BAB 6 PENGUJIAN DAN ANALISIS

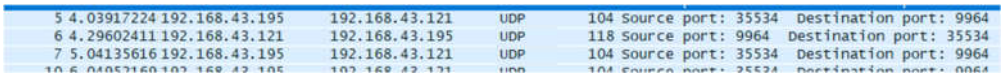
Pengujian yang dilakukan pada penelitian ini meliputi pengujian besar *packet* data, pengujian performansi sistem dan pengujian fungsionalitas aplikasi *web.*

6.1 Skenario pengujian besar *packet* data

Pengujian ini dilakukan untuk mengetahui berapa besar packet data yang dikirimkan dalam satu transaksi, hal ini akan dibandingkan dengan besar paket data yang dikirimkan dalam satu kali transaksi pada penelitian sebelumnya. Satu transaksi data yang dimaksud terdiri dari 3 proses, proses pertama yaitu besar packet ketika pertama kali perangkat terhubung ke *Manager*, proses kedua adalah besar packet data untuk pengiriman perintah *monitoring* dan proses ketiga adalah besar paket hasil *monitoring*. Hasil pengujian didapatkan melalui proses capture dari aplikasi *Wireshark.* Hasil pengujian dilakukan dengan cara melakukan proses menghubungkan *agent* pada *manager*, setelah itu proses perintah *monitoring* dari manager ke agent dan proses berikutnya adalah proses pengiriman data hasil *monitoring* dari agent ke manager, lalu melakukan capture pada besar paket data dan dibandingkan dengan sistem penelitian sebelumnya, protocol SNMP dan sistem *monitoring* perangkat IoT yang dikembangkan.

6.1.1 Hasil pengujian besar packet data

Hasil pengujian besar paket data dilakukan untuk menguji keefektifan sistem *monitoring perangkat IoT* yang dikembangkan dibandingkan dengan sistem *monitoring* penelitian sebelumnya dan protocol SNMP. Hasil tersebut dapat dilihat pada gambar.



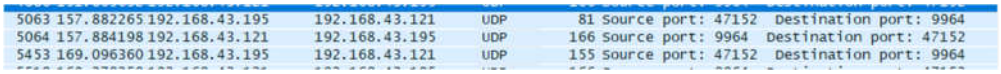
Gambar 6.1 Hasil Capture Transaksi Sistem *Monitoring* pada penelitian sebelumnya

Pada transaksi sistem *monitoring* penelitian sebelumnya dapat dilihat pada penjabaran berikut ini.

Proses 1: Perangkat *Agent* melakukan hubungan ke *Manager* menghasilkan paket sebesar **104 bytes,** seperti yang terlihat pada gambar 6.1

Proses 2: *Manager* mengirim perintah *monitoring* kepada *Agent* menghasilkan paket sebesar **118 bytes,** seperti yang terlihat pada gambar 6.1

Proses 3: *Agent* mengirim data *monitoring* yang diminta oleh *Manager* menghasilkan paket sebesar **104 bytes,** seperti yang terlihat pada gambar 6.1



5063	157.882265	192.168.43.195	192.168.43.121	UDP	81	Source port: 47152 Destination port: 9964
5064	157.884198	192.168.43.121	192.168.43.195	UDP	166	Source port: 9964 Destination port: 47152
5453	169.096360	192.168.43.195	192.168.43.121	UDP	155	Source port: 47152 Destination port: 9964

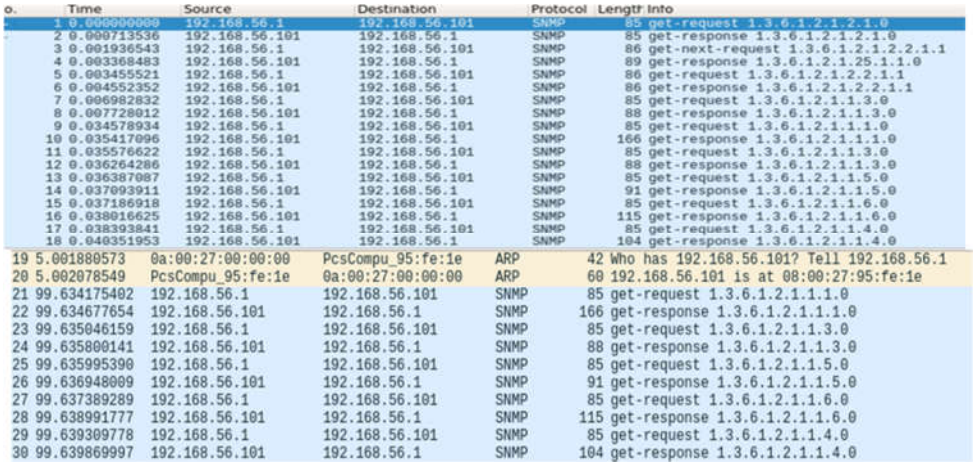
Gambar 6.2 Hasil Capture Transaksi Sistem *Monitoring* Perangkat IoT

Pada transaksi sistem *monitoring* perangkat IoT dapat dilihat pada penjabaran berikut ini.

Proses 1: Perangkat *Agent* melakukan hubungan ke *Manager* menghasilkan paket sebesar **81 bytes,** seperti yang terlihat pada gambar 6.2

Proses 2: *Manager* mengirim perintah *monitoring* kepada *Agent* menghasilkan paket sebesar **166 bytes,** seperti yang terlihat pada gambar 6.2

Proses 3: *Agent* mengirim data *monitoring* yang diminta oleh *Manager* menghasilkan paket sebesar **155 bytes,** seperti yang terlihat pada gambar 6.2



o.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.2.1.0
2	0.000713536	192.168.56.101	192.168.56.1	SNMP	85	get-response 1.3.6.1.2.1.2.1.0
3	0.001936543	192.168.56.1	192.168.56.101	SNMP	86	get-next-request 1.3.6.1.2.1.2.2.1.1
4	0.003368483	192.168.56.101	192.168.56.1	SNMP	89	get-response 1.3.6.1.2.1.25.1.1.0
5	0.003455521	192.168.56.1	192.168.56.101	SNMP	86	get-request 1.3.6.1.2.1.2.2.1.1
6	0.004552352	192.168.56.101	192.168.56.1	SNMP	86	get-response 1.3.6.1.2.1.2.2.1.1
7	0.006982832	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.3.0
8	0.007728012	192.168.56.101	192.168.56.1	SNMP	88	get-response 1.3.6.1.2.1.1.3.0
9	0.034578934	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.1.0
10	0.035417096	192.168.56.101	192.168.56.1	SNMP	166	get-response 1.3.6.1.2.1.1.1.0
11	0.035576622	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.3.0
12	0.036264286	192.168.56.101	192.168.56.1	SNMP	88	get-response 1.3.6.1.2.1.1.3.0
13	0.036387087	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.5.0
14	0.037093911	192.168.56.101	192.168.56.1	SNMP	91	get-response 1.3.6.1.2.1.1.5.0
15	0.037186918	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.6.0
16	0.038016625	192.168.56.101	192.168.56.1	SNMP	115	get-response 1.3.6.1.2.1.1.6.0
17	0.038393841	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.4.0
18	0.040351953	192.168.56.101	192.168.56.1	SNMP	104	get-response 1.3.6.1.2.1.1.4.0
19	5.001880573	0a:00:27:00:00:00	PcsCompu_95:fe:1e	ARP	42	Who has 192.168.56.101? Tell 192.168.56.1
20	5.002078549	PcsCompu_95:fe:1e	0a:00:27:00:00:00	ARP	60	192.168.56.101 is at 08:00:27:95:fe:1e
21	99.634175402	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.1.0
22	99.634677654	192.168.56.101	192.168.56.1	SNMP	166	get-response 1.3.6.1.2.1.1.1.0
23	99.635046159	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.3.0
24	99.635800141	192.168.56.101	192.168.56.1	SNMP	88	get-response 1.3.6.1.2.1.1.3.0
25	99.635995390	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.5.0
26	99.636948009	192.168.56.101	192.168.56.1	SNMP	91	get-response 1.3.6.1.2.1.1.5.0
27	99.637389289	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.6.0
28	99.638991777	192.168.56.101	192.168.56.1	SNMP	115	get-response 1.3.6.1.2.1.1.6.0
29	99.639309778	192.168.56.1	192.168.56.101	SNMP	85	get-request 1.3.6.1.2.1.1.4.0
30	99.639869997	192.168.56.101	192.168.56.1	SNMP	104	get-response 1.3.6.1.2.1.1.4.0

Gambar 6.3 Hasil Capture Transaksi SNMP

Pada transaksi SNMP dapat dilihat pada penjabaran berikut ini.

Proses 1: Perangkat *Agent* melakukan hubungan ke *Manager* menghasilkan paket sebesar **1986 bytes,** seperti yang terlihat pada gambar 6.3

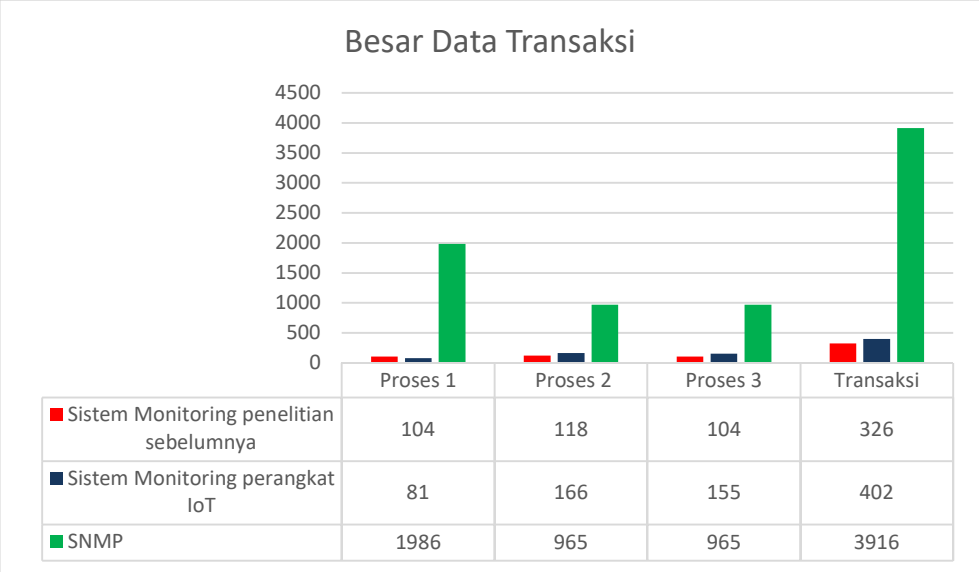
Proses 2: *Manager* mengirim perintah *monitoring* kepada *Agent* menghasilkan paket sebesar **965 bytes,** seperti yang terlihat pada gambar 6.3

Proses 3: *Agent* mengirim data *monitoring* yang diminta oleh *Manager* menghasilkan paket sebesar **965 bytes,** seperti yang terlihat pada gambar 6.3

6.1.2 Analisis pengujian besar packet data

Dari hasil pengujian besar packet data dapat dilihat bahwa total paket data yang ada dalam satu transaksi pada Sistem *Monitoring* penelitian sebelumnya adalah sebesar **326 bytes**. Pada total paket yang ada dalam satu transaksi SNMP adalah **3916 bytes** Sedangkan total paket yang ada dalam satu transaksi pada Sistem *Monitoring* perangkat IoT yang dikembangkan adalah sebesar **402 bytes.**

Hal ini dapat dilihat juga dari besar data setiap proses yang dapat diamati pada gambar 6.4.



	Proses 1	Proses 2	Proses 3	Transaksi
Sistem Monitoring penelitian sebelumnya	104	118	104	326
Sistem Monitoring perangkat IoT	81	166	155	402
SNMP	1986	965	965	3916

Gambar 6.4 Perbandingan Besar Data

Dari Gambar 6.4 dapat dilihat bahwa sistem *monitoring perangkat* IoT yang dikembangkan total paket yang ada dalam satu transaksi lebih besar karena penggunakan struktur data yang lebih kompleks tetapi pada proses 1 yaitu Perangkat Agent melakukan hubungan ke Manager menghasilkan paket yang lebih kecil dibandingkan dengan penelitian sebelumnya, Sedangkan protcol SNMP terlihat total paket yang ada dalam satu transaksi lebih besar lagi dibanding dengan penelitian sebelumnya dan Sistem *monitoring* perangkat IoT yang dikembangkan.

6.2 Skenario pengujian performansi sistem

Pengujian performansi dilakukan melalui program Agent dan program Manager yang berjalan, data yang dihimpun berupa kategori OID pada MIB dari perangkat yang diamati untuk mengetahui seberapa besar *resource* yang digunakan pada saat perangkat *Manager* dari sistem *monitoring* melayani banyak permintaan yang masuk dari *Agent*. Pengujian dilakukan dengan cara melakukan variasi pada permintaan yang masuk dari *Agent* yang terhubung, melalui perulangan menggunakan threading dengan file baru agent yang isinya nanti menggunakan data dummy sebagai cara untuk melakukan pengujian.

Source code 6.1 *Uji permintaan Agent*

```
1  from twisted.internet.protocol import DatagramProtocol
2  from twisted.internet import reactor, threads
3  import thread,json,threading
4  from uuid import getnode as get_mac
5
6  class ClientProtocol(DatagramProtocol):
7      def startProtocol(self):
8          self.transport.connect('192.168.43.195', 9974)
           print "client started"
```

```
9           result="init client"
10          self.transport.write(result)
11
12      def stopProtocol(self):
13          print 'client stopped'
14      def datagramReceived(self, datagram, (host, port)):
15          print 'Datagram received: ', repr(datagram)
16          handler = Handler(self.transport, host, port)
17          d = threads.deferToThread(handler.handleMessage, datagram)
18          if datagram == '1':
19              for i in range(160):
20                  thread.start_new_thread(handler.sendRes, ("From Thread", i, (1,
'6C:71:D9:A9:12:CB', 0.8, 1328218112, 2247737344, 1999335424)))
21          elif datagram == '2':
22              for i in range(20):
23                  thread.start_new_thread(handler.sendRes, (
24                  "From Thread", i, (1, '6C:71:D9:A9:12:CB', 12729214, 119599956,
98163, 121026)))
25          elif datagram == '3':
26              d.addCallback(handler.sendPing)
27
28  class Handler():
29      def __init__(self, transport, host, port):
30          self.host = host
31          self.port = port
32          self.transport = transport
33
34      def handleMessage(self, datagram):
35          print 'Handler Message: ', repr(datagram)
36
37      def sendRes(self, threadName,order, *arg):
38          resource= (1, '6C:71:D9:A9:12:CB', 0.8, 1328218112, 2247737344,
1999335424)
39          res=json.dumps(resource)
40          self.transport.write(res)
41          print "%s: %s: %s" % ( threadName,order,res)
42
43      def sendNetwork(self, *arg):
44          network=(2, '6C:71:D9:A9:12:CB', 12729214, 119599956, 98163, 121026)
45          net = json.dumps(network)
46          self.transport.write(net)
47          print "success"
48
49      def sendPing(self, *arg):
50          ping="up"
51          self.transport.write(ping())
52
53  def main():
54      protocol = ClientProtocol()
55      reactor.listenUDP(0, protocol)
56      #print "%s: %s:" %(thread,order)
57      reactor.run()
58
59
60  if __name__ == '__main__':
69      #for i in range(2):
70       #   my_thread=threading.Thread(target=main)
71        #  my_thread.start()#thread.start_new_thread(main, ("Thread", i))
      main()
```

Pada Source code 6.1 untuk melakukan permintaan dengan beberapa variasi baris 18 adalah digunakan untuk menetapkan variasi permintaan agent dan pada variabel thread di ikuti dengan data dummy yang di buat.

Variasi permintaan agent dilakukan dengan permintaan sebanyak *20 Agent* kemudian meningkat menjadi *40 agent, 80 agent, 160 agent, 320 agent, 640 agent, 1280 agent dan 2560 agent.*

6.2.1 Hasil pengujian performansi sistem

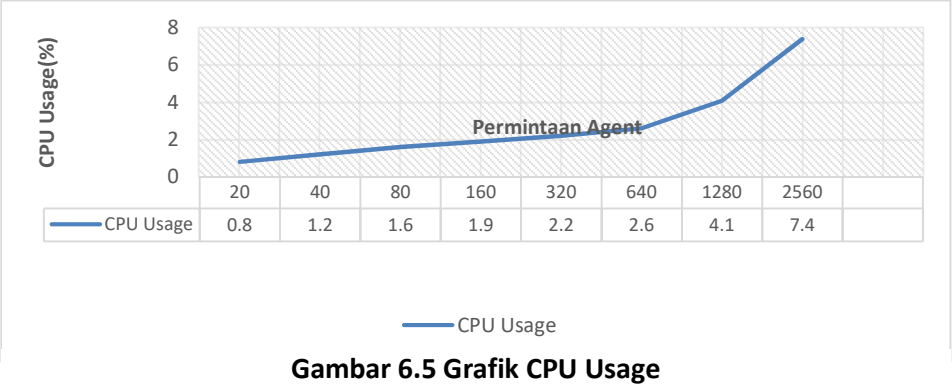
Pengujian performansi sistem dilakukan untuk mengetahui bagaiamana kemampuan sistem *monitoring* ketika menerima banyak permintaan dari *agent*. Data yang dikirimkan *agent* adalah payload yang sama dengan perintah *monitoring* dan data *monitoring* yang ada pada sistem *monitoring* IoT Dari skenario pengujian performansi sistem didapatkan hasil yang dapat dilihat pada Tabel 6.1.

Tabel 6.1 Hasil Pengujian Transmisi Data

No	Jumlah Permintaan *Agent*	CPU Usage(%)	Memori Usage(MB)	Besar Paket Data(Bytes)
1	20	0.8	5.9	1500
2	40	1.2	5.9	2700
3	80	1.6	6	5300
4	160	1.9	6	10600
5	320	2,2	6.1	21000
6	640	2.6	6.2	41900
7	1280	4.1	6.4	83800
8	2560	7.4	6.4	167000

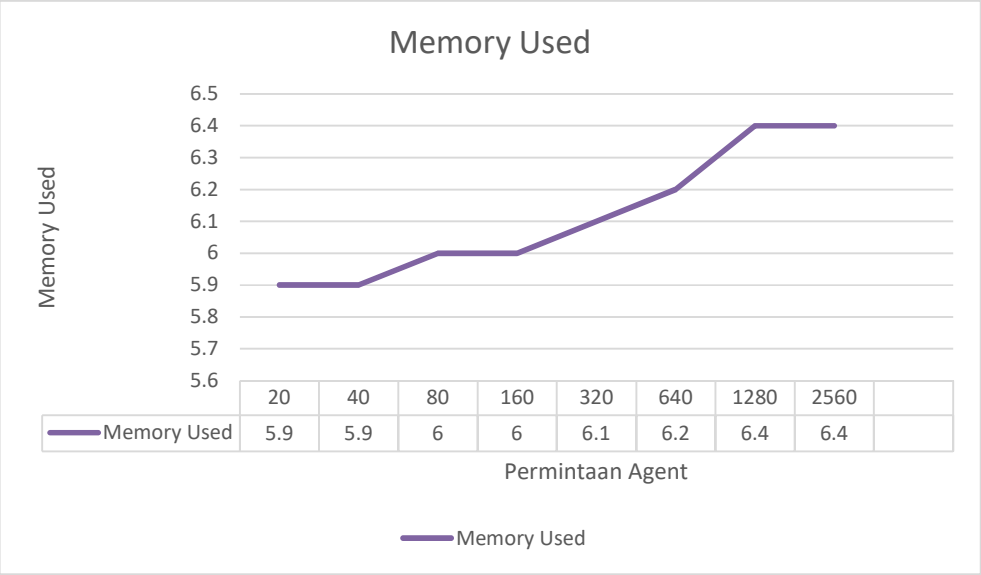
6.2.2 Analisis pengujian pada transmisi data

Dari Hasil Pengujian Transmisi Data dapat dilihat bahwa semua paket yang diharapkan sama dengan paket yang dikirim pada semua skenario pengujian. Hal ini menunjukkan bahwa Sistem *Monitoring perangkat IoT* mampu mengumpulkan informasi data yang dihimpun berupa kategori OID pada MIB dari perangkat yang diamati hal ini dapat terlihat pada gambar berikut.



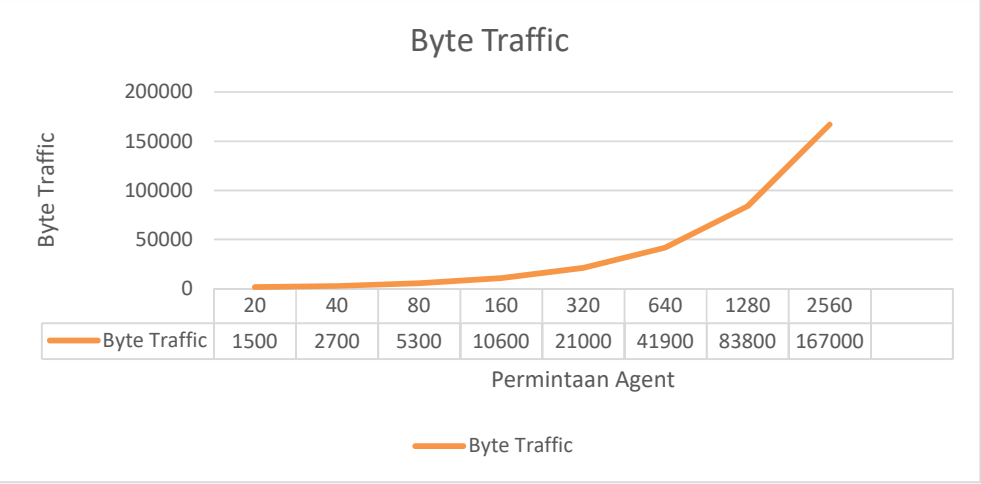
Gambar 6.5 Grafik CPU Usage

Dari Gambar 6.5 dapat dilihat bahwa CPU *Usage* meningkat seiring dengan bertambahnya permintaan *Agent*. Perbedaan signifikan ini terjadi karena penambahan jumlah permintaan *agent* yang menjadi 2 kali lipat setiap kenaikannya. Hal ini terjadi karena *request* data mempengaruhi kinerja dari sistem *monitoring IoT*.



Gambar 6.6 Grafik Memory Used

Dari Gambar 6.6 dapat dilihat komponen *memory used* memiliki pergerakan naik dipengaruhi oleh jumlah *request*. Namun jumlah penggunaan memori cenderung stabil, yaitu berada di angka 5,9 – 6 *MB*.



Gambar 6.7 Byte Traffic

Dari Gambar 6.7 dapat dilihat komponen byte *traffic* bergerak dipengaruhi oleh *request Agent*. Kenaikan byte traffic menjadi hal yang stabil karena kenaikan ini berbanding lurus dengan kenaikan jumlah permintaan *agent*. Hal ini

menunjukkan bahwa semakin banyak permintaan dari *Agent* maka *byte traffic* akan semakin tinggi.

6.3 Pengujian fungsionalitas aplikasi web

Pengujian Fungsionalitas pada aplikasi web dilakukan supaya kita dapat mengetahui apakah fitur – fitur yang ada dapat melaksakan fungsi yang diharapkan.

Tabel 6.2 Skenario Pengujian Fungsionalitas Aplikasi Web

No	Deskripsi	Masukan	Keluaran yang diaharapkan
1	Pengujian menampilkan *list device*	Table *host* pada *database*	Sistem menampilkan *list device*
2	Pengujian menampilkan status *device*	Table *hos*t pada *database*	Sistem menampilkan status *devic*e (up atau *down*)
3	Pengujian menghapus *device*	Tekan tombol *delete host*	Sistem menghapus *host* dari *database*
4	Pengujian memilih *device*	Tekan tombol monitor	Sistem menampilkan form add monitor dengan nama host sesuai yang dipilih
5	Pengujian memilih stop/start device	Tekan tombol stop/start	Sistem memberhentikan atau menjalankan host yang dimonitor
6	Pengujian penyimpanan perintah *monitoring*	Tekan tombol submit	Sistem menyimpan perintah ke *database* dan muncul tulisan *Add Success*
7	Pengujian filter berdasarkan host yang dipilih	Memilih h*ost* pada *host address* dan tekan tombol *filter*	Sistem menampilkan grafik berdasarkan host

8	Pengujian filter berdasarkan kategori OID yang di*monitoring*	Memilih OID pada *monitoring category*	Sistem menampilkan grafik berdasarkan kategori OID

6.3.1 Hasil pengujian fungsionalitas aplikasi web

Dari skenario pengujian fungsionalitas aplikasi *web* didapatkan hasil sebagai berikut:

Tabel 6.3 Hasil Pengujian Fungsionalitas Aplikasi Web

No	Deskripsi	Masukan	Keluaran yang diaharapkan	Hasil yang didapat	Kesimpulan
1	Pengujian menampilkan *list device*	Table *host* pada *database*	Sistem menampilkan *list device*	Sistem menampilkan *list device*	Benar
2	Pengujian menampilkan status *device*	Table *host* pada *database*	Sistem menampilan status *device* (up atau *down*)	Sistem menampilan status *device* (up atau *down*)	Benar
3	Pengujian menghapus *device*	Tekan tombol *delete host*	Sistem menghapus *host* dari *database*	Sistem menghapus *host* dari *database*	Benar
4	Pengujian memilih *device*	Tekan tombol monitor	Sistem menampilkan form add monitor dengan nama host sesuai yang dipilih	Sistem menampilkan form add monitor dengan nama host sesuai yang dipilih	Benar
5	Pengujian memilih stop device	Tekan tombol stop/start	Sistem memberhentikan atau menjalankan host yang dimonitor	Sistem memberhentikan atau menjalankan host yang dimonitor	Benar
6	Pengujian penyimpanan perintah *monitoring*	Tekan tombol submit	Sistem menyimpan perintah ke *database* dan muncul tulisan *Add Success*	Sistem menyimpan perintah ke *database* dan muncul tulisan *Add Success*	Benar

7	Pengujian filter berdasarkan host yang dipilih	Memilih h*ost* pada *host address* dan tekan tombol *filter*	Sistem menampilkan grafik berdasarkan host	Sistem menampilkan grafik berdasarkan host	Benar
8	Pengujian filter berdasarkan kategori OID yang di*monitoring*	Memilih OID pada *monitoring category*	Sistem menampilkan grafik berdasarkan kategori OID	Sistem menampilkan grafik berdasarkan kategori OID	Benar

6.3.2 Analisis pengujian fungsionalitas pada aplikasi web

Dari Hasil Pengujian fungsionalitas pada aplikasi *web*, setiap fungsi yang diharapkan dapat berfungsi dengan benar, output yang dikeluarkan juga sesuai dengan permintaan yang dilakukan. Hal ini menunjukkan bahwa sistem *monitoring* perangkat IoT mampu berjalan sesuai dengan struktur data dictionary dengan format JSON karena dari sistem yang dikembangkan menggunakan struktur data dictionary dengan fitur key dan valuenya. Proses pengiriman dan penerimaan data lebih baik, bisa disimpan dalam database untuk diakses lalu ditampilkan hasil *monitoring* menjadi grafik untuk ditampilkan menjadi informasi bagi pengguna.