TESIS - KI142502

PENINGKATAN PERFORMA *VERTICAL HANDOVER* PADA PERANGKAT BERGERAK DENGAN PENGURANGAN *DELAY* PADA PROSES DETEKSI TERPUTUSNYA KONEKSI BERDASARKAN *PACKET INTER-ARRIVAL TIMEOUT*

Pangestu Widodo
5115201021

DOSEN PEMBIMBING
Waskitho Wibisono, S.Kom, M.Eng, Ph.D

PROGRAM MAGISTER
RUMPUN MATA KULIAH KOMPUTASI BERBASIS JARINGAN
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
Pangestu Widodo
Nrp. 5115201021

Dengan judul :
PENINGKATAN PERFORMA VERTICAL HANDOVER PADA PERANGKAT BERGERAK DENGAN PENGURANGAN DELAY PADA PROSES DETEKSI TERPUTUSNYA KONEKSI BERDASARKAN PACKET INTER-ARRIVAL TIMEOUT

Tanggal Ujian : 12-1-2017
Periode Wisuda : 2016 Gasal

Disetujui oleh:

Waskitho Wibisono, S.Kom, M.Eng, Ph.D
NIP. 197410222000031001
(Pembimbing 1)

Tohari Ahmad, S.Kom, MIT, Ph.D
NIP. 197505252003121002
(Penguji 1)

Dr.Eng. Radityo Anggoro, S.Kom, M.Sc
NIP. 1984101620081210002
(Penguji 2)

Royyana Muslim I, S.Kom, M.Kom, Ph.D
NIP. 197708242006041001
(Penguji 3)

an. Direktur Program Pascasarjana
Asisten Direktur

Prof. Dr. Ir. Tri Widjaja, M.Eng.
NIP. 19611021 198603 1 001

Direktur Program Pasca Sarjana,

Prof.Ir.Djauhar Manfaat, M.Sc., Ph.D.
NIP. 196012021987011001

PENINGKATAN PERFORMA VERTICAL HANDOVER PADA PERANGKAT BERGERAK DENGAN PENGURANGAN *DELAY* PADA PROSES DETEKSI TERPUTUSNYA KONEKSI BERDASARKAN *PACKET INTER-ARRIVAL TIMEOUT*

Nama mahasiswa : Pangestu Widodo

NRP : 5115201021

Pembimbing : Waskitho Wibisono, S.Kom, M.Eng, Ph.D

ABSTRAK

Dewasa ini perangkat bergerak yang memiliki lebih dari satu *network interface* sudah menjadi hal yang biasa. Meskipun demikian, pada umumnya perangkat bergerak saat ini hanya memungkinkan penggunaan sebuah *network interface* saja setiap saat untuk terhubung ke *internet*. Ketika terjadi peralihan dari *netwok interface* yang satu ke *network interface* lain, diperlukan teknologi *vertical handover* untuk menjaga kualitas koneksi yang dirasakan oleh pengguna. Telah banyak penelitian yang dilakukan dan standar yang dibuat untuk mendukung *handover* pada *data link layer* dan *network layer*, namun belum ada standar untuk mendukung *handover* pada *transport layer*.

Multipath TCP merupakan pengembangan *TCP* yang memungkinkan sebuah aplikasi pada sebuah *host* untuk menggunakan beberapa koneksi *TCP* sekaligus untuk berkomunikasi dengan *host* lain. Setiap koneksi *TCP* yang digunakan dalam *Multipath TCP* disebut sebagai *subflow*. *Multipath TCP* berpotensi memberikan dukungan *handover* yang baik pada *transport layer* bagi perangkat bergerak karena perubahan *subflow* yang digunakan dapat dilakukan secara transparan sehingga tidak menganggu kinerja aplikasi. Ketika sebuah *subflow* terputus akibat peralihan dari satu *network interface* ke *network interface* lain, *Multipath TCP* dapat beralih menggunakan *subflow* lain atau membuat *subflow* baru secara transparan sehingga tidak dirasakan oleh aplikasi.

Untuk dapat melakukan *vertical handover* dengan cepat, perangkat bergerak harus dapat mendeteksi terputusnya sebuah *subflow* dengan cepat. Hal ini sulit dilakukan pada saat perangkat bergerak berperan sebagai penerima *payload* (*receiver*) tanpa mengirimkan *payload* ke *host* lain karena ia tidak dapat menggunakan mekanisme standar pada *TCP* untuk mendeteksi terputusnya *subflow*.

Pada penelitian ini diajukan metode bagi *host receiver* untuk mendeteksi terputusnya *subflow* pada sebuah koneksi *Multipath TCP* berdasarkan *packet inter-arrival timeout*. Pada metode ini *receiver* menerima informasi adanya paket yang masih akan diterima, sehingga dapat memperhitungkan *timeout* untuk paket tersebut berdasarkan interval waktu antar paket yang telah diterima.

Berdasarkan pengujian menggunakan *simulator NS2* menunjukkan bahwa metode yang diajukan mampu mendeteksi terputusnya *subflow* dengan tingkat *precision* sebesar 0,944 pada jaringan dengan tingkat *packet loss* sebesar 0,1% atau kurang, dan juga menunjukkan bahwa pada jaringan dengan tingkat *packet loss* antara 0,1% hingga 1%, penggunaan metode yang diajukan menghasilkan pengurangan waktu deteksi terputusnya *subflow* sebesar rata-rata 310,82% *round trip time* jaringan dibandingkan dengan menggunakan *retransmission timeout*.

Kata kunci: *handover*, perangkat bergerak, *Multipath TCP*, deteksi terputusnya *subflow*, *packet inter-arrival timeout*.

PERFORMANCE IMPROVEMENT OF VERTICAL HANDOVER IN MOBILE DEVICES BY REDUCING DELAY IN BROKEN CONNECTION DETECTION PROCESS BASED ON PACKET INTER-ARRIVAL TIMEOUT

By : Pangestu Widodo

Student Identity Number : 5115201021

Supervisor : Waskitho Wibisono, S.Kom, M.Eng, Ph.D

ABSTRACT

Nowadays, multiple network interfaces have become a common feature in mobile devices. Despite of having multiple network interfaces, most of those mobile devices only allow single network interface to be used at a time for connecting to the internet. When the device decides to switch between network interfaces, vertical handover technology is needed to maintain connection quality for the user. Many researches have been done and standards have been created to support handover in data link layer and network layer, but no standard have been created to support handover in transport layer.

Multipath TCP is a TCP extension that enables an application in a particular host to use several TCP connection to communicate with another host. Each of TCP connection used in Multipath TCP is called subflow. Multipath TCP holds great potential to provide good handover support in transport layer for mobile device because changes to subflow used can be done transparently so it will not disrupt application performance. When a subflow fails because of switchover between network interfaces, multipath TCP is able to switch to another subflow or create a new subflow transparently so this whole process is not visible to the application.

To perform vertical handover quickly, mobile device must be able to quickly detect subflow failure. This may be a hard thing to do when mobile device

acts as payload receiver without sending any payload to other host because it can not use mechanisms provided in standar TCP to detect subflow failure.

This research propose a novel method for a receiver host to detect subflow failure in a Multipath TCP connection based on packet inter-arrival timeout. In this method, the receiver host is informed about the existence of packets it may receive later, so it can calculate timeout for the aforementioned packets based on existing packet inter-arrival time.

Experiments using NS2 simulator in networks show that the proposed method were able to detect subflow failure with a precision of 0,944 for a network with packet loss rate of 0,1% or less, and also show the advantage of the proposed method in networks with packet loss rate of 0.1% - 1% yielding an average reduction in subflow failure detection delay of 310,82% of network round trip time compared to using standard retransmission timeout.

Keywords: handover, mobile devices, Multipath TCP, failed subflow detection, packet inter-arrival timeout.

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

BAB 1
PENDAHULUAN

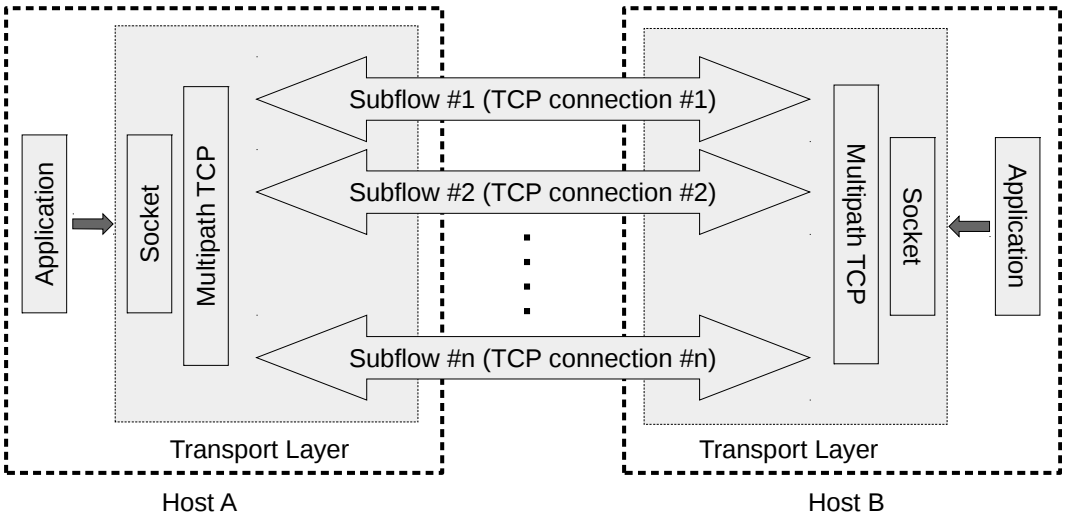
1.1 Latar Belakang

Dewasa ini perangkat bergerak mengalami perkembangan yang sangat pesat tidak hanya dari segi jumlah, namun juga teknologi yang diusung. Perangkat bergerak yang memiliki lebih dari satu *network interface* sudah menjadi hal yang biasa. Meskipun demikian, penggunaan beberapa *network interface* untuk terhubung dengan *internet* masih mengalami kendala karena pada umumnya perangkat bergerak saat ini hanya memungkinkan penggunaan sebuah *network interface* saja setiap saat untuk terhubung ke *internet*. Ketika terjadi peralihan dari *netwok interface* yang satu ke *network interface* lain, diperlukan teknologi *vertical handover* untuk menjaga kualitas koneksi yang dirasakan oleh pengguna.

Handover adalah proses dimana sebuah *node* (dalam hal ini perangkat bergerak) mempertahankan konektivitas yang dimilikinya saat terjadi perpindahan dari satu *link* ke *link* yang lain. Apabila kedua *link* tersebut menggunakan teknologi yang sama maka disebut sebagai *horizontal handover*, dan apabila kedua *link* tersebut menggunakan teknologi yang berbeda maka disebut sebagai *vertical handover* (*IEEE Computer Society*, 2009). Teknologi *handover* diharapkan dapat mendukung terjaganya kualitas koneksi yang dirasakan oleh pengguna perangkat bergerak saat berpindah dari satu jaringan ke jaringan yang lain.

Telah banyak penelitian yang dilakukan dan standar yang dibuat untuk mendukung *handover* pada perangkat bergerak, namun standar dan penelitian tersebut lebih banyak membahas tentang *handover* pada *data link layer* dan *network layer*. *IEEE 802.11r Fast BSS Transition*, *IEEE 802.11k Radio Resource Measurement* dan *IEEE 802.11v Wireless Network Management* merupakan standar yang dibuat oleh *IEEE* untuk mendukung *horizontal handover* pada jaringan yang menggunakan perangkat *WLAN*. Ketiga standar tersebut pada awalnya merupakan *amendments* terhadap standar 802.11-2007 dan telah

dimasukkan ke dalam standar 802.11-2012 (*IEEE Computer Society*, 2012). Disamping itu *IEEE* juga telah membuat standar *IEEE 802.21 Media Independent Handover* (*IEEE Computer Society*, 2009) untuk melakukan *vertical handover* antar medium yang berbeda. Semua standar tersebut menekankan dukungan *handover* pada *data link layer*. Contoh standar yang dibuat untuk mendukung *handover* pada *network layer* adalah *Mobile IP* dan *Proxy Mobile IPv6*. *Mobile IP* dan *Proxy Mobile IPv6* bertujuan agar sebuah perangkat dapat tetap terhubung ke internet menggunakan *IP address* yang sama setelah terjadinya *handover* pada *subnet* yang berbeda. *Mobile IP* terbagi menjadi dua yaitu untuk *IPv4* (Perkins, 2010) dan untuk *IPv6* (Perkins dkk, 2011). *Mobile IP* memiliki kekurangan yaitu perangkat bergerak harus diubah mendukung protokol tersebut. *Proxy Mobile IPv6* (Gundavelli dkk, 2008) memiliki kelebihan dibanding *Mobile IP* yaitu tidak diperlukan perubahan apapun pada perangkat bergerak. Terlepas dari banyaknya standar yang ada untuk mendukung *handover* pada *data link layer* dan *network layer*, belum terdapat standar untuk mendukung *handover* pada *transport layer*.



Gambar 1.1: Ilustrasi penggunaan *Multipath TCP*

Multipath TCP (Ford dkk, 2013) merupakan pengembangan *TCP* yang memungkinkan penggunaan beberapa koneksi *TCP* secara simultan. Dengan kemampuan tersebut *Multipath TCP* menjanjikan performa transfer data yang

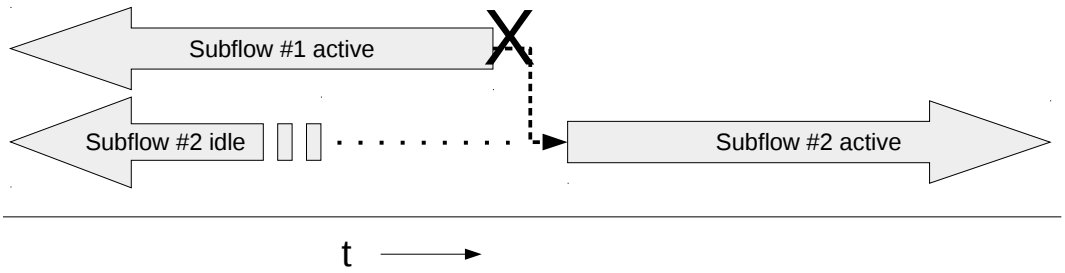
lebih tinggi dibanding *TCP* standar. Selain itu, karena arsitektur yang mampu memberikan abstraksi koneksi *TCP* terhadap *application layer*, *Multipath TCP* juga menjanjikan performa yang baik untuk mendukung mobilitas perangkat bergerak karena proses *handover* dapat dilakukan secara transparan terhadap aplikasi. Ketika sebuah koneksi *TCP* (disebut juga sebagai *subflow*) terputus akibat peralihan dari satu *network interface* ke *network interface* lain, *Multipath TCP* dapat beralih menggunakan koneksi *TCP* lain atau membuat koneksi *TCP* baru secara transparan sehingga tidak dirasakan oleh aplikasi. Ilustrasi penggunaan *Multipath TCP* dapat dilihat pada Gambar 1.1.

Secara umum terdapat dua jenis mekanisme *handover* yaitu *make-before-break* maupun *break-before-make*. Pada *make-before-break* perangkat melakukan hubungan dengan jaringan tujuan sebelum hubungan dengan jaringan asal terputus, sedangkan pada *break-before-make* perangkat baru akan melakukan hubungan dengan jaringan tujuan setelah hubungan dengan jaringan asal terputus (*IEEE Computer Society*, 2009).

Multipath TCP mendukung proses handover baik menggunakan mekanisme *make-before-break* maupun *break-before-make*. Proses *handover* dengan mekanisme *make-before-break* menggunakan *Multipath TCP* dilakukan dengan membuat *subflow* (sebuah koneksi *TCP*) baru sebelum *subflow* lama terputus. Sedangkan dukungan terhadap mekanisme *break-before-make* diperoleh melalui mekanisme asosiasi antara *subflow* yang baru dengan *subflow* lama yang terputus. Mekanisme asosiasi ini dapat dilakukan secara independen oleh *subflow* baru tanpa harus melibatkan *subflow* lama.

Salah satu masalah utama pada perangkat bergerak adalah keterbatasan energi. Penelitian yang dilakukan Carroll dan Heiser pada (Carroll dkk, 2010) menunjukkan bahwa konsumsi energi menggunakan *WiFi* lebih rendah daripada konsumsi energi menggunakan *GPRS*. Oleh karena itu, meskipun *Multipath TCP* mendukung penggunaan beberapa *network interface* untuk melakukan *transfer data* dalam waktu yang bersamaan (disebut *full mode*), akan lebih baik jika perangkat bergerak menggunakan satu *network interface* saja yang dianggap paling efisien. Cara ini disebut sebagai *single-path mode* (Kaup dkk, 2015).

Kekurangan *single-path mode* adalah delay proses *handover* cenderung tinggi. Hal ini disebabkan proses *handover* melibatkan proses aktivasi *network interface* yang akan digunakan, yang seringkali membutuhkan waktu yang relatif lama. Alternatif yang mungkin lebih baik adalah *backup mode* dimana beberapa *subflow* dibuat akan tetapi hanya satu *subflow* yang digunakan untuk transfer data pada setiap saat, sementara *subflow* lain berfungsi sebagai cadangan. Kelebihan *backup mode* adalah delay proses *handover* cenderung lebih baik daripada *single-path mode* dan konsumsi energi cenderung lebih rendah daripada *full mode.* Konsekuensi penggunaan *backup mode* ini adalah mekanisme *handover* yang digunakan adalah *make-before-break*.

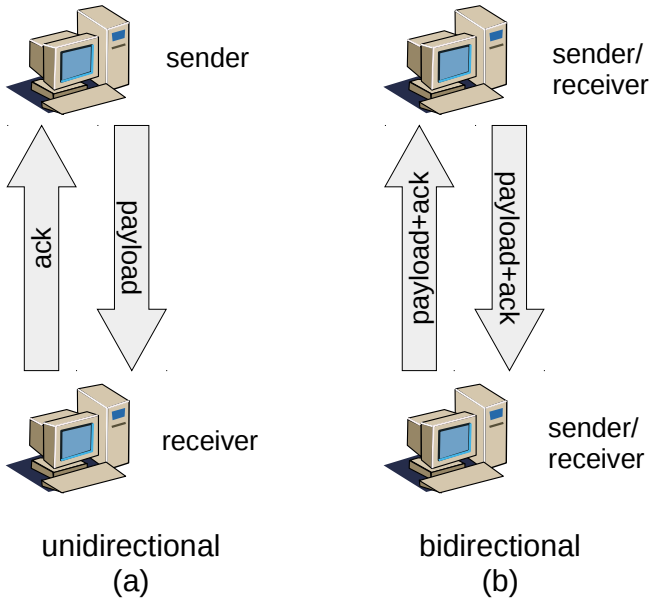


Gambar 1.2: *Timeline vertical handover* pada *backup mode Multipath TCP*

Typical timeline untuk *vertical handover* pada perangkat bergerak menggunakan *backup mode Multipath TCP* dapat dilihat pada Gambar 1.2. Pada kasus ini, ketika performa sebuah *subflow* yang sedang aktif sangat buruk (banyak terjadi *packet loss*) atau bahkan terputus, perangkat bergerak yang mendukung *Multipath TCP* dapat melakukan aktivasi salah satu *subflow* cadangan untuk menggantikan *subflow* yang lama. Ketika *subflow* terputus, performa *handover* pada kasus ini sangat bergantung pada kecepatan perangkat bergerak dalam mendeteksi terputusnya *subflow* tersebut.

Berdasarkan arah aliran *payload*, *traffic* internet dapat dibagi menjadi dua yaitu satu arah (*unidirectional*, Gambar 1.3a) dan dua arah (*bidirectional*, Gambar 1.3b). Pada *traffic* satu arah salah satu *host* berperan sebagai *sender* (pengirim *payload*) dan *host* lainnya sebagai *receiver* (penerima *payload* dan

pengirim *acknowledgement*), sedangkan pada *traffic* dua arah kedua *host* berperan sebagai *sender* sekaligus *receiver*. Penelitian yang dilakukan Sinha dkk (Sinha dkk, 2005) menunjukkan bahwa sebagian besar *traffic* internet adalah satu arah (*unidirectional*).



Gambar 1.3: Arah aliran *traffic* internet

Retransmission Timer merupakan mekanisme standar pada *TCP* untuk mendeteksi *packet loss* atau terputusnya koneksi, akan tetapi mekanisme ini hanya dapat digunakan oleh *host sender*. *Host receiver* tidak dapat menggunakan mekanisme ini untuk mendeteksi *packet loss* atau terputusnya koneksi karena *host receiver* tidak mengetahui adanya paket yang akan datang kemudian.

Hal yang sama juga berlaku pada proses *handover* perangkat bergerak yang menggunakan *Multipath TCP*. Apabila perangkat bergerak berperan sebagai *receiver*, performa deteksi terputusnya *subflow* bisa sangat buruk karena perangkat bergerak tidak mengetahui adanya paket yang masih akan diterima. Pada *single-path mode* keadaan ini bertambah rumit apabila bergerak berada di belakang *NAT* sehingga hanya perangkat bergerak yang dapat melakukan inisiasi *subflow* baru. *Server* (*sender*) tidak dapat melakukan inisiasi *subflow* baru

meskipun *server* dapat mendeteksi terputusnya *subflow* sehingga performa *handover* akan terganggu. Kondisi diatas menunjukkan dibutuhkan mekanisme bagi *receiver* untuk mendeteksi terputusnya *subflow*.

Pada penelitian ini diajukan metode bagi *host receiver* untuk mendeteksi terputusnya *subflow* pada sebuah koneksi *Multipath TCP* berdasarkan *packet-interarrival timeout*. Pada metode ini *receiver* menerima informasi adanya paket yang masih akan diterima, sehingga dapat memperhitungkan *timeout* untuk paket tersebut berdasarkan interval waktu antar paket yang telah diterima. Dengan metode tersebut diharapkan waktu yang dibutuhkan *receiver* untuk mendeteksi terputusnya *subflow* dapat dikurangi sehingga performa *handover* dapat ditingkatkan.

1.2 Rumusan Masalah

Penelitian ini akan menyajikan solusi untuk masalah-masalah sebagai berikut:

1. Bagaimana rancangan perubahan protokol *Multipath TCP* untuk meningkatkan performa *vertical handover* di *transport layer* dengan cara mengurangi *delay* pada proses deteksi *subflow* yang terputus menggunakan mekanisme *timeout* berdasarkan *packet inter-arrival time* di sisi *receiver*.
2. Bagaimana pengaruh *latency*, *Maximum Segment Size*, *bandwidth*, *jitter* dan *packet loss* pada jaringan terhadap performa metode yang diajukan dalam mendeteksi terputusnya *subflow* di sisi *receiver* yaitu dari segi *delay* dan akurasi deteksi terputusnya *subflow*.

1.3 Tujuan

Penelitian ini bertujuan untuk mengajukan sebuah metode untuk meningkatkan performa *vertical handover* di *transport layer* pada perangkat bergerak yang mengimplementasikan *Multipath TCP* dengan cara mengurangi *delay* akibat proses deteksi terputusnya *subflow*. Metode yang disebut *Packet Inter-arrival Timeout* (*PITO*) ini bekerja berdasarkan jeda waktu antar paket yang

diterima oleh *receiver*. Hasil yang diharapkan dari penelitian ini adalah rancangan perubahan protokol *Multipath TCP* yang dibutuhkan untuk mengimplementasikan metode *PITO* tersebut.

1.4 Manfaat Penelitian

Dari hasil penelitian ini, diharapkan performa *vertical handover* pada perangkat bergerak yang mengimplementasikan *Multipath TCP* dapat meningkat sehingga kualitas pengalaman pengguna (*user experience*) menjadi lebih baik.

1.5 Kontribusi Penelitian

Kondisi saat ini adalah belum ada mekanisme bagi sebuah *host receiver* untuk mendeteksi adanya *subflow* yang terputus pada sebuah koneksi *Multipath TCP*. Kontribusi penelitian ini adalah menghasilkan sebuah metode bagi *host receiver* untuk mendeteksi *subflow* yang terputus menggunakan *Packet Inter-arrival Timeout*.

1.6 Batasan Masalah

Batasan masalah pada penelitian ini adalah:

1. Pengujian dilakukan pada kasus *traffic* satu arah dimana perangkat bergerak berperan sebagai *receiver*.
2. Pengujian dilakukan pada kasus koneksi *Multipath TCP* pada mode *backup*.
3. Pengujian dilakukan menggunakan simulator *NS2*.

[Halaman ini sengaja dikosongkan]

BAB 2
KAJIAN PUSTAKA

2.1 Metode Penanganan *Packet Loss* pada *TCP* Standar

Pada bagian ini akan dijelaskan mengenai metode yang terdapat di dalam *TCP* standar yang terkait dengan terjadinya *packet loss*. Yang dimaksud dengan *TCP* standar adalah spesifikasi *TCP* yang terkandung dalam dokumen-dokumen *RFC* yang termasuk dalam kategori *Standard Protocols*, *Draft Standard Protocols*, atau *Proposed Standard Protocols* di dalam dokumen *Internet Official Protocol Standards STD-1* (RFC Editor, 2008).

2.1.1 *Retransmission Timer*

Retransmission Timer merupakan mekanisme pokok dalam *TCP* standar untuk menangani terjadinya *packet loss*. Untuk setiap paket data (*payload*) yang dikirimkan, *host sender* akan menghitung *Retransmission Timeout* untuk paket tersebut, yaitu jangka waktu maksimal diterimanya *acknowledgement* dari *host receiver* untuk paket tersebut. Ketika terjadi *timeout*, *host sender* akan melakukan transmisi ulang paket yang mengalami *timeout*, dan *Retransmission Timer* diulang kembali (*Information Science Institute University of South California*, 1981).

Perhitungan *Retransmission Timeout* (*RTO*) dilakukan berdasarkan *Smoothed Round Trip Time* (*SRTT*) dan *Round Trip Time Variance* (*RTTVAR*). *SRTT* dan *RTTVAR* dihitung berdasarkan *Round Trip Time* (*RTT*) yang terukur dengan perhitungan sebagai berikut sesuai (Paxson dkk, 2011):

1. Untuk *RTT* pertama yang didapat, dilakukan perhitungan sebagai berikut:

$$SRTT = RTT \qquad (2.1)$$

$$RTTVAR = \frac{RTT}{2} \qquad (2.2)$$

$$RTO = SRTT + max(G, 4 * RTTVAR) \qquad (2.3)$$

2. Untuk setiap *RTT* berikutnya, dilakukan perhitungan ulang sebagai berikut:

$$RTTVAR = (1-\beta) * RTTVAR + \beta * |SRTT - RTT| \qquad (2.4)$$

$$SRTT = (1-\alpha) * SRTT + \alpha * RTT \quad (2.5)$$

$$RTO = SRTT + max(G, 4 * RTTVAR) \quad (2.6)$$

dimana α = 1/8, β = 1/4 dan G = *clock granularity* pada sistem terkait.

Retransmission Timeout dapat digunakan oleh *host sender* sebagai dasar untuk mendeteksi adanya kegagalan koneksi (terputusnya *link*), misalnya berdasarkan jumlah transmisi ulang untuk paket yang sama. RFC 1122 (*Internet Engineering Task Force*, 1989) mensyaratkan penggunaan dua buah *threshold*, yaitu R1 dan R2, untuk membatasi jumlah transmisi ulang paket yang sama. R1 adalah batas waktu atau jumlah transmisi ulang secara mandiri oleh *TCP* sebelum meminta *network layer* (*IP*) untuk memeriksa kondisi *path* yang digunakan. R2 yang lebih besar dari R1 digunakan sebagai batas akhir waktu atau transmisi ulang sebelum koneksi ditutup secara paksa. Nilai minimal yang disarankan untuk R1 adalah 3 kali transmisi dan R2 adalah 100ms.

2.1.2 *Duplicate ACK*

Pada kondisi normal, setiap menerima paket data yang mengandung *sequence number* yang sesuai dengan batas kiri *receive window*, *host receiver* akan memajukan batas kiri *receive window* dan mengirimkan paket *ACK* dengan nilai sesuai batas kiri *receive window* yang baru. Ketika sebuah *host receiver* menerima paket yang tidak sesuai batas kiri pada *receive window*, *host* tersebut akan mengirimkan *ACK* dengan nilai yang sama (*duplicate ACK*) dengan *ACK* terakhir yang ia kirimkan sebelumnya. *Host sender* yang menerima 3 *duplicate ACK* akan menafsirkan hal tersebut sebagai indikasi satu buah paket yang hilang dan akan menjalankan algoritma *fast retransmit* (Allman dkk, 2009).

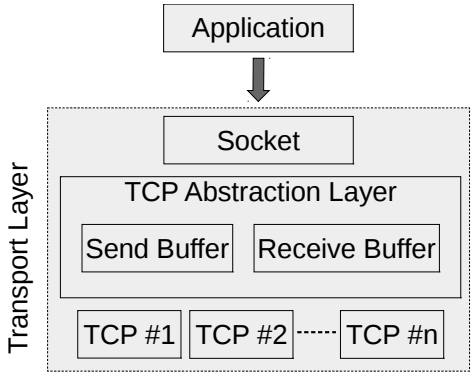
2.1.3 *Selective Acknowledgement*

Selective Acknowledgement (*SACK*) adalah mekanisme yang dapat digunakan oleh *host receiver* untuk menginformasikan adanya blok-blok data yang diterima oleh *receiver* namun dalam keadaan tidak kontinyu. *SACK* digunakan untuk melengkapi informasi pada *duplicate ACK* seperti dijelaskan diatas. Ketika *host receiver* akan mengirimkan *duplicate ACK* karena menerima paket dengan *sequence number* yang tidak sama dengan batas kiri *receive window* yang ia

miliki, *SACK option* dapat digunakan untuk memasukkan informasi batas kiri dan batas kanan masing-masing blok yang sudah diterima di dalam *receive window* tersebut (Mathis dkk, 1996).

2.2 *Multipath TCP*

Multipath TCP adalah pengembangan *TCP* yang memungkinkan bagi sebuah *host* menggunakan beberapa koneksi *TCP* sekaligus untuk terhubung dengan *host* lainnya (Ford dkk, 2013). Sebuah koneksi *Multipath TCP* terdiri atas satu atau lebih *subflow*, dimana setiap *subflow* merupakan sebuah koneksi *TCP* tersendiri. *Multipath TCP* mengunakan sebuah *send buffer* dan sebuah *receive buffer* yang digunakan bersama-sama oleh seluruh *subflow* yang ada (Barre dkk, 2011). Hal ini menghasilkan sebuah abstraksi kepada aplikasi atas *subflow* yang ada. Arsitektur *Multipath TCP* dapat dilihat pada Gambar 2.1.



Gambar 2.1: Arsitektur *Multipath TCP*

2.2.1 Inisiasi Koneksi *Multipath TCP* Baru

Sebuah koneksi *Multipath TCP* dimulai dengan menambahkan *TCP option MP_CAPABLE* pada paket *SYN* yang dikirimkan saat *3-way handshake* untuk *subflow* yang pertama. *MP_CAPABLE option* tersebut berisi *key* masing-masing *host* dan satu atau lebih *flag* sebagai informasi tambahan.

2.2.2 Pengiriman Informasi *Address ID* Alternatif yang Dapat Digunakan

Dalam *Multipath TCP*, *Address ID* digunakan untuk merepresentasikan sebuah *IP Address* pada sebuah *host*. Dengan menggunakan *Address ID* ini, proses manajemen *subflow* (misalnya penghapusan *subflow* atau pengubahan tingkat prioritas *subflow*) dapat dilakukan meskipun pada jaringan terdapat perangkat *Network Address Translation*.

Sebuah *host* dapat memberikan informasi *Address ID* lain yang dimilikinya kepada *host* lain untuk digunakan dalam membangun *subflow* baru. Hal ini dilakukan dengan mengirimkan *TCP option ADD_ADDR* yang berisi *IP Address* dan *Address ID* melalui *subflow* yang sudah dibuat.

2.2.3 Inisiasi *Subflow* Baru

Untuk menambahkan *subflow* baru ke dalam koneksi *Multipath TCP*, digunakan *token* yang dibangun berdasarkan *key* yang telah dipertukarkan pada saat inisiasi koneksi *Multipath TCP*. *Token* tersebut digunakan untuk mengidentifikasi koneksi *Multipath TCP* yang akan diasosiasikan dengan *subflow* yang akan dibangun ini. *Token* tersebut dikirimkan dalam sebuah *TCP option MP_JOIN* bersama dengan beberapa parameter lain misalnya *Address ID* sehingga *MP_*JOIN ini dapat digunakan untuk menginformasikan *Address ID* secara implisit kepada *host* lain.

Mekanisme inisiasi *subflow* baru seperti dijelaskan diatas membuat inisiasi *subflow* baru tidak perlu melibatkan *subflow* lain. Proses inisiasi *subflow* yang bersifat independen ini membuat *Multipath TCP* dapat dimanfaatkan untuk melakukan *handover* secara *break-before-make*.

2.2.4 Penghapusan *Subflow*

Sebuah *subflow* dapat dihapus dengan cara mengirimkan *TCP option REMOVE_ADDR* yang berisi salah satu *Address ID* milik *host* pengirim. Konsekuensi dari hal ini adalah seluruh *subflow* yang menggunakan *Address ID* tersebut akan dihapus dari koneksi *Multipath TCP* terkait.

Mekanisme penghapusan *subflow* ini juga dapat bermanfaat pada proses *handover*

untuk memberi informasi pada *host sender* agar berhenti mengirimkan data melalui *subflow* yang terputus.

2.2.5 Pengubahan Tingkat Prioritas *Subflow*

Pada saat inisiasi, sebuah *subflow* dapat diatur agar langsung menjadi *subflow* aktif atau menjadi *subflow* cadangan dengan memanfaatkan sebuah *flag* yang dikirimkan dalam *TCP option MP_CAPABLE* atau *MP_JOIN*. Selain itu, *host* dapat mengirimkan *TCP option MP_PRIO* untuk mengubah sebuah *subflow* dari aktif menjadi cadangan atau sebaliknya.

2.3 Penelitian dan Referensi Terkait

2.3.1 Distribusi Ukuran Paket *Internet*

Sinha dkk pada (Sinha dkk, 2005) telah menunjukkan bahwa terdapat porsi yang besar dari *traffic internet* yang berisi paket data berukuran kecil (sekitar 40 *byte*) yaitu sekitar 30%. Dari data ini dapat disimpulkan bahwa lebih dari 50% *traffic internet* bersifat satu arah dimana sebuah *host* yang terhubung dengan *internet* berperan sebagai *sender* (pengirim *payload*) dan *host* yang lain berperan sebagai *receiver* (penerima *payload* dan pengirim *acknowledgement*).

2.3.2 Interaksi *TCP* dengan *Middleboxes*

RFC1122 (*Internet Engineering Task Force*, 1989) menspesifikasikan bahwa sebuah perangkat yang mendukung *TCP* harus tidak menghiraukan *TCP option* yang tidak diimplementasikan olehnya. Hal ini memberikan kesempatan untuk menggunakan *TCP option* baru dalam metode yang diajukan dalam penelitian ini. Penelitian yang dilakukan Medina dkk (Medina dkk, 2004) juga memberikan indikasi bahwa penggunaan *TCP option* di luar standar yang ada tidak memberikan efek negatif yang berarti pada sebuah koneksi *TCP*.

2.3.3 Dukungan Mobilitas pada *Multipath TCP*

Paasch dkk (Paasch dkk, 2012) telah melakukan eksperimen mengenai *handover* pada perangkat bergerak menggunakan *Multipath TCP*. Pada penelitian tersebut, percobaan *handover* dilakukan dengan menggunakan *smartphone* yang memiliki

interface jaringan *3G* dan *WLAN*. Dalam proses pengujian, *interface WLAN* pada *smartphone* tersebut terhubung dengan sebuah *ADSL router* yang terhubung dengan *internet* dengan *latency* 30*ms*. Emulasi proses *handover* dilakukan dengan mematikan *interface WLAN* pada *ADSL router* yang digunakan untuk terhubung ke *internet*. Pada seluruh kasus pengujian, terlihat bahwa dibutuhkan waktu yang cukup lama (sekitar 200*ms*) untuk bagi *Multipath TCP* untuk melakukan *recovery* dari kegagalan jaringan *WLAN* dengan beralih menggunakan jaringan *3G*.

BAB 3
METODE PENELITIAN

Pada bab ini akan dipaparkan metode yang digunakan pada penelitian ini, yaitu 1) Studi Literatur, 2) Penelitian, dan 3) Dokumentasi. Alur kerja metode yang digunakan dapat dilihat pada Gambar 3.1.

Gambar 3.1: Alur kerja metode penelitian

3.1 Studi Literatur

Pada awal penelitian dilakukan studi pada literatur-literatur yang terkait dengan penanganan *packet loss* pada *TCP* standar, protokol *Multipath TCP*, implementasi dan simulator yang tersedia untuk *Multipath TCP*, distribusi ukuran paket di internet, interaksi *TCP* dengan *middleboxes*, dan penelitian-penelitian terkait dukungan mobilitas pada *Multipath TCP*. Sumber yang digunakan dalam penelitian ini adalah dokumen standar, kode sumber program, publikasi konferensi, dan jurnal.

Dari studi yang dilakukan didapatkan informasi-informasi penting sebagai berikut:

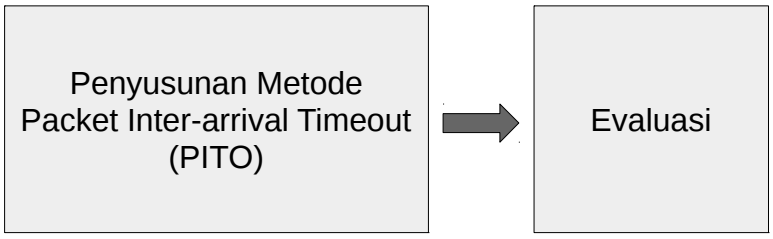
1. *RFC 1122* menyebutkan bahwa perangkat internet wajib membiarkan *TCP option* yang tidak diimplementasikan. Hal ini membuka peluang digunakannya *TCP option* baru (diluar standar yang ada) pada metode yang diajukan.
2. Berdasarkan penelitian-penelitian mengenai distribusi ukuran paket di internet, dapat disimpulkan bahwa pada setiap koneksi *TCP* sering terjadi transfer data satu arah (*unidirectional*) dimana salah satu pihak yang terhubung berperan sebagai *sender* (pengirim *payload*) dan pihak yang lain berperan sebagai *receiver* (pengirim *ACK* murni tanpa *payload*). Hal ini

berbeda dengan transfer data dua arah (*bidirectional*) dimana kedua pihak berperan sebagai *sender* sekaligus *receiver*.

3. *Retransmission Timer* dapat dimanfaatkan sebagai mekanisme pokok bagi *sender* pada sebuah koneksi *TCP* untuk mendeteksi terputusnya koneksi.
4. Belum ada mekanisme dalam TCP yang dapat dimanfaatkan *receiver* untuk mendeteksi terputusnya koneksi.
5. *Retransmission Timer* digunakan oleh *sender* pada *Multipath TCP* sebagai bahan evaluasi melakukan *retransmission* melalui *subflow* lain, dengan kata lain *Retransmission* Timer dapat dipergunakan oleh *sender* untuk mendeteksi terputusnya *subflow*.

3.2 Penelitian

Tahap ini terdiri atas dua fase yaitu 1) penyusunan metode *Packet Inter-arrival Timeout (PITO)* 2) evaluasi. Alur kerja pada tahap ini dapat dilihat pada Gambar 3.2.



Gambar 3.2: Alur kerja tahap penelitian

3.2.1 Penyusunan Metode *Packet Inter-arrival Timeout (PITO)*

Berikut ini adalah desain dasar metode yang diajukan:

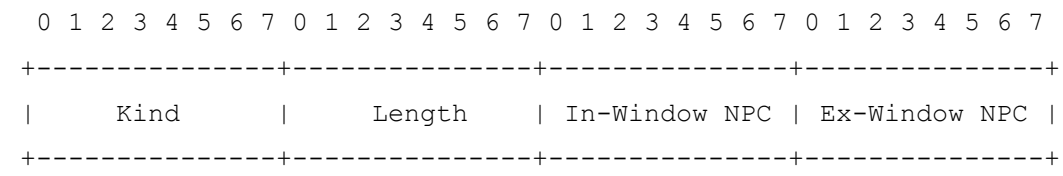
1. Bagi *receiver*, untuk mendeteksi terputusnya *subflow* dibutuhkan informasi mengenai adanya paket yang akan diterima di waktu yang akan datang.
2. Dalam setiap paket yang akan dikirimkan, *sender* dapat menyisipkan informasi dalam bentuk *TCP option* berapa jumlah paket yang akan dikirimkan setelah paket tersebut (*next packets count*). Informasi yang disisipkan ada dua yaitu *In-Window Next Packets Count (IW NPC)* dan *Ex-*

Window Next Packets Count (EW NPC). *In-Window Next Packets Count* merupakan informasi berapa banyak paket dalam *congestion window* pada *host sender* yang masih akan dikirimkan, sedangkan *Ex-Window Next Packets Count* merupakan informasi berapa banyak paket di luar *congestion window* pada *host sender* (*send buffer*) yang akan dikirimkan.

3. Setiap kali *receiver* menerima paket yang berurutan dari *sender*, *receiver* akan menghitung *timeout* untuk paket berikutnya berdasarkan informasi *IW NPC* dan *EW NPC* yang diterimanya, dan mencatat jumlah paket yang akan diterima berikutnya (apabila diperlukan).
4. Apabila tidak ada paket yang diterima *receiver* hingga terjadi timeout, maka *receiver* dapat melakukan aksi sesuai *policy* yang dimiliki. Contoh aksi yang dapat dilakukan misalnya mengaktifkan *subflow backup*, membuat *subflow* baru, mengirimkan sinyal *REMOVE_ADDR* untuk menghapus *subflow* yang mengalami *timeout*, dan lain-lain.

3.2.1.1 *Format TCP Option Next Packets Count*

Format TCP option untuk mengirimkan informasi *next packets count* dapat dilihat pada Gambar 3.3. Field *In-Window NPC* berisi informasi *In-Window Next Packets Count (IW NPC)* dan field *Ex-Window NPC* berisi informasi *Ex-Window Next Packets Count (EW NPC)*.



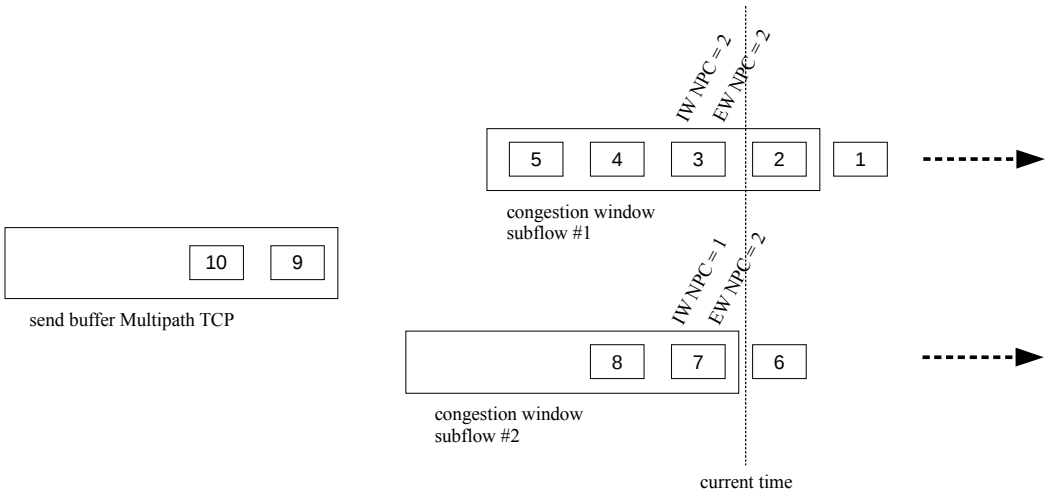
Gambar 3.3: *Format TCP option Next Packets Count*

3.2.1.2 Perhitungan *Next Packets Count*

In-Window Next Packets Count (IW NPC) diisi berdasarkan jumlah paket dalam *congestion window* pada *subflow* yang bersangkutan yang masih akan dikirimkan. Oleh karena itu *In-Window Next Packets Count* pasti terurut dari besar lalu

mengecil dan akhirnya menjadi nol, misalnya 8, 7, 6, 5, 4, 3, 2, 1, 0 .

Barre dkk pada (Barre dkk, 2011) menyebutkan bahwa *Multipath TCP* menggunakan satu buah *send buffer* yang digunakan bersama-sama oleh seluruh *subflow* yang ada. Selain itu pada penelitian yang sama disebutkan bahwa *MSS* yang digunakan pada impelementasi *Multipath TCP* dibuat sama untuk seluruh *subflow*, yaitu *MSS* yang terkecil diantara *subflow-subflow* yang ada. Oleh karena itu jumlah *Ex-Window Next Packets Count (EW NPC)* diambil dari jumlah *byte* dalam *send buffer* yang belum dikirimkan oleh salah satu *subflow* dibagi dengan *MSS* terkecil diantara *subflow-subflow* yang ada. Hal ini bertujuan untuk mengakomodasi kemungkinan penggunaan metode PITO baik pada *single-path mode*, *backup mode* dan *full mode.*



Gambar 3.4: Ilustrasi Perhitungan *Next Packets Count*

Untuk aplikasi *backup mode* dan *single-path mode*, hanya ada satu *subflow* yang aktif pada setiap saat, sehingga jumlah *Ex-Window Next Packets Count* yang dikirimkan akan terurut dari besar lalu mengecil dan akhirnya menjadi nol, misalnya 8, 7, 6, 5, 4, 3, 2, 1, 0.

Untuk *full mode*, ada kemungkinan terdapat jarak pada jumlah *Ex-Window Next Packets Count* yang dikirimkan. Sebagai contoh pada *subflow* A *Ex-Window Next Packets Count* dapat bernilai 8, 6, 4, 2, 0, dan pada *subflow* B *Ex-Window Next*

Packets Count dapat bernilai 7, 5, 3, 1, 0. Apabila pada suatu saat sebuah *subflow* menemui bahwa tidak ada data yang dapat dikirimkan, padahal sebelumnya nilai *Ex-Window Next Packets Count* yang dikirimkan melalui *subflow* tersebut lebih dari 0, maka pada *subflow* tersebut dikirimkan sebuah *Duplicate ACK* atau paket *TCP Keep Alive* yang disertai *TCP option Next Packets Count* dengan *In-Window Next Packets Count* bernilai 0. Hal ini dilakukan untuk mencegah *receiver* memberlakukan *packet inter-arrival timeout* pada *subflow* tersebut. Ilustrasi perhitungan *Next Packets Count* ini disajikan pada Gambar 3.4.

3.2.1.3 Perhitungan *Packet Inter-arrival Timeout (PITO)*

Perhitungan *Packet Inter-arrival Timeout (PITO)* diadaptasi dari perhitungan *Retransmission Timeout* pada (Paxson dkk, 2011) yang dimodifikasi. Perhitungan *PITO* dilakukan berdasarkan *Smoothed Packet Inter-arrival Time (SPIT)* dan *Packet Inter-arrival Time Variance (PITVAR)*. *SPIT* dan *PITVAR* dihitung berdasarkan *Packet Inter-arrival Time (PIT)* yang terukur.

Terdapat dua macam *timeout* dalam metode *PITO* yaitu *In-Window timeout* dan *Ex-Window timeout*. Kedua jenis *timeout* tersebut dihitung secara terpisah namun menggunakan perhitungan yang serupa yaitu sebagai berikut:

1. Apabila belum ada data *Packet Inter-arrival Time (PIT)* yang didapat, informasi *retransmission timeout* yang dimiliki TCP digunakan sementara hingga didapatkan informasi *Packet Inter-arrival Time.*
2. Untuk *Packet Inter-arrival Time* pertama yang didapat, dilakukan perhitungan sebagai berikut (berlaku untuk *In-Window timeout* maupun *Ex-Window timeout*):

$$SPIT = PIT \qquad (3.1)$$

$$PITVAR = PIT/2 \qquad (3.2)$$

$$PITO = \gamma * (SPIT + (4 * PITVAR)) \qquad (3.3)$$

3. Setiap kali *receiver* menerima paket dengan *sequence number* yang berurutan dengan paket yang diterima sebelumnya serta memenuhi syarat nilai *In-Window NPC* pada paket sebelumnya > 0 atau nilai PIT yang baru didapatkan

lebih kecil dari $PITO_{INW}/2$ maka dilakukan perhitungan ulang *In-Window timeout* sebagai berikut:

$$PITVAR_{INW} = (1-\beta) * PITVAR_{INW} + \beta * |SPIT_{INW} - PIT| \quad (3.4)$$

$$SPIT_{INW} = (1-\alpha) * SPIT_{INW} + \alpha * PIT \quad (3.5)$$

$$PITO_{INW} = \gamma * (SPIT_{INW} + (4 * PITVAR_{INW})) \quad (3.6)$$

dimana $\alpha = 1/32$, $\beta = 1/16$, dan $\gamma = 4$.

4. Setiap kali *receiver* menerima paket dengan *sequence number* yang berurutan dengan paket yang diterima sebelumnya serta memenuhi syarat nilai *In-Window NPC* pada paket sebelumnya = 0 dan nilai PIT yang baru didapatkan lebih besar dari $PITO_{INW}/2$ maka dilakukan perhitungan ulang *Ex-Window timeout* sebagai berikut:

$$PITVAR_{EXW} = (1-\beta) * PITVAR_{EXW} + \beta * |SPIT_{EXW} - PIT| \quad (3.7)$$

$$SPIT_{EXW} = (1-\alpha) * SPIT_{EXW} + \alpha * PIT_{EXW} \quad (3.8)$$

$$PITO_{EXW} = \gamma * (SPIT_{EXW} + (4 * PITVAR_{EXW})) \quad (3.9)$$

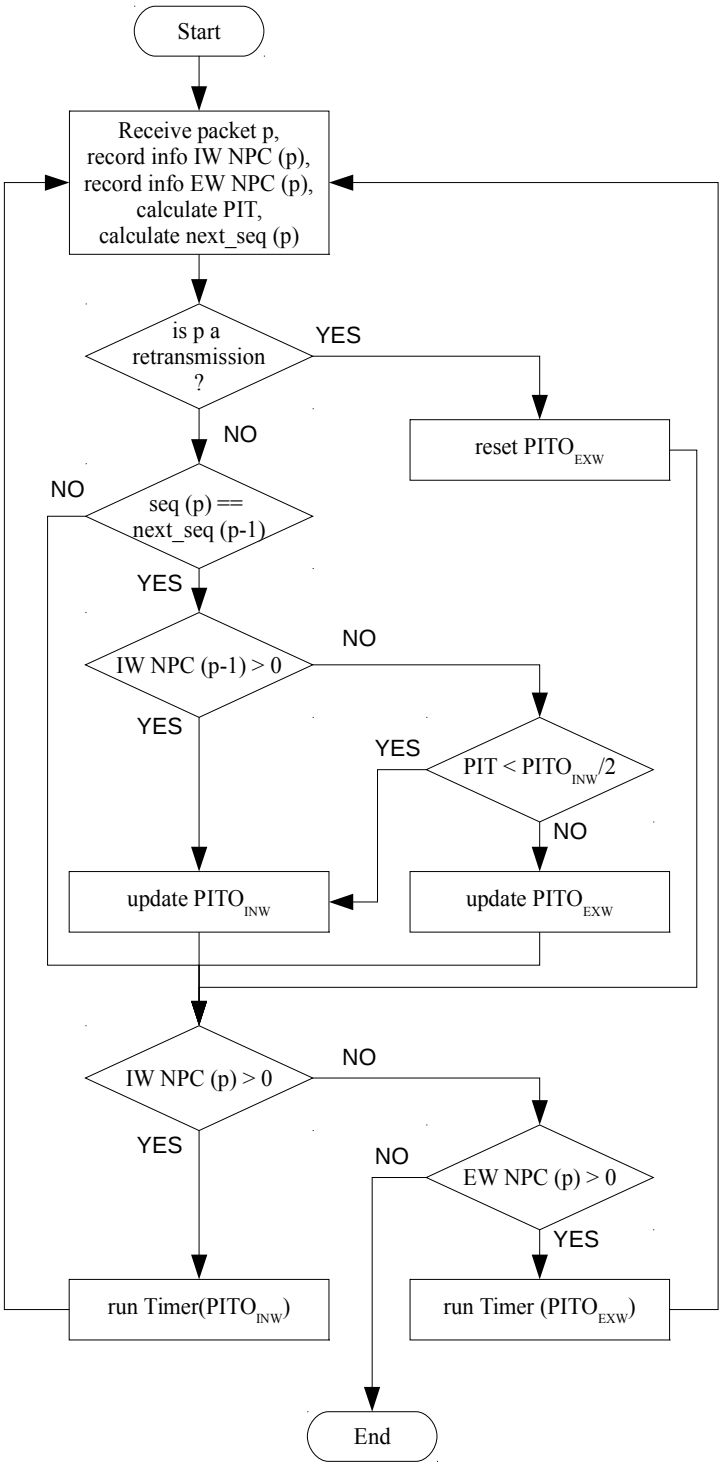
dimana $\alpha = 1/64$, $\beta = 1/32$, dan $\gamma = 1, 1.1, 1.2, \ldots 4$. Penambahan nilai γ dilakukan apabila $PITO_{EXW} < PITO_{INW}$.

5. Apabila paket yang diterima memiliki sequence number yang melompat atau merupakan awal dari rangkaian paket retransmisi (sehingga tidak terurut) maka tidak dilakukan perhitungan ulang (dilewatkan).
6. Apabila paket yang diterima merupakan retransmisi (karena *retransmission timeout* ataupun *fast restransmit*) maka *Ex-Window timeout* dihitung ulang menggunakan *PIT* pertama yang didapat pada langkah 2 diatas.
7. Apabila paket yang diterima saat ini memiliki *In-Window NPC* = 0 maka $PITO_{EXW}$ digunakan sebagai timer. Sebaliknya apabila nilai *In-Window NPC* > 0 maka $PITO_{INW}$ digunakan sebagai timer.

Diagram alir untuk algoritma diatas disajikan pada Gambar dengan keterangan sebagai berikut

- IW NPC (p) dan EW NPC (p) : informasi IW NPC di paket p
- p-1 : paket yang diterima sebelum p
- seq (p) : sequence number pada paket p

- next_seq (p) : sequence number paket setelah p



Gambar 3.5: Diagram Alir Metode *PITO* pada *Host Receiver*

3.2.2 Evaluasi

Evaluasi dilakukan dalam beberapa fase, yaitu:

1. Pembuatan simulasi *vertical handover menggunakan* NS2 untuk aliran *traffic unidirectional* dimana *server* berperan sebagai *sender* dan perangkat bergerak berperan sebagai *receiver* pada aplikasi *backup mode* dimana terdapat lebih dari satu *subflow* namun hanya ada satu *subflow* saja yang aktif (digunakan untuk melakukan *transfer data*) pada setiap waktu sementara *subflow* lain berfungsi sebagai cadangan. Pembuatan simulasi dilakukan berdasarkan protokol *Multipath TCP* yang dilengkapi metode *PITO*.
2. Analisis dan evaluasi pengaruh parameter-parameter dalam jaringan seperti *latency/round-trip-time*, *Maximum Segment Size*, *bandwidth*, *jitter*, dan *packet loss* terhadap performa metode *PITO*. Kemudian dilakukan seleksi nilai parameter yang akan digunakan dalam proses pengujian di fase selanjutnya berdasarkan hasil analisis dan evaluasi tersebut.
3. Pengukuran performa metode *PITO* dalam mendeteksi terputusnya *subflow* untuk kasus diatas berdasarkan (1) *timeout delay*, yaitu jeda waktu antara paket terakhir yang diterima oleh *receiver* pada subflow yang sedang aktif hingga terjadinya *timeout* pada *receiver*, dan (2) akurasi metode *PITO* dalam mendeteksi terputusnya *subflow*. Tingkat akurasi tersebut dinyatakan dalam bentuk *precision* yang dihitung menggunakan rumus sebagai berikut:

$$precision = \frac{TP}{TP + FP} \qquad (3.10)$$

dimana:

- TP adalah banyaknya *vertical handover* yang dilakukan ketika *subflow* benar-benar terputus.
- FP adalah banyaknya *vertical handover* yang dilakukan sebelum *subflow* benar-benar terputus.

Pengukuran setiap metrik diatas dilakukan menggunakan simulasi yang telah dijelaskan sebelumnya dengan parameter jaringan yang dihasilkan dari proses seleksi pada fase sebelumnya.

3.3 Dokumentasi

Pada tahap ini dibuat laporan hasil penelitian yang dilakukan. Tujuan dari pembuatan dokumentasi ini adalah untuk menyajikan metode yang diusulkan beserta hasil evaluasi dalam bentuk tertulis.

Jadwal kerja untuk penelitian ini disajikan pada Tabel 3.1.

Tabel 3.1: Jadwal aktivitas penelitian

Aktivitas	Bulan (Tahun 2016)																	
	Agustus				September				Oktober				November				Desember			
Studi Literatur			■	■	■	■	■	■	■	■	■	■								
Penyusunan Metode *Packet Inter-arrival Timeout*							■	■	■	■	■	■	■	■						
Evaluasi											■	■	■	■	■	■	■	■		
Dokumentasi											■	■	■	■	■	■	■	■	■	■

[Halaman ini sengaja dikosongkan]

BAB 4
HASIL DAN PEMBAHASAN

Pada bab ini akan dipaparkan implementasi dan pengujian metode *Packet Inter-arrival Timeout (PITO)*. Pengujian dilakukan dengan mengubah parameter-parameter jaringan seperti *latency*, *jitter*, *packet loss*, dan *bandwidth*. Evaluasi terhadap hasil pengujian disajikan pada bagian pembahasan di akhir bab.

4.1 Pembuatan Lingkungan Pengujian

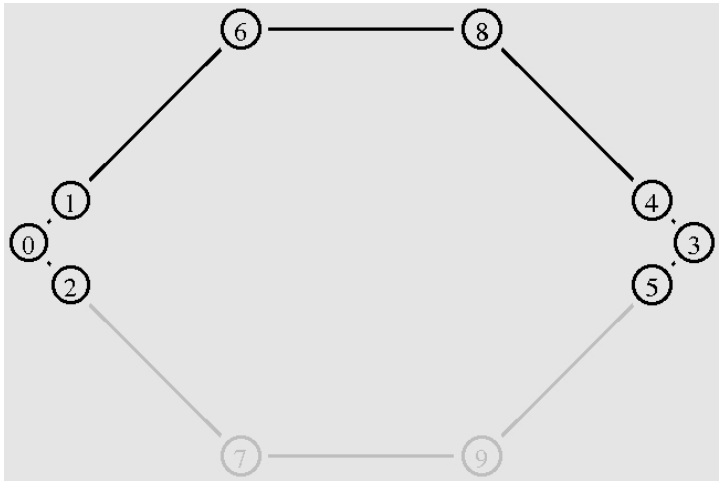
Implementasi metode *PITO* dilakukan menggunakan *simulator NS2* yang telah dilengkapi protokol *Multipath TCP*. Implementasi dilakukan dalam tiga tahap yaitu 1) Implementasi jaringan di *NS2* 2) Implementasi algoritma metode *PITO* di *NS2* 3) Implementasi parameter jaringan di *NS2*.

4.1.1 Implementasi Jaringan di *NS2*

Pada tahap ini dilakukan perancangan dan implementasi jaringan menggunakan *simulator NS2*. Pada saat penelitian ini dilakukan, impelementasi *Multipath TCP* yang terdapat pada NS2 tidak mendukung skenario *vertical handover* pada jaringan nirkabel, oleh karena itu pada pengujian ini digunakan jaringan kabel dengan beberapa penyesuaian agar dapat merepresentasikan terjadinya *vertical handover* pada jaringan nirkabel menggunakan Multipath TCP. Topologi jaringan yang digunakan dalam pengujian disajikan pada Gambar 4.1.
Node 0 merupakan *host sender* yang terhubung dengan *Node 3* yang merupakan *host receiver*. *Node 0* memiliki dua buah *network interface* yang direpresentasikan sebagai *Node 1* dan *Node 2* pada gambar diatas. *Node 3* juga memiliki dua buah *network interface* yaitu *Node 4* dan *Node 5*.

Jalur berwarna hitam yang menghubungkan *Node 1* dan *Node 4* merupakan jalur yang aktif digunakan untuk transmisi pada saat pengujian, sedangkan jalur yang berwarna abu-abu yang menghubungkan *Node 2* dan *Node 5*

merupakan jalur cadangan/*backup*. Setiap segmen pada jaringan memiliki *bandwidth* yang sama dan *latency* yang sama.



Gambar 4.1: Topologi jaringan dalam pengujian

Agar dapat merepresentasikan skenario *vertical handover* pada jaringan nirkabel, dilakukan beberapa penyesuaian sebagai berikut:

1. Dilakukan implementasi *random packet loss* secara acak untuk merepresentasikan *packet loss* akibat penggunaan media nirkabel.
2. *Node 9* merepresentasikan perangkat *Access Point* pada jaringan nirkabel dengan jangkauan sangat luas. Ketika terjadi *vertical handover* akan dilakukan aktivasi *subflow* yang melewati jalur berwarna abu-abu.
3. *Node 8* merepresentasikan perangkat *Access Point* pada jaringan nirkabel dengan jangkauan terbatas. Setelah periode waktu tertentu, dilakukan pemutusan segmen antara *Node 8* dan *Node 4* untuk merepresentasikan *host receiver* (*Node 3*) berada di luar jangkauan *Access Point Node 8*. Pada saat inilah terjadi *vertical handover* dan dilakukan deaktivasi terhadap *subflow* yang melewati jalur berwarna hitam.

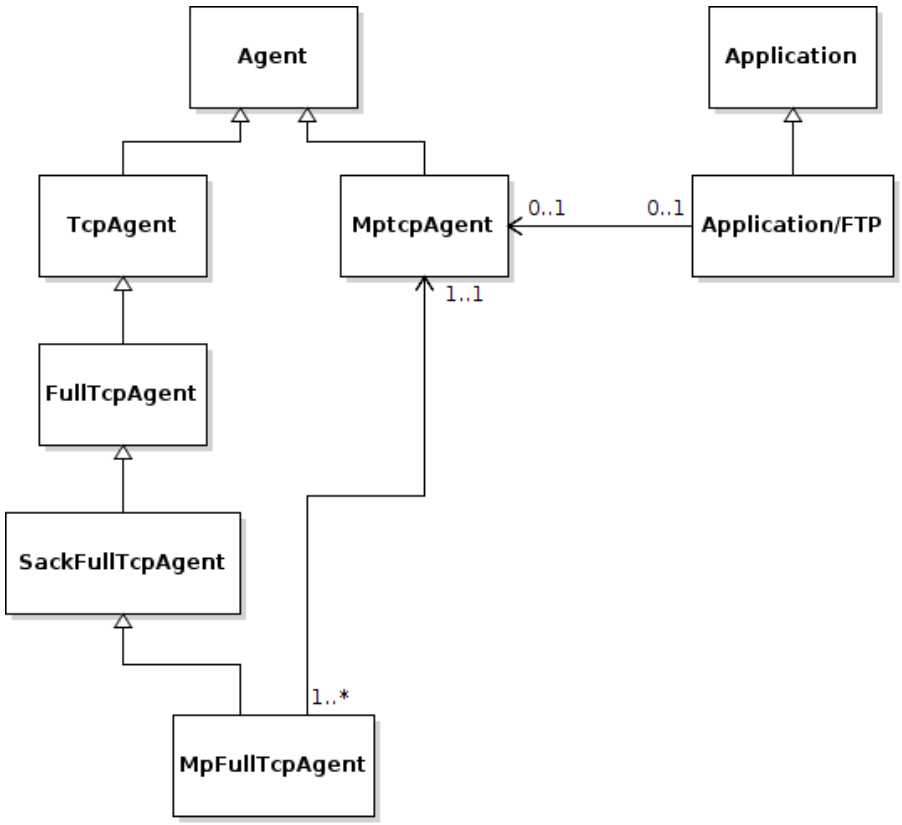
4.1.2 Implementasi Algoritma Metode *PITO* di *NS2*

Pada tahap ini dilakukan implementasi algoritma metode PITO di sisi *host sender*

maupun *host receiver*. Pada *host sender* algoritma yang diimplementasikan adalah algoritma untuk membangun dan menyisipkan *TCP option Next Packets Count*, sedangkan pada *host receiver* algoritma yang diimplementasikan adalah algoritma untuk melakukan perhitungan *In-Window timeout* dan *Ex-Window timeout* serta penerapan *timeout* tersebut menggunakan fasilitas *timer* pada *NS2*.

4.1.2.1 Struktur Kelas

Secara umum struktur kelas yang terlibat dalam metode *PITO* dapat dilihat pada Gambar 4.2.



Gambar 4.2: *Class Diagram* Implementasi Algoritma Metode *PITO* di *NS2*

Pada *NS2* kelas *Agent* merupakan *superclass* seluruh kelas yang mengimplementasikan protokol *TCP*. Kelas *TcpAgent* yang merupakan turunan dari kelas *Agent* mengimplementasikan protokol *TCP* namun hanya bersifat satu

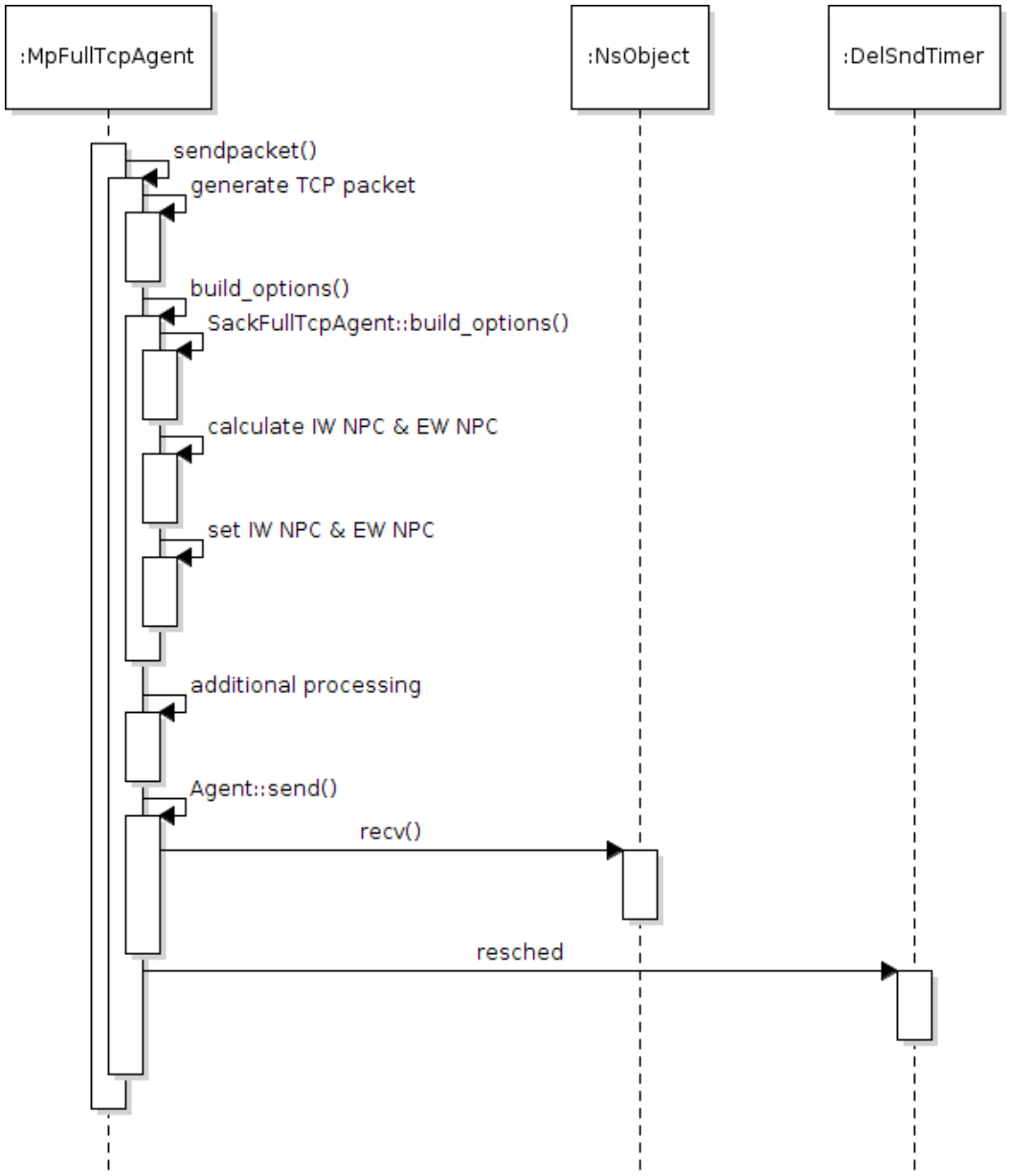
arah. Kelas *FullTcpAgent* merupakan turunan dari kelas *TcpAgent* yang mengimplementasikan protokol *TCP* dua arah. Kelas *SackFullTcpAgent* merupakan turunan dari kelas *FullTcpAgent* yang mengimplementasikan *Selective Acknowledgement*. Kelas *MpFullTcpAgent* merupakan kelas turunan dari kelas *SackFullTcpAgent* yang merepresentasikan *endpoint* dari sebuah *subflow*. Kelas *MptcpAgent* merupakan turunan dari kelas *Agent* dan berfungsi melakukan koordinasi terhadap satu atau lebih *MpFullTcpAgent*. Implementasi protokol *Multipath TCP* terdapat pada kelas *MptcpAgent* dan *MpFullTcpAgent*. Kelas *Application/FTP* berfungsi sebagai penyedia data yang digunakan dalam pengujian.

4.1.2.2 Implementasi Algoritma Metode *PITO* pada *Host Sender*

Dalam pengujian *TCP option Next Packets Count* disisipkan setiap kali *host sender* akan mengirimkan paket data dengan ukuran *payload* > 0 byte. Proses perhitungan dan penyisipan *TCP option Next Packets Count* disajikan pada Gambar 4.3.

Perhitungan dan penyisipan *In-Window Next Packets Count* (*IW NPC*) dan *Ex-Window Next Packets Count* (*EW NPC*) ke dalam paket data dilakukan setiap kali fungsi *sendpacket()* pada kelas *MpFullTcpAgent* dipanggil. Fungsi *sendpacket()* akan melakukan pemanggilan terhadap fungsi *build_options()* yang berperan membangun seluruh *TCP option* yang akan disisipkan ke dalam paket data, termasuk salah satunya adalah *TCP option Next Packets Count*.

Setelah seluruh proses persiapan pengiriman paket data selesai dilakukan, fungsi *sendpacket()* akan memanggil fungsi *recv()* pada object bertipe *NsObject* yang menjadi perantara pengiriman paket menuju *host receiver*. *Object* tersebut biasanya merupakan sebuah *link*. Setelah itu *sendpacket()* akan memanggil fungsi *resched()* pada *object* bertipe *DelSndTimer* untuk mensimulasikan waktu yang dibutuhkan untuk mengirimkan paket data yang disebutkan diatas.

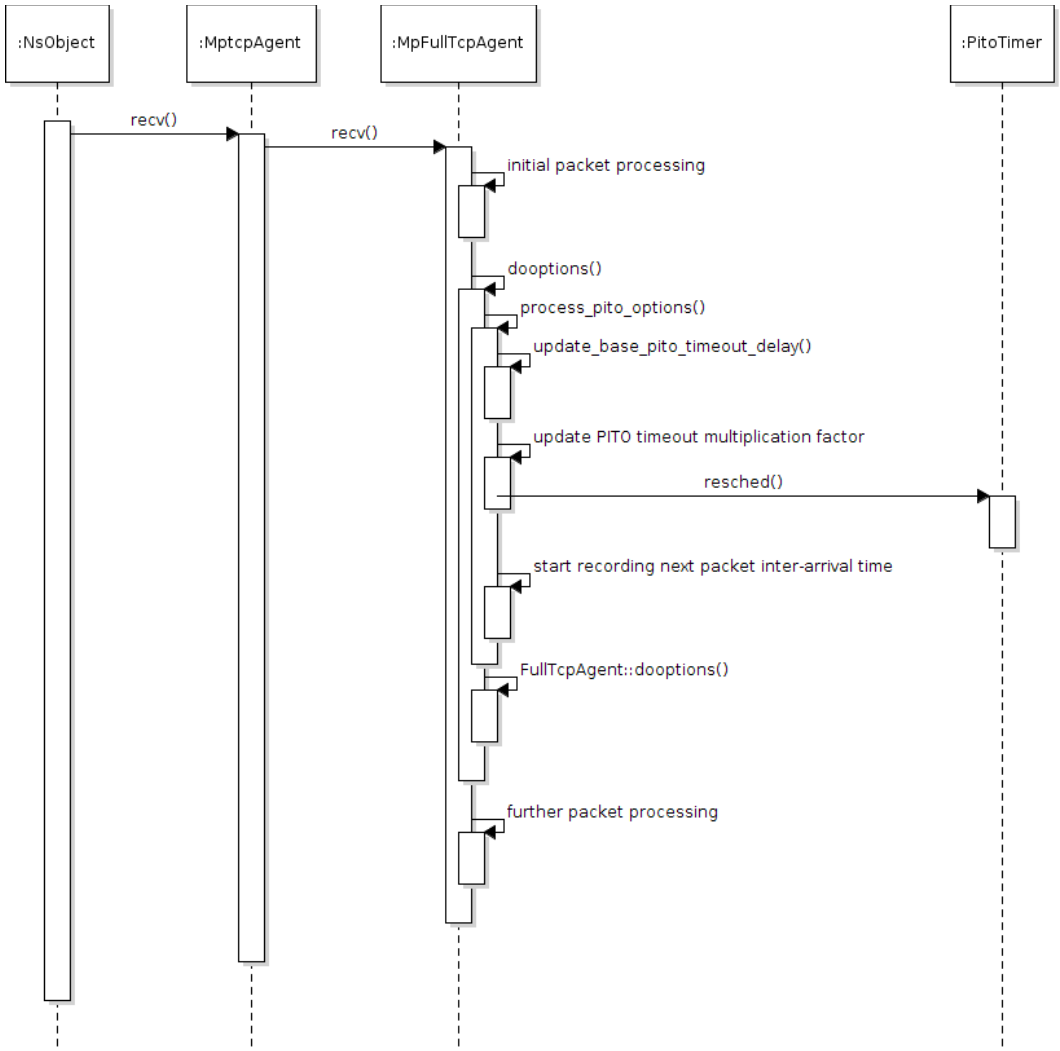


Gambar 4.3: *Sequence Diagram* Penambahan *TCP Option Next Packets Count* pada Paket Data di *Host Sender*

4.1.2.3 Implementasi Algoritma Metode *PITO* pada *Host Receiver*

Di sisi *host receiver*, pemrosesan terhadap paket data dimulai dengan diterimanya paket data melalui pemanggilan fungsi *recv()* yang terdapat di kelas *MptcpAgent*.

Kemudian paket data tersebut disampaikan kepada *object* bertipe *MpFullTcpAgent* melalui fungsi *recv()* yang dimilikinya.



Gambar 4.4: *Sequence Diagram* Pemrosesan *TCP Option Next Packets Count* pada Paket Data di *Host Receiver*

Fungsi *recv()* pada *MpFullTcpAgent* merupakan sebuah fungsi yang kompleks. Namun yang perlu menjadi perhatian adalah bahwa di bagian awal fungsi tersebut terjadi pemanggilan fungsi *dooptions()* yang berfungsi melakukan pemrosesan terhadap seluruh *TCP option* yang terkandung dalam paket data yang diterima. Di

dalam fungsi *dooptions()* ini dilakukan pemanggilan fungsi *process_pito_options()* yang bertugas melakukan pemrosesan terhadap *TCP option Next Packets Count* yang ada di dalam paket data.

Proses yang terjadi di dalam fungsi *process_pito_options()* meliputi perhitungan *In-Window timeout* dan *Ex-Window timeout*, serta pemanggilan fungsi *resched()* pada *object* bertipe *PitoTimer* untuk menjalankan fungsi *timer* yang sebenarnya. Ketika *timer* pada *object* bertipe *PitoTimer* tersebut mengalami *timeout*, maka dikatakan *PITO timeout* telah terjadi dan menandakan dimulainya proses *vertical handover* ke *subflow* cadangan. Seluruh pemrosesan *TCP option Next Packets Count* yang diterima *host receiver* digambarkan pada Gambar 4.4.

4.1.3 Implementasi Algoritma *Vertical Handover* di NS2

Implementasi algoritma *vertical handover* di *NS2* memanfaatkan *object* bertipe *PitoTimer* yang telah dijelaskan diatas. *Object* bertipe *PitoTimer* ini dimiliki oleh *object* bertipe *MpFullTcpAgent* yang berperan menjalankan fungsi-fungsi pada sebuah *subflow Multipath TCP*. Ketika *timer* pada *object* tersebut mengalami *timeout*, *object* bertipe *MpFullTcpAgent* yang memiliki *object timer* tersebut akan mengirimkan infomasi terjadinya *timeout* tersebut pada *object* bertipe *MptcpAgent* yang bertugas melakukan koordinasi terhadap beberapa *object* betipe *MpFullTcpAgent*. *Object* bertipe *MptcpAgent* inilah yang kemudian melakukan aktivasi *subflow* cadangan dan melakukan deaktivasi *subflow* yang mengalami kegagalan.

4.1.4 Implementasi Parameter Jaringan di *NS2*

Tabel 4.1: Daftar kelas yang diubah dan parameter yang diimplementasikan

Kelas	Parameter
LinkDelay	*jitter* dan waktu terputusnya koneksi
DropTail	*packet loss* dan waktu terputusnya koneksi

Sebagian besar parameter jaringan yang akan diuji coba pada penelitian ini sudah didukung oleh *NS2*, diantaranya adalah *Maximum Segment Size*, *latency/round-trip-time*, dan *bandwidth*. Untuk melengkapi fungsionalitas yang dibutuhkan dalam penelitian ini, dilakukan pengubahan kode sumber *NS2* sesuai Tabel 4.1.

4.2 Pengujian

Tahap ini terdiri atas tiga fase yaitu 1) analisis pengaruh parameter jaringan terhadap performa metode *PITO* 2) pengujian performa metode *PITO* 3) evaluasi hasil pengujian metode *PITO*.

4.2.1 Analisis pengaruh parameter jaringan terhadap metode *PITO*

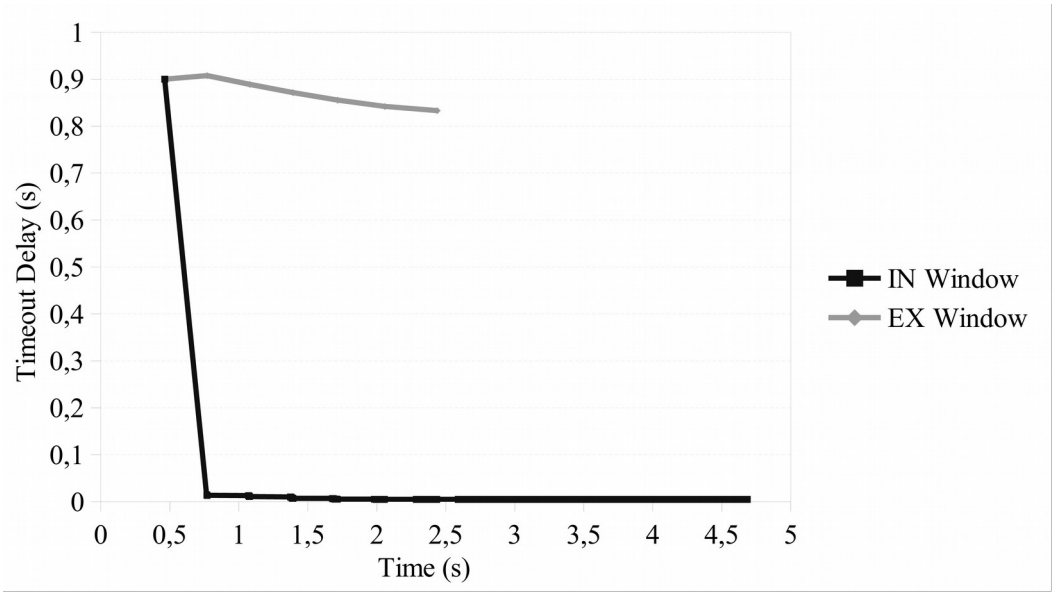
Pada bagian ini dilakukan analisis pengaruh beberapa parameter jaringan terhadap performa metode *PITO*. Parameter-parameter yang dianalisa adalah *latency/round-trip-time*, *Maximum Segment Size*, *bandwidth*, *jitter*, dan *packet loss*.

4.2.1.1 *Latency/Round-Tip-Time*

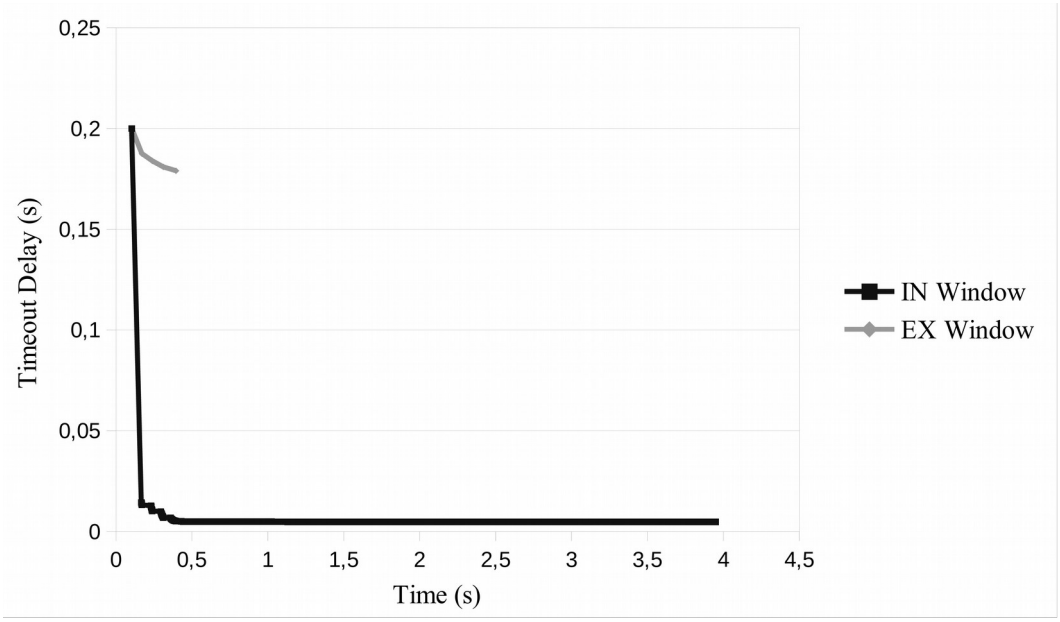
Besarnya nilai *round trip time* (*RTT*) menentukan banyaknya paket yang dibutuhkan untuk memenuhi seluruh segmen di dalam jaringan. Hal ini berarti semakin besar *round tip* time, semakin besar pula *congestion window* yang dibutuhkan untuk mengisi penuh jaringan. Pada awal koneksi, nilai *congestion window* yang kecil mengakibatkan *Packet Inter-arrival Time* yang besar pada setiap akhir *window*. Hal ini berakibat tingginya nilai awal *Ex-Window timeout*. Setelah *congestion window* semakin bertambah besar maka nilai *Ex-Window timeout* menjadi semakin kecil.

Contoh grafik *timeout* yang dihasilkan metode *PITO* pada jaringan dengan *latency* 150ms (*round trip time* 300ms) dapat dilihat pada Gambar 4.5. Sebagai perbandingan, Gambar 4.6 menampilkan contoh grafik *timeout* yang dihasilkan pada *latency* 30ms (*round trip time* 60ms). Parameter yang digunakan untuk

kedua grafik tersebut dapat dilihat pada Tabel 4.2.



Gambar 4.5: *Timeout Delay* pada *latency* 150ms.



Gambar 4.6: *Timeout Delay* pada *latency* 30ms.

Selain nilai awal *Ex-Window timeout*, efek lain yang dihasilkan oleh besarnya nilai *round trip time* adalah besarnya rentang waktu pengguaan *Ex-Window timeout*. Dari kedua gambar dapat dilihat bahwa semakin kecil nilai *latency* maka waktu penggunaan *Ex-Window timeout* semakin kecil. Hal ini disebabkan oleh cepatnya pertumbuhan *congestion window* pada jaringan dengan *latency* yang kecil.

Tabel 4.2: Parameter Jaringan pada Proses Analisis
Pengaruh Parameter *Latency*

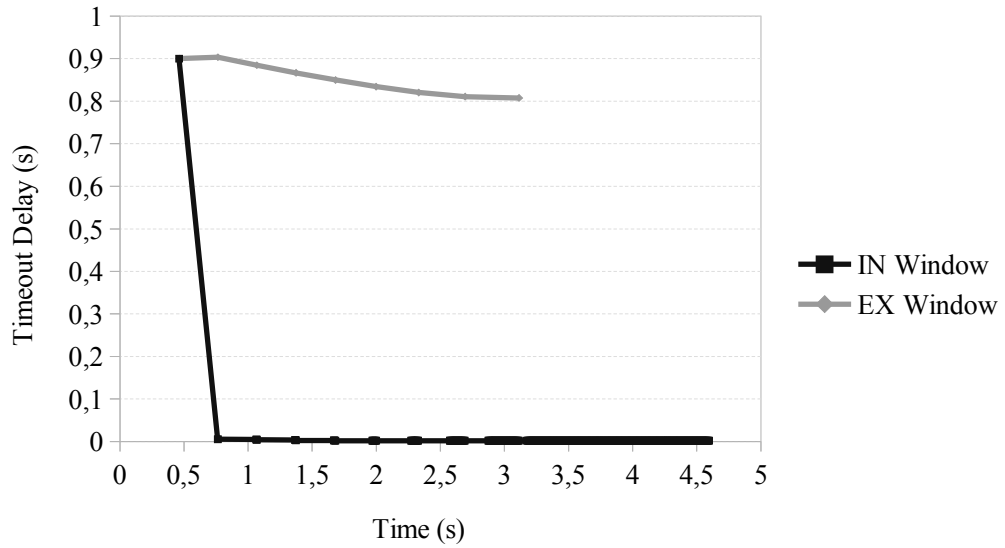
Parameter	**Nilai**
Maximum Segment Size	1440 bytes
Bandwidth	10Mbps
Latency	30ms, 150ms
Maximum Jitter	0ms
Packet Loss	0%

4.2.1.2 *Maximum Segment Size*

Maximum Segment Size (*MSS*) menentukan interval waktu minimal antar paket data. Semakin besar *MSS* maka nilai *In-Window timeout* dan *Ex-Window timeout* terendah yang dapat dicapai akan semakin besar. Di sisi lain semakin besar *MSS* maka efisiensi penggunaan jaringan akan semakin besar (*bandwidth overhead* akibat *packet header* semakin kecil). Keuntungan lain akibat menggunakan *MSS* besar adalah periode penggunaan *Ex-Window timeout* lebih rendah daripada menggunakan *MSS* kecil. Hal ini diakibatkan besarnya nilai *MSS* akan mengurangi ukuran *congestion window* yang dibutuhkan untuk memenuhi jaringan dengan paket data.

Contoh grafik *timeout* yang dihasilkan metode *PITO* pada jaringan dengan *MSS*

536 byte dapat dilihat pada Gambar 4.7. Sebagai perbandingan, Gambar 4.5 menampilkan contoh grafik *timeout* yang dihasilkan *MSS* 1440 byte. Parameter yang digunakan untuk kedua grafik tersebut dapat dilihat pada Tabel 4.3.



Gambar 4.7: *Timeout Delay* dengan *MSS* 536 byte.

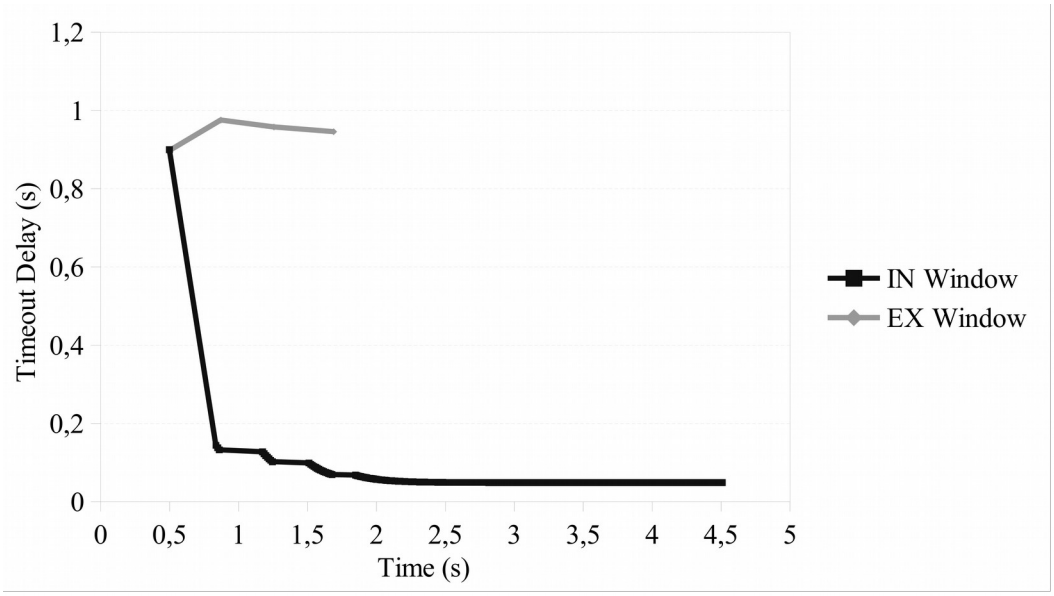
Tabel 4.3: Parameter Jaringan pada Proses Analisis Pengaruh Parameter *MSS*

Parameter	**Nilai**
Maximum Segment Size	536byte,1440byte
Bandwidth	10Mbps
Latency	150ms
Maximum Jitter	0ms
Packet Loss	0%

4.2.1.3 *Bandwidth*

Tabel 4.4: Parameter Jaringan pada Proses Analisis Pengaruh Parameter *Bandwidth*

Parameter	**Nilai**
Maximum Segment Size	1440byte
Bandwidth	10Mbps,1Mbps
Latency	150ms
Maximum Jitter	0ms
Packet Loss	0%



Gambar 4.8: *Timeout Delay* dengan *bandwidth 1Mbps*.

Bandwidth juga menentukan interval waktu minimal antar paket data. Semakin besar *bandwidth* maka nilai *In-Window timeout* dan *Ex-Window timeout* terendah yang dapat dicapai akan semakin kecil. Hal ini berlawanan dengan *Maximum Segment Size* (*MSS*) dimana semakin besar MSS berakibat nilai minimum *In-Window timeout* dan *Ex-Window timeout* terendah yang dapat dicapai akan

semakin besar.

Contoh grafik *timeout* yang dihasilkan metode *PITO* pada jaringan dengan *bandwidth* 1Mbps dapat dilihat pada Gambar 4.8. Sebagai perbandingan, Gambar 4.5 menampilkan contoh grafik *timeout* yang dihasilkan dengan *bandwidth* 10Mbps. Parameter yang digunakan dapat dilihat pada Tabel 4.4.

4.2.1.4 *Jitter*

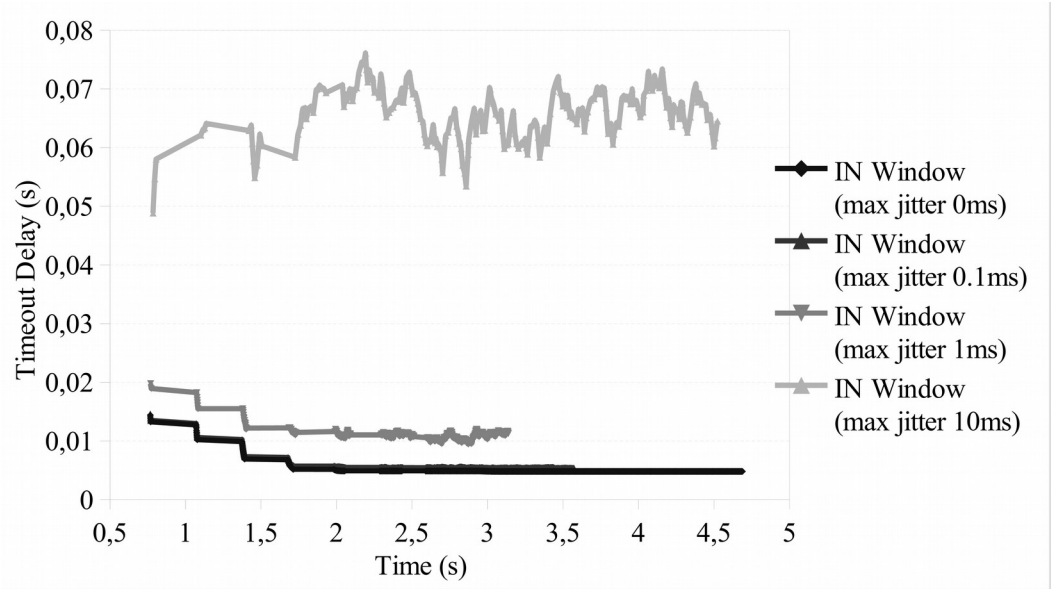
Jitter digunakan secara langsung dalam perhitungan nilai timeout yang dihasilkan oleh metode *PITO*. *Jitter* berbanding lurus dengan *timeout delay* yang dihasilkan metode *PITO*. Semakin besar *jitter*, maka semakin besar pula *timeout delay* yang dihasilkan. Hal ini berakibat buruk bagi metode *PITO* karena kecepatan deteksi terputusnya koneksi semakin lambat, namun *jitter* dapat mengurangi kemungkinan terjadinya kesalahan deteksi terputusnya koneksi.

Tabel 4.5: Parameter Jaringan pada Proses Analisis Parameter *Jitter*

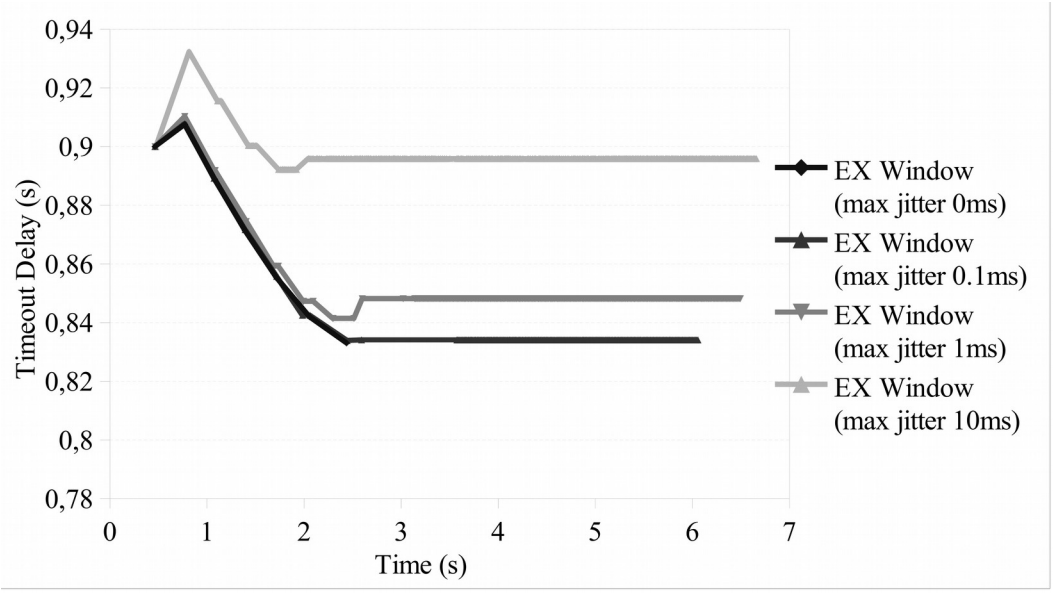
Parameter	**Nilai**
Maximum Segment Size	1440byte
Bandwidth	10Mbps
Latency	150ms
Maximum Jitter	0ms,0.1ms, 1ms,10ms
Packet Loss	0%

Analisis pengaruh *jitter* dilakukan dengan menambahkan *delay* bernilai acak dengan batas maksimal tertentu pada segmen antar *Node 6* dan *Node 8* pada Gambar 4.1 (berlaku terhadap kedua arah aliran paket data). Gambaran pengaruh *jitter* pada *In-Window timeout* dapat dilihat pada Gambar 4.9 (tidak dimulai dari detik ke 0 untuk memperjelas tampilan grafik), sedangkan gambaran pengaruh

jitter pada *Ex-Window timeout* dapat dilihat pada Gambar 4.10.



Gambar 4.9: Pengaruh *Jitter* terhadap *In-Window Timeout Delay*.



Gambar 4.10: Pengaruh *Jitter* terhadap *Ex-Window Timeout Delay*.

4.2.1.5 *Packet Loss*

Ketika terjadi sebuah *packet loss*, maka *host sender* akan mengurangi *congestion window* menjadi setengah dari jumlah data yang sedang beredar di jaringan kemudian memasuki fase *congestion avoidance*. Hal ini berarti setelah terjadi *packet loss* maka sebagian besar *PITO timeout* yang digunakan adalah *Ex-Window timeout*. *Host receiver* yang mendeteksi adanya *packet loss* ini akan melakukan *resetting* nilai *Ex-Window timeout* ke nilai *Ex-Window timeout* pertama yang didapat.

Packet loss merupakan parameter jaringan yang paling mempengaruhi performa metode *PITO* karena *packet loss* mengakibatkan ketidakteraturan *packet inter-arrival time* pada *host receiver*. Tingkat *packet loss* yang tinggi dapat meningkatkan resiko kesalahan deteksi terputusnya koneksi oleh metode *PITO*.

4.2.2 Pengujian performa metode *PITO*

Pengujian performa metode *PITO* dilakukan dalam dua tahap yaitu 1) pengujian akurasi metode *PITO* dalam proses deteksi terputusnya koneksi 2) evaluasi nilai *timeout* yang dihasilkan metode *PITO* dibandingkan dengan metode standar pada *TCP* yaitu *retransmission timeout*.

4.2.2.1 Skenario pengujian performa metode *PITO*

Pengujian performa metode *PITO* dilakukan dengan melakukan pemutusan aliran data pada segmen antara *Node 8* dan *Node 4* pada Gambar 4.1. Hal ini dilakukan untuk merepresentasikan kondisi *host receiver* (*Node 3*) berada di luar jangkaian *Access Point Node 8*. Pemutusan aliran data dilakukan dengan cara melakukan pembuangan (*dropping*) seluruh paket data yang melewati segmen tersebut setelah 20 detik.

Skenario pengujian diatas dibuat untuk memicu terjadinya *vertical handover* dari *subflow* yang melalui *Node 1*, *6*, *8*, *4* ke *subflow* yang melalui *Node 2*, *7*, *9*, *5* pada koneksi *Multipath TCP* antara *Node 0* dan *Node 3* pada Gambar 4.1. Ketika *host*

receiver (*Node 3*) yang mengimplementasikan metode *PITO* mendeteksi terputusnya *subflow* yang melewati *Node 1*, *6*, *8*, *4* maka *host receiver* akan melakukan *vertical handover* dengan cara melakukan aktivasi *subflow* cadangan yang melewati *Node 2*, *7*, *9*, *5*. Pengujian dihentikan setelah 30 detik.

4.2.2.2 Pengujian akurasi metode *PITO*

Pengujian akurasi metode *PITO* dilakukan dengan melakukan 10 (sepuluh) kali percobaan untuk setiap kombinasi nilai parameter yang terdapat pada Tabel 4.6. *Ground truth* yang digunakan dalam pengukuran akurasi metode *PITO* ini adalah titik waktu paket terakhir yang melewati segmen yang terputus diterima *host receiver*. Terjadinya *PITO timeout* sebelum titik *ground truth* tersebut dianggap sebagai kesalahan *metode PITO* dalam mendeteksi terputusnya koneksi (terjadi *false positive*). Oleh karena itu setiap percobaan dikatakan berhasil apabila *PITO timeout* hanya terjadi setelah titik *ground truth* tersebut (terjadi *true positive*). Tingkat keberhasilan pada setiap kombinasi nilai parameter dinyatakan dalam bentuk *precision* yang dihitung berdasarkan jumlah percobaan yang berhasil dan yang gagal. Hasil pengujian akurasi metode *PITO* disajikan pada Tabel 4.7 hingga Tabel 4.18.

Tabel 4.6: Parameter Jaringan pada Proses Pengujian Akurasi Metode *PITO*

Parameter	**Nilai**
Maximum Segment Size	1440byte, 1000byte, 516byte
Bandwidth	10Mbps, 5Mbps, 1Mbps
Latency	150ms, 90ms, 30ms
Maximum Jitter	0ms, 3ms, 6ms
Packet Loss	0%, 0,01%, 0,1%, 1%

Tabel 4.7: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0% dan *jitter* 0ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0%	0ms	150ms	10Mbps	1440byte	1
0%	0ms	150ms	10Mbps	1000byte	1
0%	0ms	150ms	10Mbps	516byte	1
0%	0ms	150ms	5Mbps	1440byte	1
0%	0ms	150ms	5Mbps	1000byte	1
0%	0ms	150ms	5Mbps	516byte	1
0%	0ms	150ms	1Mbps	1440byte	1
0%	0ms	150ms	1Mbps	1000byte	1
0%	0ms	150ms	1Mbps	516byte	1
0%	0ms	90ms	10Mbps	1440byte	1
0%	0ms	90ms	10Mbps	1000byte	1
0%	0ms	90ms	10Mbps	516byte	1
0%	0ms	90ms	5Mbps	1440byte	1
0%	0ms	90ms	5Mbps	1000byte	1
0%	0ms	90ms	5Mbps	516byte	1
0%	0ms	90ms	1Mbps	1440byte	1
0%	0ms	90ms	1Mbps	1000byte	1
0%	0ms	90ms	1Mbps	516byte	1
0%	0ms	30ms	10Mbps	1440byte	1
0%	0ms	30ms	10Mbps	1000byte	1
0%	0ms	30ms	10Mbps	516byte	1
0%	0ms	30ms	5Mbps	1440byte	1
0%	0ms	30ms	5Mbps	1000byte	1
0%	0ms	30ms	5Mbps	516byte	1
0%	0ms	30ms	1Mbps	1440byte	1
0%	0ms	30ms	1Mbps	1000byte	1
0%	0ms	30ms	1Mbps	516byte	1

Tabel 4.8: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0% dan *jitter* 3ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0%	3ms	150ms	10Mbps	1440byte	1
0%	3ms	150ms	10Mbps	1000byte	1
0%	3ms	150ms	10Mbps	516byte	1
0%	3ms	150ms	5Mbps	1440byte	1
0%	3ms	150ms	5Mbps	1000byte	1
0%	3ms	150ms	5Mbps	516byte	1
0%	3ms	150ms	1Mbps	1440byte	1
0%	3ms	150ms	1Mbps	1000byte	1
0%	3ms	150ms	1Mbps	516byte	1
0%	3ms	90ms	10Mbps	1440byte	1
0%	3ms	90ms	10Mbps	1000byte	1
0%	3ms	90ms	10Mbps	516byte	1
0%	3ms	90ms	5Mbps	1440byte	1
0%	3ms	90ms	5Mbps	1000byte	1
0%	3ms	90ms	5Mbps	516byte	1
0%	3ms	90ms	1Mbps	1440byte	1
0%	3ms	90ms	1Mbps	1000byte	1
0%	3ms	90ms	1Mbps	516byte	1
0%	3ms	30ms	10Mbps	1440byte	1
0%	3ms	30ms	10Mbps	1000byte	1
0%	3ms	30ms	10Mbps	516byte	1
0%	3ms	30ms	5Mbps	1440byte	1
0%	3ms	30ms	5Mbps	1000byte	1
0%	3ms	30ms	5Mbps	516byte	1
0%	3ms	30ms	1Mbps	1440byte	1
0%	3ms	30ms	1Mbps	1000byte	1
0%	3ms	30ms	1Mbps	516byte	1

Tabel 4.9: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0% dan *jitter* 6ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0%	6ms	150ms	10Mbps	1440byte	1
0%	6ms	150ms	10Mbps	1000byte	1
0%	6ms	150ms	10Mbps	516byte	1
0%	6ms	150ms	5Mbps	1440byte	1
0%	6ms	150ms	5Mbps	1000byte	1
0%	6ms	150ms	5Mbps	516byte	1
0%	6ms	150ms	1Mbps	1440byte	1
0%	6ms	150ms	1Mbps	1000byte	1
0%	6ms	150ms	1Mbps	516byte	1
0%	6ms	90ms	10Mbps	1440byte	1
0%	6ms	90ms	10Mbps	1000byte	1
0%	6ms	90ms	10Mbps	516byte	1
0%	6ms	90ms	5Mbps	1440byte	1
0%	6ms	90ms	5Mbps	1000byte	1
0%	6ms	90ms	5Mbps	516byte	1
0%	6ms	90ms	1Mbps	1440byte	1
0%	6ms	90ms	1Mbps	1000byte	1
0%	6ms	90ms	1Mbps	516byte	1
0%	6ms	30ms	10Mbps	1440byte	1
0%	6ms	30ms	10Mbps	1000byte	1
0%	6ms	30ms	10Mbps	516byte	1
0%	6ms	30ms	5Mbps	1440byte	1
0%	6ms	30ms	5Mbps	1000byte	1
0%	6ms	30ms	5Mbps	516byte	1
0%	6ms	30ms	1Mbps	1440byte	1
0%	6ms	30ms	1Mbps	1000byte	1
0%	6ms	30ms	1Mbps	516byte	1

Tabel 4.10: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.01% dan *jitter* 0ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.01%	0ms	150ms	10Mbps	1440byte	1
0.01%	0ms	150ms	10Mbps	1000byte	0.9
0.01%	0ms	150ms	10Mbps	516byte	1
0.01%	0ms	150ms	5Mbps	1440byte	1
0.01%	0ms	150ms	5Mbps	1000byte	1
0.01%	0ms	150ms	5Mbps	516byte	1
0.01%	0ms	150ms	1Mbps	1440byte	1
0.01%	0ms	150ms	1Mbps	1000byte	1
0.01%	0ms	150ms	1Mbps	516byte	1
0.01%	0ms	90ms	10Mbps	1440byte	1
0.01%	0ms	90ms	10Mbps	1000byte	1
0.01%	0ms	90ms	10Mbps	516byte	1
0.01%	0ms	90ms	5Mbps	1440byte	1
0.01%	0ms	90ms	5Mbps	1000byte	1
0.01%	0ms	90ms	5Mbps	516byte	1
0.01%	0ms	90ms	1Mbps	1440byte	1
0.01%	0ms	90ms	1Mbps	1000byte	1
0.01%	0ms	90ms	1Mbps	516byte	1
0.01%	0ms	30ms	10Mbps	1440byte	1
0.01%	0ms	30ms	10Mbps	1000byte	1
0.01%	0ms	30ms	10Mbps	516byte	0.9
0.01%	0ms	30ms	5Mbps	1440byte	1
0.01%	0ms	30ms	5Mbps	1000byte	1
0.01%	0ms	30ms	5Mbps	516byte	1
0.01%	0ms	30ms	1Mbps	1440byte	1
0.01%	0ms	30ms	1Mbps	1000byte	1
0.01%	0ms	30ms	1Mbps	516byte	1

Tabel 4.11: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.01% dan *jitter* 3ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.01%	3ms	150ms	10Mbps	1440byte	1
0.01%	3ms	150ms	10Mbps	1000byte	1
0.01%	3ms	150ms	10Mbps	516byte	1
0.01%	3ms	150ms	5Mbps	1440byte	1
0.01%	3ms	150ms	5Mbps	1000byte	1
0.01%	3ms	150ms	5Mbps	516byte	1
0.01%	3ms	150ms	1Mbps	1440byte	1
0.01%	3ms	150ms	1Mbps	1000byte	1
0.01%	3ms	150ms	1Mbps	516byte	1
0.01%	3ms	90ms	10Mbps	1440byte	1
0.01%	3ms	90ms	10Mbps	1000byte	1
0.01%	3ms	90ms	10Mbps	516byte	1
0.01%	3ms	90ms	5Mbps	1440byte	1
0.01%	3ms	90ms	5Mbps	1000byte	1
0.01%	3ms	90ms	5Mbps	516byte	1
0.01%	3ms	90ms	1Mbps	1440byte	1
0.01%	3ms	90ms	1Mbps	1000byte	1
0.01%	3ms	90ms	1Mbps	516byte	1
0.01%	3ms	30ms	10Mbps	1440byte	1
0.01%	3ms	30ms	10Mbps	1000byte	1
0.01%	3ms	30ms	10Mbps	516byte	1
0.01%	3ms	30ms	5Mbps	1440byte	1
0.01%	3ms	30ms	5Mbps	1000byte	1
0.01%	3ms	30ms	5Mbps	516byte	1
0.01%	3ms	30ms	1Mbps	1440byte	1
0.01%	3ms	30ms	1Mbps	1000byte	1
0.01%	3ms	30ms	1Mbps	516byte	1

Tabel 4.12: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.01% dan *jitter* 6ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.01%	6ms	150ms	10Mbps	1440byte	1
0.01%	6ms	150ms	10Mbps	1000byte	1
0.01%	6ms	150ms	10Mbps	516byte	1
0.01%	6ms	150ms	5Mbps	1440byte	1
0.01%	6ms	150ms	5Mbps	1000byte	1
0.01%	6ms	150ms	5Mbps	516byte	1
0.01%	6ms	150ms	1Mbps	1440byte	1
0.01%	6ms	150ms	1Mbps	1000byte	1
0.01%	6ms	150ms	1Mbps	516byte	1
0.01%	6ms	90ms	10Mbps	1440byte	1
0.01%	6ms	90ms	10Mbps	1000byte	1
0.01%	6ms	90ms	10Mbps	516byte	1
0.01%	6ms	90ms	5Mbps	1440byte	1
0.01%	6ms	90ms	5Mbps	1000byte	1
0.01%	6ms	90ms	5Mbps	516byte	1
0.01%	6ms	90ms	1Mbps	1440byte	1
0.01%	6ms	90ms	1Mbps	1000byte	1
0.01%	6ms	90ms	1Mbps	516byte	1
0.01%	6ms	30ms	10Mbps	1440byte	1
0.01%	6ms	30ms	10Mbps	1000byte	1
0.01%	6ms	30ms	10Mbps	516byte	0.9
0.01%	6ms	30ms	5Mbps	1440byte	1
0.01%	6ms	30ms	5Mbps	1000byte	1
0.01%	6ms	30ms	5Mbps	516byte	1
0.01%	6ms	30ms	1Mbps	1440byte	1
0.01%	6ms	30ms	1Mbps	1000byte	1
0.01%	6ms	30ms	1Mbps	516byte	1

Tabel 4.13: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.1% dan *jitter* 0ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.1%	0ms	150ms	10Mbps	1440byte	1
0.1%	0ms	150ms	10Mbps	1000byte	1
0.1%	0ms	150ms	10Mbps	516byte	0.9
0.1%	0ms	150ms	5Mbps	1440byte	1
0.1%	0ms	150ms	5Mbps	1000byte	0.9
0.1%	0ms	150ms	5Mbps	516byte	0.9
0.1%	0ms	150ms	1Mbps	1440byte	0.9
0.1%	0ms	150ms	1Mbps	1000byte	1
0.1%	0ms	150ms	1Mbps	516byte	0.8
0.1%	0ms	90ms	10Mbps	1440byte	1
0.1%	0ms	90ms	10Mbps	1000byte	1
0.1%	0ms	90ms	10Mbps	516byte	0.9
0.1%	0ms	90ms	5Mbps	1440byte	0.9
0.1%	0ms	90ms	5Mbps	1000byte	0.9
0.1%	0ms	90ms	5Mbps	516byte	0.9
0.1%	0ms	90ms	1Mbps	1440byte	1
0.1%	0ms	90ms	1Mbps	1000byte	1
0.1%	0ms	90ms	1Mbps	516byte	1
0.1%	0ms	30ms	10Mbps	1440byte	1
0.1%	0ms	30ms	10Mbps	1000byte	1
0.1%	0ms	30ms	10Mbps	516byte	0.7
0.1%	0ms	30ms	5Mbps	1440byte	1
0.1%	0ms	30ms	5Mbps	1000byte	1
0.1%	0ms	30ms	5Mbps	516byte	0.8
0.1%	0ms	30ms	1Mbps	1440byte	1
0.1%	0ms	30ms	1Mbps	1000byte	1
0.1%	0ms	30ms	1Mbps	516byte	1

Tabel 4.14: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.1% dan *jitter* 3ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.1%	3ms	150ms	10Mbps	1440byte	1
0.1%	3ms	150ms	10Mbps	1000byte	0.9
0.1%	3ms	150ms	10Mbps	516byte	1
0.1%	3ms	150ms	5Mbps	1440byte	1
0.1%	3ms	150ms	5Mbps	1000byte	1
0.1%	3ms	150ms	5Mbps	516byte	1
0.1%	3ms	150ms	1Mbps	1440byte	1
0.1%	3ms	150ms	1Mbps	1000byte	1
0.1%	3ms	150ms	1Mbps	516byte	1
0.1%	3ms	90ms	10Mbps	1440byte	1
0.1%	3ms	90ms	10Mbps	1000byte	1
0.1%	3ms	90ms	10Mbps	516byte	1
0.1%	3ms	90ms	5Mbps	1440byte	1
0.1%	3ms	90ms	5Mbps	1000byte	1
0.1%	3ms	90ms	5Mbps	516byte	1
0.1%	3ms	90ms	1Mbps	1440byte	1
0.1%	3ms	90ms	1Mbps	1000byte	1
0.1%	3ms	90ms	1Mbps	516byte	1
0.1%	3ms	30ms	10Mbps	1440byte	1
0.1%	3ms	30ms	10Mbps	1000byte	1
0.1%	3ms	30ms	10Mbps	516byte	0.9
0.1%	3ms	30ms	5Mbps	1440byte	1
0.1%	3ms	30ms	5Mbps	1000byte	1
0.1%	3ms	30ms	5Mbps	516byte	1
0.1%	3ms	30ms	1Mbps	1440byte	1
0.1%	3ms	30ms	1Mbps	1000byte	1
0.1%	3ms	30ms	1Mbps	516byte	1

Tabel 4.15: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 0.1% dan *jitter* 6ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
0.1%	6ms	150ms	10Mbps	1440byte	1
0.1%	6ms	150ms	10Mbps	1000byte	1
0.1%	6ms	150ms	10Mbps	516byte	1
0.1%	6ms	150ms	5Mbps	1440byte	1
0.1%	6ms	150ms	5Mbps	1000byte	1
0.1%	6ms	150ms	5Mbps	516byte	0.9
0.1%	6ms	150ms	1Mbps	1440byte	1
0.1%	6ms	150ms	1Mbps	1000byte	1
0.1%	6ms	150ms	1Mbps	516byte	1
0.1%	6ms	90ms	10Mbps	1440byte	1
0.1%	6ms	90ms	10Mbps	1000byte	0.9
0.1%	6ms	90ms	10Mbps	516byte	1
0.1%	6ms	90ms	5Mbps	1440byte	1
0.1%	6ms	90ms	5Mbps	1000byte	1
0.1%	6ms	90ms	5Mbps	516byte	1
0.1%	6ms	90ms	1Mbps	1440byte	1
0.1%	6ms	90ms	1Mbps	1000byte	1
0.1%	6ms	90ms	1Mbps	516byte	1
0.1%	6ms	30ms	10Mbps	1440byte	1
0.1%	6ms	30ms	10Mbps	1000byte	1
0.1%	6ms	30ms	10Mbps	516byte	0.9
0.1%	6ms	30ms	5Mbps	1440byte	1
0.1%	6ms	30ms	5Mbps	1000byte	1
0.1%	6ms	30ms	5Mbps	516byte	1
0.1%	6ms	30ms	1Mbps	1440byte	1
0.1%	6ms	30ms	1Mbps	1000byte	1
0.1%	6ms	30ms	1Mbps	516byte	0.9

Tabel 4.16: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 1% dan *jitter* 0ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
1%	0ms	150ms	10Mbps	1440byte	0.5
1%	0ms	150ms	10Mbps	1000byte	0.4
1%	0ms	150ms	10Mbps	516byte	0.6
1%	0ms	150ms	5Mbps	1440byte	0.3
1%	0ms	150ms	5Mbps	1000byte	0.6
1%	0ms	150ms	5Mbps	516byte	0.5
1%	0ms	150ms	1Mbps	1440byte	0.5
1%	0ms	150ms	1Mbps	1000byte	0.9
1%	0ms	150ms	1Mbps	516byte	0.7
1%	0ms	90ms	10Mbps	1440byte	0.5
1%	0ms	90ms	10Mbps	1000byte	0.3
1%	0ms	90ms	10Mbps	516byte	0.3
1%	0ms	90ms	5Mbps	1440byte	0.6
1%	0ms	90ms	5Mbps	1000byte	0.5
1%	0ms	90ms	5Mbps	516byte	0.6
1%	0ms	90ms	1Mbps	1440byte	0.7
1%	0ms	90ms	1Mbps	1000byte	0.4
1%	0ms	90ms	1Mbps	516byte	0.6
1%	0ms	30ms	10Mbps	1440byte	0
1%	0ms	30ms	10Mbps	1000byte	0
1%	0ms	30ms	10Mbps	516byte	0.1
1%	0ms	30ms	5Mbps	1440byte	0
1%	0ms	30ms	5Mbps	1000byte	0
1%	0ms	30ms	5Mbps	516byte	0.1
1%	0ms	30ms	1Mbps	1440byte	0.8
1%	0ms	30ms	1Mbps	1000byte	0.6
1%	0ms	30ms	1Mbps	516byte	0

Tabel 4.17: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 1% dan *jitter* 3ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
1%	3ms	150ms	10Mbps	1440byte	0.7
1%	3ms	150ms	10Mbps	1000byte	0.9
1%	3ms	150ms	10Mbps	516byte	0.9
1%	3ms	150ms	5Mbps	1440byte	0.9
1%	3ms	150ms	5Mbps	1000byte	0.8
1%	3ms	150ms	5Mbps	516byte	0.6
1%	3ms	150ms	1Mbps	1440byte	0.8
1%	3ms	150ms	1Mbps	1000byte	1
1%	3ms	150ms	1Mbps	516byte	0.6
1%	3ms	90ms	10Mbps	1440byte	0.9
1%	3ms	90ms	10Mbps	1000byte	0.8
1%	3ms	90ms	10Mbps	516byte	0.7
1%	3ms	90ms	5Mbps	1440byte	0.7
1%	3ms	90ms	5Mbps	1000byte	0.9
1%	3ms	90ms	5Mbps	516byte	0.9
1%	3ms	90ms	1Mbps	1440byte	0.8
1%	3ms	90ms	1Mbps	1000byte	0.8
1%	3ms	90ms	1Mbps	516byte	0.7
1%	3ms	30ms	10Mbps	1440byte	0.4
1%	3ms	30ms	10Mbps	1000byte	0.3
1%	3ms	30ms	10Mbps	516byte	0.3
1%	3ms	30ms	5Mbps	1440byte	0.4
1%	3ms	30ms	5Mbps	1000byte	0.3
1%	3ms	30ms	5Mbps	516byte	0.6
1%	3ms	30ms	1Mbps	1440byte	0.7
1%	3ms	30ms	1Mbps	1000byte	0.7
1%	3ms	30ms	1Mbps	516byte	0.3

Tabel 4.18: Hasil Pengujian Akurasi Metode *PITO* dengan *packet loss* 1% dan *jitter* 6ms

Parameter					***Precision***
Packet Loss	***Jitter***	***Latency***	***Bandwidth***	***MSS***	
1%	6ms	150ms	10Mbps	1440byte	0.9
1%	6ms	150ms	10Mbps	1000byte	0.6
1%	6ms	150ms	10Mbps	516byte	1
1%	6ms	150ms	5Mbps	1440byte	0.9
1%	6ms	150ms	5Mbps	1000byte	0.8
1%	6ms	150ms	5Mbps	516byte	0.8
1%	6ms	150ms	1Mbps	1440byte	0.8
1%	6ms	150ms	1Mbps	1000byte	0.8
1%	6ms	150ms	1Mbps	516byte	0.7
1%	6ms	90ms	10Mbps	1440byte	0.7
1%	6ms	90ms	10Mbps	1000byte	0.5
1%	6ms	90ms	10Mbps	516byte	0.6
1%	6ms	90ms	5Mbps	1440byte	1
1%	6ms	90ms	5Mbps	1000byte	0.7
1%	6ms	90ms	5Mbps	516byte	0.8
1%	6ms	90ms	1Mbps	1440byte	1
1%	6ms	90ms	1Mbps	1000byte	1
1%	6ms	90ms	1Mbps	516byte	0.6
1%	6ms	30ms	10Mbps	1440byte	0.4
1%	6ms	30ms	10Mbps	1000byte	0.6
1%	6ms	30ms	10Mbps	516byte	0.1
1%	6ms	30ms	5Mbps	1440byte	0.7
1%	6ms	30ms	5Mbps	1000byte	0.6
1%	6ms	30ms	5Mbps	516byte	0.3
1%	6ms	30ms	1Mbps	1440byte	0.6
1%	6ms	30ms	1Mbps	1000byte	0.8
1%	6ms	30ms	1Mbps	516byte	0.5

4.2.2.3 Evaluasi *timeout* yang dihasilkan metode *PITO*

Pada bagian ini akan dilakukan perbandingan *timeout* yang dihasilkan metode *PITO* dengan *retransmission timeout* yang merupakan bagian dari TCP standar. Parameter jaringan yang digunakan pada proses evaluasi ini dipilih yang diperkirakan dapat merepresentasikan penggunaan *internet* pada perangkat bergerak saat ini dan disajikan pada Tabel 4.19.

Evaluasi *timeout* yang dihasilkan metode *PITO* dilakukan dengan menggunakan hasil pengujian akurasi metode *PITO* diatas yang tidak mengalami kesalahan deteksi, kemudian membandingkan rata-rata *timeout delay* yang dihasilkan metode PITO pada *host receiver* dengan *retransmission timeout* yang terjadi pada *host sender*. Selisih antara kedua nilai tersebut kemudian dibandingkan dengan nilai *round trip time* (*RTT*) pada percobaan tersebut. Hasil yang didapat berupa rata-rata penghematan waktu deteksi terputusnya koneksi yang diukur dalam satuan waktu *round trip time*. Penggunaan *round trip time* sebagai satuan waktu dimaksudkan untuk melakukan normalisasi terhadap nilai penghematan waktu tersebut yang sangat dipengaruhi oleh *round trip time*. Hasil evaluasi tersebut dapat dilihat pada Tabel 4.20.

Tabel 4.19: Parameter Jaringan pada Proses Evaluasi *Timeout Delay* yang Dihasilkan Metode *PITO*

Parameter	**Nilai**
Maximum Segment Size	1440byte
Bandwidth	10Mbps, 5Mbps, 1Mbps
Latency	150ms, 90ms, 30ms
Maximum Jitter	3ms
Packet Loss	0,1%, 1%

Tabel 4.20: Hasil Evaluasi Pengurangan *Delay Timeout*

Parameter					**Pengurangan *Delay Timeout* (*RTT*)**
***Packet Loss* (%)**	***Jitter* (ms)**	***Latency* (ms)**	***Bandwidth* (Mbps)**	***MSS* (byte)**	
0,1	3	150	10	1440	-0,227s (-75,64% *RTT*)
0,1	3	90	10	1440	-0,161s (-89,48% *RTT*)
0,1	3	30	10	1440	0,033s (54,68% *RTT*)
0,1	3	150	5	1440	-0,299s (-99,99% *RTT*)
0,1	3	90	5	1440	0,187s (104,35% *RTT*)
0,1	3	30	5	1440	0,108s (179,32% *RTT*)
0,1	3	150	1	1440	1,754s (583,62% *RTT*)
0,1	3	90	1	1440	2,688s (1493,42% *RTT*)
0,1	3	30	1	1440	2,256s (3760,38% *RTT*)
1	3	150	10	1440	-0,538s (-179,61% *RTT*)
1	3	90	10	1440	-0,193s (-107,03% *RTT*)
1	3	30	10	1440	0,04s (66,33% *RTT*)
1	3	150	5	1440	-0,356s (-118,62% *RTT*)
1	3	90	5	1440	-0,218s (-121,13% *RTT*)
1	3	30	5	1440	0,032s (52,96% *RTT*)
1	3	150	1	1440	-0,374s (-124,72% *RTT*)
1	3	90	1	1440	-0,264s (-146,6% *RTT*)
1	3	30	1	1440	0,089s (148,45% *RTT*)

4.3 Pembahasan hasil pengujian metode *PITO*

Hasil pengujian akurasi metode *PITO* yang disajikan pada Tabel 4.7 hingga Tabel 4.18 dapat dirangkum menjadi Tabel 4.21. Data pada Tabel 4.21 menunjukkan bahwa besarnya tingkat *packet loss* sangat mempengaruhi akurasi metode *PITO* dalam mendeteksi terputusnya koneksi. Ketika tidak terjadi *packet loss* (nilai parameter *packet loss* 0%) maka metode *PITO* dapat melakukan deteksi

terputusnya koneksi tanpa kesalahan sama sekali meskipun nilai *jitter* bervariasi. Hal ini disebabkan karena ketika tidak terjadi *packet loss* maka nilai *jitter* sebesar 0ms (tidak ada variasi jeda waktu antar paket yang diterima *host receiver*) sudah cukup untuk mencegah terjadinya kesalahan pada metode *PITO* dalam mendeteksi terputusnya koneksi. Ketika tingkat *packet loss* bertambah maka akurasi metode *PITO* semakin menurun. Pada tingkat *packet loss* 0.01% rata-rata nilai *precision* mencapai 0,996, sedangkan untuk tingkat *packet loss* 0,1% dan 1% rata-rata nilai *precision* masing-masing adalah 0,974 dan 0,596. Hal ini sekaligus menunjukkan bahwa metode PITO dapat menghasilkan performa yang baik pada jaringan dengan tingkat *packet loss* antara 0% hingga 0,1%.

Tabel 4.21: Rata-rata *Precision* Berdasarkan *Packet Loss* dan *Jitter*

Nilai *Packet Loss*	**Nilai *Jitter***	***Precision***
0%	0ms	1
0%	3ms	1
0%	6ms	1
0,01%	0ms	0,993
0,01%	3ms	1
0,01%	6ms	0,996
0,1%	0ms	0,944
0,1%	3ms	0,993
0,1%	6ms	0,985
1%	0ms	0,411
1%	3ms	0,681
1%	6ms	0,696

Pengaruh *jitter* terhadap akurasi metode *PITO* terlihat jelas pada kasus tingkat *packet loss* 0,1% dan 1% dimana nilai *precision* meningkat ketika terdapat *jitter* dalam jaringan. Hal ini disebabkan karena semakin besar nilai *jitter* maka nilai

PITO timeout akan meningkat yang menyebabkan tingkat toleransi metode *PITO* terhadap variasi jeda waktu antar paket (*packet inter-arrival time*) dalam jaringan bertambah. Pada kasus tingkat *packet loss* 0,01% terjadi peningkatan *precision* ketika *jitter* bertambah dari 0ms ke 3ms, namun menurun kembali ketika *jitter* bernilai 6ms. Pada kasus ini penurunan tersebut disebabkan oleh terjadinya satu kali kesalahan deteksi seperti yang disajikan pada Tabel 4.12. Kesalahan deteksi pada kasus ini terjadi secara acak dan dapat dianggap sebagai *outlier*.

Tabel 4.22: Jumlah Skenario yang Mengalami Pengurangan dan Penambahan *Delay Timeout*

Nilai *Latency*	**Jumlah Skenario yang Mengalami Pengurangan *Delay Timeout***	**Jumlah Skenario yang Mengalami Penambahan *Delay Timeout***
150ms	1	5
90ms	3	3
30ms	6	0

Keuntungan penggunaan metode *PITO* terlihat dari hasil evaluasi pengurangan *timeout delay*. Dari total 18 skenario percobaan pada Tabel 4.20, 10 diantaranya menunjukkan pengurangan waktu deteksi terputusnya koneksi. Hal ini ditunjukkan pada Tabel 4.22. Secara keseluruhan rata-rata pengurangan waktu deteksi terputusnya koneksi adalah 310,82% *round trip time* (*RTT*). Apabila proses pengiriman informasi *vertical handover* dari *host receiver* ke *host sender* membutuhkan waktu sebesar 50% *RTT*, maka rata-rata penghematan waktu bersih yang dihasilkan metode *PITO* adalah sebesar 260,82% *RTT*. Hasil ini menunjukkan bahwa metode *PITO* efektif dalam mengurangi *delay* pada proses *vertical handover* akibat proses deteksi terputusnya koneksi.

Berdasarkan Tabel 4.22 terdapat pola pada hasil evaluasi pengurangan *timeout*

delay: untuk nilai *packet loss*, *jitter*, dan *bandwidth* yang sama, jika *latency* semakin kecil maka kemungkinan terjadinya pengurangan *delay timeout* semakin besar. Pola ini terjadi karena *latency* yang kecil akan menyebabkan nilai *Ex-Window timeout* yang kecil. *Ex-Window timeout* merupakan batas bawah *timeout* yang dihasilkan oleh metode *PITO*, sehingga *latency* yang kecil secara keseluruhan akan mengurangi waktu deteksi terputusnya koneksi.

[Halaman ini sengaja dikosongkan]

BAB 5
KESIMPULAN

Pada bab ini akan diuraikan kesimpulan yang diambil dari hasil analisis, pengujian, dan pembahasan pada bab sebelumnya. Di akhir bab akan disertakan saran untuk penelitian selanjutnya.

5.1 Kesimpulan

Beberapa kesimpulan yang dapat diambil dari penelitian metode *Packet Inter-arrival Timeout* untuk mendeteksi terputusnya koneksi adalah sebagai berikut:

1. Akurasi metode *PITO* dalam mendeteksi terputusnya koneksi sangat dipengaruhi oleh tingkat *packet loss* dalam jaringan. Semakin besar tingkat *packet loss* maka akurasi metode *PITO* semakin berkurang.
2. Metode *PITO* masih dapat menghasilkan tingkat akurasi yang baik pada jaringan dengan tingkat packet *loss* sebesar 0,1% atau kurang dengan nilai *precision* sebesar 0,944 atau lebih.
3. *Lantecy* jaringan berpengaruh terhadap kemungkinan terjadinya pengurangan *delay timeout*. Semakin kecil *latency* maka semakin besar kemungkinan terjadinya pengurangan *delay timeout*.
4. Dari total 180 kali pengujian dengan tingkat *packet loss* antara 0,1% hingga 1%, penggunaan metode *PITO* secara keseluruhan menghasilkan pengurangan waktu deteksi terputusnya koneksi sebesar rata-rata 310,82% *round trip time* jaringan dibandingkan dengan menggunakan *retransmission timeout*.

5.2 Saran

Metode *PITO* menjanjikan peningkatan performa yang besar dalam mendeteksi terputusnya koneksi. Namun metode *PITO* menggunakan perhitungan

yang cukup rumit dan terdapat perhitungan yang menggunakan angka *floating point*. Hal ini dapat berakibat menyita energi yang cukup besar dalam operasinya. Oleh karena itu perlu dilakukan evaluasi penggunaan energi oleh metode *PITO*.

DAFTAR PUSTAKA

Allman, M., Paxson, V., Blanton, E. (2009), *TCP Congestion Control*, Internet Engineering Task Force.

Barre, S., Paasch, C., Bonaventure, O. (2011), "Multipath TCP: From Theory to Practice", *Proceedings of 10th International IFIP TC 6 Networking Conference*, Eds: Domingo-Pascual, J. et al., International Federation for Information Processing, Valencia, hal. 444-457.

Carroll, A., Heiser, G. (2010), "An Analysis of Power Consumption in a Smartphone", *Proceedings of 2010 USENIX Annual Tehnical Conference*, USENIX Association, Berkeley, hal. 21-21.

Ford, A., Raiciu, C., Handley, M., Bonaventure, O. (2013), *TCP Extensions for Multipath Operation with Multiple Addresses*, Internet Engineering Task Force.

Gundavelli, S., Leung, K., Devarapalli, V., Chowdhury, K., Patil, B. (2008), *Proxy Mobile IPv6*, Internet Engineering Task Force.

IEEE Computer Society. (2009), *IEEE Standard for Local and Metropolitan Area Networks – Part 21: Media Independent Handover Services*, The Institute of Electrical and Electronics Engineer, Inc, New York.

IEEE Computer Society. (2012), *IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, The Institute of Electrical and Electronics Engineer, Inc, New York.

Information Sciences Institute. (1981), *Transmission Control Protocol*, DARPA, Marina del Rey.

Internet Engineering Task Force. (1989), *Requirements for Internet Hosts – Communication Layers*, Internet Engineering Task Force.

Kaup, F., Wichtlhuber, M., Rado, S., Hausheer, D. (2015), "Can Multipath TCP Save Energy? A Measuring and Modeling Study of MPTCP Energy Consumption", *Proceedings of 40th Annual IEEE Conference on Local Computer Networks*, Eds: Kanhere, S. et al., The Institute of Electrical and Electronics Engineer, Inc, Clearwater Beach, hal. 442-445.

Mathis, M., Mahdavi, J., Floyd, S., Romanow, A. (1996), *TCP Selective Acknowledgement Options*, Internet Engineering Task Force.

Medina, A., Allman, M., Floyd, S. (2004), "Measuring Interactions between Transport Protocols and Middleboxes", *Proceedings of 4th ACM SIGCOMM Conference on Internet Measurement*, Association of Computer Machinery, Taormina, hal. 336-341.

Paasch, C., Detal, G., Duchene, F., Raiciu, C., Bonaventure, O. (2012), "Exploring Mobile/WiFi Handover with Multipath TCP", *Proceedings of 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*, Association of Computer Machinery, Valencia, hal. 31-36.

Paxson, V., Allman, M., Chu, J., Sargent, M. (2011), *Computing TCP's Retransmission Timer*, Internet Engineering Task Force.

Perkins, C. (2010), *IP Mobility Support for IPv4, Revised*, Internet Engineering Task Force.

Perkins, C., Johnson, D., Arkko, J. (2011), *Mobility Support in IPv4*, Internet Engineering Task Force.

RFC Editor. (2008), *STD 1 – Internet Official Protocol Standards*, University of Southern California/Informations Sciences Institute, Marina del Rey.

Sinha, R., Papadopoulos, C., Heidemann, J. (2007), *Internet Packet Size Distributions: Some Observations*, Technical Report ISI-TR-2007-643, University of Southern California/Informations Sciences Institute, Marina del Rey.

[Halaman ini sengaja dikosongkan]

BIOGRAFI PENULIS

Penulis adalah pria kelahiran Malang, 21 Mei 1983. Ia adalah anak ketiga dari tiga bersaudara. Pendidikan SD, SMP, dan SMA dijalani penulis di kota kelahirannya, di Malang. Setelah lulus dari SMU Negeri 3 Malang pada tahun 2001, penulis merantau ke Bandung untuk melanjutkan pendidikan ke jenjang S1 di Teknik Informatika, Institut Teknologi Bandung. Pengalaman di dunia profesional mulai didapat penulis sejak tahun 2006. Selama beberapa tahun penulis berikutnya mendapatkan pengalaman bekerja di berbagai bidang, mulai dari teknik, pemasaran, hingga akademik, sebelum akhirnya penulis memutuskan untuk melanjutkan studi ke jenjang S2 di Teknik Informatika, Institut Teknologi Sepuluh Nopember pada tahun 2015. Pada tahun 2017 penulis dinyatakan lulus dan mendapatkan gelar M.Kom.