Collection Tree Protocol

Christan Dietz
Betreuer: Dr. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: dietzc@in.tum.de

KURZFASSUNG

Das Sammeln von Messdaten gehört zu den Hauptaufgaben von drahtlosen Sensornetzwerken. Die von den Sensorknoten erfassten Messwerte sollen möglichst vollständig und über mehrere Hops zu einer zentralen Sammeleinheit weitergeleitet werden. Die in drahtlosen Sensornetzwerken auftretenden Beeinträchtigungen, wie etwa stark asymmetrische Verbindungsqualitäten und häufige Topologieänderungen durch Knotenmobilität oder -ausfälle, stellen hierbei hohe Anforderungen an das Routingprotokoll. Vorliegende Arbeit stellt das Collection Tree Protocol (CTP) vor, ein für dieses Einsatzgebiet entwickeltes Routingprotokoll, welches die oben genannten Probleme adressiert. Ausgehend von der populären CTP-Implementierung des Sensorknotenbetriebssystems TinyOS wird die Funktionsweise von CTP detailliert erläutert.

Schlüsselworte

Collection, CTP, Routing, Sensornetze

1. EINLEITUNG

Drahtlose Sensornetzwerke finden aufgrund sinkender Kosten und Fortschritte in der Forschung immer weitere Verbreitung. Besonders im Bereich der Überwachung bieten sich zahlreiche Anwendungsgebiete: Sie werden beispielsweise genutzt, um die Wasserqualität in einem großem Gelände zu kontrollieren [5] oder um den Stromverbrauch in Bürogebäuden zu überprüfen [11]. Tierforscher erlangen mithilfe drahtloser Sensornetzwerke Aufschluss über das Gruppenverhalten von wilden Tieren [14].

Bei vielen dieser Einsatzgebiete wird eine große Anzahl an Sensorknoten, die in regelmäßigen Abständen Messungen vornehmen, im zu untersuchenden Bereich verteilt. Diese Messwerte müssen anschließend an zentrale Knoten weitergereicht werden, da diese mehr Ressourcen zur Verfügung haben, um die Daten zu verarbeiten und zu speichern. Wegen der großen räumlichen Ausdehnung der Senornetzwerke muss der Übertragungsweg mehrere Zwischenknoten beinhalten, welche die Nachrichten weiterleiten. Aufgabe des Routingprotokolls ist es nun, möglichst kostengünstige und stabile Pfade zu finden und zu verwalten.

Viele Gegebenheiten in einem drahtlosen Sensornetzwerk erschweren das: Da die Sensorknoten aus Gründen des Stromverbrauchs und wegen der Anschaffungskosten nur Funkmodule mit niedriger Leistung besitzen, sind die Verbindungsqualitäten ständigen Schwankungen unterworfen. Knoten können auch ausfallen, oder wechseln zeitweise in einen Schlafmodus. In vielen Sensornetzen sind die Knoten zudem mobil. Dies führt zu häufigen Änderungen der Topologie, auf die das Routingprotokoll zeitnah reagieren muss.

Das in dieser Arbeit erläuterte CTP geht durch mehrere Ansätze auf diese Probleme ein: Es verwendet den modernen Four-Bit Link Estimator um möglichst gute Abschätzungen über Verbindungsqualitäten zu erhalten und so für stabile Routen zu sorgen. Durch eine variable Beaconrate wird versucht, den Control Overhead gering zu halten, um die Luftschnittstelle zu entlasten. Interferenzen werden zusätzlich durch künstliche Verzögerungen beim Weiterleiten von Nachrichten gemieden. CTP besitzt des Weiteren einen Mechnismus, um Routingschleifen zu erkennen und aufzulösen.

CTP ist mittlerweile weit verbreitet und in vielen Betriebssystemen für Sensorknoten verfügbar. Dazu zählen unter anderem Contiki OS [3], Mantis OS [1] und TinyOS [6]. Die verschiedenen Implementierungen unterscheiden sich aber in Details. Deshalb behandelt diese Arbeit speziell die CTP-Implementierung von TinyOS, da es sich um die bekannteste handelt. Beschrieben wird das Verhalten von CTP in TinyOS 2.1.1, die aktuellste Version zum Zeitpunkt der Anfertigung dieser Arbeit.

Die Arbeit gliedert sich folgendermaßen: Zuerst wird in Kapitel 2 vorgestellt, welche Netzwerktopologie CTP bildet und welche Nachrichten dabei ausgetauscht werden müssen, um diese zu erreichen. Kapitel 3 beschreibt, auf welche Art und Weise CTP die Kosten von Verbindungen schätzt und Kapitel 4 behandelt die Routenauswahl. Das Senden und Weiterleiten von Datenpaketen wird in Kapitel 5 erläutert. Das darauf folgende Kapitel 6 erklärt, wie CTP auf Topologieänderungen reagiert. Kapitel 7 fasst zwei Leistungsbewertungen von CTP zusammen. Ein Fazit bildet in Kapitel 8 den Schluss.

2. METRIK/BEACONS

CTP ist ein proaktives Distance Vector Routing Protokoll. Jeder Knoten wählt aus seinen Nachbarn einen Elternknoten so aus, dass seine Kosten zum Wurzelknoten möglichst gering sind. Die Kosten eines Knotens resultieren aus den Kosten des Elternknotens plus denen der Verbindung zwischen den beiden [10]. Alle Wurzelknoten haben die Kosten 0. Dadurch baut CTP eine baumartige Topologie auf. Es können auch mehrere Wurzelknoten existieren, wodurch sich ein Wald von Bäumen ergibt. Datenpakete werden entlang dieses Kostengefälles zu den Wurzelknoten weitergeleitet. Jeder Knoten nimmt Pakete von Kindknoten entgegen und schickt sie an seinen Elternknoten [4]. Wurzelknoten selbst leiten keine Nachrichten im Sensornetzwerk weiter, sondern

doi: 10.2313/NET-2011-07-1_08

fungieren als reine Datensenken. Sie besitzen auch keine Elternknoten. Es gibt in CTP keine Möglichkeit, bestimmte Datensenken direkt zu addressieren. Die Daten werden zu dem, bezogen auf den Kostenabstand, nächsten Wurzelknoten geleitet. Es handelt sich in diesem Sinne um ein Anycast-Protokoll. Unterschiedliche Instanzen des gleichen Datenpaketes können aber auch mehrere Wurzelknoten erreichen, falls es zu Paketduplikaten kommt. CTP garantiert hier weder, dass alle Pakete an den Datensenken ankommen, noch, dass sie nur an jeweils einer Datensenke ankommen. Obwohl CTP diese Verlässlichkeit nicht bietet, wird versucht, möglichst viele Datenpakete auszuliefern und Paketduplikate zu unterdrücken (vgl. Abschnitt 5).
Als Kostenmetrik verwendet die TinyOS-Implementierung von CTP Expected Transmissions (ETX), also die Anzahl der Sendeversuche, die im Mittel durchgeführt werden müssen, damit eine Nachricht erfolgreich übertragen wird.

2.1 Beacons

Durch regelmäßige Beacons informieren Knoten Nachbarn über ihren Elternknoten und ihre Kosten. Solche Routing Frames, wie in Abbildung 1 gezeigt, enthalten den gewählten Next Hop (parent) und die Kosten bis zum Wurzelknoten (ETX).

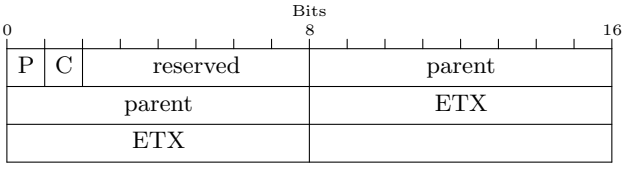


Abbildung 1: CTP Routing Frame

Nachbarknoten, die solche Beacons empfangen, fügen das Tupel (Knotenadresse, parent, ETX) in ihre Routingtabelle ein oder aktualisieren die Werte parent und ETX, falls es für diese Knotenadresse schon einen Eintrag in der Tabelle gibt. Knoten, die noch keine gültige Route besitzen, setzen das „parent"-Feld in ihren Beacons auf INVALID_ADDR (0xffff).
Das Pull-Bit (P) wird von Knoten gesetzt, welche keinen Elternknoten besitzen, also neu zum Netzwerk hinzukommen oder kurzzeitig von den anderen Sensorknoten isoliert waren. Sie fordern damit Nachbarknoten, die den Beacon mit dem gesetzten Pull-Bit empfangen, auf, ihre Beaconrate zu erhöhen, damit der isolierte Knoten schnell in das Netzwerk eingegliedert werden kann.
Das Congestion-Bit (C) bot die Möglichkeit, Nachbarknoten über eine Überlastung zu informieren. Es wurde von Knoten gesetzt, deren Sendepuffer zu viele Pakete enthielt. Knoten mit gesetztem C-Bit wurden nicht als Elternknoten gewählt. Dies hatte allerdings, wie auf der TinyOS-Mailingliste diskutiert, insgesamt negative Folgen auf die Leistung des Routingprotokolls, weshalb das C-Bit in der aktuellen TinyOS-Version 2.1.1 ignoriert wird.

2.2 Trickle

Im Gegensatz zu anderen Collection-Protokollen wie MultiHopLQI [2] versendet CTP Beacons nicht in gleichmäßigen Zeitabständen, sondern passt das Beaconintervall dynamisch an die Gegebenheiten im Netzwerk an [10]. Um die Zeitpunkte für Routing Frame Broadcasts zu bestimmen, verwendet CTP eine Abwandlung des Trickle-Algorithmus [13]. Der Zeitabstand zwischen den Beacons schwankt zwischen τ und $2 \cdot \tau$. Die genaue Intervalldauer wird jedes Mal zufällig gewählt, um eine ungewollte Synchronisation der Sensorknoten zu vermeiden. τ wird nach jedem Beacon verdoppelt, bis es den Wert τ_h erreicht, wodurch die Zeit zwischen den Beacons exponentiell ansteigt und die Beaconrate abnimmt. In TinyOS bewegt sich das Intervall τ zwischen $\tau_l = 64ms$ und $\tau_h = 256s$. Abhängig vom eingesetzten Sensornetzwerk muss darauf geachtet werden, dass der Minimalwert τ_l nicht zu niedrig gewählt ist. Bei dichter Knotenverteilung kann ein zu kleiner Wert von τ_l viele Kollisionen hervorrufen, was die Leistung des Netzwerkes negativ beeinflusst. Richtig eingestellt hat das dynamische Beaconintervall den Vorteil, dass wenig Control Overhead erzeugt wird, nachdem sich das Netzwerk stabilisiert hat. Um dennoch schnell auf Topologieänderungen reagieren zu können, wird τ bei bestimmten Ereignissen auf den ursprünglichen Wert τ_l zurückgesetzt.
Dies geschieht, wenn ein Routing- oder Data-Frame empfangen wird, bei dem das Pull-Bit gesetzt ist. Das hilft, die Routingtabelle des anfragenden Knoten zeitnah zu füllen, damit dieser einen Next-Hop wählen kann. Solange Knoten keinen gültigen Elter besitzen, setzen sie nicht nur das Pull-Bit in ihren Paketen, sondern halten τ auch konstant auf dem Minimalwert τ_l.
Knoten setzen das Beaconintervall auch zurück, wenn sie Inkonsistenzen oder Änderungen in der Netzwerktopologie feststellen, um das Netzwerk schnell wieder in einen stabilen Zustand zurückzuführen. CTP reagiert auf diese Weise, wenn Schleifen erkannt werden oder die Kosten eines Knoten drastisch sinken.

3. LINK ESTIMATION

Der Link Estimator ist in CTP dafür zuständig, die Kosten von Verbindungen zwischen Knoten zu schätzen. Diese Kosten werden auch *1-hop*-ETX, ETX_{1hop} [4] oder single-hop ETX [6] genannt. Der ETX-Wert einer Verbindung, bei der keine Pakete verloren gehen ist 1. Für eine Verbindung mit 50% Erfolgswahrscheinlichkeit auf Hin- und Rückweg gilt $\text{ETX}_{1hop} = \frac{1}{0.5 \cdot 0.5} = 4$. CTP verwendet sowohl Beacons (ETX_b) als auch Unicastpakete (ETX_d), um diese Kosten zu schätzen. Beide Schätzungen werden nicht direkt als Wert von ETX_{1hop} übernommen. Damit kurzzeitige Schwankungen in der Verbindungsqualität nicht zu drastischen Änderungen der Kosten führen, wird der Prozess geglättet, indem der alte ETX-Wert mit einfließt:

$$\text{ETX}_i = \alpha \cdot \text{ETX}_{i-1} + (1 - \alpha) \cdot \text{ETX}_x \qquad (1)$$

ETX_i steht hier für den *1-hop*-ETX-Wert nach der i-ten Aktualisierung und ETX_x für Kostenschätzungen entweder durch Beacons (ETX_b) oder durch Datenpakete (ETX_d). Der Gewichtungsfaktor α hat in TinyOS den Wert 0,9.
Beacon-basierte und Datenpaket-basierte ETX-Schätzungen beeinflussen die *1-hop*-ETX durch Gl. 1 auf gleiche Weise. Allerdings aktualisieren sie ETX_{1hop} unterschiedlich oft. ETX_b wird nach Ankunft einer gewissen Anzahl an Beacons und ETX_d nach dem Senden von Datenpaketen erneuert. In einem stabilen Netzwerk mit niedriger Beaconrate überwiegt der Datenverkehr und ETX_d hat dann mehr Einfluss auf die Kostenschätzung als ETX_b [6]. Untersuchungen

doi: 10.2313/NET-2011-07-1_08

haben ergeben, dass die Schätzung der Verbindungskosten durch Unicastpakete eine deutliche Verbesserung gegenüber der reinen Beacon-basierten Schätzung darstellt. Sowohl im Hinblick auf Paketverlust [15] als auch auf die durchschnittliche Anzahl der Hops, die ein Paket auf dem Weg zum Wurzelknoten passiert [7].
Der Link Estimator verwaltet eine Tabelle mit bekannten Nachbarknoten und den *1-hop*-ETX-Werten der dazugehörigen Verbindungen.

3.1 Linkestimation durch Beacons

Sensorknoten berechnen ihre Eingangsqualitäten durch Beacons, die sie von Nachbarknoten empfangen und Beacons, die verloren gehen. Die Eingangsqualität eines Knotens K_1 bezüglich des Knotens K_2 ist gleichzeitig die Ausgangsqualität von K_2 bezüglich K_1 [8]. Die Eingangsqualität wird durch das Link Estimation Exchange Protocol (LEEP) [8] benachbarten Knoten mitgeteilt, sodass diese ihre Ausgangsqualität kennen. Dazu werden an CTP Routing Frames ein LEEP-Header und ein LEEP-Footer angefügt, wie Abbildung 2 zeigt. Der CTP Beacon wird also immer als Payload in einem LEEP Frame übertragen.

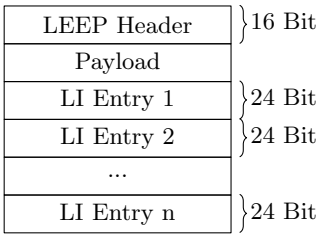


Abbildung 2: LEEP Frame

Der LEEP Header (Abbildung 3) enthält neben einer Sequenznummer (seqno) die Anzahl (nentry) der angefügten Link Information Entries (Abbildung 4). Der Footer besteht aus 0 bis n LI Entries, wobei n durch die maximale Paketgröße auf Layer 2 limitiert ist: Der gesamte LEEP Frame darf nicht größer sein als die maximale Payloadlänge von Verbindungsschichtpaketen [8]. In jedem dieser LI Entries wird die Adresse eines Nachbarknoten (node id) und die Eingangsqualität bezüglich dieses Nachbarn (link quality) übertragen. Knoten, die einen solchen LEEP Frame empfangen, lesen ihre Ausgangsqualität aus dem entsprechenden LI Entry mit ihrer Knotenadresse aus.

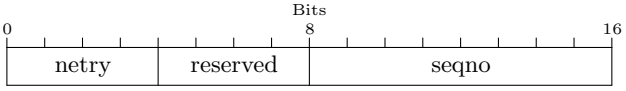


Abbildung 3: LEEP Header

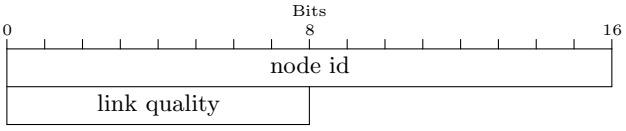


Abbildung 4: LEEP Link Information Entry

Wird ein Beacon von einem Knoten empfangen, der sich noch nicht in der LE-Tabelle befindet, wird ein neuer Eintrag für diesen angelegt. Dabei wird das „mature“-Flag auf 0 gesetzt, was bedeutet, dass es noch keine zuverlässige Kostenschätzung für diesen Knoten gibt und er nicht als Elternknoten verwendet werden darf. Folgende Beacons von diesem Knoten werden gezählt und anhand der Sequenznummer im LEEP-Header wird die Anzahl verlorener Beacons festgestellt. Nach einem Fenster von BLQ_PKT_WINDOW (5) Beacons wird Q_{in} bestimmt durch das Verhälnis der empfangenen Beacons b_{rcv} und der gesamten Beacons, die dieser Nachbarknoten versendet hat b_{sent}:

$$Q_{in} = \frac{b_{rcv}}{b_{sent}} \tag{2}$$

Liegt diese erste Schätzung der Eingangsqualität vor, wird dieser Knoten in der LE-Tabelle als „mature“ deklariert und so zur Verwendung als Next-Hop freigegeben. Weiterhin wird Q_{in} alle BLQ_PKT_WINDOW Routing Frames aktualisiert. Allerdings wird von nun an der alte Wert von Q_{in} gewichtet durch α mit einbezogen:

$$Q_{in,i} = \alpha \cdot Q_{in,i-1} + (1-\alpha) \cdot \frac{b_{rcv}}{b_{sent}} \tag{3}$$

Der LE-Tabellen-Eintrag wird zurückgesetzt, falls die Differenz der Sequenznummern zweier hintereinander empfangenen LEEP Frames größer ist als MAX_PKT_GAP (10). Dieser Nachbar verliert seinen „mature“-Status und die Verbindungsqualität zu ihm muss neu geschätzt werden.
Ein- und Ausgangsqualität bezüglich eines Knotens werden schließlich verwendet, um den Beacon-gestützten ETX-Wert zu bestimmen:

$$\mathrm{ETX}_b = \frac{1}{Q_{in} \cdot Q_{out}} \tag{4}$$

3.2 Linkestimation auf Datenebene

Sobald eine gültige Route durch einen Elternknoten verfügbar ist und Unicastpakete zu diesem Next-Hop gesendet werden, wird auch die Datenebene in die Linkschätzung mit einbezogen:

$$\mathrm{ETX}_d = \frac{d_{sent}}{d_{acked}} \tag{5}$$

Das Verhältnis aus gesendeten Unicastpaketen zu einem Knoten d_{sent} und den erfolgreich übertragenen d_{acked} ist ein direktes Maß für den ETX-Wert dieser Verbindung [10]. ETX_d wird aktualisiert, nachdem DLQ_PKT_WINDOW (3) Pakete jeweils durch ein entsprechendes ACK quittiert wurden oder durch Timeouts als verloren gelten. Sollten innerhalb eines solchen Paketfensters keine ACKs empfangen werden ($d_{acked} = 0$), so wird $\mathrm{ETX}_d = d_{sent}$ gesetzt [7]. CTP versendet ACKs nicht selbst, sondern erwartet, dass Schicht 2 diese Funktionalität anbietet. Es muss daher eine Sicherungsschicht verwendet werden, welche synchrone ACKs für Unicast-Pakete unterstützt [6].

3.3 Four-Bit Link Estimator

Seit September 2007 (Revision 3604) wird in TinyOS der von Fonseca und Gnawali in [7] vorgeschlagene Four-Bit Link Estimator verwendet, wodurch sich einige Abwandlungen von dem oben beschriebenen, alten Verfahren zur Verbindungskostenschätzung ergeben. Der Four-Bit Link Estimator nutzt Informationen aus weitereren Schichten, um *1-hop*-ETX-Werte zu bestimmen. Diese Informationen, namentlich White-, ACK-, Compare- und Pin-Bit, werden von der physikalischen Schicht (White-Bit), der Verbindungsschicht

doi: 10.2313/NET-2011-07-1_08

(ACK-Bit) und der Netzwerkschicht (Compare- und Pin-Bit) bereitgestellt.
Das White-Bit ist ein Hinweis auf eine gute Verbindungsqualität. Es wird dann gesetzt, wenn der Kanal laut Schicht 1 bei Empfang eines Paketes eine gute Qualität aufweist. Viele der in drahtlosen Sensornetzwerken eingesetzten Funkmodule wie das CC1000 [16] oder CC2420 [17] unterstützen diese Funktionalität.
Das ACK-Bit wird bei Paketen gesetzt, die erfolgreich durch ein ACK quittiert wurden. Es ermöglicht eine genaue Schätzung der bidirektionalen Verbindungsqualität. TinyOS implementiert das ACK-Bit durch die in Abschnitt 3.2 beschriebene Funktionalität.
Mit dem Pin-Bit kann die Netzwerkschicht dem Link Estimator mitteilen, dass ein Eintrag nicht aus der LE-Tabelle entfernt werden darf. Bei TinyOS wird das Pin-Bit bei dem Knoten gesetzt, der derzeit als Elternknoten verwendet wird und aus diesem Grund nicht ersetzt werden soll. Es wird auch bei LE-Tabellen-Einträgen gesetzt, deren zugehörige Nachbarknoten Wurzelknoten mit ETX-Wert 0 sind [4], da direkte Verbindungen zu Wurzelknoten denen über mehrere Hops vorgezogen werden.
Das Compare-Bit ermöglicht es dem Link Estimator, mit Hilfe der Netzwerkschicht festzustellen, ob ein Nachbarknoten vorteilhaft erscheint. Es wird gesetzt, falls die Kosten dieses Knotens geringer sind, als die von mindestens einem Eintrag der Routingtabelle. Compare- und White-Bit werden bei der Ersetzungsstrategie von Einträgen in der LE-Tabelle verwendet.
Beim Berechnen der Beacon-basierten Verbindungskosten, ETX_b, verzichtet der Four Bit Link Estimator auf die Ausgangqualität. Die Eingangqualitäten werden zwar durch das Link Estimation Exchange Protocol (siehe Abschnitt 3.1) an Nachbarknoten verteilt, um mit der CTP-Spezifikation [6] kompatibel zu bleiben. Sie werden aber von empfangenden Knoten ignoriert. Die Eingangsqualität wird lediglich als erste, grobe Schätzung der Verbindungsqualität verwendet, welche später durch Unicastpakete (Ack-Bit), wie in Abschnitt 3.2 beschrieben, genauer geschätzt werden kann.

3.4 Verwalten der LE-Einträge

Speicher ist auf Sensorknoten eine stark limitierte Ressource, wodurch die Link Estimation-Tabelle nur eine begrenzte Anzahl an Einträgen verwalten kann. Gleichzeitig ist es für den Link Estimator wichtig, Informationen über Nachbarknoten zu speichern, da die Verbindungsqualitäten zu ihnen nur über einen längeren Zeitraum genau geschätzt werden können. Es ist also von großer Bedeutung, nur die besten und vielversprechendsten Nachbarn in die LE-Tabelle mit aufzunehmen. Eine unvorteilhafte Auswahl an Nachbarknoten kann fatale Auswirkungen auf die resultierende Netzwerktopologie haben [12]. Aus diesem Grund verwendet CTP relativ viel Energie auf die Ersetzungsstrategie.
Falls noch Platz in der LE-Tabelle ist, kann ein neu entdeckter Nachbarknoten einfach eingefügt werden.
Ist die Tabelle voll, versucht CTP den schlechtesten Eintrag mit „mature“-Status zu finden, dessen *1-hop*-ETX-Wert oberhalb des Schwellwertes EVICT_ETX_THRESHOLD (TinyOS: 6,5) liegt. Gelingt es, einen solchen Eintrag zu finden, so wird er durch den Neuen ersetzt. Einträge, deren Pin-Bit gesetzt ist (der derzeitige Elternknoten und Wurzelknoten), werden hierbei nicht beachtet, da sie nicht aus der Tabelle entfernt werden dürfen.
Scheitert auch dieser Versuch, kann der Knoten nicht in die Tabelle aufgenommen werden, es sei denn, der Four-Bit Link Estimator wird verwendet. In diesem Fall überprüft CTP, ob das White- und das Compare-Bit bei dem Beaconpaket, durch das der neue Knoten sich meldet, gesetzt sind. Sind beide Bits gesetzt, wird ein zufälliger, unpinned, nicht-„mature“ Eintrag aus der LE-Tabelle entfernt und durch den neuen Knoten ersetzt, da dieser laut Netzwerkschicht (Compare-Bit) und physikalischer Schicht (White-Bit) sehr gute Kosten und Verbindungsqualitäten aufweist. Gibt es keinen solchen Eintrag, wird der Knoten nicht in die LE-Tabelle eingefügt.

4. ROUTING

In regelmäßigen Zeitabständen (8 Sekunden in TinyOS) wählen Kindknoten in CTP ihren Vaterknoten neu. Die Auswahlprozedur wird auch immer dann ausgeführt, bevor ein Beacon gesendet wird oder wenn der derzeitige Elternknoten nicht mehr erreichbar ist.
Aus der Routingtabelle wird der Knoten mit den geringsten Kosten bestimmt. Die Summe aus Knoten-ETX und *1-hop*-ETX zu diesem Knoten ist hierbei ausschlaggebend. In Betracht gezogen werden hierbei nur Nachbarknoten, die einen gültigen Elternknoten besitzen. Des Weiteren muss der zugehörige Eintrag in der LE-Tabelle den Status „mature“ besitzen und die Verbindungsqualität zu diesem potentiellen Next-Hop muss sich oberhalb eines gewissen Schwellwertes befinden: $ETX_{1hop} < 5$. Es werden auch keine direkten Kindknoten als Elternknoten zugelassen, da sich sonst Schleifen der Länge 2 bilden würden [4].
Wird ein Knoten gefunden, auf den obige Eigenschaften zutreffen, entscheidet CTP, ob es einen Wechsel zu diesem neuen Knoten gibt, oder ob der alte Next-Hop beibehalten wird. Damit nur geringfügig schwankende Verbindungsqualitäten nicht zu häufigen Topologieänderungen führen, wird nur zu einem neuen Elternknoten gewechselt, wenn der Pfad über diesen „signifikant“ besser ist. „Signifikant“ bedeutet nach [10], einen um 1,5 Übertragungen niedrigereren ETX-Wert zu besitzen.

5. DATENPAKETE

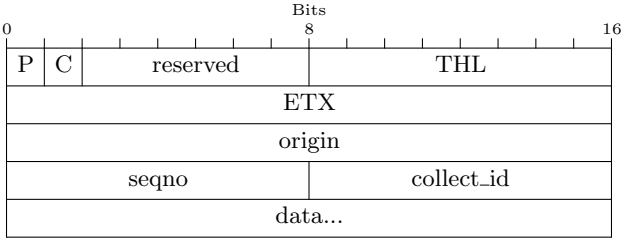


Abbildung 5: CTP Data Frame

CTP verwendet für Datenpakete Header wie in Abbildung 5 gezeigt. Der Header enthält wieder die aus dem Routing Frame (Abbildung 1) bekannten Pull- und Congestion-Bits. Ihre Funktion ist die gleiche wie in Abschnitt 2.1 beschrieben. Das Time-Has-Lived-Feld (THL) ist eine Art invertiertes TTL-Feld. Es startet mit dem Wert 0 und wird von jedem Hop inkrementiert. „origin“ enthält die Adresse des Ursprungsknotens und „seqno“ ist die Sequenznummer für dieses Paket. „collect_id“ kann von höheren Schichten verwendet werden, um dem Datenpaket einen bestimmten Typ

doi: 10.2313/NET-2011-07-1_08

zuzuweisen. Die Felder „origin“, „seqno“ und „collect_id“ werden von dem Sensorknoten gesetzt, welcher das Paket in Umlauf bringt, und von weiterleitenden Knoten nicht verändert [6]. In „ETX“ werden die Kosten des sendenden oder weiterleitenden Knotens übertragen. Diese Information wird benötigt, um Routingschleifen aufzulösen (siehe Abschnitt 6).

5.1 Sendepuffer

Jeder Sensorknoten verwaltet einen Sendepuffer, in welchem zu sendende Nachrichten nach FIFO-Strategie zwischengespeichert werden. Dieser Puffer hat in TinyOS eine Kapazität von 12 Einträgen für weiterzuleitete Pakete. Er besitzt zusätzlich Speicherplatz für jeweils ein Paket pro lokalem Client, der via CTP Daten versenden möchte. Pakete, die gesendet oder weitergeleitet werden sollen, werden an das Ende der Sendepuffers angefügt. Ist kein Speicher für weitere Pakete verfügbar, so wird das Paket verworfen.
Den Kopf dieses FIFO-Puffers bildet das Paket, das gerade versendet werden soll. Das Senden eines Paketes erfolgt synchron. Das heißt, es wird kein weiterer Sendevorgang ausgeführt, bis entweder ein ACK empfangen wird, oder ein Timeout eintritt. Wird das Paket durch ein entsprechendes ACK quittiert, so gilt es als erfolgreich übertragen und wird aus dem Puffer entfernt. Nach einer Zeitspanne t_{ack} wird der Sendevorgang für das nächste Paket im Sendepuffer eingeleitet. Konnte allerdings das ACK oder das Paket nicht erfolgreich übertragen werden, wird nach einem Timeout versucht, das Paket bis zu 30 weitere Male zu versenden. Zwischen den Sendeversuchen bleibt wieder eine Zeitspanne t_{nack}.
Die Verzögerungen t_{ack} und t_{nack} sind wichtig, um Interferenzen mit eigenen, früheren Paketen, die von Nachbarknoten weitergleleitet werden, zu vermeiden. Abbildung 6 veranschaulicht dies. Hier möchte Knoten K_1 zwei Pakete P_1 und P_2 an Knoten K_2 übertragen, der diese an K_3 weiterleitet. Die Kreise symbolisieren die jeweilige Funkreichweite von K_1 und K_3. Nachdem K_1 das erste Paket verschickt hat, muss er mit dem Sendevorgang von Paket P_2 mindestens solange warten, bis K_3 P_1 versendet hat, da sonst, falls K_1 und K_3 gleichzeitig senden, bei K_2 Interferenzen auftreten und K_2 P_2 nicht empfangen kann.

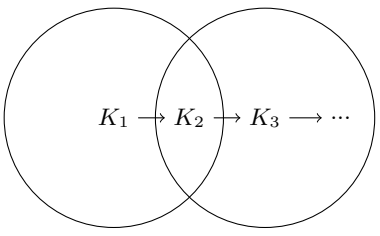


Abbildung 6: Mögliche Interferenz bei K_2

5.2 Umgang mit Paketduplikaten

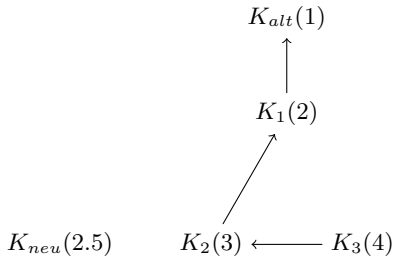
Durch stark asymmetrische Links oder kurzzeitige Schwankungen in der Linkqualität kann der Fall auftreten, dass ein Datenpaket beim Empfangsknoten ankommt, dieser das Paket durch ein ACK quittiert, das ACK aber nicht vom Sendeknoten empfangen wird. Da der Sendeknoten keine Quittierung erhält, muss er davon ausgehen, dass auch sein Paket verloren gegangen ist, weshalb er das Datenpaket erneut sendet.
Solche Paketduplikate stellen ein Problem für das drahtlose Sensornetz dar, da sie im ungünstigsten Fall bei jedem Hop verdoppelt werden und ihre Zahl somit exponentiell mit der Anzahl weiterleitender Knoten wächst [10]. Aus diesem Grund versucht CTP, duplizierte Pakete zu erkennen und ihre Weiterleitung zu unterdrücken.
Zwei Pakete gelten als Duplikate, wenn sie in den Feldern „origin“, „seqno“, „collect_id“ und „THL“ übereinstimmen. Die Überprüfung des Time-Has-Lived-Feldes ist notwendig, da es aufgrund von Schleifen vorkommen kann, dass ein Paket einen Knoten mehrmals passiert. Damit in einem solchen Fall das Paket nicht fälschlicherweise als Duplikat identifiziert und verworfen wird, muss auch der THL-Wert mit einbezogen werden. Denn falls die Schleifenlänge nicht ein Vielfaches von 256 beträgt, unterscheidet sich der THL-Wert nach jedem Schleifendurchgang.[10]
Bevor ein Paket zum Weiterleiten in den Sendepuffer eingefügt wird, überprüft CTP zunächst, ob das Paket bereits darin vorliegt. Wird eine Übereinstimmung in den obigen Header-Feldern gefunden, so wird das Paket verworfen.
Weil der Sendepuffer im optimalen Fall nicht sehr voll ist und es sein kann, dass das gesuchte Paket schon erfolgreich übertragen wurde, verwaltet CTP zusätzlichen Speicher für versendete Pakete, den sogenannten Transmit Cache [10]. Diese FIFO-Schlange enthält die für eine Paketinstanz charakteristischen Felder („origin“, „seqno“, „collect_id“ und „THL“) der n zuletzt weitergeleiteten Pakete. Dadurch kann eine Übereinstimmung mit früheren Paketen geprüft werden. TinyOS speichert auf diese Weise $n = 4$ Pakete. In [9] wird ein Verfahren beschrieben, bei dem nicht die charakteristischen Tupel der zuletzt weitergeleiteten Pakete, sondern ein Hashwert gespeichert wird. TinyOS 2.1.1 verwendet aber direkt die Tupel, wie in [10] erläutert.
Nur wenn weder im Transmit Cache noch im Sendepuffer Paketduplikate gefunden werden, kann das Paket zum Versenden in den Sendepuffer aufgenommen werden.

6. TOPOLOGIEÄNDERUNGEN

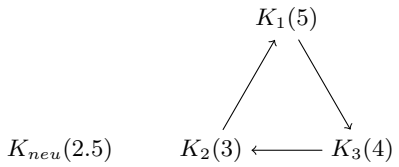
Sensornetzwerke unterliegen ständigen Topologieänderungen: Die Knoten haben oft nur sehr wenig Batteriekapazität zur Verfügung, wodurch sie immer wieder ausfallen und ersetzt werden müssen. Auch die Verbindungsqualitäten zwischen den Knoten können durch Knotenbewegung oder abschirmende Objekte schwanken.
Der von CTP verwendete Link Estimator kann durch seine kleinen Aktualisierungsfenster schnell auf solche Veränderungen reagieren, sodass die Netzwerktopologie die physikalischen Gegebenheiten zeitnah widerspiegelt.[10]
Verbessert sich der ETX-Wert eines Knotens um mehr als 1,5 (2,0 in TinyOS) Übertragungen, stellt er möglicherweiße einen günstigereren Next-Hop für seine Nachbarn dar. Aus diesem Grund verringert er das Beaconintervall auf τ_l, um seine Nachbarknoten innerhalb kurzer Zeit über die neuen Kosten zu informieren.[10]
Distance Vector Routing Protokolle wie CTP sind sehr anfällig gegenüber Schleifenbildung. Routingschleifen können entstehen, wenn Knoten aufgrund von Topologieänderungen, die noch nicht von allen Nachbarknoten wahrgenommen wurden, Pfade wählen, welche einen eigenen Kindknoten enthalten.
Die Abbildungen 7a und 7b veranschaulichen dies. In diesem Beispiel fällt die Verbindung von Knoten K_1 zu K_{alt} aus. K_1 wählt nun K_3 als seinen Elternknoten und es entsteht eine

doi: 10.2313/NET-2011-07-1_08

Schleife. Da K_2 noch veraltete Kosteninformationen über K_1 besitzt, die unter denen von K_{neu} liegen, wählt er keinen neuen Elternknoten. Die Zahlen in Klammern stehen für beispielhafte Knotenkosten.



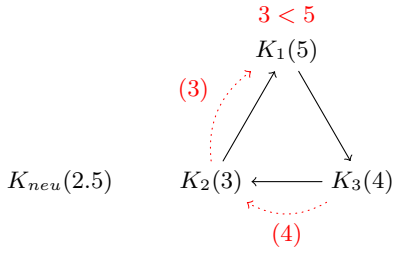
(a) Die Verbindung $K_1 \rightarrow K_{alt}$ fällt aus



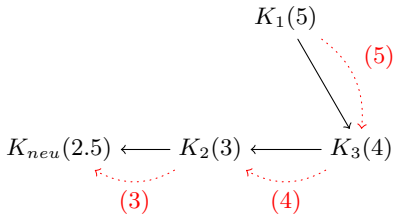
(b) Eine Schleife hat sich gebildet

Abbildung 7: Entstehung von Routingschleifen

CTP kann solche Schleifen erkennen und aktiv auflösen. Dazu wird im Header eines jeden Datenpaketes (siehe Abbildung 5) der ETX-Wert des weiterleitenden Knotens übertragen. Empfängt ein Sensorknoten ein Paket mit einem ETX-Wert, der geringer ist als sein eigener, deutet das auf eine Schleife hin. Denn in einer konsistenten Topologie muss der ETX-Wert von Hop zu Hop in Richtung Wurzelknoten monoton abnehmen.



(a) K_1 stellt eine Schleife fest



(b) K_2 wählt einen neuen Vaterknoten

Abbildung 8: Auflösen einer Schleife

Der Knoten, der eine solche Inkonsistenz feststellt, setzt sein Beaconintervall auf den Minimalwert τ_l zurück, um benachbarte Knoten schnell zu informieren. Um den Routing Frames Vorrang zu gewähren, werden eine kurze Zeitspanne lang keine Datenpakete versendet.
CTP verwirft Datenpakete nicht, bis die Schleife beseitigt ist, sondern nutzt sie, um die Topologie aktiv zu reparieren: Die Datenpakete zirkulieren so lange in der Schleife, bis ein Knoten einen Next-Hop wählt, mit dem ein Weg aus der Schleife gefunden wird. Sie veranlassen dabei die Hops auf ihrem Pfad, die Beaconrate zu erhöhen, um die Reparatur zu beschleunigen.[10]
Abbildung 8 zeigt ein Beispiel dieses Vorgangs. Die Knoten K_1 bis K_3 bilden eine Schleife. Das von K_3 in Umlauf gebrachte Datenpaket erreicht schließlich K_1. Dieser stellt eine mögliche Schleife fest, da das empfangene Paket einen geringeren ETX-Wert aufweist als er. Er sendet mit erhöhter Frequenz Beacons und informiert dadurch unter anderem K_2 über seine gestiegenen Kosten. Das Paket zirkuliert währenddessen weiter in der Schleife. Bei der nächsten Ankunft bei K_2 besitzt dieser Knoten aber aktualisierte Informationen über die Kosten von K_1 und hat deshalb K_{neu} als seinen Elternknoten gewählt. Das Datenpaket verlässt über die Verbindung $K_2 \rightarrow K_{neu}$ die Schleife.

7. LEISTUNGSBEWERTUNG

Gnawali, Fonseca et al. dokumentieren in [10] ihre Experimente mit CTP. Dabei wurden 12 Testbeds unterschiedlicher Größe von $6m$ x $4m$ bis $50m$ x $25m$ x $10m$ untersucht. Die Testbeds enthielten jeweils 35 bis 310 Sensorknoten vom Typ Tmote, TelosB, Mica2dot, MicaZ, Epic-Quanto, eyesIFXv2 oder Blaze. Durch die unterschiedliche Verteilung und Dichte hatten die Knoten einen Grad von 6 bis 305. Die Experimente liefen für mindestens 3 Stunden, in denen die Sensorknoten alle 16 Sekunden Daten an die Wurzel sendeten. In allen Fällen kamen mindestens 90% der auf den Sensorknoten generierten Daten bei den Wurzelknoten an. In mehr als der Hälfte der Testkonfigurationen lag der Wert sogar über 99%. Hierbei ist allerdings anzumerken, dass CTP so konfiguriert war, nach einem Paketverlust die Nachricht bis zu 30 weitere Male erneut zu versenden. Das führt einerseits zu einer hohen Verlässlichkeit, dass Datenpakete die Wurzelknoten erreichen, kann bei hohem Verkehrsaufkommen aber auch die Leistung des Protokolls beeinträchtigen. Viele erneute Sendeversuche führen nicht nur zu einer temporären Überlastung der Luftschnittstelle, sondern erzeugen auch Paketstau auf den Knoten.
Verglichen mit dem ebenfalls sehr bekannten Protokoll MultiHopLQI erzielte CTP bessere Ergebnisse. MultiHopLQI wies einen niedrigereren Bruchteil an erfolgreich übertragenen Datenpaketen von durchschnittlich 85% auf, welcher zudem größeren Schwankungen unterworfen war. Durch seine konstante Beaconrate benötigte MultiHopLQI 73% mehr Control Overhead als CTP, das die Beaconrate dynamisch anpasst.
Die simulative Leistungsbewertung von Colesanti und Santini in [4] bestätigt die Ergebnisse von Gnawali et al. größtenteils. In einem simulierten Testbed der Größe $250m$ x $250m$ wurden 100 Knoten gleichmäßig verteilt und folgendes Experiment durchgeführt. Alle Knoten wachen in regelmäßigen Intervallen gleichzeitig auf, erzeugen die Baumtopologie und senden Daten zum Wurzelknoten. Anschließen gehen sie wieder in einen Schlafzustand über. Jeder der Knoten nimmt in

doi: 10.2313/NET-2011-07-1_08

der Wachphase aktiv am Erstellen der Netzwerktopologie teil. Ob er aber in einem Zyklus auch Daten versendet, wird durch eine vordefinierte Wahrscheinlichkeit p bestimmt. Die Schlafphase dauert $44s$ und das Zeitfenster, in dem die Knoten aktiv sind, ist unterteilt in $11s$ für das Erstellen der Topologie und $5s$ um die Daten weiterzureichen. Colesanti und Santini werteten auf diese Weise 50 Topologien mit jeweils unterschiedlicher Sendewahrscheinlichkeit p aus. Sie kamen zu dem Ergebnis, dass mit $p = 0{,}5$ mehr als 95% der gesendeten Pakete die Datensenke erreichen. Mit höherer Sendewahrscheinlichkeit $p = 1$ werden allerdings in manchen Fällen nur 80% oder weniger erreicht. Im Durchschnitt kamen aber für alle Werte von p über 95% der gesendeten Datenpakete bei den Wurzelknoten an.[4]

8. FAZIT

In dieser Arbeit wurde das Routingprotokoll CTP beschrieben. Der Schwerpunkt lag dabei darauf, wie es mit den Herausforderungen in drahtlosen Sensornetzwerken umgeht. Denn CTP besitzt eine Vielzahl an Eigenschaften, die auf dieses Einsatzgebiet hin zugeschnitten sind. Dies zeigt, dass Routing in solchen Netzwerken kein einfaches Thema ist.
Dennoch ist es mit CTP möglich, in einem hohen Grad verlässliches Routing zu betreiben, was die weite Verbreitung des Protokolls erklärt. Mit seinen Ideen hat CTP auch die Entwicklung des IPv6 Routingprotokolls RPL wesentlich beeinflusst. RPL adaptiert beispielsweise das in CTP erprobte dynamische Beaconintervall.
Dabei gibt es für CTP und allgemein auf dem Gebiet Routing in drahtlosen Sensornetzwerken durchaus noch deutlichen Forschungs- und Verbesserungsbedarf. So wäre zum Beispiel im Fall von CTP eine stabil funktionierende Methode, überlastete Knoten zu umgehen, wünschenswert.

9. LITERATUR

[1] *Mantis OS*. `http://mantisos.org/documentation/api/html-unstable/ctp_8h-source.html`, Okt. 2007.

[2] *MultiHopLQI Routing Protocol*. `http://www.tinyos.net/tinyos-2.1.0/tos/lib/net/lqi/`, Aug. 2008.

[3] *Contiki 2.5 Release Candidate 1*. `http://www.sics.se/contiki/news/contiki-2.5-release-candidate-1-avaliable.html`, Nov. 2010.

[4] Colesanti, U. und S. Santini: *A Performance Evaluation of the Collection Tree Protocol Based on its Implementation for the Castalia Wireless Sensor Networks Simulator*. Technical Report 681, Department of Computer Science, ETH Zurich, Zurich, Switzerland, Aug. 2010.

[5] Dinh, T. L., W. Hu, P. Sikka, P. Corke, L. Overs und S. Brosnan: *Design and Deployment of a Remote Robust Sensor Network: Experiences from an Outdoor Water Quality Monitoring Network*. In: *32nd IEEE Conference on Local computer Networks, 2007. LCN 2007.*, S. 799 –806, Okt. 2007.

[6] Fonseca, R., O. Gnawali, K. Jamieson, S. Kim, P. Levis und A. Woo: *TEP 123: The Collection Tree Protocol*. `http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html`, Feb. 2007.

[7] Fonseca, R., O. Gnawali, K. Jamieson und P. Levis: *Four Bit Wireless Link Estimation*. In: *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*, Nov. 2007.

[8] Gnawali, O.: *TEP 124: The Link Estimation Exchange Protocol (LEEP)*. `http://www.tinyos.net/tinyos-2.x/doc/html/tep124.html`, Feb. 2007.

[9] Gnawali, O., R. Fonseca, K. Jamieson und P. Levis: *CTP: Robust and Efficient Collection through Control and Data Plane Integration*. Technical Report, The Stanford Information Networks Group (SING), Stanford University, Feb. 2008.

[10] Gnawali, O., R. Fonseca, K. Jamieson, D. Moss und P. Levis: *Collection tree protocol*. In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, S. 1–14, Nov. 2009.

[11] Kazandjieva, M., O. Gnawali, B. Heller, P. Levis und C. Kozyrakis: *Identifying Energy Waste through Dense Power Sensing and Utilization Monitoring*. Technical Report, Computer Science Lab, Stanford University, März 2010.

[12] Langendoen, K., A. Baggio und O. Visser: *Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture*. In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, S. 8 pp., Apr. 2006.

[13] Levis, P., N. Patel, D. Culler und S. Shenker: *Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks*. In: *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, S. 15 – 28. USENIX Association, März 2004.

[14] Romer, K. und F. Mattern: *The design space of wireless sensor networks*. Wireless Communications, IEEE, 11(6):54 – 61, Dez. 2004.

[15] Rothery, S., W. Hu und P. Corke: *An empirical study of data collection protocols for wireless sensor networks*. In: *Proceedings of the workshop on Real-world wireless sensor networks*, REALWSN '08, S. 16–20, 2008.

[16] Texas Instruments: *CC1000 Datasheet*. `http://www.ti.com/lit/gpn/cc1000`, Feb. 2007.

[17] Texas Instruments: *CC2420 Datasheet*. `http://www.ti.com/lit/gpn/cc2420`, März 2007.

doi: 10.2313/NET-2011-07-1_08