen zu den Informationen Elements in Kapitel 3.1.1). Daher kann bei der Node Synchronisation auf zwei Arten unterschieden werden.

- Frame-basiert
- Acknowledgment-basiert

wobei in beiden Fällen der Zeitunterschied gemessen wird, wann das jeweilige Synchronisations-Paket eintreffen soll mit dem tatsächlichen Zeitpunkt.

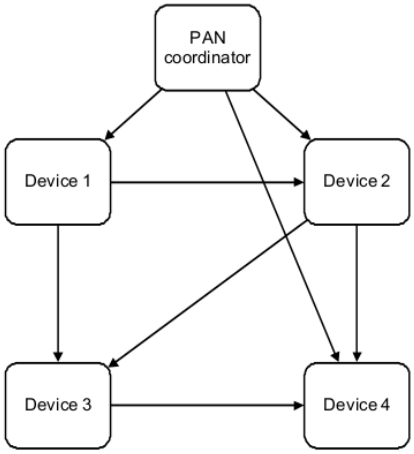

Abbildung 2.5: Beispielsynchronisation innerhalb eines PAN [4]

In ns3 fehlt eine aktive Implementierung eines Sychronisationsalgorithmus, da innerhalb des Simulators alle Knoten auf den Beginn der Simulation synchronisiert werden. In *lr-wpan-tsch-mac.cc* Zeile 1368 kann beispielhaft gezeigt werden, dass keine aktive Synchronisierung besteht.

```
1  void
   LrWpanTschMac::MlmeTschModeRequest (MlmeTschModeRequestParams params)
3  {
     NS_LOG_FUNCTION(this);
5    MlmeTschModeConfirmParams confirmParams;
     confirmParams.TSCHMode = params.TSCHMode;
7
     switch(params.TSCHMode) {
9      case MlmeTschMode_ON:

11       /*TODO: Check if it is synced
         if (its not synced) {
13         confirmParams.Status = LrWpanMlmeTschModeConfirmStatus_NO_SYNC;
         }*/
15   }
   }
```

3 Network Formation Process

Dieses Kapitel beschreibt im Detail den Ablauf sowie die notwendigen Funktinalitäten für den Network Formation Process eines IEEE 802.15.4e TSCH Netzwerkes. Dabei wird zuerst das Konzept der Enhanced Beacons EBs zur Informationsweiterleitung beschrieben mitsamt einem beispielhaften Minimalaufbau, bevor der Network Formation Process anhand verschiedener Sichtweisen erläutert wird.

3.1 Enhanced Beacons EB

Enhanced Beacons EBs sind anwendungsbezogene Beacons, welche auf dem Beacon Format beruhen und werden vor allem im DSME und TSCH Mode verwendet. Sie bilden einen Mechanismus um Informationen aus höheren Schichten auch auf der MAC Schicht zu nutzen. Erkenntlich werden EBs durch das *Frame Version Field* welches den Wert *0b10* bekommt. Die gesendeten Informationen werden dabei durch die Anbindung von Information Elements IE übermittelt.
Im TSCH Mode werden EBs vor allem angewendet um die Synchronisation der Knoten innerhalb des PAN sicherzustellen, sowie im Network Formation Process.

3.1.1 Information Elements IE

Information Elements IEs sind ein bekannter Mechanismus aus dem Standard IEEE 802.11 (WLAN) und sollen Informationen auf dem MAC Sublayer transportieren, welche vorwiegend für das Management des Netzwerkes dienen.
Man kann dabei zwischen zwei Arten an IEs unterscheiden,

Header IE sind Teil des MAC Headers und liefert Informationen damit der Media Access Control Layer das Frame verarbeiten kann.

Payload IE wird innerhalb des MAC Payload gesendet und soll als Schnittstelle zu höheren Schichten dienen und Informationen weitergeben.

Der Aufbau von IEs ist dabei in beiden Fällen identisch, er beginnt mit dem *Type Descriptor* einem Flag von 1 Bit länge, welcher mit 0 ein Header IE und mit 1 ein Payload IE angibt. Dannach folgt die 8 Bit lange *Element ID* um Angaben “uber den Inhalt anzugeben, dem *Length Field* mit 7 Bit und abschliessend den 11 Bit langen *Content*.
Der exemplarische Aufbau von IEs kann in der Implementierung in *lr-wpan-mac-header.cc* ab Zeile 1270 betrachtet werden. Die Anwendung dieser und spezielle IEs, gerade für den Network Formation Process, fehlen dagegen vollständig und müssen zur weiteren Verwendung erst implementiert werden.

```
//802.15.4 IE Header
if (m_fctrlFrmVer == 2 && m_fctrlIEListPresent == 1) {
  uint8_t lastid;

  do {
    HeaderIE* newie = new HeaderIE;
    uint16_t head = i.ReadLsbtohU16 ();
    newie->length = (head >> 9); //7 bits
    newie->id = (head >> 1); //8bits
    newie->type = 0; //1bit

    for (int j = 0;j<newie->length ;j++) {
      newie->content.push_back(i.ReadU8 ());
    }
    headerie.push_back(*newie);
    lastid = newie->id;
  } while (lastid != 0x7e && lastid != 0x7f);
}
```

3.1.2 Minimaler Aufbau eines Enhanced Beacons

Nachfolgend ist der Aufbau eines EB dargestellt, welcher minimal notwendig ist um die Kommunikation mittels TSCH zu gewährleisten. Das Beispiel mitsamt der Erklärung ist dabei dem Internet-Draft 6tisch-minimal entnommen.

```
1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Len1 =   0   |Element ID=0x7e|0|     Len2 = 26         |GrpId=1|1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Len3 =   6    |Sub ID = 0x1a|0|             ASN
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               ASN                                | Join Priority |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Len4 = 0x01  |Sub ID = 0x1c|0| TT ID = 0x00  |   Len5 = 0x01
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |ID=0x9 |1| CH ID = 0x00  | Len6 = 0x0A   |Sub ID = 0x1b|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   #SF = 0x01  | SF ID = 0x00  |   SF LEN = 0x65 (101 slots)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| #Links = 0x01 |       SLOT OFFSET = 0x0000      |    CHANNEL
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  OFF  = 0x0000 |Link OPT = 0x0F|          NO MAC PAYLOAD
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Der Aufbau entspricht dabei dem Header IE sowie dem Payload IE eines Enhanced Beacons.

Header IE Header Len1 = Header IE Länge, mit dem Wert 0, dem Element ID 0x7e, um anzuzeigen das sofort der Payload IE anschliesst, sowie dem Type mit Wert 0.

Payload IE Header der Payload IE Header gibt mit *Len2* seine Länge mit 26 Byte an, der *GroupID* von 1, was einem MLME Header entspricht sowie dem Type 1.

MLME-SubIE TSCH Synchronization für die Synchronisation wird mit *Len3* ein 6 Byte langer Payload beschrieben, *SubID* gibt den Wert *0x1a* an, was für einen MLME-SubIE TSCH Synchronization Header wirbt. Zusätzlich kommt der *Type* Short durch den Wert 0 zur anwendung, sowie der *Absolut Sequence Number* mit einer Länge von 5 Byte, sowie der *Join Priority*.

MLME-SubIE TSCH TimeSlot beschreibt die Timeslots beginnend mit *Len4* von 1 Byte, der *SubID* mit dem Wert *0x1c* der das IE für den Timeslot (MLME-SubIE Timeslote) angibt. Dazu wird der *Type* als Short (0) sowie der standard *Timeslot template ID* von *0x00* angegeben.

MLME-SubIE Ch. Hopping liefert die Informationen für das Channel Hopping, mit *Len5* einer Längenfeld von 1 Byte, der *SubID* von *0x09* welche den Header als *MLME-SubIE Ch. Hopping* ausgibt, dem *type* und seinem Wert long (1), sowie der *Channel Hopping Sequence ID*.

MLME-SubIE TSCH Slotframe and Link der letzte IE liefert schliesslich die Informationen “uber das TSCH Slotframe und dem sendenen Knoten. Beginnen mit der Länge in Feld *Len6* von 10 Byte, der *SubID* mit Wert *0x1b* der den Anwendungsfall des IE angibt (MLME-SubIE TSCH Slotframe and Link) einem *type* von *short* (0), der *Number of slotframes* mit *0x01*, dem *SlotFrame Handle* mit Wert *0x00*. Das *SlotFrame Size* Feld liefert die Grösse von 101 Slots an (Wert von 0x65), gefolgt von der *Number of Links*, dem aktuellen *Timeslot*, *Channel Offset* und den *Link Option* der mit dem Wert *0x0F* die Optionen *tx, rx, shared* und *timekeeping* angibt.

3.2 Network Formation Process

Der Network Formation Process lässt sich dabei in zwei Bereiche aufteilen. Das “Werben“ (Advertising) um neue Teilnehmer für das Netzwerk, durch bereits im Netzwerk befindliche Knoten, sowie einem “Bootstrapping“-Verfahren das ein Knoten durchführen muss um in das Netzwerk einzutreten. In der nachfolgenden Abbildung 3.1 wird der Prozess in einem Kontrollflussgraphen dargestellt.

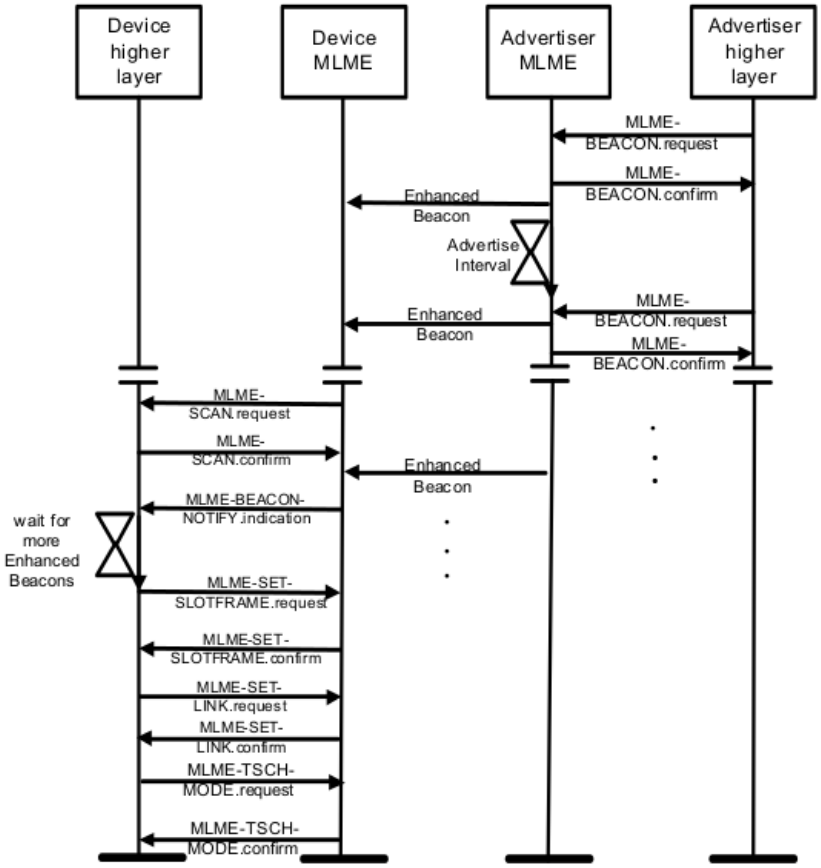

Abbildung 3.1: TSCH Prozedur für Enhanced Beacons [4]

3.2.1 Advertising

Ein Personal Area Network PAN wird erstellt, indem ein Gerät, in der Regel der PAN-Coordinator die Existenz des Netzwerkes bekannt gibt. Dies geschieht durch das Aussenden von Enhanced Beacons, welche Informationen innerhalb von Information Elements enthalten damit Knoten sich zum PAN synchronisieren können. Wie in Abbildung 3.1 abgebildet, beginnt das "Werben" indem ein *MLME-BEACON.request* aus einer höheren Schicht die Initiative gibt, Enhanced Beacons zu senden, welches regelmässig wiederholt wird.
Die enthaltenen Informationen der Enhanced Beacons, wurden konkret bereits im vorherigen Kapitel beschrieben und enthalten:

- Zeit Informationen, damit Knoten sich zum PAN synchronisieren können.
- Channel Hopping Informationen
- Timeslot Informationen, um zu erfahren wann Knoten als Receiver arbeiten um Frames zu empfangen und Acknowledgments zu senden.
- Initial Link- und Slotframe Informationen, "ubermitteln die Informationen damit ein neuer Knoten weiss, wann er auf ankommende Verbindungen hören soll und wann er selber an seinen Advertisment Knoten senden kann.

Im aktuellen Entwicklungsstand wird deutlich, dass die benötigten IEs weder vollständig implementiert sind noch deren gebrauch eingebettet ist. Als Beispiel kann die Methode *McpsDataRequest* aus dem Modul *lr-wpan-tsch-mac.cc* ab Zeile 241 gezeigt werden.

```
1  void
   LrWpanTschMac::McpsDataRequest (TschMcpsDataRequestParams params, Ptr<
       Packet> p)
3  {
   ...
5  if (params.m_frameControlOptions.IesIncluded)
     {
7      macHdr.SetIEField();
       //TODO: Insert IEs
9    }
   else
11   {
       macHdr.SetNoIEField();
13   }
   ...
15 }
```

3.2.2 “Bootstrapping“

Der “Bootstrapping“-Algorithmus beschreibt das Verhalten von Knoten ausserhalb des PAN und deren Aktivitäten diesem beizutreten.
Wie in Abbildung 3.1 gezeigt, wird der Knoten nach dem Erhalt der Primitive *MLME-SCAN.request* nach ankommenden EBs auf mehreren Frequenzen suchen. Dadurch erhält der Knoten die notwendigen Informationen indem er diese aus den EBs extrahiert und nun mit dem PAN sychronisiert. Im Detail wird der Knoten sich an die Attribute der Slotframes und Timeslots anpassen, da er sowohl die ASN wie auch die Timing-Informationen aus den IEs erhalten hat um schließlich die Primitive *MLME-SET-SLOTFRAME* zu aktivieren. Anschliessend konfiguriert der Knoten seinen eigenen MLME Sublayer mit *MLME-SET-LINK* sowie dem aktivieren des TSCH Modes mit *MLME-TSCH-MODE* zur Kommunikation innerhalb des PAN. Ist dieser Algorithmus abgeschlossen, ist der Knoten mit dem PAN innerhalb des richtigen Slotframe und Timeslot synchronisiert und kann durch eigenständiges Senden und Empfangen an diesem teilnehmen. Alle drei Primitiven sind vollständig im Modul *lr-wpan-tsch-mac.cc* implementiert.

- MLME-SET-SLOTFRAME ist ab Zeile 1180 für den Request implementiert.

```
1       void
        LrWpanTschMac::MlmeSetSlotframeRequest (
    MlmeSetSlotframeRequestParams params)
3       {
        ...
5       }
```

- Die Implementierung für MLME-SET-LINK folgt ab Zeile 1258.

```
1       void
        LrWpanTschMac::MlmeSetLinkRequest (MlmeSetLinkRequestParams
    params)
3       {
        ...
5       }
```

- MLME-TSCH-MODE ab Zeile 1368 braucht noch die Abfrage der Synchronisation

```
1       void
        LrWpanTschMac::MlmeTschModeRequest (MlmeTschModeRequestParams
    params)
3       {
```

```
        ...
5       }
```

4 Kernfunktionalitäten in der Entwicklung mit ns-3-dev-TSCH

Dieses Kapitel beschreibt die Kernfunktionalitäten welche für den Umgang und die Entwicklung mit dem Entwicklungsrepository ns-3-dev-TSCH wichtig sind. Dabei werden zuerst die Kernmethoden aus der ns-3 Helperklasse *lr-wpan-tsch-helper* erläutert, bevor weitere allgemeine ns-3 Funktionen näher betrachtet werden.

4.1 Kernmethoden der Helperklasse LrWpanTschHelper

4.1.1 LrWpanTschHelper::Install

Die Methode *LrWpanTschHelper::Install* ab Zeile 309 enhält einen NodeIterator welcher für jeden erstellten Knoten das dazugehörige NetDevice erstellt und ihm den richtigen Channel zuordnet. Wie im Kommentar ersichtlich fehlt bislang die Möglichkeit den Adressenmodus sowie die PanID für die NetDevices bei Erstellung festzulegen, statt dessen muss dieser Vorgang in einem zusätzlichen Schritt durchgeführt werden.

```
1  NetDeviceContainer
   LrWpanTschHelper::Install (NodeContainer c)
3  {
     NetDeviceContainer devices;
5    for (NodeContainer::Iterator i = c.Begin (); i != c.End (); i++)
       {
7        Ptr<Node> node = *i;

9        Ptr<LrWpanTschNetDevice> netDevice = CreateObject<LrWpanTschNetDevice
       > ();
         netDevice->SetChannel (m_channel);
11       node->AddDevice (netDevice);
         netDevice->SetNode (node);
13       // \todo add the capability to change short address, extended
         // address and panId. Right now they are hardcoded in LrWpanTschMac::
       LrWpanTschMac ()
15       devices.Add (netDevice);
       }
17   return devices;
   }
```

4.1.2 LrWpanTschHelper::AssociateToPan

Die Methode *LrWpanTschHelper::AssociateToPan* beginnend ab Zeile 390 durchläuft alle erstellten NetDevices und legt deren PanId und ShortAdresse für den TSCH (NMAC) und Nicht-TSCH (OMAC) Mode fest. Diese Funktionalitäten sollten in jeweils eigene Methoden ausgelagert werden, im besonderen das Festlegen der Adressen um die hardgecodete Implementierung zu ersetzen.

```
void
LrWpanTschHelper::AssociateToPan (NetDeviceContainer c, uint16_t panId)
{
  NetDeviceContainer devices;
  uint16_t id = 1;
  uint8_t idBuf[2];

  for (NetDeviceContainer::Iterator i = c.Begin (); i != c.End (); i++)
    {
      Ptr<LrWpanTschNetDevice> device = DynamicCast<LrWpanTschNetDevice> (*
    i);
      if (device)
        {
          idBuf[0] = (id >> 8) & 0xff;
          idBuf[1] = (id >> 0) & 0xff;
          Mac16Address address;
          address.CopyFrom (idBuf);

          device->GetOMac ()->SetPanId (panId);
          device->GetOMac ()->SetShortAddress (address);
          device->GetNMac ()->SetPanId (panId);
          device->GetNMac ()->SetShortAddress (address);
          id++;
        }
    }
  return;
}
```

4.1.3 LrWpanTschHelper::ConfigureSlotframeAllToPan

Die Methode *LrWpanTschHelper::ConfigureSlotframeAllToPan* ab Zeile 1068 definiert den Schedule der Kommunikation, durch die Reihenfolge der Link-Methoden *AddLink*, *AddAdvLink* und *AddBcastLinks*. Die Methode erstellt am Anfang eine Anzahl an Slotframes in Abhängigkeit zur Anzahl der NetDevices, Empty Timeslots sowie einem zusätzlichen Broadcast und der Möglichkeit der bidirektionalen Verbindung. Anschliessend werden ein Advertisment Link im Timeslot 0 hinzugefügt, der optionale Broadcast Link sowie den Direktverbindung an beteiligten NetDevices in einfacher und bidirektionaler Verbindung. Diese Methode sowie deren verwendeten Helper-Methoden müssen überarbeitet werden, damit ein selbstgewählter Schedule mit definierten Variablen anwendbar ist, ansonsten können nur die im Rahmen der Parameter entsprechend Kommunikationsabläufe definiert werden.

```
void
LrWpanTschHelper::ConfigureSlotframeAllToPan(NetDeviceContainer devs, int
    empty_timeslots, bool bidir, bool bcast)
{
  int size = (bcast ? 2 : 1) + (bidir ? 2 : 1)*(devs.GetN ()-1) +
    empty_timeslots;

  AddSlotframe(devs,m_slotframehandle,size);

  //Add links
  AddLinkParams alparams;
  alparams.slotframeHandle = m_slotframehandle;
  alparams.channelOffset = 0;

  alparams.linkHandle = 0;
  alparams.timeslot = 0;
  AddAdvLink (devs,0,alparams);

  if (bcast) {
    alparams.linkHandle = 1;
    alparams.timeslot = 1;
    AddBcastLinks (devs,0,alparams);
  }

  uint16_t c=(bcast ? 2 : 1);

  for (u_int32_t i = 0; i < devs.GetN ()-1; i++,c++)
    {
      alparams.linkHandle = c;
      alparams.timeslot = c;
      AddLink(devs,i+1,0,alparams,false);
    }

  if (bidir)
```

```
    for (u_int32_t i = 0; i < devs.GetN ()-1; i++,c++)
34  {
      alparams.linkHandle = c;
36    alparams.timeslot = c;
      AddLink(devs,0,i+1,alparams,false);
38  }
   m_slotframehandle++;
40 }
```

4.1.4 LrWpanTschHelper::AddAdvLink und LrWpanTschHelper::AddBcastLinks

Die beiden Methoden *LrWpanTschHelper::AddAdvLink* in Zeile 868 und *LrWpanTschHelper::AddBcastLinks* in Zeile 904 beschreiben den Advertisment- bzw. Broadcast-Link. Nachfolgend ist der Code für den Advertisment Link abgebildet. Der Unterschied zwischen beiden Linktypen liegt in der Verarbeitung der Sende und Empfangsverbindung. Der Broadcast Link erstellt MlmeSetLinkRequests an alle Knoten und sendet an die Adresse *ff:ff* während der Transmitphase. Im Laufe der Receive Phase hört er auf die Adresse *ff:ff* und "offnet Links an alle Knoten mit einem weiteren MlmeSetLinkRequest. Im aktuellen Entwicklungsstand ist der Broadcast Link nicht anwendbar, da säemtliche Links innerhalb des PANs in den Sendenmodus geschaltet werden und auch diesen während des Timeslots nicht verlassen, weshalb zwar Pakete gesendet aber nicht empfangen werden können. Für unseren Anwendungsfall soll dieser allerdings den "Join Request" eines beitretenden Knoten beinhalten, weshalb die Implementierung geändert werden muss. Da zumindest der PAN Koordinator die Pakete empfangen muss gehen wir in Zukunft davon aus, dass der PAN Koordinator das erste NetDevice darstellt. Konkret wird künftig nicht erste NetDevice nicht mehr in den Transmit Modus geschalten, indem die for-Schleife erst mit dem zweiten NetDevice beginnt zu zählen.

```
//10000000 to transmit
2 linkRequest.linkOptions.reset();
linkRequest.linkOptions.set(0,1);
4 linkRequest.linkOptions.set(2,1);
linkRequest.linkType = MlmeSetLinkRequestlinkType_ADVERTISING;
6 linkRequest.nodeAddr = Mac16Address("ff:ff");
for ( u_int32_t i = 1;i < devs.GetN ();i++)
8   {
        linkRequest.TxID = i;
10      linkRequest.RxID = coordinatorPos;
        linkRequest.linkFadingBias = FadingBias[coordinatorPos][i];
12      devs.Get(i)->GetObject<LrWpanTschNetDevice> ()->GetNMac()->
    MlmeSetLinkRequest (linkRequest);
  }
```

Der Advertisment Link dagegen sendet in der Transmit Phase ebenfalls an die Adresse *ff:ff*, erstellt aber nur einen MlmeSetLinkRequest an ein bestimmten NetDevice. Im Verlauf der Receive Phase wird nur auf die Adresse des Senders gehört und nur zu diesem ein Link aufgebaut.
Die Methode *LrWpanTschHelper::AddAdvLink* wird für den Network Formation Process benötigt und erhäelt in der Variable *u_int32_t senderPos* das beizutretende NetDevice. Aufgrund der momentanen Implementierung, können wir den Broadcast Link leider nicht für Broadcast Transmits verwenden, weshalb der Advertisment Link diese Aufgabe “ubernehmen wird, solange dieser Umstand nicht geändert wird.

```
void
LrWpanTschHelper::AddAdvLink(NetDeviceContainer devs, u_int32_t senderPos,
    AddLinkParams params)
{
  MlmeSetLinkRequestParams linkRequest;
  linkRequest.Operation = MlmeSetLinkRequestOperation_ADD_LINK;
  linkRequest.linkHandle = params.linkHandle;
  linkRequest.slotframeHandle = params.slotframeHandle;
  linkRequest.Timeslot = params.timeslot;
  linkRequest.ChannelOffset = params.channelOffset;

  //10000000 to transmit
  linkRequest.linkOptions.reset();
  linkRequest.linkOptions.set(0,1);
  linkRequest.linkType = MlmeSetLinkRequestlinkType_ADVERTISING;
  linkRequest.nodeAddr = Mac16Address("ff:ff");
  linkRequest.linkFadingBias = NULL;
  linkRequest.TxID = senderPos;
  linkRequest.RxID = 0;
  devs.Get(senderPos)->GetObject<LrWpanTschNetDevice> ()->GetNMac()->
    MlmeSetLinkRequest (linkRequest);

  //01010000 to receive
  linkRequest.linkOptions.reset();
  linkRequest.linkOptions.set(1,1);
  linkRequest.linkOptions.set(3,1);
  linkRequest.nodeAddr = Mac16Address::ConvertFrom(devs.Get(senderPos)->
    GetAddress());
  for (u_int32_t i = 0;i < devs.GetN ();i++)
      if (i != senderPos) {
          linkRequest.linkFadingBias = FadingBias[i][senderPos];
          linkRequest.TxID = senderPos;
          linkRequest.RxID = i;
          devs.Get(i)->GetObject<LrWpanTschNetDevice> ()->GetNMac()->
    MlmeSetLinkRequest (linkRequest);
      }
}
```

4.1.5 LrWpanTschHelper::GenerateTraffic

Mit den bislang beschriebenen Hilfsmethoden wird der Schedule innerhalb des TSCH Mode fest und entsprechen damit den Möglichkeiten die höhere Schichten besitzen um Routing und Datenverkehr zu beschreiben. Dadurch wird zwar vorgegeben wann und wie die Kommunikation innerhalb eines PAN stattfindet, der eigentliche Austausch von Paketen zwischen den Knoten wird aber von einer anderen Hilfsmethode geregelt.
Diese Funktionalität wird von der Methode *LrWpanTschHelper::GenerateTraffic* ab Zeile 1129 wiedergespiegelt. Dabei wird ein Simulator Schedule gestartet, welcher einen Ablauf an zeitdiskreten Events beschreibt wodurch MAC-Frames zwischen dem ausgewählten NetDevice *dev* und seinem Ziel *dst* gesendet werden. Die Parameter *start* und *duration* geben dabei die Gesamtlänge der Kommunikation an. Wichtig ist zusätzlich der Parameter *interval*, welcher die zeitlichen Abstände zwischen zwei Events angibt. Die Gewichtung dieses Parameter wird im

```
1 void
  LrWpanTschHelper::GenerateTraffic(Ptr<NetDevice> dev, Address dst, int
      packet_size, double start, double duration, double interval)
3 {
    double end = start+duration;
5   Simulator::Schedule(Seconds(start),&LrWpanTschHelper::SendPacket,this,dev
      ,dst,packet_size,interval,end);
  }
```

4.2 Logging und Tracing

Das Logging in ns3 ermöglicht genaue Informationen “uber den genauen Ablauf der Netzwerksimulation zu erhalten wird das Logging eingesetzt, damit können genauere Informationen erhalten werden, als beim “ublichen Tracing. Da die Informationsfülle extrem ansteigt wurde für die einzelnen Loggingkomponenten eine Hilfsmethode geschrieben.

```
  void
2 LogComponents(bool phy, bool csmaca, bool diverse)
  {
4   LogComponentEnableAll (LOG_PREFIX_TIME);
    LogComponentEnableAll (LOG_PREFIX_FUNC);
6   LogComponentEnable ("LrWpanTschMac", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanTschNetDevice", LOG_LEVEL_ALL);
8   if (phy)
    {
10  LogComponentEnable ("LrWpanPhy", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanSpectrumSignalParameters", LOG_LEVEL_ALL);
12  LogComponentEnable ("LrWpanSpectrumValueHelper", LOG_LEVEL_ALL);
    }
14  if (csmaca)
```

```
    {
16  LogComponentEnable ("LrWpanCsmaCa", LOG_LEVEL_ALL);
    }
18  if (diverse)
    {
20  LogComponentEnable ("LrWpanErrorModel", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanInterferenceHelper", LOG_LEVEL_ALL);
22  }
}
```

Als Beispiel wird ein Ausschnitt herangezogen, der am Anfang eines neuen diskreten Events steht und die TSCH Funktionalität der Simulation angibt. Darin meldet zuerst das LrWpanTschNetDevice *00:02*, dass Daten an das Gerät mit der MAC-Adresse *02-02-00:01* gesendet werden. Nachdem die Primitiven gesetzt wurden, inkrementieren zuerst alle Geräte iher ASN und geben ihre Aktion in diesem Timeslot an. Im Beispiel empfängt *00:01* Daten und richtet sich darauf ein, diese zu Empfangen. Das Gerät *00:02* erkennt ein Paket in seiner Warteliste und startet die Sendeprozedur.

```
1  0.02s LrWpanTschNetDevice:Send(0x14ee770, 0x14fd5f0, 02-02-00:01, 34525)
   0.02s LrWpanTschNetDevice:GetMtu(0x14ee770)
3  0.02s [address 00:02] LrWpanTschMac:McpsDataRequest(0x14ee800, 0x14fd5f0)
   0.02s [address 00:02] LrWpanTschMac:SetTxLinkQueue(0x14ee800)
5  0.02s [address 00:02] LrWpanTschMac:SetTxLinkQueue(): Enqueuing packet with
       SeqNum = 108 in queue with link sequence = 0
   0.02s [address 00:01] LrWpanTschMac:IncAsn(0x14ec530)
7  0.02s [address 00:02] LrWpanTschMac:IncAsn(0x14ee800)
   0.02s [address 00:03] LrWpanTschMac:IncAsn(0x14f0c50)
9  0.02s [address 00:04] LrWpanTschMac:IncAsn(0x14f3080)
   0.02s [address 00:05] LrWpanTschMac:IncAsn(0x14f54e0)
11 0.02s [address 00:06] LrWpanTschMac:IncAsn(0x14f7980)
   0.02s [address 00:07] LrWpanTschMac:IncAsn(0x14f9da0)
13 0.02s [address 00:08] LrWpanTschMac:IncAsn(0x14fc1f0)
   0.02s [address 00:01] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
      Queue size = 0
15 0.02s [address 00:01] LrWpanTschMac:ScheduleTimeslot(): Link found at
      timeslot 2
   0.02s [address 00:01] LrWpanTschMac:ScheduleTimeslot(): TSCH Changing to
      channel 18
17 0.02s [address 00:01] LrWpanTschMac:ScheduleTimeslot(): 0x14ec530setting
      for channel 18 fading bias: 1
   0.02s [address 00:01] LrWpanTschMac:ScheduleTimeslot(): Start timeslot
      receiving procedure
19 0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
      Queue size = 1
   0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): Link found at
      timeslot 2
21 0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): TSCH Changing to
      channel 18
```

```
0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): 0x14ee800setting
    for channel 18 fading bias: 1
0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): Queue contained
    link size = 1
0.02s [address 00:02] LrWpanTschMac:FindTxPacketInEmptySlot(0x14ee800)
0.02s [address 00:02] LrWpanTschMac:FindTxPacketInEmptySlot(): Find Tx
    packet in queue with link position = 0 with queue size = 1
0.02s [address 00:02] LrWpanTschMac:ScheduleTimeslot(): Start timeslot
    transmiting procedure, seqnum = 108
0.02s [address 00:03] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:03] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
0.02s [address 00:04] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:04] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
0.02s [address 00:05] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:05] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
0.02s [address 00:06] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:06] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
0.02s [address 00:07] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:07] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
0.02s [address 00:08] LrWpanTschMac:ScheduleTimeslot(): Timeslot 2 ts = 2
    Queue size = 0
0.02s [address 00:08] LrWpanTschMac:ScheduleTimeslot(): No link in this
    timeslot, turning off the radio
```

Zum tracen der einzelnen Pakete, wird allen vorran die Methode *EnablePcap* verwendet um die pcap Dateien mitzuschneiden, welche anschliessend zur Netzwerkanalyse in Wireshark ausgewertet werden können.

```
lrWpanHelper.EnablePcap (std::string ("tsch_scenario"), panCoordNetDevice,
    true);
```

5 Experimente mit ns-3-dev-TSCH

Zur Analyse des Codes und des künftigen Anwendungsfall wurden mehrere Beispielscripte erstellt um verschiedene Aspekte zu analysieren, welche nachfolgend erläutert werden.

5.1 Kommunikation und Scheduling

Als Grundlage und Kernaufgabe wurde die normale Kommunikation innerhalb des TSCH Mode definiert, wobei an ein solch erstelltes PAN mehrere Anfoderungen festgelegt wurden.

- Star bzw. Mesh TSCH Netzwerk,
- 7 FFDs und 1 PAN-Coordinator,
- Slotframe Size von 8, wobei der erste Slot für das Senden der EBs vom PAN-Coordinator festgelegt wird,
- 100 Bytes an Daten pro gesendetem Paket von jedem der FFDs zum PAN-Coordinator, wobei eine Fragmentierung der Pakete vermieden werden soll,
- kein Network-Formation,
- und das Network Scheduling ist beliebig, wobei Kollisionen und Shared-Slots auch vermieden werden.

Für die Topologie ist das bereits im vorherigen Kapitel zum Thema Mobility vorgestellte Codesnipped verantwortlich. Dabei wird ein Netzwerk aufgebaut, wobei alle 8 Knoten anhand eines Grids auf zwei Reihen auf dem Koordinatensystem verteilt werden.
Wie im vorherigen Kapitel angeschnitten, definierten die Hilfsmethoden *AddAdvLink*, *AddLink* und *AddBcastLinks* das Scheduling zwischen den Knoten und die Methode *GenerateTraffic* die tatsächliche Kommunikation.
Im Script wurde dabei zuerst die vorgegebene Hilfsmethode *ConfigureSlotframeAllToPan* mit unseren vorgegebenen Anfoderungen verwendet, welche folgenden Schedule vorgibt:

1. Advertisment Link Dauer: 1 Timeslot
2. Broadcast Link Dauer: 1 Timeslot
3. Unidirektionale Verbindung zwischen den FFDs und dem PAN Koordinator 7 Timeslots

Dieser Schedule mit einer Dauer von 9 Timeslots wird nun anschliessend für die Dauer des Scripts wiederholt. Für die tatsächliche Kommunikation ist nun das nachfolgende Codesnipped verantwortlich, womit zuerst allgemein der PAN Koordinator an alle FFDs ein Frame sendet und anschliessend in aneinanderfolgenden Timeslots jedes FFD sein Frame an den PAN Koordinator sendet.
Ein wichtiger Unterschied muss erläutert werden, zwischen dem Schedule eines Slotframes im Gegensatz zur tatsächlichen Kommunikation zwischen den Knoten. Der Schedule umfasst dabei einen vollen Slotframe und wird wiederholt und dauert wie im Beispiel angegeben einen Zeitraum von 9 Timeslots. Die Kommunikation dagegen selber wird durch die Methode *LrWpanHelper::GenerateTraffic* vorgegeben, wodurch im Abstand von vorgegeben Intervallzyklen, durch die Variable *interval*, jeweils ein zeitdiskretes Event zum Senden der Frames festgelegt.

```
// Packets from panCoord
Ptr<Node> panCoordNode = panCoord.Get (0);
Ptr<NetDevice> panCoordNetDevice = panCoordNode->GetDevice (0);
Address address_panCoord = panCoordNetDevice->GetAddress ();
//Broadcast address
Mac16Address brdcst ("ff:ff");
lrWpanHeelper.GenerateTraffic (panCoordNetDevice, brdcst, pktsize, 0.0,
    duration, interval);
// FFDs
for (NodeContainer::Iterator i = ffds.Begin (); i != ffds.End (); i++)
{
  Ptr<Node> node = *i;
  Ptr<NetDevice> device = node->GetDevice (0);
  lrWpanHelper.GenerateTraffic (device, address_panCoord, pktsize,
    starttopancoord, duration, interval);
  starttopancoord += 0.01;
}
```

5.2 Script Kollision und Mobility

Durch folgendes Script soll ermittelt werden wie sich der Entwicklungsstand bei Kollisionen innerhalb des gleichen Timeslots verhält. Die dafür verwendete Topologie besteht aus einem PAN Koordinator an welchen zwei FFDs im gleichen Timeslot “uber einen Shared Link diesem ein Paket senden womit es zum Kollisionsfall kommt. Wie sich später herausstellt, beeinflusst die Position der Knoten zueinander das Ergebnis.
Das Mobility Modul beschreibt die Position und Bewegung von Knoten im Netzwerk. Die wichtigsten Einstellungen für unseren Anwendungszweck sind dabei,

Mobility Model definiert die Bewegung von Knoten, wobei wir diese dauerhaft auf einer festen Position fixieren wollen. Dafür wird die Einstellung *ns3::ConstantPositionMobilityModel* verwendet.

Position Allocator setzt die (Start-)Position der Knoten fest. Dabei stehen mehrere Allokatoren zur Auswahl, wobei wir den *ns3::GridPositionAlloctor* verwenden, der alle Knoten anhand eines Koordinatensystem mit festen Abständen zueinander erstellt

Als Vorlage zur Positionsfestlegung diente dabei folgendes Codesnipped:

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "GridWidth", UintegerValue(2),
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5),
                               "DeltaY", DoubleValue (5),
                               "LayoutType", StringValue ("RowFirst"));
mobility.Install (allNodes);
```

Daraus ergibt sich wie in nachfolgender Darstellung sichtbar, die Verteilung der Knoten auf dem Grid, woran wir die Ergebnisse erläutern.

```
// FFD2 --
// PANC -- FFD1
```

Unterschiedliche Entfernung der FFDs zum PAN Koordinator Dieser Fall wird durch die Variablen *DeltaX* und *DeltaY* festgelegt, dabei zeigt das Ergebnis, dass beide FFDs nach einem erfolgreichen Clear Channel Assessment ihr Frame senden. Der PAN Koordinator akzeptiert aber nur das Frame dessen FFD das eine geringere

Entfernung aufweist, da dieses eine grössere Signalstärke aufweist. Deshalb sendet der PAN Koordinator ein Ack-Frame mit der Sequenznummer des akzeptieren FFD. Im nächsten Schritt empfangen beide FFDs dieses ACK-Frame und prüfen dessen Sequenznummer, wobei nur ein FFD dieses akzeptiert. Das andere FFD wird daraufhin im nächsten möglichen Timeslot versuchen das somit abgelehnte Frame abermals zu senden, sobald der Backoff-Algorithmus dies zulässt.

Gleiche Entfernung zueinander Ein Sonderfall ergibt sich, wenn beide FFDs die selber Entfernung und somit die gleiche Signalstärke zum PAN Koordinator aufweisen. Dabei wird zwar der gleiche Ablauf sichtbar, dass ein FFD erfolgreich sein Frame senden und das ACK-Frame empfangen und akzeptieren kann. Dieses FFD ist aber grundsätzlich immer FFD Nummer 1. Dies wird durch die Implementierung von ns3 als diskreter Eventsimulator festgelegt, wobei ns3 iterativ alle Knoten pro Zeitpunkt durchläuft und damit zuerst FFD1 berechnet.

Das Verhalten der Knoten nach einer solchen Kollision wird durch den weiteren Schedule innerhalb des Slotframes festgelegt. Werden zusätzliche dedizierte Links zwischen den FFDs zum PAN Koordinator zu einem späteren Timeslot innerhalb des selben Slotframes festgelegt, wird das zum Kollisionszeitpunkt abgelehnte Frame erfolgreich gesendet. Damit ist der TSCH CSMA-CA Backoff Algorithmus korrekt implementiert.
Fehlen diese dedizierten Links dagegen, wird das abgelehnte Frame erst im nächsten Slotframe gesendet und kann bei anhaltender Kollision für einen langen Zeitraum blockiert werden, womit alle in der Warteschlange befindlichen Frames des Knoten und damit dieser selber blockiert.
Das allgemeine Verhalten im TSCH Mode wiederspricht aber den Vorgaben während einer Kollision, weshalb das Kollisionsverhalten mit der ursprünglichen LRWPAN Version ausserhalb des TSCH Mode verglichen wurde. Die Anforderungen und Vorraussetzungen sind beibehalten worden.
Dabei zeigt sich das im normalen LRWPAN Modus im Kollisionsfall das Clear Channel Assessment einem Knoten, in Abhängigkeit von der Entfernung zum PAN Koordinator, eine Backoffzeit zuteilt und diesen damit daran hindert sein Frame zu senden. Damit wurde eine mögliche Kollision verhindert und die Kommunikation wird fortgesetzt.
Aufgrund diesen Vergleichs mit der normalen LRWPAN Version zeigt sich, dass das Clear Channel Assessment im TSCH Mode Fehler aufweist, indem es im Kollisionsfall nicht aktiv wird. Dafür wurde aber im Standard der TSCH CSMA-CA Backoff Algorithmus eingeführt, damit genau dieser Sonderfall behandelt werden kann. Wie Abbildung 5.1 zeigt, wird im Kollisionsfall eine Backoffzeit eingesetzt, in welcher der Knoten nicht versucht Frames zu senden obwohl der Schedule aktive Links festgesetzt hat. Da unsere Beispiele dieses Verhalten gezeigt haben, muss abschließend festgestellt werden, dass das Kollisionsverhalten vollständig und korrekt implementiert ist.

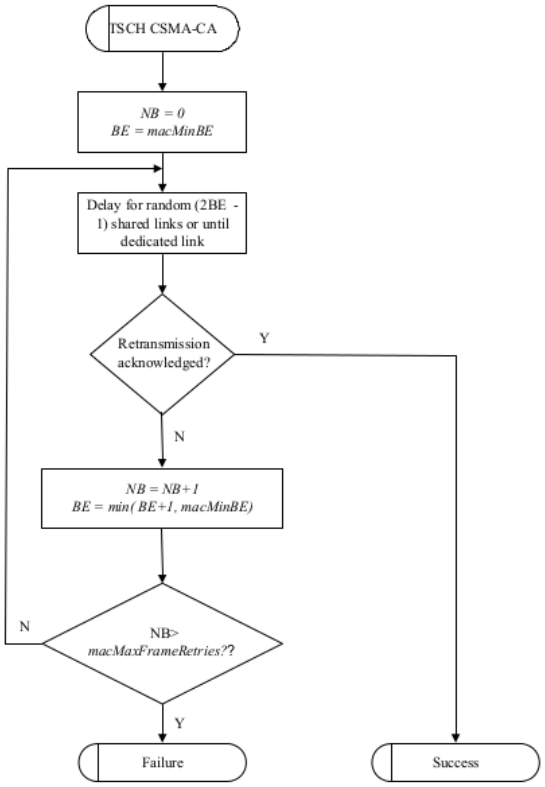

Abbildung 5.1: TSCH CSMA-CA Backoff Algorithmus [4]

6 Fazit und weiterführende Arbeiten

6.1 Grober Entwicklungsplan

Anhand des Implementierungsstands müssen einige Entwicklungsarbeiten vorgenommen werden, damit der Network Formation Process durchgefüehrt werden kann. Nachfolgend wird nun im Groben beschrieben was ein solcher Entwicklungsplan beinhaltet.

1. Anpassen bzw. Neuentwicklung der Helpermethoden zur Erstellung von beliebigen Schedules mit beliebigen Variablenwerten für Slotframes, Timeslots, ChannelOffset, SlotframeHandle mit den Linktypen Link, Advertisment Link und Broadcast Link.

2. Analyse und Anpassung der aktuellen Grobstruktur der Information Elements

3. Einbettung der IEs in die Module zur aktiven Anwendung

4. Entwicklung von Helper Methoden zum Senden und Empfangen von IEs
5. Helper Methoden und Einbettung der konkreten IEs innerhalb des Network Formation Process, wie in 3.1.2 beschrieben
   - Header IE Header
   - Payload IE Header
   - MLME-SubIE TSCH Synchronization
   - MLME-SubIE TSCH Timeslot
   - MLME-SubIE Channel Hopping
   - MLME-SubIE TSCH Slotframe and Link
6. Beispiel zur Anwendung der IEs
7. Implementierung des Advertisment Modes zum Aussenden der Enhanced Beacons
8. Anpassen des “Bootstrapping“-Algorithmus anhand der neuen Implementierung.

7 Appendix: Scriptimplementierung

7.1 tsch_scenario.cc

```
#include <ns3/log.h>
#include <ns3/core-module.h>
#include <ns3/network-module.h>
#include <ns3/lr-wpan-module.h>
#include <ns3/simulator.h>
#include <ns3/single-model-spectrum-channel.h>
#include <ns3/packet.h>
#include <ns3/mobility-module.h>
#include <ns3/spectrum-helper.h>
#include <string>

#include <iostream>

using namespace ns3;

////////////////////////////////
// Configuration
////////////////////////////////
bool verbose = true;        // enable logging
uint32_t numberOfFFDs = 7;       // FFDs
bool brdcst_as_join = true;
bool collision = true;

int pktsize = 91;            //size of packets, in bytes
double duration = 1;        //simulation total duration, in seconds
double starttopancoord = 0.02;         // Starttime transmitting FFDs
double interval = 0.2;
double starttoffds = 0.08;

void
LogComponents(bool phy, bool csmaca, bool diverse)
{
  LogComponentEnableAll (LOG_PREFIX_TIME);
  LogComponentEnableAll (LOG_PREFIX_FUNC);
  LogComponentEnable ("LrWpanTschMac", LOG_LEVEL_ALL);
  LogComponentEnable ("LrWpanTschNetDevice", LOG_LEVEL_ALL);
  if (phy)
  {
    LogComponentEnable ("LrWpanPhy", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanSpectrumSignalParameters", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanSpectrumValueHelper", LOG_LEVEL_ALL);
  }
  if (csmaca)
  {
    LogComponentEnable ("LrWpanCsmaCa", LOG_LEVEL_ALL);
  }
```

```
  if (diverse)
  {
    LogComponentEnable ("LrWpanErrorModel", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanInterferenceHelper", LOG_LEVEL_ALL);
  }

}

void
EnableTsch(LrWpanTschHelper* lrWpanHelper,NetDeviceContainer& netdev)
{
  lrWpanHelper->EnableTsch(netdev,0,duration);
}

int main (int argc, char** argv)
{

  CommandLine cmd;
  cmd.AddValue ("verbose", "Print trace information if true", verbose);
  cmd.Parse (argc, argv);
  GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

  // ------------------------------
  // Nodes
  NodeContainer ffds;
  NodeContainer panCoord;
  ffds.Create (numberOfFFDs);
  panCoord.Create (1);
  NodeContainer lrwpanNodes(panCoord, ffds);

  /////////////////////////////////
  // Mobility
  /////////////////////////////////

  MobilityHelper mobility;
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "GridWidth", UintegerValue(4),
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (5),
                                 "DeltaY", DoubleValue (5),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.Install (lrwpanNodes);

  /////////////////////////////////
  // Channel
  /////////////////////////////////
  SpectrumChannelHelper channelHelper = SpectrumChannelHelper::Default ();
  channelHelper.SetChannel ("ns3::MultiModelSpectrumChannel");
```

```
  Ptr<SpectrumChannel> channel = channelHelper.Create ();
98
  ////////////////////////////////
100 // Configure lrwpan nodes
  ////////////////////////////////
102 LrWpanTschHelper lrWpanHelper(channel,numberOfFFDs+1,false,true);

104 // Add and install the LrWpanTschNetDevice for each node
  NetDeviceContainer netdev = lrWpanHelper.Install (lrwpanNodes);
106
  // AssociateToPan
108 lrWpanHelper.AssociateToPan(netdev,123);

110 // Slotframes
  lrWpanHelper.ConfigureSlotframeAllToPan(netdev,0,false,true); //
    slotframes = 8
112
  // start with TSCH
114 Simulator::Schedule(Seconds(0),&EnableTsch,&lrWpanHelper,netdev);

116 // Packets from panCoord
  Ptr<Node> panCoordNode = panCoord.Get (0);
118 Ptr<NetDevice> panCoordNetDevice = panCoordNode->GetDevice (0);
  Address address_panCoord = panCoordNetDevice->GetAddress ();
120
  // panCoord
122 // Broadcast Address
  Mac16Address brdcst ("ff:ff");
124 // Advertisment Link
  lrWpanHelper.GenerateTraffic (panCoordNetDevice, brdcst, pktsize, 0.0,
    duration, interval);
126
  // Broadcast Link
128 if (brdcst_as_join)
  {
130   // Send File from FFD to pancoord during Brdcst link
    Ptr<Node> brdcst_ffd_one = ffds.Get (0); // 00:01
132   Ptr<NetDevice> brdcst_ffd_one_netdev = brdcst_ffd_one->GetDevice (0);
    Ptr<Node> brdcst_ffd_two = ffds.Get (1); // 00:02
134   Ptr<NetDevice> brdcst_ffd_two_netdev = brdcst_ffd_one->GetDevice (0);

136   if (collision)
    {
138     // send to pancoord
      lrWpanHelper.GenerateTraffic (brdcst_ffd_one_netdev, address_panCoord
    , pktsize, 0.1, duration, interval);
140     lrWpanHelper.GenerateTraffic (brdcst_ffd_two_netdev, address_panCoord
    , pktsize, 0.1, duration, interval);
      // send brdcst
```

```
142         lrWpanHelper.GenerateTraffic (brdcst_ffd_one_netdev, brdcst, pktsize,
        0.1, duration, interval);
            lrWpanHelper.GenerateTraffic (brdcst_ffd_two_netdev, brdcst, pktsize,
        0.1, duration, interval);
144       }
          else
146       {
            lrWpanHelper.GenerateTraffic (brdcst_ffd_one_netdev, address_panCoord
        , pktsize, 0.1, duration, interval);
148       }
      }
150   else
      {
152     //Address brdcst = panCoordNetDevice->GetBroadcast (); // ff:ff
        lrWpanHelper.GenerateTraffic (panCoordNetDevice, brdcst, pktsize, 0.1,
        duration, interval);
154   }

156   // FFDs
      for (NodeContainer::Iterator i = ffds.Begin (); i != ffds.End (); i++)
158   {
        Ptr<Node> node = *i;
160     Ptr<NetDevice> device = node->GetDevice (0);
        lrWpanHelper.GenerateTraffic (device, address_panCoord, pktsize,
        starttopancoord, duration, interval);
162     starttopancoord += 0.01; // starttopancoord = 0.02
      }
164
      //Enable PCAP and Ascii Tracing
166   AsciiTraceHelper ascii;
      Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("tsch_scenario.
        tr");
168   lrWpanHelper.EnablePcapAll (std::string ("tsch_scenario"), true);
      lrWpanHelper.EnablePcap (std::string ("tsch_scenario"), panCoordNetDevice
        , true);
170   lrWpanHelper.EnableAsciiAll (stream);
      if (verbose)
172     {
          //lrWpanHelper.EnableLogComponents ();
174       LogComponents(false, false, false);
        }
176
      // ----------------------------------

178
      Simulator::Run ();
180   Simulator::Destroy ();
      return 0;
182 }
```

7.2 tsch_scenario_collision.cc

```
#include <ns3/log.h>
#include <ns3/core-module.h>
#include <ns3/network-module.h>
#include <ns3/lr-wpan-module.h>
#include <ns3/simulator.h>
#include <ns3/single-model-spectrum-channel.h>
#include <ns3/packet.h>
#include <ns3/mobility-module.h>
#include <ns3/spectrum-helper.h>
#include <string>

#include <iostream>

using namespace ns3;

////////////////////////////////
// Configuration
////////////////////////////////
bool verbose = true;        // enable logging
uint32_t numberOfFFDs = 2;       // FFDs

int pktsize = 91;            //size of packets, in bytes
double duration = 1;        //simulation total duration, in seconds
double starttopancoord = 0.01;         // Starttime transmitting FFDs
double interval = 0.1;
double starttoffds = 0.08;


void
LogComponents(bool phy, bool csmaca, bool diverse)
{
  LogComponentEnableAll (LOG_PREFIX_TIME);
  LogComponentEnableAll (LOG_PREFIX_FUNC);
  LogComponentEnable ("LrWpanTschMac", LOG_LEVEL_ALL);
  LogComponentEnable ("LrWpanTschNetDevice", LOG_LEVEL_ALL);
  if (phy)
  {
    LogComponentEnable ("LrWpanPhy", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanSpectrumSignalParameters", LOG_LEVEL_ALL);
    LogComponentEnable ("LrWpanSpectrumValueHelper", LOG_LEVEL_ALL);
  }
  if (csmaca)
  {
    LogComponentEnable ("LrWpanCsmaCa", LOG_LEVEL_ALL);
  }
  if (diverse)
  {
    LogComponentEnable ("LrWpanErrorModel", LOG_LEVEL_ALL);
```

```
49      LogComponentEnable ("LrWpanInterferenceHelper", LOG_LEVEL_ALL);
    }

}

void
EnableTsch(LrWpanTschHelper* lrWpanHelper,NetDeviceContainer& netdev)
{
  lrWpanHelper->EnableTsch(netdev,0,duration);
}

int main (int argc, char** argv)
{

  CommandLine cmd;
  cmd.AddValue ("verbose", "Print trace information if true", verbose);
  cmd.Parse (argc, argv);
  GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

  // ------------------------------
  // Nodes
  NodeContainer ffds;
  NodeContainer panCoord;
  ffds.Create (numberOfFFDs);
  panCoord.Create (1);
  NodeContainer lrwpanNodes(panCoord, ffds);

  ////////////////////////////////
  // Mobility
  ////////////////////////////////

  MobilityHelper mobility;
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "GridWidth", UintegerValue(4),
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (5),
                                 "DeltaY", DoubleValue (5),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.Install (lrwpanNodes);

  ////////////////////////////////
  // Channel
  ////////////////////////////////
  SpectrumChannelHelper channelHelper = SpectrumChannelHelper::Default ();
  channelHelper.SetChannel ("ns3::MultiModelSpectrumChannel");
  Ptr<SpectrumChannel> channel = channelHelper.Create ();

  ////////////////////////////////
```

```
99  // Configure lrwpan nodes
    ////////////////////////////////
101 LrWpanTschHelper lrWpanHelper(channel,numberOfFFDs+1,false,true);

103
    // Add and install the LrWpanTschNetDevice for each node
105 NetDeviceContainer netdev = lrWpanHelper.Install (lrwpanNodes);

107 // ------------------------------;

109 // AssociateToPan
    lrWpanHelper.AssociateToPan(netdev,123);
111
    // Slotframes
113 int size = 20;
    lrWpanHelper.AddSlotframe(netdev, 0, size);
115
    //Add links
117 AddLinkParams alparams;
    alparams.slotframeHandle = 0;
119 alparams.channelOffset = 0;

121 uint16_t c = 1;

123 for (u_int32_t i = 0; i < netdev.GetN ()-1; i++,c++)
      {
125     alparams.linkHandle = c;
        alparams.timeslot = 1;
127     lrWpanHelper.AddLink(netdev,i+1,0,alparams,true);
      }
129
    // start with TSCH
131 Simulator::Schedule(Seconds(0),&EnableTsch,&lrWpanHelper,netdev);

133 // Packets from panCoord
    Ptr<Node> panCoordNode = panCoord.Get (0);
135 Ptr<NetDevice> panCoordNetDevice = panCoordNode->GetDevice (0);
    Address address_panCoord = panCoordNetDevice->GetAddress ();
137 std::cout << address_panCoord << " ";

139 // Broadcast Address
    Mac16Address mac16_brdcst ("ff:ff");
141
    // FFDs
143 // Send File from FFD to pancoord during Brdcst link
    Ptr<Node> ffd_one = ffds.Get (0); // 00:01
145 Ptr<NetDevice> ffd_one_netdev = ffd_one->GetDevice (0);
    Ptr<Node> ffd_two = ffds.Get (1); // 00:02
147 Ptr<NetDevice> ffd_two_netdev = ffd_two->GetDevice (0);
```

```
149   // send to pancoord
      lrWpanHelper.GenerateTraffic (ffd_one_netdev, address_panCoord, pktsize,
        0.1, duration, interval);
151   lrWpanHelper.GenerateTraffic (ffd_two_netdev, address_panCoord, pktsize,
        0.1, duration, interval);

153   //Enable PCAP and Ascii Tracing
      AsciiTraceHelper ascii;
155   Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("
        tsch_scenario_collision.tr");
      lrWpanHelper.EnablePcapAll (std::string ("tsch_scenario_collision"), true
        );
157   lrWpanHelper.EnablePcap (std::string ("tsch_scenario_collision"),
        panCoordNetDevice, true);
      lrWpanHelper.EnableAsciiAll (stream);
159   if (verbose)
        {
161       //lrWpanHelper.EnableLogComponents ();
          // bool phy, bool csmaca, bool diverse
163       LogComponents(true, true, true);
        }
165
      // ------------------------------
167   Simulator::Run ();
      Simulator::Destroy ();
169   return 0;
}
```

Literatur

[1] "A Performance Analysis of the Network Formation Process in IEEE 802.15.4e TSCH Wireless Sensor/Actuator Networks". In: *2014 IEEE Symposium on Computers and Communications (ISCC)* (2014). DOI: 10.1109/ISCC.2014.6912607, S. 6.

[2] Giuseppe Anastasi Domenico De Guglielmo und Alessio Seghetti. "From IEEE 802.15.4 to IEEE 802.15.4e: A Step Towards the Internet of Things". In: *Advances onto the Internet of Things, Advances in Intelligent Systems and Computing*. DOI: 10.1007/978-3-319-03992-3$_1$0. Springer International Publishing Switzerland, 2014.

[3] Thomas Watteyne Pascal Thubert. *Deterministic IPv6 over IEEE802.15.4e Timeslotted Channel Hopping (6TSCH) BoF*.

[4] IEEE Computer Society. *IEEE Standard for Local and metropolitan area networksâ Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer*. en. 3 Park Avenue New York, NY 10016-5997 USA, 2012.