Ruby アソシエーション　開発助成金 2020
「ruby の標準ライブラリ Socket への Happy Eyeballs Version 2 (RFC8305) の導入」
メンター報告書

田中哲

2021 年 3 月 21 日

1　はじめに

本プロジェクトは IPv4/IPv6 dual stack 環境において、名前解決と TCP 接続の確立に必要以上に長い時間がかかる問題への対策を行うものである。

2　問題

IPv4/IPv6 dual stack 環境における TCP 接続に不必要に時間がかかる原因として、以下の点があげられる。

- DNS ではそのプロトコルの制約として、IPv6 アドレスの問い合わせと IPv4 アドレスの問い合わせは別々の問い合わせとして行わなければならない。そのため、IPv6 アドレスの問い合わせを最初に行い、それが失敗した場合に IPv4 の問い合わせを行うとした場合、IPv6 アドレスの問い合わせの返事がない場合 (たとえば DNS サーバが IPv6 アドレスの問い合わせに対して反応しない場合、また query や response のパケットが落ちた場合) 失敗するまでにタイムアウトを待つ必要があり、これは長い時間がかかる。この動作は C 言語の API のレベルでは、getaddrinfo 関数が失敗するまでに長い時間がかかるという現象となる。getaddrinfo は libc の関数であるため、具体的なタイムアウトの時間は libc が決定する。
- DNS の問い合わせの結果として IPv6 アドレスと IPv4 アドレスの両方が得られたと仮定して、IPv6 アドレスに対して TCP 接続を最初に試み、失敗した場合に IPv4 アドレスに対して問い合わせを行うとした場合、IPv6 アドレスに対する TCP handshake の返事 (3-way handshake の SYN-ACK) が到着しない場合に失敗するまでにタイムアウトを待つ必要があり、これは長い時間がかかる。この動作は C 言語の API のレベルでは、connect 関数が失敗するまでに長い時間がかかるという現象となる。connect はシステムコールであるため、具体的なタイムアウトの時間は kernel が決定する。なお、DNS は問い合わせに対して複数のアドレスを返すことがあるため、IPv4/IPv6 dual stack でなくても、複数のアドレスが返され、その中に TCP 接続に反応しないアドレスがあれば、同様の問題が生じる。

なお、getaddrinfo 関数は、protocol independent に呼び出す、つまり IPv6 や IPv4 といった指定をせずに呼び出すこともできる。このようにした場合、getaddrinfo は (原理的には) 内部的に IPv6 と IPv4 の問い合わせを同時に行うことができる。しかし、この場合でも IPv6 と IPv4 の両方の問い合わせが終了するまでは getaddrinfo を return できない。そのため、このような呼び出し方では、問い合わせが先に終了したほうから (もう一方が終わる前に) TCP 接続を始めることはできない。

3 RFC 8305: Happy Eyeballs Version 2

前述の問題に対し、Happy Eyeballs Version 2 は以下の解決を行う。(複数の DNS サーバの選択などについては省略し、Ruby における実装に関わる部分のみを記述する。)

- DNS による IPv6 アドレスの問い合わせと IPv4 アドレスの問い合わせを同時に開始する。これにより、先に返事を得られたほうから処理を進めることが可能になる。getaddrinfo 関数を使う場合、IPv6 アドレスの問い合わせを行う getaddrinfo と IPv4 アドレスの問い合わせを行う getaddrinfo を別のスレッドで呼び出すことになる。
- 複数のアドレスが得られた場合、TCP 接続の開始を順に行うが、kernel のタイムアウトにより接続の失敗が確定しなくても、短い時間 (Connection Attempt Delay = 250 ms) で接続の結果 (成功または失敗) が判明しなければ、次のアドレスに対する接続を開始する。この結果、ある時点で複数のアドレスに対する TCP 接続が並行に行われることがあるが、最初にひとつ接続に成功した時点でその接続を用いることにし、他は破棄する。このために non-blocking connect を用いる。

ここで、IPv4 よりも IPv6 を優先するため、DNS による IPv4 アドレスの問い合わせが IPv6 アドレスの問い合わせよりも先に終了した場合、得られた IPv4 アドレスに対して即座に接続を開始するのではなく、ごく短い時間 (Resolution Delay = 5 ms) IPv6 アドレスの問い合わせの終了を待つ。

また、IPv4 と IPv6 のアドレスの両方が複数得られた場合、TCP 接続の試みは可能なかぎり IPv4 と IPv6 を交互に行う。

これを状態遷移図として表現すると、図 1 となる。なお、TCP 接続開始の間隔を保証する Connection Attempt Delay によるタイムアウトは直前の TCP 接続開始を起点とした時間とする。また、TCP 接続の試みの失敗が判明した場合、それに利用したソケットは適宜クローズするものとする。

4 Happy Eyeballs Version 2 実装の困難な点

ここでは、開発助成の期間に判明して修正された、実装が困難な点をひとつ説明する。

図 1 の状態 v4c と v6c では、それらの状態からの遷移として、Connection Attempt Delay によるタイムアウト、TCP の connection が成立する、getaddrinfo が終了する、という 3 種がある。このため、この 3 種のイベントのうち、最初に発生するものを待たなければならない。

旧来から Unix において複数のイベントを待つシステムコールには select がある。しかし、select ではタイムアウトと connection の成立を同時に待てるが、getaddrinfo の終了を待つことはできない。これは select では (タイムアウト以外に) file descriptor に起きたイベントを待てるが、getaddrinfo という関数の終了 (ないしは getaddrinfo を呼んだスレッドの終了) において file descriptor に関するイベントは起きないためである。

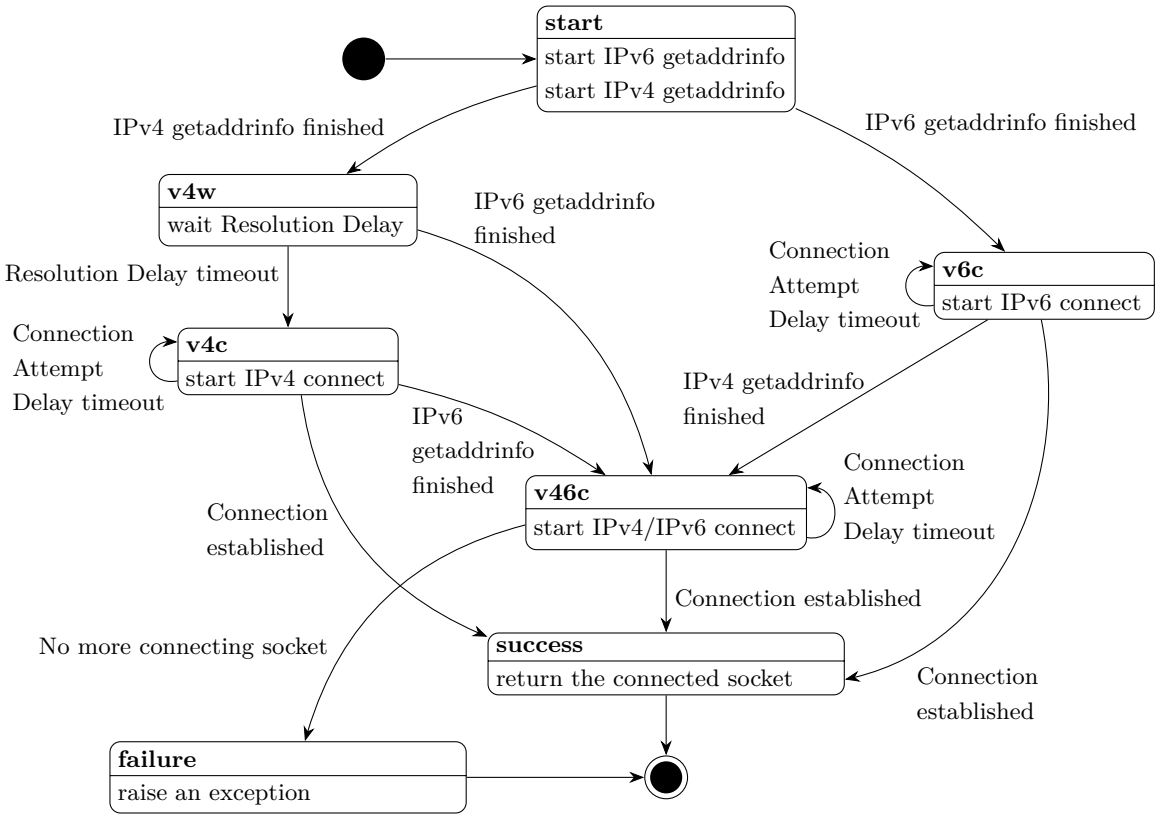

図 1　Happy Eyeballs Version 2 の状態遷移図

松下氏による初期実装において、最終報告書に記載されているように、無駄な待ち時間が発生したのは、この点が考慮されていなかったためである。pipe を用いてこの問題を解決したというのは、getaddrinfo が終了したときに pipe に終了したことを意味するメッセージを書き込むことにより getaddrinfo が終了したことを select で検出できるようにした、ということを意味している。

5　まとめ

松下氏は以前から getaddrinfo_a の利用を試みており、この助成に関連する問題に取り組んでいる。

本助成期間に行われた開発は Ruby 本体へのマージには至っていないものの、アルゴリズムの理解とその実装に進捗があったと認められる。