Mobile Network Project

1. Introduction

Ad-hoc network is a decentralized type of wireless network. The network is ad-hoc because it does not rely on a pre-existing infrastructure such as routers and access points in wired network. Therefore, the stations in network participate in routing by forwarding data for other nodes and in hosting. One of the important features of Ad-hoc network is that wireless mobile ad hoc networks are self-configuring and dynamic which means the nodes in the network are free to move. Because of this, routing protocols used in wired network could not be applied to ad hoc network and researches for ad hoc routing protocols have been proceeded.

Ad-hoc routing protocols usually consists of two parts: a table-driven protocol which updates and maintains the routing table and an on-demand protocol which updates the routing table as the needs of the case demand. Destination Sequenced Distance Vector (DSDV) routing protocol is a representative of table-driven protocol. DSDV is a hop-by-hop distance vector routing protocol requiring each node to periodically broadcast routing updates based on the idea of classical Bellman-Ford routing algorithm. Each node maintains a routing table listing the next hop for each reachable destination, number of hops to reach destination and the sequence number assigned by destination node. The stations periodically transmit their routing tables to their immediate neighbors. A station also transmits its routing table if a significant change has occurred in its table from the last update sent. So, the update is both time-driven and event driven. The routing table updates can be sent in two ways: a full dump or an incremental update.

In the existing DSDV routing protocol, the time at which the routing table is periodically updated at each node is fixed to a constant value. This setting is always constant regardless of the presence or absence of data packets that must be transmitted from one node to another. Therefore, this protocol has a problem that it cannot appropriately deal with the situation of the network because it uses a fixed value.

In this project, the number of packets to be transferred from each node to another node will be recorded to solve the problem. Based on this records, a new method to dynamically implement the time to periodic update of routing table is proposed and the performance of this method will be shown through simulations.

2. A New Approach to Improve Performance of DSDV Algorithm

As mentioned earlier, the existing DSDV routing protocol has the fixed time value of updating routing table. For example, we could easily found that the default time value of periodic updates is 15 seconds in NS3 environment.

It does not matter when there is a lot of traffic in network. However, unnecessary updates are made and control overheads are increased when there is no traffic.

Figure 1. The basic time value of periodic update in NS3

To overcome the problem of DSDV, dynamic time to update routing table according to the presence or absence of data packets is preferable which has been proposed in Adaptive Destination Sequenced Distance Vector (A-DSDV). In each node, A-DSDV will perform following steps:

1. At first, node X checks the number of periodic updates and the number of data packets to be transferred.
2. When the node X get more than three data packets to send while the periodic updates happened twice, the time interval between periodic update will be decreased.
3. When the node X get less than three data packets to send while the periodic updates happened twice, the time interval between periodic update will be increased.

For example, suppose that a node counts the number of updates that occur periodically and the number of data packets to be sent. Also, the default time interval is 5 seconds and data packets are checked every 10 seconds. At 10 seconds, it the number of data packets to be transferred does not exceed two, the time interval will increase to 7.5 seconds, which is 3/2 times of 5 seconds. In the opposite case, the interval will be reduced as 2.5 seconds, which is 1/2 times of 5 seconds. As a result, A-DSDV reduces unnecessary updates to a great extent when there is not much traffic. This overall flowchart of A-DSDV is shown step by step in Figure 2.

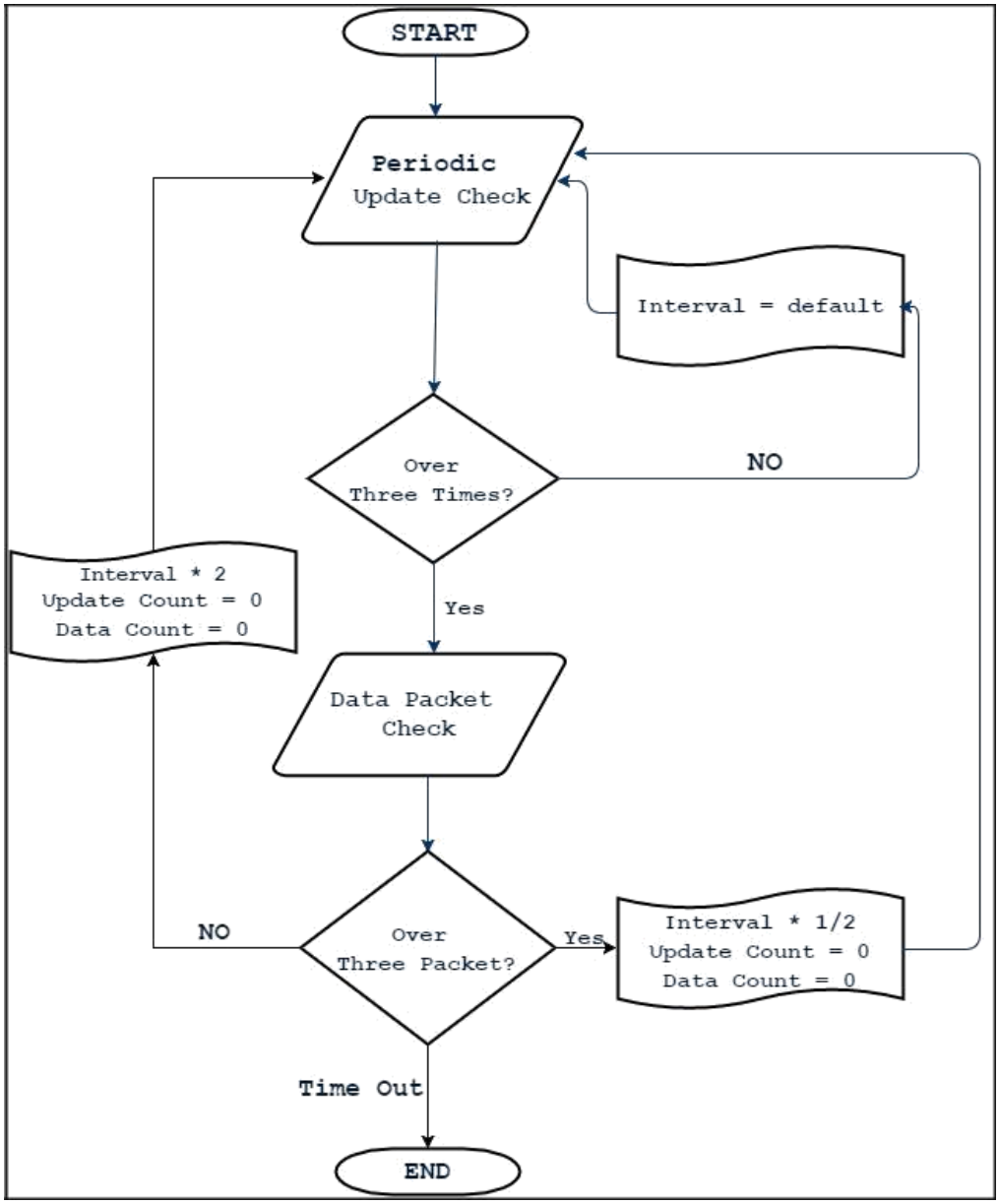

Figure 2. Flowchart of A-DSDV reducing unnecessary updating of table

3. Modification of source code

3.1. Declaring variable

The most important part of A-DSDV is counting the number of periodic updates and data packets to be sent. To count how many packets are coming in while there are two updates, the first thing I did was make the variable in 'dsdv-routing-protocol.h'. The variable for periodic updates is called as 'pupdate_cnt' and the other which is needed to count the packets is named as 'data_cnt'. Also, for initialize these variables, I mentioned those in 'dsdv-routing-protocol.cc'. This modification of code is shown in figure 3 and figure 4.

```
public:
  ////////////////////Elly////////////////////////////////
  uint32_t data_cnt; //counting sending packets from node
  uint32_t pupdate_cnt; // how many periodic update have occurec
  ////////////////////////////////////////////////////////
```

Figure 3. Declaration of variables in 'dsdv-routing-protocol.h'

```
RoutingProtocol::RoutingProtocol ()
  : m_routingTable (),
    m_advRoutingTable (),
    m_queue (),
    m_periodicUpdateTimer (Timer::CANCEL_ON_DESTROY)
{
  data_cnt = 0;
  pupdate_cnt = 0;
  m_uniformRandomVariable = CreateObject<UniformRandomVariable> ();
}
```

Figure 4. Initialization of variables in 'dsdv-routing-protocol.cc'

3.2. Counting the number of data packets and periodic updates

To count how many packets are coming, I firstly found the part which the packets from other nodes are enqueued and count the number. Also, I checked the how many periodic updates of routing table are occurring through the source code. I just added the code on the existing source code, 'dsdv-routing-protocol.cc'. The additions are shown in figure 5 and 6.

```
void
RoutingProtocol::DeferredRouteOutput (Ptr<const Packet> p,
                                      const Ipv4Header & header,
                                      UnicastForwardCallback ucb,
                                      ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p << header);
  NS_ASSERT (p != 0 && p != Ptr<Packet> ());
  QueueEntry newEntry (p,header,ucb,ecb);
  bool result = m_queue.Enqueue (newEntry);
  ////////////////////////////////////////////////////////////////////
  data_cnt++;
  ////////////////////////////////////////////////////////////////////
  if (result)
    {
      NS_LOG_DEBUG ("Added packet " << p->GetUid () << " to queue.");
    }
}
```

Figure 5. Addition of 'data_cnt' in 'dsdv-routing-protocol.cc'

```
void
RoutingProtocol::SendPeriodicUpdate ()
{

  std::map<Ipv4Address, RoutingTableEntry> removedAddresses, allRoutes;
  m_routingTable.Purge (removedAddresses);
  MergeTriggerPeriodicUpdates ();
  m_routingTable.GetListOfAllRoutes (allRoutes);
  ////////////////////Elly/////////////////////////////////////////
  pupdate_cnt++;  //how many periodic updates are occured
  ///////////////////////////////////////////////////////////////
  if (allRoutes.empty ())
    {
      return;
    }
  NS_LOG_FUNCTION (m_mainAddress << " is sending out its periodic update");
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketA
ddresses.begin (); j
       != m_socketAddresses.end (); ++j)
    {
      Ptr<Socket> socket = j->first;
      Ipv4InterfaceAddress iface = j->second;
      Ptr<Packet> packet = Create<Packet> ();
      for (std::map<Ipv4Address, RoutingTableEntry>::const_iterator i = allRoute
s.begin (); i != allRoutes.end (); ++i)
```

Figure 6. Addition of ‘pupdate_cnt’ in ‘dsdv-routing-protocol.cc’

3.3. Applying periodic changing

This is the main part of A-DSDV. In A-DSDV, the algorithm will check only two periodic updates. Thus, in the first check, it shows how many packets have been received up to this point, and this is the first check. The time interval is not changed and is like before. In the second periodic check, if the data count exceeds the threshold, the interval will be reduced. Also, if the data count does not exceeds the threshold, the interval will be increased. Compared to the existing source code, a new method could dynamically implement the time to periodic update of routing table.

```
 if ( pupdate_cnt != 2 )
    {
      if ( data_cnt < 3 )
        {
          std::cout << "                                                    "<<std::en
dl;
          std::cout << " pupdate<3 data<3 m_periodicInterval:" << m_periodicUpda
teInterval << data_cnt <<std::endl;
          std::cout << " This is pupdate_cnt:" << pupdate_cnt <<std::endl;
          std::cout << " This is data_cnt:" << data_cnt <<std::endl;
          std::cout << " ____________________________________5Cut__________"<<std::e
ndl;
          m_periodicUpdateTimer.Schedule (m_periodicUpdateInterval + MicroSecond
s (25 * m_uniformRandomVariable->GetInteger (0,1000)));
        }
      else
        {
          std::cout << "                                                    "<<std::en
dl;
          std::cout << " pupdate<3 data>3 m_periodicInterval" << m_periodicUpdat
eInterval<<std::endl;
          std::cout << " This is pupdate_cnt:" << pupdate_cnt <<std::endl;
          std::cout << " This is data_cnt:" << data_cnt <<std::endl;
          std::cout << "____________________________________5Cut_________"<<std::
endl;

          m_periodicUpdateTimer.Schedule (m_periodicUpdateInterval + MicroSecond
s (25 * m_uniformRandomVariable->GetInteger (0,1000)));
        }
    }
```

Figure 7. The first periodic check in ‘dsdv-routing-protocol.cc’

```
  else if (pupdate_cnt == 2)
    {
      if ( data_cnt < 3 )
        {
          m_periodicUpdateInterval = m_periodicUpdateInterval + (m_periodicUpdateInt
erval / 2);

          std::cout << "                                                    "<<std::endl;
          std::cout << " pupdate == 2,data<3 m_periodicInterval" << m_periodicUpdate
Interval <<std::endl;
          std::cout << " This is pupdate_cnt:" << pupdate_cnt <<std::endl;
          std::cout << " This is data_cnt:" << data_cnt <<std::endl;
          std::cout << " ____________________________________10Cut_________"<<std::endl
;

          pupdate_cnt = 0;
          data_cnt = 0;

          m_periodicUpdateTimer.Schedule (m_periodicUpdateInterval + MicroSeconds (2
5 * m_uniformRandomVariable->GetInteger (0,1000)));
        }
       else
        {
          m_periodicUpdateInterval = m_periodicUpdateInterval - (m_periodicUpdateInt
erval / 2);
          if ( m_periodicUpdateInterval < Seconds(0) )
           {
              m_periodicUpdateInterval = Seconds(1);
           }

          std::cout << "                                                    "<<std::endl;

          std::cout << " pupdate==2,data>3 m_periodicInterval" << m_periodicUpdateIn
terval <<std::endl;
          std::cout << " This is pupdate_cnt:" << pupdate_cnt <<std::endl;
          std::cout << " This is data_cnt:" << data_cnt <<std::endl;
          std::cout << " ____________________________________10Cut_______"<<std::endl;


          pupdate_cnt = 0;
          data_cnt = 0;

          m_periodicUpdateTimer.Schedule (m_periodicUpdateInterval + MicroSeconds (2
5 * m_uniformRandomVariable->GetInteger (0,1000)));
        }

   }
```

Figure 8. The second periodic check in ‘dsdv-routing-protocol.cc’

```
  m_periodicUpdateTimer.Schedule (m_periodicUpdateInterval + MicroSeconds (25 * m_un
iformRandomVariable->GetInteger (0,1000))); //yogi
```

Figure 9. The original part of periodic checking in ‘dsdv-routing-protocol.cc’

4. Simulation and Result

4.1. Simulation

In A-DSDV, the basic problem of DSDV has been improved by reducing the unnecessary number of updating routing table. It could be shown by using the version of NS3. In the simulation, it has been shown that the performance of A-DSDV is much better than DSDV. Also, the performance of A-DSDV stood out in low traffic.

Here it is assumed that, the ad-hoc network consists of 3, 5, 7, 10 and 20 wireless nodes, moving about over 500 X 500 flat space for 200 seconds of simulated time. In order to enable direct and fair comparisons between the DSDV and the A-DSDV, it was critical to challenge the protocols with identical loads and environmental conditions. Each run of the simulator a scenario file as input that describes the exact motion of each node and the exact sequence of packets. Also, to make the low traffic environment, the interval between data packets is set as 4seconds and for the high traffic the interval is set as 1 seconds.

In this scenario, the first node is always the source node and the last node is the destination. For example, if there are 10 nodes, the node #0 is the source and the node #9 is the destination. The following pictures are different depending on the number of nodes.

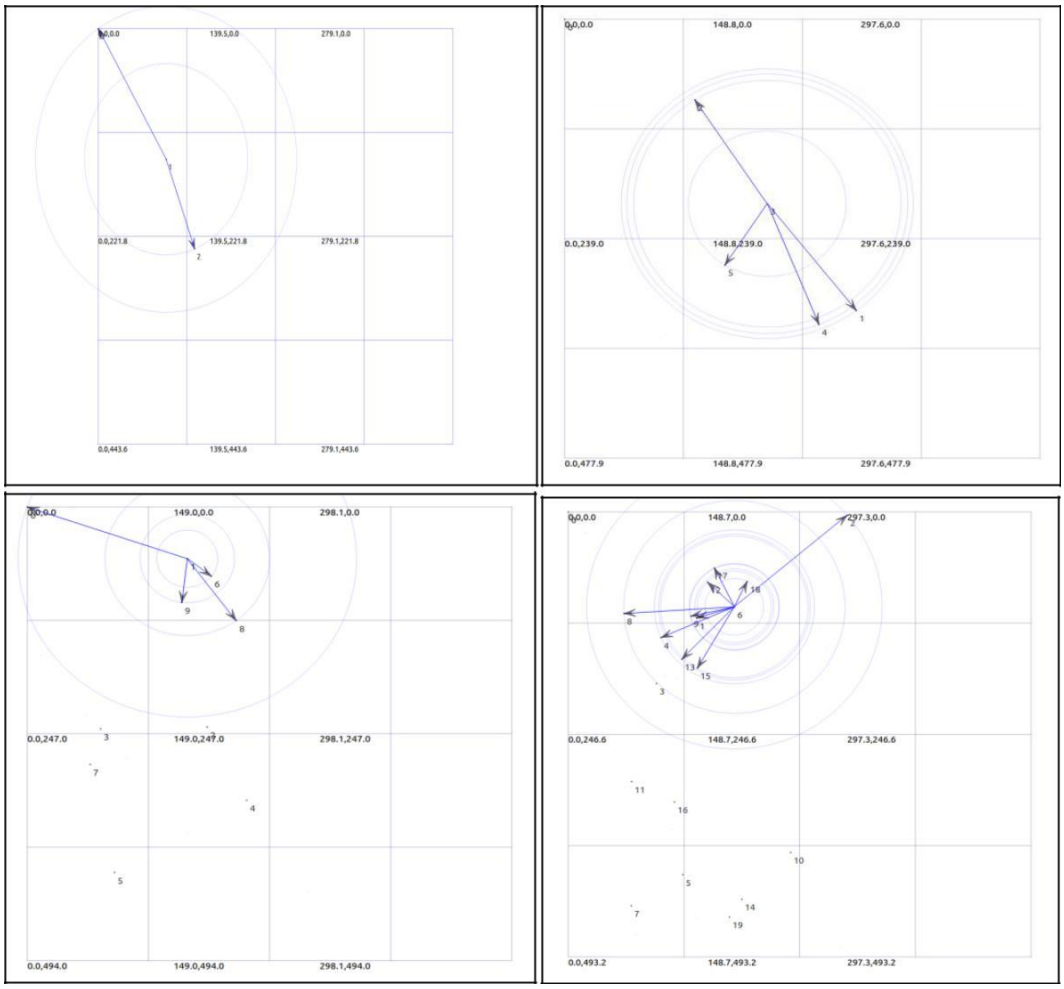

Figure 10. Screen shots when there are 3 nodes, 6 nodes, 20 nodes and 10 nodes in the ad hoc network. Look at the pictures in a clockwise direction.

4.2. Result

A. Control Packet Overhead

For the low traffic, A-DSDV reduces the updates of routing table. Figure 11. Reveals this concept. From the graph, it can be concluded that as the number of nodes increase in a network, number of control packet is reduced in A-DSDV than that of DSDV.

However, in the high traffic, A-DSDV have no impact on reducing the updates of routing table. In particular, the number of control packets suddenly increases in figure 12. This is probably natural. If there is a lot of packet to be sent, the period of updating routing tables will be shorten and have no effect on reducing control packets.

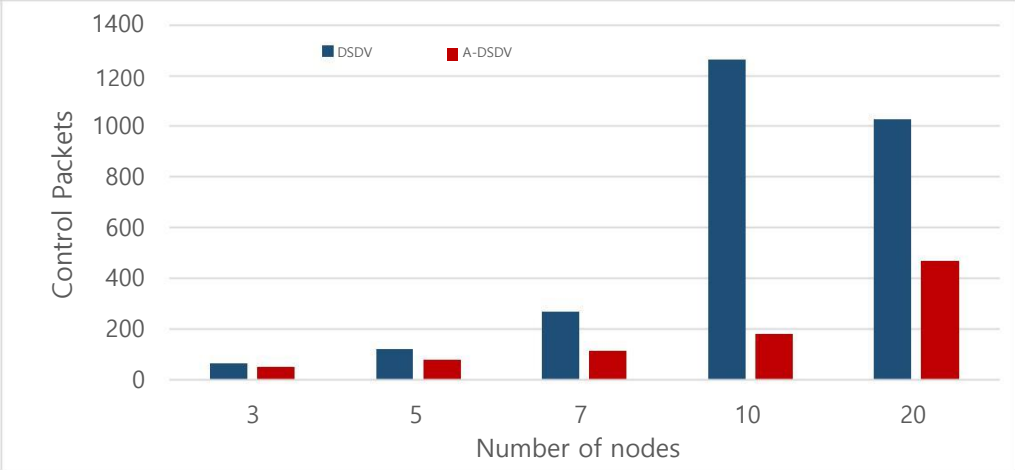

Figure 11. Comparison of DSDV and A-DSDV in case of reducing Control Packets for a network of 3, 5, 7, 10 and 20 nodes in lower traffic.

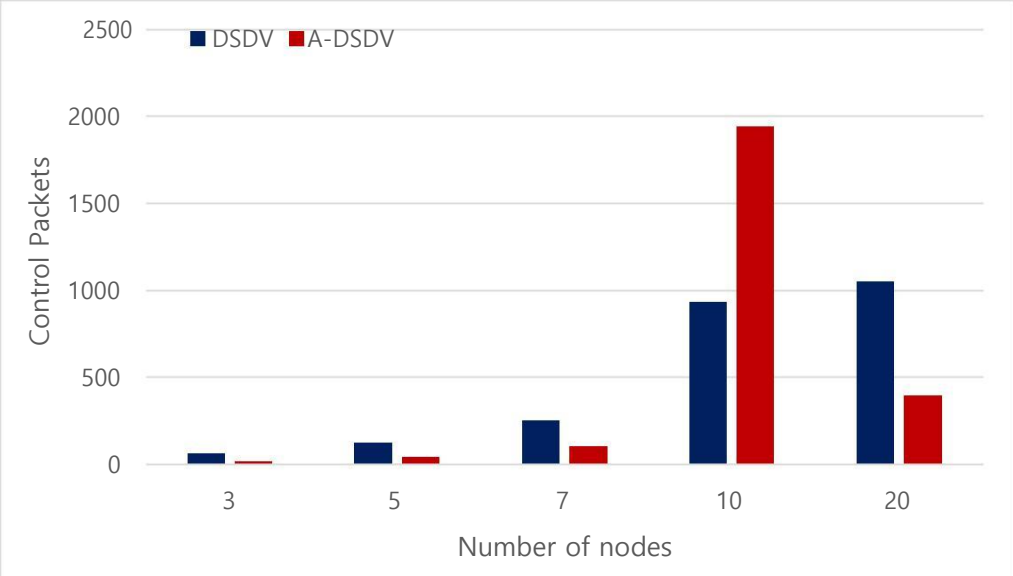

Figure 12. Comparison of DSDV and A-DSDV in case of reducing Control Packets for a network of 3, 5, 7, 10 and 20 nodes in higher traffic.

Also, I compared A-DSDV with the other routing protocol, AODV. From the graph, it can be concluded that when there are relatively less nodes in a network, A-DSDV is more advantageous at the number of control packets. However, as the number of nodes increases, AODV could be better than that of A-DSDV.

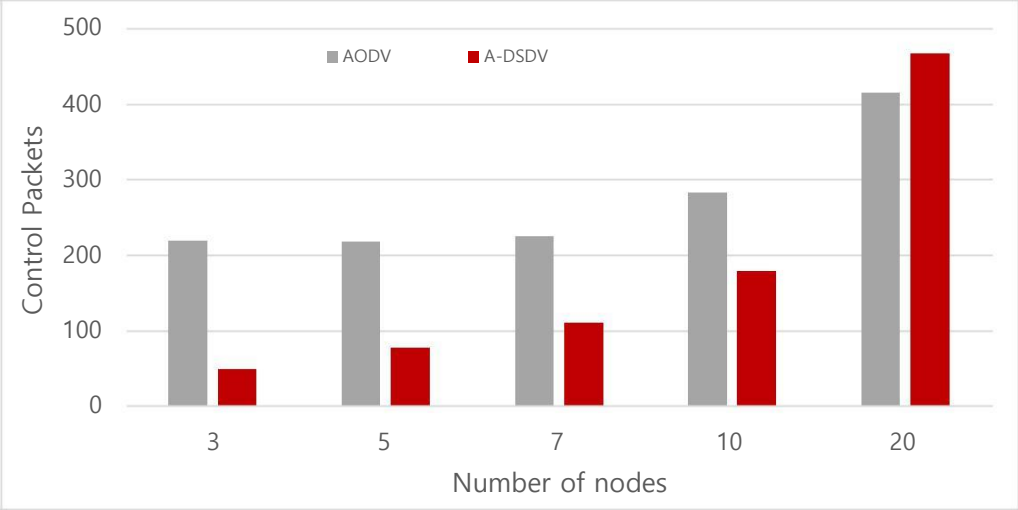

Figure 13. Comparison of A-DSDV and AODV in case of reducing Control Packets for a network of 3, 5, 7, 10 and 20 nodes.

B. Throughput

In the same scenario, the throughput of A-DSDV is closer to DSDV but higher than DSDV as shown in figure 13.

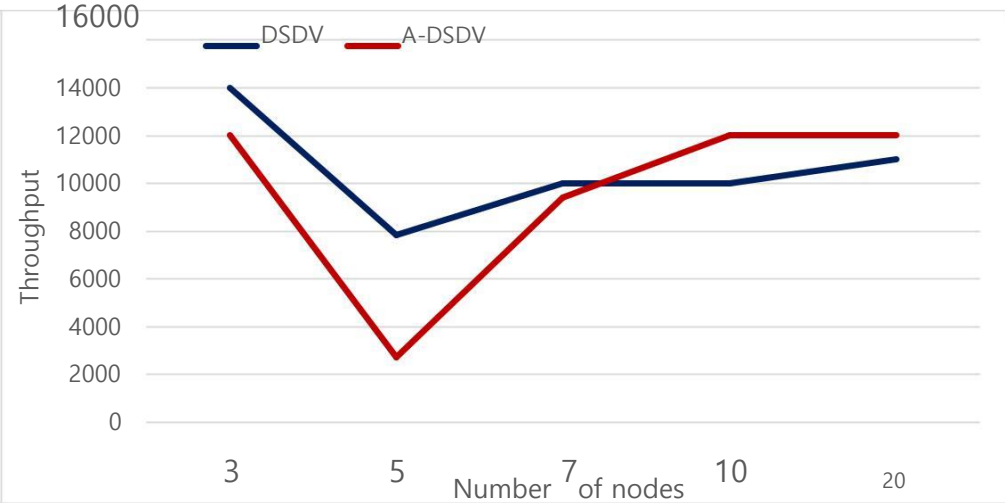

Figure 14. Comparison of A-DSDV and AODV in case of throughput for a network of 3, 5, 7, 10 and 20 nodes.

5. Conclusion

This project has proposed modified DSDV algorithm A-DSDV with a new concept: Reducing the number of control packet in DSDV. Performance of A-DSDV is elevated in respect of some simulation metrics. Unlike other source routing protocols, the A-DSDV observes the number of packets to be sent and quickly changes the time interval of updating routing table. From this algorithm, it was shown that a new method to dynamically implement the time to periodic update of routing table had effect on reducing the number of control packets.

From the simulation, it can be concluded that as the number of nodes increase in a network, number of control packet is reduced in A-DSDV than that of DSDV especially in lower traffic. However, in the high traffic, A-DSDV have no impact on reducing the updates of routing table. In particular, the number of control packets suddenly increases. This is probably natural. If there is a lot of packet to be sent, the period of updating routing tables will be shorten and have no effect on reducing control packets.

Also, there was one more disadvantage of A-DSDV that the number of packets to be transferred next could not be predicted. When I classify the number of incoming packets according to time, sometimes the number of packets arriving at a certain period exceeded the threshold and decreased the interval. However, the number of packets again did not exceed the threshold in the next time period. Therefore, for the more accurate measurements, making a prediction may be a good idea.