Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

a) Flow Monitor in “wifi-simple-adhoc-grid.cc” using “flowmon-parse-results.py” to evaluate results

The “wifi-simple-adhoc-grid.cc” file was simply modified with the standard call for FlowMonitor. The difficult part was in developing the code to parse the XML output data. The example program “flowmon-parse-results.py” was modified to suit this purpose. Snippets of the modified code can be found in the appendix.

b) Traffic trials with flow monitor

All trials were done at distance of 100m and a simulation time of 33.0 seconds, which gave 3 seconds for data transfer after the OSLR was converged. The standard command line inputs were:

1. ./waf
2. ./waf --run "wifi-simple-adhoc-grid.cc --numPackets=<int> --interval=<float>"
3. python3 ~/repos/wireless_comm_lab/Lab5/flowmon-parse-results_Lab5.py wifi-simple-adhoc-grid.xml

See below for a few example outputs.

```
FlowID: 1 (UDP 10.1.1.25/49153 --> 10.1.1.1/80)
        TX bitrate: None
        RX bitrate: None
        Mean Delay: 127.94 ms
        Packet Loss Ratio: 0.00 %
        Hop Count: 8

Default Parameters----
timeFirstTxPacket: 30.00
timeLastTxPacket: 30.00 s
timeFirstRxPacket: 30.13 s
timeLastRxPacket: 30.13 s
txBytes: 1028.0
txPackets: 1
rxBytes: 1028.0
rxPackets: 1
lostPackets: 0
timesForwarded: 7
```

Figure 1: Nptkt=1, interval=1.0 (default)

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

```
FlowID: 1 (UDP 10.1.1.25/49153 --> 10.1.1.1/80)
        TX bitrate: 19.74 kbit/s
        RX bitrate: 20.19 kbit/s
        Mean Delay: 84.10 ms
        Packet Loss Ratio: 0.00 %

Default Parameters----
timeFirstTxPacket: 30.00
timeLastTxPacket: 32.50 s
timeFirstRxPacket: 30.13 s
timeLastRxPacket: 32.57 s
txBytes: 6168.0
txPackets: 6
rxBytes: 6168.0
rxPackets: 6
lostPackets: 0
timesForwarded: 42
```

Figure 2: Npkt=10, interval=0.5

Table 1: "wifi-simple-adhoc-grid" Traffic Trials

Number of Packets	**1**	**10**	**10**	**10**	**50**	**50**	**100**	**100**	**100**
Interval	**1.0**	**1.0**	**0.5**	**0.1**	**0.1**	**0.05**	**0.05**	**0.01**	**0.04**
txPackets	1	3	6	10	30	50	60	100	75
rxPackets	1	3	6	10	30	48	57	14	31
txBytes	1028	3084	6168	10280	30840	51400	61680	102800	77100
rxBytes	1028	3084	6168	10280	30840	49344	58596	14392	31868
Tx Bitrate	None	12.34	19.74	91.38	85.08	167.84	167.27	830.71	208.38
Rx Bitrate	None	12.69	20.19	97.41	86.74	169.01	168.28	68.53	140.29
Mean Delay	127.94	90.8	84.1	80.4	75.07	124.06	115.88	776.94	641.89

Table 1 shows the results of the trials when varying the number of packets and interval time. The optimal conditions appeared to occur when approximately 60 packets were transferred and an interval time around 0.05 sec was selected. Below that time, the number of packets received fell off drastically, the delay time increases significantly, and the resultant received bitrate falls off.

c) Additional background flows

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

Additional flows were added by including a 2nd sink/source and adding randomly spaced flows (see appendix). The sources/sinks were arranged, respectively:

1. sinkNode=0, sourceNode=24
2. sinkNode2=3, sourceNode2=20
3. sinkNode3=10, sourceNode3=5
4. sinkNode4=8, sourceNode4=18

The case of 10 packets and 0.1 interval time from Table 1 (above) was used as a benchmark. Table 2 shows the results of the trails by varying the number of flows when looking at the original flow (node 24 to node 0).

Table 2: Additional Flow model trials

Number of Packets	10	10	10	10
Interval	0.1	0.1	0.1	0.1
Flows	**1**	**2**	**3**	**4**
txPackets	10	10	10	10
rxPackets	10	7	7	9
txBytes	10280	10280	10280	10280
rxBytes	10280	7196	7196	9252
Tx Bitrate	91.38	91.38	91.38	91.38
Rx Bitrate	97.41	52.21	66.15	83.46
Mean Delay	80.4	187.15	37.35	49.27

Table 2 shows that increasing the congestion on the overall network didn't necessarily have a detrimental effect the on the one flow of note. Going up to two flows increased the delay time and reduced the bitrate, but each additional flow seemed to increase the bitrate and decrease the delay time. Perhaps, as long as the network is not saturated, the number of hops that the packet needs to take by itself is decreased.

```
rse-results_Lab5.py wifi-simple-adhoc-grid.xml
Reading XML file

. done.
FlowID: 1 (UDP 10.1.1.21/49153 --> 10.1.1.4/81)
        TX bitrate: 19.19 kbit/s
        RX bitrate: 19.53 kbit/s
        Mean Delay: 70.54 ms
        Packet Loss Ratio: 0.00 %
        Hop Count: 7

Default Parameters----
timeFirstTxPacket: 29.90
timeLastTxPacket: 32.90 s
timeFirstRxPacket: 30.01 s
timeLastRxPacket: 32.96 s
txBytes: 7196.0
txPackets: 7
rxBytes: 7196.0
rxPackets: 7
lostPackets: 0
timesForwarded: 42
FlowID: 2 (UDP 10.1.1.25/49153 --> 10.1.1.1/80)
        TX bitrate: 19.74 kbit/s
        RX bitrate: 20.22 kbit/s
        Mean Delay: 84.43 ms
        Packet Loss Ratio: 0.00 %
        Hop Count: 8

Default Parameters----
timeFirstTxPacket: 30.00
timeLastTxPacket: 32.50 s
timeFirstRxPacket: 30.13 s
timeLastRxPacket: 32.57 s
txBytes: 6168.0
txPackets: 6
rxBytes: 6168.0
rxPackets: 6
lostPackets: 0
timesForwarded: 42
```

Figure 3: Example output with 2 flows

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

d) Takagami Loss Propagation Model

Table 3: Packet/Interval Trials with Takagami Loss Propagation Model

Number of Packets	**10**	**10**	**10**	**50**	**50**	**100**	**100**
Interval	**1.0**	**0.5**	**0.1**	**0.1**	**0.05**	**0.05**	**0.01**
txPackets	3	6	10	30	50	60	100
rxPackets	3	6	10	30	50	60	100
txBytes	3084	6168	10280	30840	51400	61680	102800
rxBytes	3084	6168	10280	30840	51400	61680	102800
Tx Bitrate	12.34	19.74	91.38	85.08	167.84	167.27	830.71
Rx Bitrate	12.41	19.83	92.53	85.41	168.61	167.91	824.79
Mean Delay	12.49	10.62	10.34	9.35	9.11	9.05	42.64

The implementation with the Takagami Loss Propagation model didn't show much difference on the low end, when there were few packets and the interval times were large. However, the biggest difference was seen the Mean Delay time, which showed much lower values than the previous Friis Propagation Loss Model. This resulted in the simulation being able to take the transfers up to much higher numbers of packets and smaller interval times before the performance started falling off.

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

Appendix (code)

b)

```
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();
```

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

```
    @param self The object pointer.
    @param flow_el The element.
    '''
    self.flowId = int(flow_el.get('flowId'))
    rxPackets = float(flow_el.get('rxPackets'))
    txPackets = float(flow_el.get('txPackets'))


    # my added variables
    self.timeFirstTxPacket = float(parse_time_ns(flow_el.get('timeFirstTxPacket')))
    self.timeLastTxPacket = float(parse_time_ns(flow_el.get('timeLastTxPacket')))
    self.timeFirstRxPacket = float(parse_time_ns(flow_el.get('timeFirstRxPacket')))
    self.timeLastRxPacket = float(parse_time_ns(flow_el.get('timeLastRxPacket')))
    self.txBytes = float(flow_el.get('txBytes'))
    self.txPackets = float(flow_el.get('txPackets')) # also member above
    self.rxBytes = float(flow_el.get('rxBytes'))
    self.rxPackets = float(flow_el.get('rxPackets')) # also member above
    self.lostPackets = float(flow_el.get('lostPackets'))
    self.timesForwarded = float(flow_el.get('timesForwarded'))


    tx_duration = (parse_time_ns (flow_el.get('timeLastTxPacket')) - parse_time_ns(flow_el.get('timeFirstTxPacket')))*1e-9
    rx_duration = (parse_time_ns (flow_el.get('timeLastRxPacket')) - parse_time_ns(flow_el.get('timeFirstRxPacket')))*1e-9
    self.rx_duration = rx_duration
    self.probe_stats_unsorted = []
    if rxPackets:
        self.hopCount = float(flow_el.get('timesForwarded')) / rxPackets + 1
    else:
        self.hopCount = -1000
    if rxPackets:
        self.delayMean = float(flow_el.get('delaySum')[:-2]) / rxPackets * 1e-9
        self.packetSizeMean = float(flow_el.get('rxBytes')) / rxPackets
    else:
        self.delayMean = None
        self.packetSizeMean = None
    if rx_duration > 0:
        self.rxBitrate = float(flow_el.get('rxBytes'))*8 / rx_duration
    else:
        self.rxBitrate = None
    if tx_duration > 0:
        self.txBitrate = float(flow_el.get('txBytes'))*8 / tx_duration
    else:
        self.txBitrate = None
    lost = float(flow_el.get('lostPackets'))
    #print "rxBytes: %s; txPackets: %s; rxPackets: %s; lostPackets: %s" % (flow_el.get('rxBytes'), txPackets, rxPackets, lost)
    if rxPackets == 0:
        self.packetLossRatio = None
    else:
        self.packetLossRatio = (lost / (rxPackets + lost))

    interrupt_hist_elem = flow_el.find("flowInterruptionsHistogram")
    if interrupt_hist_elem is None:
        self.flowInterruptionsHistogram = None
    else:
        self.flowInterruptionsHistogram = Histogram(interrupt_hist_elem)
```

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

```
idx = ("timeFirstTxPacket timeLastTxPacket timeFirstRxPacket timeLastRxPacket "
    "txBytes txPackets rxBytes rxPackets lostPackets timesForwarded").split()
for sim in sim_list:
    for flow in sim.flows:
        t = flow.fiveTuple
        proto = {6: 'TCP', 17: 'UDP'} [t.protocol]
        print("FlowID: %i (%s %s/%s --> %s/%i)" % \
            (flow.flowId, proto, t.sourceAddress, t.sourcePort, t.destinationAddress, t.destinationPort))
        if flow.txBitrate is None:
            print("\tTX bitrate: None")
        else:
            print("\tTX bitrate: %.2f kbit/s" % (flow.txBitrate*1e-3,))
        if flow.rxBitrate is None:
            print("\tRX bitrate: None")
        else:
            print("\tRX bitrate: %.2f kbit/s" % (flow.rxBitrate*1e-3,))
        if flow.delayMean is None:
            print("\tMean Delay: None")
        else:
            print("\tMean Delay: %.2f ms" % (flow.delayMean*1e3,))
        if flow.packetLossRatio is None:
            print("\tPacket Loss Ratio: None")
        else:
            print("\tPacket Loss Ratio: %.2f %%" % (flow.packetLossRatio*100))

        print()
        print("Default Parameters----")
        timeFirstTxPacket = flow.timeFirstTxPacket * 1e-9
        print(f"timeFirstTxPacket: {timeFirstTxPacket:.2f} s")
        timeLastTxPacket = flow.timeLastTxPacket * 1e-9
        print(f"timeLastTxPacket: {timeLastTxPacket:.2f} s")
        timeFirstRxPacket = flow.timeFirstRxPacket * 1e-9
        print(f"timeFirstRxPacket: {timeFirstRxPacket:.2f} s")
        timeLastRxPacket = flow.timeLastRxPacket * 1e-9
        print(f"timeLastRxPacket: {timeLastRxPacket:.2f} s")
        print(f"txBytes: {flow.txBytes}")
        print(f"txPackets: {flow.txPackets}")
        print(f"rxBytes: {flow.rxBytes}")
        print(f"rxPackets: {flow.rxPackets}")
        print(f"lostPackets: {flow.lostPackets}")
        print(f"timesForwarded: {flow.timesForwarded}")
        # my dataframe
        # data = [flow.timeFirstTxPacket,flow.timeLastTxPacket,flow.timeFirstRxPacket,
        #         flow.timeLastRxPacket,flow.txBytes,flow.txPackets,flow.rxBytes,
                # flow.rxPackets,flow.lostPackets,flow.timesForwarded]
        # df = pd.DataFrame(data,index=idx)
```

c)

Igor Povarich
Comp Sci 5430: Lab5
10/13/2021

```
TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");

// 1st sink
Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (sinkNode), tid);
InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
recvSink->Bind (local);
recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

// 2nd sink
Ptr<Socket> recvSink2 = Socket::CreateSocket (c.Get (sinkNode2), tid); // 2nd flow
InetSocketAddress local2 = InetSocketAddress (Ipv4Address::GetAny (), 81);
recvSink2->Bind (local2);
recvSink2->SetRecvCallback (MakeCallback (&ReceivePacket));

// 1st source
Ptr<Socket> source = Socket::CreateSocket (c.Get (sourceNode), tid);
InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, 0), 80);
source->Connect (remote);

// 2nd source
Ptr<Socket> source2 = Socket::CreateSocket (c.Get (sourceNode2), tid);
InetSocketAddress remote2 = InetSocketAddress (i.GetAddress (sinkNode2, 0), 81);
source2->Connect (remote2);
```