Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Implementierung einer protokollübergreifenden FIB für RIOT

Martin Landsmann

Projekt 1

Fakultechnik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

Das *Internet of Things*[1] ist derzeit in aller Munde. Kommunizierende Mikrocontroller, die mit dem Internet interagieren können, finden immer mehr Anwendung im privaten sowie kommerziellen Sektor. Sie werden zum vernetzten steuern von Geräten, als auch, zum überwachen von Messwerten eingesetzt. Durch die Anbindung an das Internet können sie Berichte über Messwerte gezielt an spezifische Hosts die mit dem Internet verbunden sind senden, und gleichzeitig von diesen Hosts gesteuert und überwacht werden. Die Heterogenität der hierbei eingesetzten Mikrocontroller, die meist mit einer leistungsschwachen MCU und wenigen kB Speicher ausgestattet sind, benötigen ein Betriebssystem um Entwicklungsprozesse für Anwendungen zu optimieren. RIOT[2] ist ein stetig weiterentwickeltes Betriebssystem, dass sich die Erfüllung dieser Anforderung zum Ziel gesetzt hat. Eine einmal implementierte Anwendung für RIOT, soll auf verschiedener Hardware benutzt werden können ohne die Funktionalität der Anwendung speziell auf die jeweilige Hardware anpassen zu müssen. Im Kontext des *Internet of Things* ist die Kommunikationsfähigkeit einzelner Geräte miteinander, und mit dem Internet, ein entscheidendes Kriterium für ein erfolgreiches Betriebssystem. Hierfür ist in RIOT ein Netzwerkstack implementiert, der die Kommunikation zwischen Knoten und dem Internet ermöglicht. RIOT wird stetig weiterentwickelt, um die Vielzahl an Anforderungen für seinen konkreten Einsatz zu optimieren. Derzeit wird der RIOT Netzwerkstack umgebaut, um eine saubere Netzwerkschichten-Architektur für die Kommunikation und eine klare Abgrenzung der Aufgabenbereiche der eingesetzten Module im Netzwerkstack bereitzustellen. Dieser Umbau führt unter anderem eine protokollübergreifende *Forwarding Information Base* (FIB) in den Netzwerkstack ein, um zentral durch das Betriebssystem *next-hop* Informationen verwalten zu können.

Diese Ausarbeitung beschreibt, wie eine protokollübergreifende FIB in RIOT umgesetzt werden kann. Dabei werden Probleme der aktuellen RIOT Netzwerkstack Implementierung im Bezug auf das Weiterleiten von Paketen aufgezeigt, mögliche Lösungsansätze diskutiert und die konkrete Umsetzung beschrieben.

2 Motivation und Hintergrund

Eine FIB wird durch Routing-Protokolle oder manuelle Eingaben mit *next-hop* Informationen befüllt. Soll ein Paket zu einer Zieladresse weitergeleitet werden, wird die FIB nach der Adresse des nächsten Knotens auf dem Weg zu jenem Ziel angefragt. Die FIB trennt ein eingesetztes Routing-Protokoll von der reinen Paketweiterleitung, und grenzt damit das Aufgabengebiet eines Routing-Protokolls, Routen zu lernen, klar vom Forwarding der Pakete im Netzwerkstack. Beim Forwarding von Paketen wird eine schnelle Antwortzeit von der FIB gefordert, um die Paketverarbeitung und Weiterleitung nicht zu verlangsamen. Diese Verwaltungsstruktur für *next-hop* Informationen im Netzwerkstack wird von populären Betriebssystemen wie beispielsweise Linux implementiert[3]. Die FIB im Linux kernel[1] gruppiert und verwaltet die *next-hop* Informationen in verschiedenen *hash*-Tabellen und Listen mit Querverweisen, um schnelle Antwortzeiten zu gewährleisten. Der dadurch erreichte Geschwindigkeitszuwachs geht auf Kosten von Speicherbedarf für die einzelnen Einträge der FIB. Dieser Ansatz ist im Kontext des *Internet of Things*, im Bezug auf verfügbare Ressourcen auf den eingesetzten Knoten, zu schwergewichtig.
RIOT bedarf einer FIB, die den Anforderungen leistungsschwacher, mit wenigen kB Speicher ausgestatteter Knoten genügt. Da der Netzwerkstack von RIOT derzeit einen Architekturumbau erfährt, wird in den folgenden Unterabschnitten das Forwarding eines Pakets im aktuellen Netzwerkstack, in Abschnitt 2.1, und im zukünftigen Netzwerkstack, in Abschnitt 2.2, skizziert.

2.1 Forwarding im aktuellen RIOT Netzwerkstack

In RIOT wird derzeit ein Routing-Protokoll direkt in die Weiterleitung von Paketen involviert, und ist damit sehr eng an die RIOT IPv6 Implementierung gekoppelt. Die IPv6 Implementierung wiederum, baut direkt auf einer früheren *IPv6 over Low-power Wireless Personal Area Network* (6LoWPAN)[4] Implementierung auf. Ein Routing-Protokoll registriert während seiner Initialisierung eine Callback-Funktion in der Vermittlungsschicht von RIOT[2]. Dies geschieht durch

[1] `https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/include/net/ip_fib.h`
[2] `https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/ip.c#L58`

den Aufruf der Registrierfunktion `ipv6_iface_set_routing_provider(...)`[3]. Die registrierte Callback-Funktion wird von der RIOT Vermittlungsschicht in `ipv6_send_packet(...)`[4] aufgerufen, wenn ein *next-hop* für die Weiterleitung eines Pakets zu einer Zieladresse benötigt wird. Die registrierte Callback-Funktion erwartet die Ziel IPv6 Adresse als Parameter und gibt den Pointer auf die vom Routing-Protokoll als *next-hop* ausgewählte IPv6 Adresse zurück. Der zurückgegebene *next-hop* ist in der derzeitigen Implementierung des RIOT Netzwerkstacks die reine IPv6 Adresse und an keine spezifische Kommunikationsschnittstelle gebunden. Anschließend wird die *next-hop* Adresse an die *Neighbour Discovery*[5] von RIOT weitergereicht[5], um einen zugehörigen *Neighbour Cache Entry* zu finden, der zur Parametrierung der Paketweiterleitung[6] genutzt wird. Der *Neighbour Cache Entry* enthält die Information über die zu nutzende Kommunikationsschnittstelle für die Weiterleitung des Pakets zum *next-hop*. Daraufhin wird ein MAC-Frame[7] vorbereitet und an den Treiber der spezifisch genutzten Kommunikationsschnittstelle zum Übertragen weitergegeben. In Abb. 2.1 ist die derzeitige, ab der Vermittlungsschicht nach unten hin sehr monolithische Implementierung des RIOT Netzwerkstacks dargestellt.

2.2 Forwarding im zukünftigen RIOT Netzwerkstack

Der aktuelle auf 6LoWPAN aufgebaute RIOT Netzwerkstack ist derzeit im Umbau[8], um Modularität und Flexibilität der einzelnen Schichten auszubauen. Dabei wird insbesondere die Speichernutzung zum erstellen und weiterreichen von Paket-Headern sowie Paketdaten im Netzwerkstack, die Kommunikation zwischen den Schichten, und eine klare Abgrenzung der eingesetzten Module und ihrer Aufgabengebiete überarbeitet. Zusätzlich werden generalisierte Schnittstellen geschaffen, die erlauben unterschiedliche Transceiver flexibel beim RIOT Netzwerkstack anmelden und nutzen zu können.
Hierfür werden zwischen den einzelnen Schichten Funktionsaufrufe durch eine strikte *message passing* Kommunikation ersetzt. Eine zentrale Verwaltungsstelle, die `netreg`[9], erlaubt dafür die Registrierungen von Netzwerkstack-Modulen mit ihren Angebotenen Funktionen sowie Nachrichtentypen an denen die spezifischen Module interessiert sind. Die registrierten Module kommunizieren über einen zentralen Verteiler, der `netapi`[10], der die Nachrichten im Netzwerk-

[3] https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/ip.c#L861
[4] https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/ip.c#L71
[5] https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/icmp.c#L1510
[6] https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/ip.c#L87
[7] https://github.com/RIOT-OS/RIOT/blob/master/sys/net/network_layer/sixlowpan/mac.c#L296
[8] https://github.com/RIOT-OS/RIOT/issues/2278
[9] https://github.com/RIOT-OS/RIOT/blob/master/sys/include/net/ng_netreg.h
[10] https://github.com/RIOT-OS/RIOT/blob/master/sys/include/net/ng_netapi.h

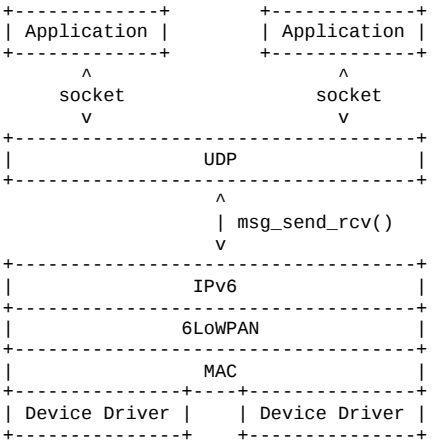

Abbildung 2.1: Aktueller Netzwerkstack

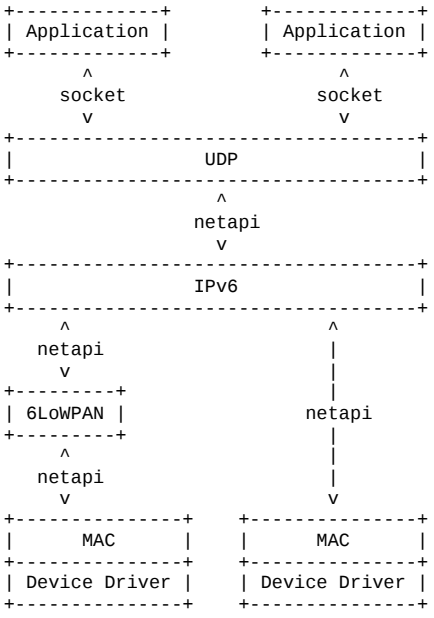

Abbildung 2.2: Zukünftiger Netzwerkstack

stack anhand der in der `netreg` registrierten Funktionen und Nachrichtentypen verschickt. In Abb. 2.2 ist die beschriebene zukünftige Architektur des RIOT Netzwerkstacks dargestellt. Durch das neue Konzept des Netzwerkstacks wird der Prozess des Forwarding von Paketen flexibler. Wird ein UDP[6] Paket über 6LoWPAN an eine Zieladresse geschickt, bereitet der UDP-Thread das UDP Paket vor, und lässt über die `netapi` den IPv6-Thread benachrichtigen, dass ein IP Paket verschickt werden soll. Der IPv6-Thread verarbeitet das Paket und lässt die `netapi` den 6LoWPAN-Thread benachrichtigen. Der fragt die FIB nach einem *next-hop* und adaptiert das IPv6 Paket. Dann wird der Link-Layer-Thread mittels `netapi` benachrichtigt, der die *Neighbor Discovery* für die Link-Layer Adresse zu dem *next-hop* anfragt, und letztendlich das Paket an die Kommunikationsschnittstelle übergibt, der es abschickt. Wie das Beispiel zeigt, sind im neuem Netzwerkstack direkte Funktionsaufrufe zwischen den einzelnen Schichten durch das *message passing* ersetzt. Ebenso wurde die direkte Kommunikation mit Routing-Protokollen aufgelöst und durch die Interaktion mit der FIB ersetzt.

Diese kurze Zusammenfassung des neuen Netzwerkstack-Konzepts basiert auf den Ergebnissen des RIOT Network Stack Taskforce Meeting vom 5-6 Februar 2015 an der FU Berlin.[11]

[11] `https://github.com/RIOT-OS/RIOT/wiki/minutes-network-task-force-feb-2015`

3 Problemstellung

RIOT nutzt in seinem derzeitigen Netzwerkstack ausschließlich IPv6 Adressen. Diese werden durch ein speziellen Datentyp repräsentiert der durchweg in der Implementierung des Netzwerkstacks genutzt wird. Dieser Datentyp ist mit einer festen Größe von 16 Bytes definiert, um eine komplette IPv6 Adresse speichern zu können.
Die Einschränkung auf den IPv6 Datentyp führt dazu, dass

- IPv4[7] Adressen im RIOT Netzwerkstack nur in IPv6[8] Adressen eingebettet[9] genutzt werden können. Damit ist eine Ausnutzung der nativ kleineren Adressgröße zur Reduzierung des Speicherverbrauchs von IPv4 mit 4 Bytes gegenüber IPv6 mit 16 Bytes effektiv nicht möglich.

- proprietäre Adresstypen, die in IP-fremden Routing-Protokollen zum Einsatz kommen, derzeit nicht eingesetzt werden können, da der erwartete Typ immer IPv6 ist und keine Möglichkeit besteht den Typ mit der Adresse anzugeben.

- eine potentielle Kompression von IPv6 Adressen, wie sie in 6LoWPAN durch die *IP header compression* (IPHC)[10] möglich ist, nicht zur Reduktion des für die Adressen genutzten Speichers verwendet wird.

Es ist derzeit nicht möglich neben der *Neighbor Discovery*, mehr als ein Layer 3 Routing-Protokoll gleichzeitig im RIOT-Netzwerkstack zu betreiben. Die RIOT Vermittlungsschicht hat keinen Mechanismus sowie Verwaltungsstrukturen, um mehrere Registrierungen von Routing-Protokoll Callbacks durchzuführen.
Das bedeutet, dass

- ein RIOT Knoten nur Teilnehmer in einem einzigen Netzwerk sein kann. Dies gilt sowohl wenn er potentiell durch mehrere verfügbare Kommunikationsschnittstellen mit verschiedenen Netzwerken und Protokollen interagieren könnte, als auch, wenn die verfügbare Kommunikationsschnittstelle des Knotens Verbindung zu verschiedenen Netzwerken aufnehmen könnte.

- eine potentielle Gateway-Funktionalität, zur Kommunikation zwischen Protokoll-Fremden Netzwerken mittels Protokollumsetzung, beispielsweise *Bluetooth Low Energy* (BLE)[11] zu 6LoWPAN, nicht möglich ist.

4 Anforderungen an eine protokollübergreifende FIB für RIOT

Aus der in Abschnitt 3 vorgestellten Problemstellungen, werden in diesem Kapitel Anforderungen abgeleitet.

Kompatibilität zu Adresskompression Im Kontext von *Internet of Things*- und Sensor-Netzwerken ist es erforderlich, dass die FIB mit Kompression von Ziel und *next-hop* Adresse umgehen kann. Kompression von Adressen wird beispielsweise in 6LoWPAN[12] durch die IPHC umgesetzt. Die IPHC wurde speziell für de Einsatz in IEEE 802.15.4[13] basierten Netzwerken entwickelt, um den Speicherbedarf von Adressen in 6LoWPAN Paket-Headern zu minimieren. Wird beispielsweise IPHC im RIOT Netzwerkstack zur Reduzierung der zu speichernden Adress-Bytes genutzt, muss die FIB die Art der Adress-Kompression speichern und bei Anfrage nach einem *next-hop* zurückliefern können.

Unterstützung IP-fremder Adressen Die FIB muss IP-fremde Adressen unterstützen. Beispielsweise wie im ZigBee[14]-Protokoll mit einer 2 Byte Adresse und einem zusätzlichem Byte für einen *Endpoint-Identifier*, dem nrf24r01+ Transceiver[15] mit einer 5 Byte Adresse für einen Knoten, oder mit 6 Byte für *Bluetooth device*-Adressen.

Feste Tabellengrößen für FIB-Einträge Die Tabellengröße für FIB Einträge muss frei wählbar, jedoch zur Compile-Zeit fest eingestellt sein. Das statische Festlegen des genutzten Speichers für die FIB-Einträge garantiert, dass der damit zur Laufzeit angelegte Speicher exklusiv der FIB zur Verfügung steht. Konkurrierendes allozieren von Speicher durch die FIB für ihre Einträge, wird damit während der Laufzeit vermieden.

Unterstützung mehrerer Kommunikationsschnittstellen Die FIB muss *next-hop*s eindeutig der genutzten Kommunikationsschnittstelle zuordnen, die mit dem Netzwerk zur Weiterleitung in Richtung Zieladresse verbunden ist. Dafür muss in den FIB-Einträgen die ID der zugeordneten Kommunikationsschnittstelle zu dem *next-hop* mit abgelegt sein.

Unterstützung von reaktivem Routing Die Vermittlungsschicht in RIOT muss ausschließlich die FIB nach *next-hop*s zu einer Zieladresse anfragen. Routing-Protokolle hingegen, werden von der Vermittlungsschicht nicht mehr direkt nach *next-hop*s angefragt. Sie nutzen die FIB, um aus den gelernten Routen die *next-hop* Einträge abzulegen und sie damit der Vermittlungsschicht zugänglich zu machen.
Hierbei ist das Verhalten von (a) proaktiven und (b) reaktiven Routing-Protokollen zu differenzieren.

(a) Proaktive Routing-Protokolle lernen Routen zu Zieladressen in der Topologie, noch bevor eine konkrete Kommunikation zwischen Knoten benötigt wird. Aus den gelernten Routen, weisen die proaktiven Routing-Protokolle einem gelerntem Ziel einen eindeutigen *next-hop* zur Weiterleitung von Paketen zu. Wenn die Routen des proaktiven Routing-Protokolls konvergieren, ist potentiell jeder von dem Knoten aus erreichbaren Zieladresse in der Topologie ein eindeutiger *next-hop* zugewiesen. Wird nun ein *next-hop* von der Vermittlungsschicht zu einer bekannten Zieladresse benötigt, wird dieser in den bereits zuvor eingetragenen Einträgen nachgeschlagen, und zurückgegeben wenn er existiert.

(b) Reaktive Routing-Protokolle lernen Routen zu Zieladressen erst, wenn eine konkrete Kommunikation zwischen Knoten in der Topologie erforderlich ist. Wird hier ein *next-hop* von der Vermittlungsschicht zu einer bekannten Zieladresse benötigt, ist dieser potentiell unbekannt, sodass die konkrete Kommunikation nicht unmittelbar stattfinden kann.
Da die FIB Routing-Protokolle vom direkten Zugriff aus der Vermittlungsschicht trennt, muss sie einen Mechanismus implementieren, der mit proaktivem sowie reaktivem Verhalten umgehen kann. Das Verhalten eines proaktiven Routing-Protokolls eignet sich exakt für die Nutzung einer FIB, um die gelernten *next-hop*s in der FIB-Tabelle zu sammeln, und diese für die Vermittlungsschicht verfügbar zu halten. Das reaktive Routing-Protokoll hingegen, muss zum lernen einer Route, und damit zum befüllen der FIB mit einem spezifischem *next-hop* zu einer Zieladresse, aktiv angestoßen werden. Dafür muss ein Mechanismus implementiert werden, der die FIB mit einem reaktiven Routing-Protokoll bekannt macht, und das reaktive Routing-Protokoll durch die FIB in den Weiterleitungsprozess eines Pakets einbindet.

Parallel operierende Routing-Protokolle Die *next-hop* Einträge der FIB müssen eindeutig sein. Diese Einzigartigkeit der FIB-Einträge muss bestand haben, auch wenn mehrere Routing-Protokolle parallel betrieben *next-hop*s zu Zieladressen liefern. Die Einzigartigkeit eines FIB-Eintrags muss durch die Kombination aus Zieladresse und zugehöriger Kommunika-

tionsschnittstelle für den *next-hop* erfolgen. Dies muss auch gewährleistet sein, wenn mehrere Routing-Protokolle sich eine Kommunikationsschnittstelle zur Kommunikation mit Netzwerken Teilen, beispielsweise in einem Szenario, in dem der RIOT Knoten als „einarmiger"-Router arbeitet. Dabei müssen die Routing-Protokolle selbständig vermeiden, Schleifen in der Topologie durch ungünstige *next-hop* Einträge zu erzeugen, beispielsweise indem Adressbereiche logisch, auf die einzelnen Routing-Protokolle getrennt und verteilt werden.
Die FIB muss so implementiert sein, dass Zugriff von parallel operierenden Routing-Protokollen möglich ist. Sind mehrere Routing-Protokolle zur selben Zeit auf RIOT aktiv, kann dies zu einem konkurrierender Zugriff auf die FIB führen, beispielsweise wenn die Routing-Protokolle einen *next-hop* eintragen. Ebenso kann die Anfrage nach einem *next-hop* zu einer Zieladresse durch die Vermittlungsschicht konkurrierend erfolgen. Daher muss die FIB konkurrierenden Zugriff handhaben können.

5 Konzept und Implementierung einer protokollübergreifenden FIB für RIOT

Die FIB wird in der RIOT Vermittlungsschicht implementiert. Sie basiert auf Ereignis-gesteuertem Verhalten und benötigt keine eigenen Threads. Da die FIB von mehreren Quellen aus zugegriffen werden kann, beispielsweise durch parallel operierende Routing-Protokolle und die Vermittlungsschicht, wird konkurrierender Zugriff auf die FIB-Einträge durch Mutexe verhindert. Die FIB wird unterteilt in (i) einen Datenbereich zum halten der tatsächlichen Adressdaten, und (ii) eine Verwaltungsstruktur für ihre Einträge.

(i) Die Repräsentation der tatsächlichen Adressdaten wird in eine Containerklasse ausgelagert[1]. Die Größe eines Containers, sowie die maximale Anzahl der Container-Einträge wird in einer statisch angelegten Tabelle zur Compile-Zeit festgelegt. Die einzelnen Container enthalten die Adressbytes, die Anzahl durch die gespeicherte Adresse tatsächlich genutzten Bytes und einen Referenzzähler der angibt wie oft diese Adresse in FIB-Einträgen genutzt wird. Die Containerklasse bietet der FIB Schnittstellenfunktionen an, um Adressen hinzuzufügen, zu entfernen und miteinander zu vergleichen. Dabei wird beim hinzufügen einer bereits existierenden Adresse in der Container-Tabelle kein neuer Eintrag erstellt, sondern ein Referenzzähler inkrementiert. Hierfür implementiert die Containerklasse intern eine Funktion, die zur übergebenen Adresse einen identischen Eintrag sucht. Analog wird beim entfernen einer Adresse der Referenzzähler des entsprechenden Containers dekrementiert.
Im folgenden wird das Konzept der Zugriffsfunktionen erläutert:

- `address_container_t *address_add(uint8_t *addr, size_t addr_size);`

  Diese Funktion erstellt einen neuen Container-Eintrag aus der übergebenen Adresse `addr`, falls die Adresse nicht zuvor bereits in einen Container eingetragen wurde. Der Referenzzähler des neuen, oder schon bestehenden Container-Eintrags wird inkrementiert. Als Rückgabewert erhält die FIB einen Pointer auf den Container-Eintrag, in der die Adresse gespeichert ist.

[1] `https://github.com/BytesGalore/RIOT/blob/add_fib/sys/include/net/ng_fib/ng_universal_address.h`

- `void address_remove(address_container_t *entry);`

  Mithilfe dieser Funktion kann die FIB einen zuvor erstellten Container-Eintrag, identifiziert durch den Übergabeparameter `entry`, löschen. Dabei wird nur der Referenzzähler des Container-Eintrags dekrementiert. Erreicht der Referenzzähler 0, gilt der entsprechende Container-Eintrag als ungenutzt und kann mit neuen Adressdaten überschrieben werden.

- `int address_get(address_container_t *entry, uint8_t *addr, size_t *addr_size);`

  Diese Funktion kopiert die Adressdaten, aus dem durch `entry` identifizierten Container-Eintrag, in den Kontext des Aufrufers. Dabei werden die Adressdaten an die Stelle des Ausgangsparameters `addr` kopiert.

- `int address_compare(address_container_t *entry, uint8_t *addr, size_t *addr_size);`

  Diese Funktion vergleicht die Adressdaten, aus dem durch `entry` identifizierten Container-Eintrag mit den übergebenen Adressdaten `addr`, und gibt das Ergebnis des Vergleichs als Rückgabewert zurück.

(ii) Die FIB-Verwaltungsstruktur enthält Datenstrukturen[2] und interne Funktionen, um FIB-Einträge geordnet in einer Tabelle abzulegen und zu verwalten[3]. Die dafür genutzten Datenstrukturen verknüpfen zunächst ein Minimalset eines FIB-Eintrags, bestehend aus Zieladresse, der *next-hop* Adresse sowie der genutzten Kommunikationsschnittstelle miteinander[16]. Des weiteren, enthält die Datenstruktur je ein Element, dass zum speichern von Eigenschaften der beinhalteten Adressen genutzt werden kann, beispielsweise zur Identifikation des Typs der Adresse und die Art der eingesetzten Kompression. Zusätzlich enthält die Datenstruktur ein Feld für die Lebensdauer des Eintrags in ms vgl. Abb. 5.1.
Die FIB enthält eine statisch allozierte Tabelle für ihre FIB-Einträge, deren Größe zur Compile-Zeit festgelegt wird. Diese Tabelle wird durch Aufruf von Schnittstellenfunktionen durch Routing-Protokolle, oder manuelle Eingaben, mit *next-hop* Informationen befüllt. Der Vermittlungsschicht von RIOT wird eine FIB-Schnittstellenfunktion angeboten, um einem *next-hop* zu einer Zieladresse anzufragen. Intern implementiert die FIB eine Suchfunktion, die per *Longest Präfix Match* eine passende *next-hop* Adresse sowie die zu nutzende Kommunikationsschnittstelle zu einer gegebenen Zieladresse liefert. Zusätzlich enthält die FIB einen Mechanismus, der beim Ablauf der Lebensdauer eines FIB-Eintrags, diesen zur Wiederverwendung durch

[2] `https://github.com/BytesGalore/RIOT/blob/add_fib/sys/include/net/ng_fib/ng_fib_table.h`
[3] `https://github.com/BytesGalore/RIOT/blob/add_fib/sys/include/net/ng_fib/ng_fib.h`

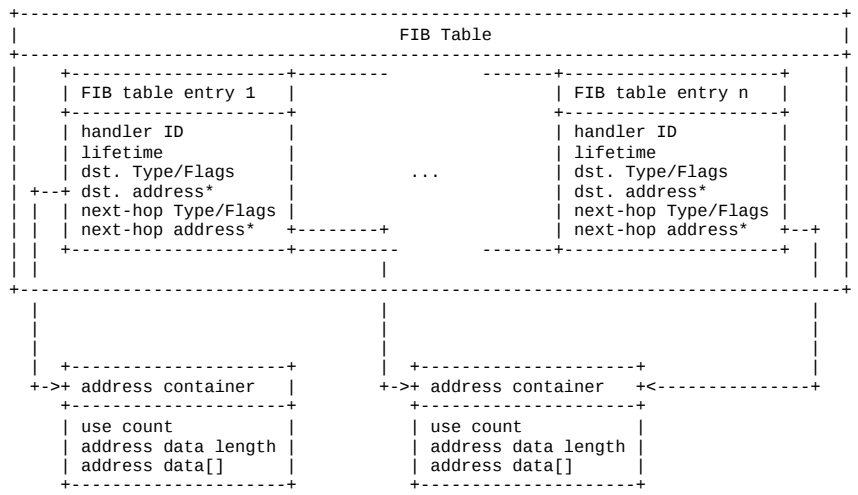

Abbildung 5.1: Datenbereiche und Einträge der FIB Tabelle

neue *next-hop* Informationen freigibt. Ein definierter Wert von $2^{32} - 1$ wird als „unendlich" interpretiert und gibt an, dass die Lebensdauer des Eintrags niemals abläuft. Die Lebensdauer und die automatische In-Validierung eines Eintrags wird in Abschnitt 5.1 genauer diskutiert. Zur Handhabung von reaktiven Routing-Protokollen bietet die FIB eine Registrierfunktion an, mit der sich ein reaktives Routing-Protokoll bei der FIB anmelden kann. Ist die Suche nach einem *next-hop* zu einer Zieladresse in der FIB erfolglos, werden alle zuvor registrierten reaktiven Routing-Protokolle angestoßen eine Route zu der gesuchten Zieladresse zu ermitteln, und einen gefundenen *next-hop* in der FIB einzutragen. Die Handhabung von reaktiven Routing-Protokollen wird in Abschnitt 5.2 genauer diskutiert.

Im folgenden Teil wird das Konzept der angebotenen Schnittstellenfunktionen erläutert. Diese sind als Komponenten-Diagramm in Abb. 5.2 skizziert.

- `int fib_add(kernel_pid_t handler_id, uint8_t *dst, size_t dst_size, uint8_t *next_hop, size_t next_hop_size, uint32_t next_hop_flags, uint32_t lifetime);`

  Diese Funktion wird Routing-Protokollen und für manuelle Eingaben angeboten, um neue Einträge in die FIB einzufügen. Als Parameter übergibt der Aufrufer die ID des zur Weiterleitung zu benutzenden *Handlers* `handler_id`, die Zieladresse `dst`, die *next-hop* Adresse `next_hop` mit seinen flags `next_hop_flags` sowie die Lebensdauer `lifetime` des Eintrags. Der *Handler* ist hierbei die PID des Treibers für eine spezifische Kommunikationsschnittstelle.

- `int fib_update(kernel_pid_t handler_id, uint8_t *dst, size_t dst_size, uint8_t *next_hop, size_t next_hop_size, uint32_t next_hop_flags, uint32_t lifetime);`

  Diese Funktion wird zur Aktualisierung der *next-hop* Adresse sowie der Lebensdauer eines FIB-Eintrags benutzt, der bereits in der FIB eingetragen wurde und durch die `handler_id` sowie die Zieladresse `dst` eindeutig identifizierbar ist.

- `void fib_remove(uint8_t *dst, size_t dst_size);`

  Diese Funktion wird Routing-Protokollen und für manuelle Eingaben angeboten, um Einträge aus der FIB zu entfernen. Als Parameter wird die Zieladresse `dst` übergeben.

- `void fib_register_rrp(void);`

  Mit dieser Funktion meldet sich ein reaktives Routing-Protokoll bei der FIB an. Die FIB speichert dabei die PID des reaktiven Routing-Protokolls, um sie zum initiieren einer Routenermittlung zu nutzen.

- `int fib_get_next_hop(uint8_t *dst, size_t dst_size, kernel_pid_t *handler_id, uint8_t *next_hop, size_t *next_hop_size, uint32_t* next_hop_flags);`

  Diese Funktion wird von der Vermittlungsschicht von RIOT genutzt, um *next-hop* Informationen von der FIB zum Weiterleiten eines Pakets in Richtung Zieladresse zu erhalten. Als Eingangs-Parameter bekommt die Funktion die Zieladresse `dst`. Die *Handler* ID `handler_id` und die *next-hop* Adresse `next_hop` mit flags `next_hop_flags`, wird als Ausgangs-Parameter mit den Werten aus dem gefundenen FIB-Eintrag überschrieben.

  Existiert kein *next-hop* Eintrag zu der gegebenen Zieladresse, und es haben sich reaktive Routing-Protokolle bei der FIB registriert, wird an die PIDs der registrierten reaktiven Routing-Protokolle eine Nachricht mit der Zieladresse mittels RIOT *message passing* Mechanismus verschickt, um das lernen einer Route zu der Zieladresse `dst` anzustoßen.

5.1 Automatische Invalidierung von FIB-Einträgen

Durch die Nutzung einer Lebensdauer in FIB-Einträgen wird ein Mechanismus angeboten, der die In-Validierung der FIB-Einträge automatisch durch die FIB, und unabhängig vom Verfasser des Eintrags vornehmen kann. Die Lebensdauer eines FIB-Eintrags ist auf eine ms genaue Auflösung festgelegt. Wird eine gröbere Granularität von einem Routing-Protokoll benötigt, muss dieses selbstständig eine Umrechnung vornehmen.
Bei jedem Zugriff auf einen FIB-Eintrag wird dessen Lebensdauer neu berechnet, wenn sie

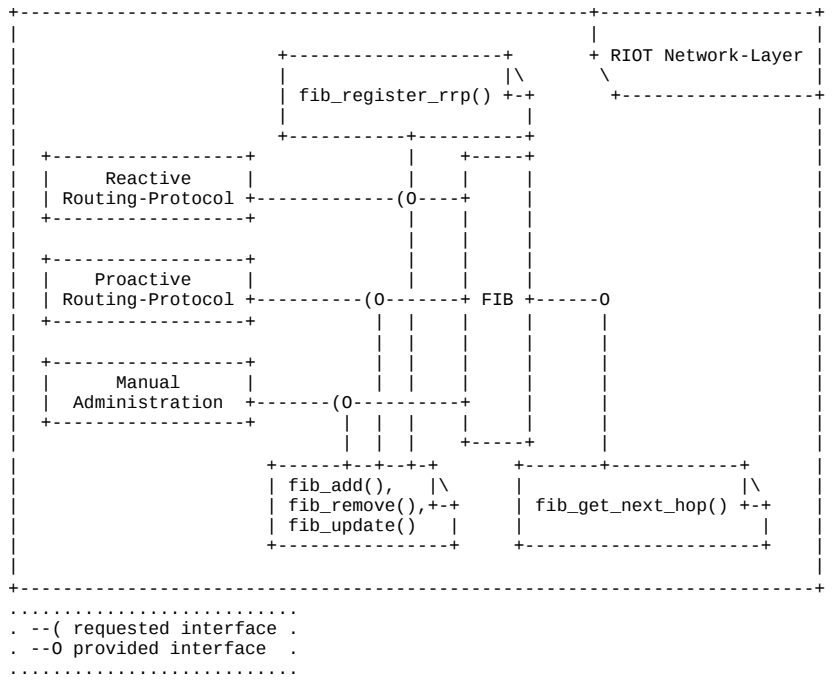

Abbildung 5.2: Komponentendiagramm der FIB Implementierung

beim einfügen nicht auf „unendlich" festgelegt wurde. Ist die Lebensdauer abgelaufen, kann der FIB-Eintrag durch einen neuen Eintrag ersetzt werden. Wird ein FIB-Eintrag durch ein Routing-Protokoll, oder durch manuelle Eingabe entfernt, wird nur die Lebensdauer des Eintrags auf 0 gesetzt. Damit ist die Lebensdauer gleichzeitig der Indikator ob ein FIB-Tabellenplatz frei ist. Nutzt ein Routing-Protokoll diese angebotene Eigenschaft eines FIB-Eintrags nicht und setzt die Lebensdauer seiner eingetragenen *next-hop*s auf „unendlich", muss es selbständig die FIB-Einträge löschen.

5.2 Handhabung reaktiver Routing-Protokolle

Wird auf einem RIOT-Knoten ein reaktives Routing-Protokoll (RRP) benutzt, muss es sich bei der FIB anmelden, um für die Routenermittlung angestoßen zu werden. Ein direktes anstoßen durch Funktionen der RIOT Vermittlungsschicht findet nicht mehr statt. Das reaktive Routing-Protokoll ruft zum Registrieren bei der FIB die `fib_register_rrp()` auf. Die Funktion speichert die `kernel_pid_t`[4] PID des Aufrufers in einer Tabelle, mit zur Compile-Zeit festgelegter maximaler Anzahl an möglichen Einträgen. Ist das starten einer Routenermittlung durch

[4] `https://github.com/RIOT-OS/RIOT/blob/master/core/include/kernel_types.h#L83`

reaktive Routing-Protokolle notwendig, weil kein *next-hop* Eintrag in der FIB gefunden wurde, wird der folgende Ablauf zum benachrichtigen durchgeführt:

- Die FIB bereitet eine Nachricht für die reaktiven Routing-Protokolle vor. Diese beinhaltet einen den RRPs und der FIB gemeinsam bekannten Identifikator, der die Nachricht als Aufruf zur Routenermittlung auszeichnet sowie die Zieladresse zu der die Route gefunden werden soll. Hierbei ist zu beachten, dass die Nachricht nur die Referenz zur Zieladresse beinhaltet und somit die Adressdaten vom RRP in seinen eigenen Kontext kopiert werden müssen.
- Die FIB iteriert durch die PID-Tabelle der registrierten reaktiven Routing-Protokolle, und sendet mittels RIOT *message passing* an jeden Eintrag die zuvor vorbereitete Nachricht. Das bedeutet, dass jedes RRP eine Funktion bereitstellen muss, die auf eine Nachricht zur Routenermittlung von der FIB blockierend wartet.
- Das benachrichtigte RRP liest die empfangene Nachricht, überprüft den Identifikator und kopiert die Zieladresse in seinen eigenen Kontext für die Weiterverarbeitung.
- Anschließend sendet das RRP eine Bestätigung an die FIB über den erfolgreichen Empfang der Nachricht.
- Die FIB wartet auf diese Bestätigung, bevor das nächste RRP in der PID-Tabelle benachrichtigt wird.

Wurde ein RRP benachrichtigt und hat den Empfang bestätigt, kann es seine Routenermittlung starten. Nun existieren zwei Möglichkeiten des weiteren Ablaufs:

a. Die FIB meldet der Vermittlungsschicht, dass kein *next-hop* für die Zieladresse existiert, jedoch ein RRP versucht eine Route und einen passenden *next-hop* zu finden. Da die Vermittlungsschicht potentiell viele Pakete zu unterschiedlichen Zielen zustellen möchte und nicht sicher einen *next-hop* für das aktuelle Paket erwarten kann, führt dies dazu, dass die betroffenen Pakete als unzustellbar markiert und verworfen werden. Zur Konsequenz hat diese Variante, dass Pakete eventuell nach einer gewissen Konvergenz-Zeit in der das RRP eine Route findet, zu einer zunächst nicht erreichbaren Zieladresse erfolgreich hätten weitergeleitet werden könnten, jedoch bereits verworfen wurden.

b. Die FIB blockiert, und wartet bis ein RRP einen *next-hop* lernt, oder bis alle RRPs erfolglos die suche beenden. Das RRP sendet im Erfolgsfall eine Nachricht mit der *next-hop* Information

an die FIB. Diese fügt die *next-hop* Information in die FIB-Tabelle ein und gibt sie der Vermittlungsschicht als Ergebnis zurück. Ist das RRP bei der Suche nach einer Route gescheitert, wird dies der FIB mit einer entsprechenden Nachricht mitgeteilt. Das führt zu einer Warteschlange in der Zustellung von Paketen, bis ein *next-hop* für das aktuelle Paket gefunden wurde, oder es aufgrund einer nicht erreichbaren Zieladresse verworfen wird. In Konsequenz könnten so die Informationen aus den wartenden Paketen vor dem Weiterleiten veralten, obwohl sie hätten weitergeleitet werden können.

Variante a. verlangt, dass das Senden von Paketen optimistisch, potentiell mehrfach probiert werden muss, da möglicherweise ein RRP auf dem Knoten aktiv ist und nach einer gewissen Zeit einen *next-hop* liefert. Hierbei ist eine generelle Einschätzung wie häufig oder nach welcher Zeit ein erneutes Senden stattfinden muss nicht möglich, und müsste für jeden spezifischen Anwendungsfall neu festgelegt werden.
Variante b. stellt hingegen den erwarteten Verlauf der Paketweiterleitung in einem Netzwerk mit einem einzigen RRP dar. In einem Szenario, in dem ein Knoten mit verschiedenen Netzwerken und Routing-Protokollen interagiert, ist sie jedoch mindestens hinderlich.
Da der Aktuelle RIOT Netzwerkstack komplett umgebaut wird, ist noch nicht abschließend festgelegt, welches Verhalten die sinnvoller zu nutzende Variante darstellt. Die aktuelle FIB implementiert derzeit Variante b., kann jedoch leicht auf Variante a. angepasst werden.

6 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurde die protokollübergreifende FIB für RIOT vorgestellt und anschließend ihre tatsächliche Umsetzung beschrieben. Im ersten Schritt wurde kurz der aktuelle und der zukünftige RIOT Netzwerkstack vorgestellt. Anschließend wurden Probleme des aktuellen RIOT Netzwerkstacks identifiziert und in konkrete Anforderungen für ein Konzept protokollübergreifenden FIB für RIOT eingeordnet.

Wird eine Kompression von IPv6 Adressen im Netzwerkstack genutzt, können Teile, oder sogar die komplette IPv6 Adresse eines Knotens aus kontextabhängigen Zusatzinformationen in einer Topologie, wie einem Netzwerkpräfix und der Link-Lokalen IP Adresse, abgeleitet werden. Die FIB ist dahingehend umgesetzt, dass die kontextabhängige Information in einem FIB Eintrag gespeichert wird. Diese Information wird der Vermittlungsschicht zur Verfügung gestellt, wenn sie einen *next-hop* zu einer Zieladresse anfragt.

Der zukünftige RIOT Netzwerkstack implementiert generalisierte Schnittstellen an der sich Transceiver anmelden können. Das erlaubt RIOT Knoten, sich mit Netzwerken zu verbinden die potentiell IP-fremde Adressen nutzen, wie beispielsweise *Bluetooth Low Energy*. Die FIB ist dahingehend implementiert, dass sie mit beliebigen Adresstypen umgehen und diese verwalten kann.

Die Implementierung der FIB erlaubt die Tabellengröße für FIB Einträge, sowie die erforderliche Speichergröße für die konkreten Adressdaten, beliebig zur Compile-Zeit festzulegen. Dynamisches allozieren von Speicher für die Einträge ist nicht erforderlich.

Durch die Unterstützung generalisierter Schnittstellen seitens des zukünftigen Netzwerkstacks, kann ein RIOT Knoten eine beliebige Anzahl an Kommunikationsschnittstellen parallel betreiben. Die eindeutige Zuordnung eines FIB-Eintrags zu einer Kommunikationsschnittstelle, wird anhand einer im Eintrag gespeicherten *Handler* ID vollzogen.

Die FIB implementiert einen Mechanismus zur Handhabung reaktiver Routing-Protokolle. Diese werden durch die FIB angestoßen, wenn kein *next-hop* für die Zieladresse eingetragen ist, um eine Route zur Zieladresse zu suchen und letztendlich den *next-hop* für die Weiterleitung von Paketen auf dieser Route in die FIB einzutragen.

Die Implementierung der FIB trennt Routing-Protokolle vom direktem Zugriff seitens der Vermittlungsschicht. Die *next-hop*s zu einer Zieladresse werden bei der FIB angefragt, die wiederum durch Routing-Protokolle in die FIB eingetragen werden. Die FIB ist dahingehend implementiert, dass konkurrierender Zugriff auf die Daten verhindert wird und damit Routing-Protokolle parallel betrieben werden können.

In folgenden Schritten müssen die in RIOT eingesetzten proaktiven Routing-Protokolle *Optimized Link State Routing* (OLSR)[17] und *Routing Protocol for Low power and Lossy Networks* (RPL)[18] sowie das reaktive Routing-Protokoll *Ad-hoc On-demand Distance Vector v2* (AODVv2)[19] dahingehend angepasst werden, dass sie die FIB wie hier vorgesehen nutzen.

Literaturverzeichnis

[1] Sergio Gusmerol and Karel Wouters and Maurizio Tomasella and Kostas Kalaboukas and Harald Vogt and Mark Harrison and Ovidiu Vermesan, *Vision and Challenges for Realising the Internet of Things*, European Comission, 2010.

[2] Emmanuel Baccelli, Oliver Hahm, Mesut Günes, Matthias Wählisch, and Thomas C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*, Piscataway, NJ, USA, 2013, IEEE Press.

[3] Sameer Seth and M. Ajaykumar Venkatesulu, *TCP/IP Architecture, Design and Implementation in Linux*, Wiley, 2008.

[4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, IETF, September 2007.

[5] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, IETF, September 2007.

[6] J. Postel, "User Datagram Protocol," RFC 768, IETF, August 1980.

[7] Jon Postel, "Internet Protocol," RFC 791, IETF, September 1981.

[8] Stephen E. Deering and Robert M. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, IETF, December 1998.

[9] Robert M. Hinden and Stephen E. Deering, "IP Version 6 Addressing Architecture," RFC 2373, IETF, July 1998.

[10] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, IETF, September 2011.

[11] Bluetooth SIG, *Bluetooth Specification 4.2*, Bluetooth SIG, December 2014.

[12] Zach Shelby and Carsten Bormann, *6LoWPAN: The Wireless Embedded Internet*, Wiley, 2009.

[13] IEEE Std. 802.15.4-2011, *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE, 2011.

[14] ZigBee Alliance Profile Specification:, *ZigBee Profile: Retail Services 0x010A, Version 1.0, Revision 17*, ZigBee Alliance, October 2013.

[15] Nordic Semiconductor, *nRF24L01+ Single Chip 2.4GHz Transceiver, Product Specification v1.0*, Nordic Semiconductor, September 2008.

[16] Fred Baker, "Requirements for IP Version 4 Routers," RFC 1812, IETF, June 1995.

[17] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, IETF, October 2003.

[18] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, IETF, March 2012.

[19] Charles Perkins, Stan Ratliff, John Dowdell, and Lotte Steenbrink, "Dynamic MANET On-demand (AODVv2) Routing," Internet-Draft – work in progress 06, IETF, December 2014.