Performance Analysis of DSDV Protocol

Andrew Briscoe
School of Computer Science and Software Engineering
UWA-CITS4419
Email: 21332512@student.uwa.edu.au

Caitlin Wood
School of Computer Science and Software Engineering
UWA-CITS4419
Email: 21324675@student.uwa.edu.au

Sanabel Abu Jwades
School of Computer Science and Software Engineering
UWA-CITS4419
Email: 21910099@student.uwa.edu.au

***Abstract*—In this paper, we evaluate the performance of the proactive routing protocol Destination Sequenced Distance Vector (DSDV). The major goal of this study is to analyse the performance of DSDV under low, medium and high mobility scenarios. The performance is analysed with respect to Throughput. Simulation results verify that DSDV can not achieve high throughput levels achieved by other protocols. However, small modifications in the protocol such as having uncooperative nodes, ignoring the sequence number or having longer intervals for periodic updates can improve the throughput under certain mobility levels.**

I. Introduction

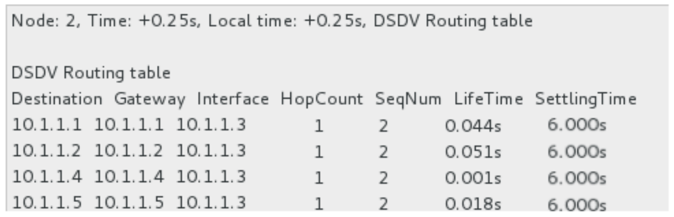
The DSDV routing protocol is an extension of the Routing Information Protocol (RIP) for ad-hoc networks where each node within the network maintains a local copy of the network topology. Information regarding each of the node's possible destinations is held within each node in the form of a *routing table*. If Node A intends to send a packet to Node B, it will run a shortest path algorithm on its routing table so that it can decide which intermediary node it should send the packet to next in order for it to ultimately reach its destination. Since ad-hoc networks are mobile, the network topology is constantly changing. Every time a node senses a movement that will cause a change in its routing table, it must broadcast an update packet so that the other nodes in the network are aware of this change. Furthermore, periodic updates are often sent in order to further reduce contention issues among the nodes. Since nodes are constantly receiving updates from various nodes, a *sequence number* is utilised so that the receiving node can determine which update has the most recent information. It is important to note that the routing information held within each node may not be consistent or correct at any given time. This is because the DSDV protocol is a *distributed algorithm* so each node is only capable of maintaining local information [1][2].

In this project, we will be exploring ns-3s implementation of the DSDV routing protocol. NS-3s DSDV implementation maintains a DSDV routing table as shown in Figure 1 that stores a nodes IP address, hop count to destination, last known sequence number, next hop towards destination and a timestamp for last update received [3]. The DSDV updates are transmitted either periodically or when a received DSDV packet modifies the routing table. By modifying the existing DSDV protocol and measuring the resulting network throughput, we will observe the effects of some of the defining features of the protocol.

In order to conduct some experiments on the DSDV protocol, NS-3s implementation that can be found at [4] is used and modified. In this paper, four main experiments are conducted as well as two additional ones. An extra experiment was conducted based on the suggested future work found in the DSDV literature [5]. Furthermore, a final meta-experiment has been implemented. This repeats experiments 1 to 5 with extra data points. This is an attempt to obtain more information about the nature of the DSDV protocol, especially in extreme scenarios. Below is the list of the investigated questions:

1) What is the effect of using *partial updates* over full routing table dumps?
2) What is the effect of several nodes not forwarding the updates *(not cooperating)*?
3) What is the effect of *incomplete routing* information?
4) What is the effect of *sequence number* on the DSDV protocol?
5) What is the optimal *update interval time*?
6) What extra information can graphs with *extra data points* reveal?

The data collected from the experiments done in this paper show similar performance results for DSDV to those shown in [6]. Results from [6] show that the throughput of the DSDV protocol averages at about 50 percent. This is unlike some other ad-hoc protocols explored in the paper that yielded throughput values closer to 100 percent[1].


```
Node: 2, Time: +0.25s, Local time: +0.25s, DSDV Routing table

DSDV Routing table
Destination  Gateway   Interface  HopCount  SeqNum  LifeTime  SettlingTime
10.1.1.1   10.1.1.1  10.1.1.3       1        2      0.044s     6.000s
10.1.1.2   10.1.1.2  10.1.1.3       1        2      0.051s     6.000s
10.1.1.4   10.1.1.4  10.1.1.3       1        2      0.001s     6.000s
10.1.1.5   10.1.1.5  10.1.1.3       1        2      0.018s     6.000s
```
Fig. 1: Simulation Results

II. Design Details

The performance of DSDV routing protocol is investigated in this study. Several experiments are conducted for this investigation. Below is a detailed description of the system set-up.

[1]Source code and data available at https://github.com/SuperPanda/cits4419-dsdv

A. System Parameters

Simulations are conducted using Network Simulator-3 (NS-3), version 3.25. This is the latest version of the network simulation package at the time of this work. The parameters used in the simulations are listed in Table 1. These parameters are defined as follows:

- *Area for mobility model*: defines the maximum area that bounds the mobility of all nodes during the simulation.
- *Number of Nodes*: is the total number of nodes in the simulated network.
- *CBR rate*: is the constant bit rate at which each node generates traffic.
- *Settling Time*: is the time a node would wait before advertising a change in its neighborhood in high mobility scenarios. This is done to reduce the effect of route fluctuations.
- *Periodic Update Time*: defines the interval at which periodic updates are sent out. In this update, a node broadcasts out its entire routing table (default 15s).
- *Start Data Transition Time*: is the times that CBR nodes will begin generating traffic. A significant time gap here (approximately 50 seconds), gives the DSDV protocol adequate time to converge. That is, at the time that nodes begin generating traffic, each node has some idea about the topology of the network.
- *Simulation Time*: the number of discrete time steps for which the simulation will run.

Table 1: System Parameters

Parameter Name	Value
Area for mobility model	1500m x 1500m
Number of Nodes	50
CBR rate	1Mbps
Settling Time	6 seconds
Periodic Update Time	15 seconds
Start Data Transmission Time	50s
Simulation Time	100s

B. Experiments Set-up

The following set-up is adopted for all experiment conducted using NS-3 in this paper:

- The density of nodes is set to 50 nodes in an area of 1500X1500 meters. This is achieved by allocating random positions for 50 nodes using an ns3::RandomRectanglePositionAllocator with maximum dimensions of 1500x1500.
- The mobility model for all nodes is set to random waypoint mobility model.
- Three mobility speeds are used for performance comparisons: 1 m/sec, 10 m/sec and 20 m/sec. Mobility of nodes is configured by setting the mobility model of the Mobility Helper class to ns3::RandomWaypointMobilityModel with user-specific speeds.
- The number of sources for CBR traffic is set to 10, 20 or 30 nodes. This is achieved by installing OnOff application to 10, 20 or 30 nodes only.

Figure 2 shows an example simulation set-up as generated by NetAnim.

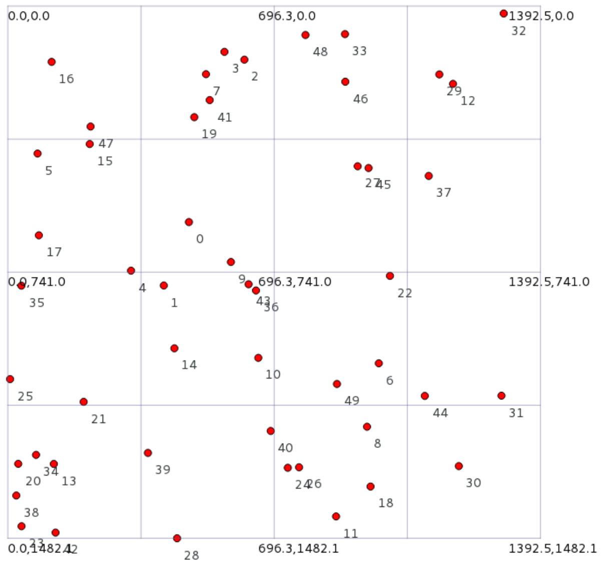


Fig. 2: 50-node random allocation in a 1500X1500 meters area

C. Performance Metric

In this paper, throughput is considered as the performance metric to compare the performance of different protocol variations. Throughput is the rate of successfully transmitted data packets in a unit time in the network during the simulation [6].

IP flow monitoring is enabled on all nodes to gather statistics about the networks throughput in each experiment. The number of received packets in each flow in the network is summed to get the total received packets. The overall throughput is then calculated by dividing the total received packets by the time gap, in seconds, between the first sent packet and the last received packet. To obtain this throughput data, ns3::FlowMonitorHelper was used.

III. Protocol Modifications Details

For each of the investigations conducted, some modifications needed to be made to the existing implementation of the DSDV protocol.

A. Experiment 1 - Effect of using partial updates versus full updates

The first experiment involves comparing the performance of the existing DSDV protocol and a modified DSDV protocol that only sends full updates. As it stands, DSDV allows nodes to send two kinds of routing table updates to other nodes. These are *periodic updates* and *triggered updates*. Periodic updates broadcast the entire routing table and are sent out after m_periodicUpdateInterval(default:15s) by each node. Triggered updates, on the other hand, are simply partial updates that only contain the columns of the routing table that have been modified. These are only sent out when a node receives a packet that results in a change of its routing table, irrespective of what that change might be [7]. In ns3s DSDV implementation, the updates described are sent by calling one of the two following functions:

- Void RoutingProtocol::SendTriggeredUpdate()
- Void RoutingProtocol::SendPeriodicUpdate()

A simple modification that was made in order to ensure that only full updates are sent. By duplicating the code from the SendPeriodicUpdate function in the SendTriggeredUpdate function, every update now occurs as if it is a Periodic Update. Therefore, the *whole routing table* is sent each time an update is triggered.

B. Experiment 2 - Effect of some updates not being forwarded

This experiment involves introducing 5, 10 and 15 nodes that don't send any updates (triggered or periodic) and comparing performance with the original DSDV protocol. This would mean that nodes within the network might have less accurate information because none of the nodes will receive a notification of any changes in the routing table of the designated uncooperative nodes. This was implemented by randomly choosing the specified number of uncooperative nodes from a list. Therefore, if we are introducing 5 nodes that do not send any updates then 5 nodes will be randomly chosen from a list of all the nodes. This way, the uncooperative nodes that are chosen are not biased towards or away from the CBR generated nodes. An IsDisabled flag has also been set up. If a node is selected to be uncooperative then that node will have its IsDisabled flag set to true.

Both update functions that were described in Experiment 1 (for periodic updates and triggered updates) have a socket.sendTo and a socket.send function. The purpose of these functions is to send out routing tables to other nodes. If the node that calls the update function has an IsDisabled = true, then both of the socket.send and socket.sendTo are disabled. If these functions are disabled then no update is sent out to the other nodes.

C. Experiment 3 - Effect of incomplete routing information

In this experiment, a column is deleted randomly from the routing table with probabilities p=0.4, 0.5 and 0.6. The idea of this is to learn the effects of incomplete routing information. At *configuration time*, those nodes are looped through and will be selected to ignore a column with the specified probability. So, if p=0.4 this would mean that approximately 40% of the nodes will have an ignored column in their routing table. If a node had been selected to ignore a column, a single column will randomly be chosen from that nodes routing table.

From here, the column number of the ignored column for each node is passed into that nodes instance of the DSDV routing protocol. It as added as an attribute in the dsdv-routing-protocol.cc file using the following code:

```
.AddAttribute ("IgnoreColumn",
"Routing Table Entry to ignore",
 IntegerValue(-1),
 MakeIntegerAccessor
(&RoutingProtocol::m_ignoreColumn),
 MakeIntegerChecker<int32_t> ());
```

Where the default value is -1. This means that if there is no column to be ignored on a node, this attribute will be set to -1.

The DSDV protocol has been modified so that whenever an interaction is made with the routing table (i.e. an add route, delete route, lookup route, etc.), the column number is passed into the function as a parameter.

For example,

```
m_advRoutingTable.AddRoute
(rmItr->second,m_ignoreColumn);
```

In the code above, the AddRoute method is being called and a column number is passed in as a parameter. Then in the dsdv-rtable.cc file, the corresponding function takes the m_ignoreColumn parameter

```
bool
RoutingTable::AddRoute
(RoutingTableEntry& rt, int32_t ignoredColumn)
{
  if (!IsValidColumn
     (rt.GetDestination(),ignoredColumn))
             return false;
     return AddRoute(rt);
}
```

And if the column is to be ignored, then that function call is simply ignored.

D. Experiment 4 - Effect of sequence numbers

In this experiment sequence numbers are ignored. According the the ns3 documentation [4], the original DSDV protocol will accept an update if the sequence number attached to the update is higher than the current sequence number. If the sequence numbers are equal, however, the update with the lowest hopCount value is taken. This experiment was implemented by focusing on the following function:

```
Void RoutingProtocol::RecvDsdv
     (Ptr<Socket> socket)
```

This function allows the node to receive DSDV packets and routing table updates from other nodes. If a routing table update is received, the sequence number of the update will be checked. This is achieved using the following conditional statement in the original DSDV implementation.

```
// (1)
if (dsdvHeader.GetDstSeqno () >
     advTableEntry.GetSeqNo ())
```

If the above statement is true (that, if packet has a better sequence number) then the update will be received regardless of the hop count. If the received sequence number is equal to the current sequence number, however, the following statement will hold true.

```
// (2)
else if (dsdvHeader.GetDstSeqno () ==
        advTableEntry.GetSeqNo ())
```

In this case, the update will be received only if it has a better hop count than the previously received update.

For this experiment, the goal is to ignore sequence numbers while retaining hop count information. Therefore, statement (1) described above was set to if(false) so that this whole block was ignored. Meanwhile statement (2) is set to if(true). This way, every time a packet is received, the hop count will be checked and the update packet will be accepted or declined based on the hop count rather than the sequence number.

E. *Experiment 5 - Finding the Optimal Update Interval Period*

This experiment is about modifying the periodic update time so that an optimal setting can be observed. This experiment is based on the future work suggested by the authors in [5]. The periodic update time, as described earlier, is the time interval at which periodic routing table updates are sent. To implement this experiment, different values of the *m_periodicUpdateInterval* parameter were iterated through and the results were observed for each setting. This parameter has a default value of 15 seconds in the ns3 implementation.

F. *Experiment 6 - A Meta-Experiment In Simulation Scaling*

This final experiment involved scaling up experiments 1-5 to see what further information can be gathered about the nature of the protocol in extreme situations. This experiment was implemented by creating utilities to run 195 simulations across 15 computers, taking a couple of hours. Using the existing code from experiments 1-5, extra trials were added by iterating over an increased number of values for some of the parameters.

By making the code, data, all tools and utilities used in these experiments, other users are invited to use these methods to generate larger datasets, and to check that results are replicable. By scaling the simulations, we found new noteworthy features in the experimental data. These will be further discussed in the results section below.

IV. Results

A. *Experiment 1 - Effect of using partial updates versus full updates*

This experiment yielded some very interesting results. Theoretically, by switching to full updates throughput is expected to decrease. This is because a full update is much larger than a partial update. If every node is sending full routing table updates, it is expected that the traffic generated by these update packets will consume the majority of the channels bandwidth.

The experimental results (as shown in Figure 3) show that sending full updates only had less of an effect when node mobility is low. This can easily be explained as when there is low node mobility, less partial updates (triggered updates) are sent since less nodes are experiencing changes to their routing tables. Therefore, if these partial updates are switched to full updates, the change not noticeable. For a moderate node mobility speed, on the other hand, a significant decrease in bandwidth can be observed. This is because triggered updates will be sent much more often and if these update packets are very large, the channel becomes flooded.

It is important to note that these observation are most prominent when there are fewer nodes generating CBR traffic. This is because, once the number of nodes generating traffic is high, the channel becomes quite flooded and DSDV performs very poorly. Therefore, any changes have less of an impact in this region as a very large percentage of the packets are being dropped. This pattern is also the case in experiments 2 to 6.

B. *Experiment 2 - Effect of some updates not being forwarded*

It is expected that when some nodes refuse to send update packets, other nodes may have an inaccurate view of the network topology. For example, as shown in Figure4 if node A moves into the range of node B and if node B is an uncooperative node, it will not inform nodes C and D that node A is within reach. Therefore, when making routing decisions, nodes C and D will not be aware that any packets can reach node A. This would theoretically cause throughput degradation as more packets will be dropped unnecessarily.

The experimental results, on the other hand, are very surprising. It appears that when uncooperative nodes are included, there seems to be a general throughput increase. This is especially the case when the number of nodes generating CBR traffic is low. This observed improved performance may be the case for similar reasons that AODV has improved performance. AODV is similar to DSDV, but it doesn't send updates to everyone. Instead, nodes using AODV access routing information in an on request manner [8]. This means that there are much less stale and old route update packets flooding the network at any given time. It is reasonable to assume that the researchers who came up with AODV may have noticed an anomaly with DSDV. The improved throughput could be due to the fact that update packets are less likely to flood the channel when less nodes are sending them, therefore more of the generated CBR traffic is able to reach its destination with less chance of collision.

C. *Experiment 3 - Effect of incomplete routing information*

With a column ignored in some of the routing tables, the overall network will have some incomplete routing information. With incomplete routing information, some nodes may not be able to forward their packets to their intended destination. Alternatively, some packets may not be able to take the shortest path to their destination as some intermediary nodes may be missing from their routing tables. Overall, it is expected that these issues would result in decreased throughput. Results from this experiment are shown in Figure 5.

The results are very similar to Experiment 2 above. For faster node mobilities, it appears that there is a throughput increase when the number of nodes generating CBR traffic is

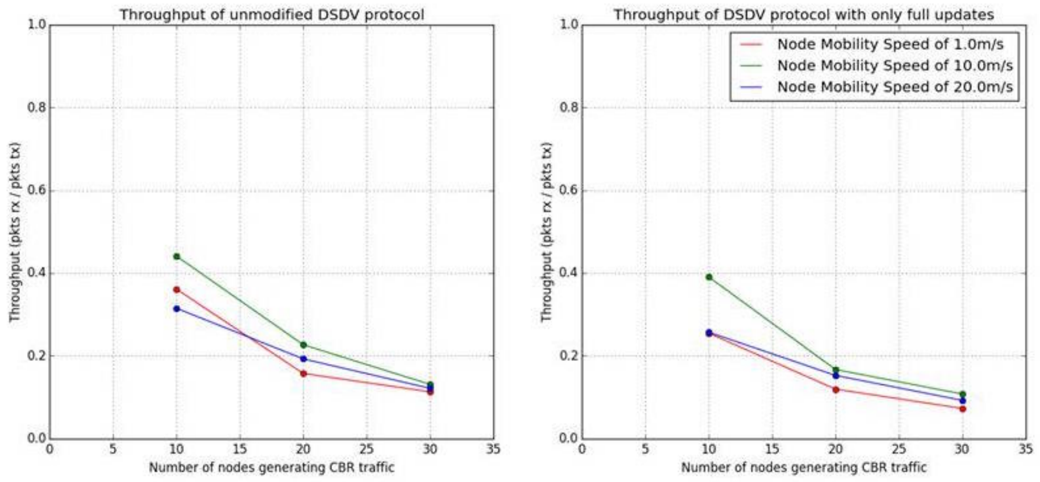


Fig. 3: Experiment 1 results - Effect of using partial updates versus full updates

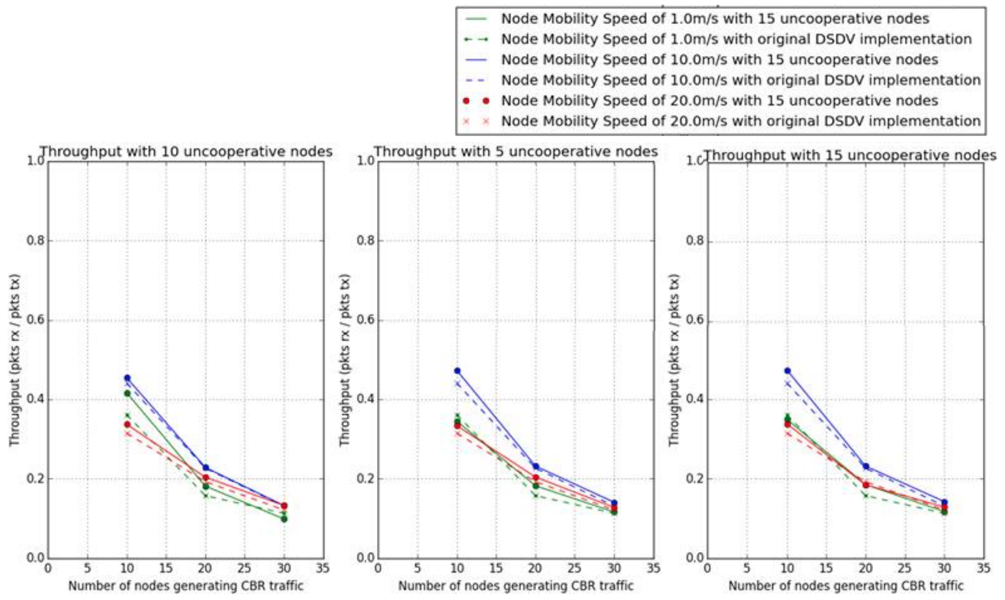


Fig. 4: Experiment 2 results - Effect of some updates not being forwarded

low. This could be because less triggered updates are being sent at these faster mobilities than in the original DSDV protocol. If a node detects a change to a column that is being ignored, this will not trigger an update. This means that less of the channel is utilised with update packets. Interestingly, when the number of CBR nodes that are generating traffic increases, this throughput improvement is no longer prominent. This could be due to an increased number of packets that are not reaching their destination due to the incorrect information in the routing tables. Alternatively it could be since packets are not taking the shortest paths to their destination and spending too much time in the channel thus are more likely to collide with one another.

D. Experiment 4 - Effect of sequence numbers

Sequence numbers are included in the DSDV protocol with the intention of the issues experienced with infinite looping in the conventional distance vector routing protocol [1]. Furthermore, sequence numbers help nodes to maintain the most recent routing information that it has received from a particular node. It is expected that with sequence numbers ignored, nodes within the network may be attempting to retain old routing information. This means that they're more likely to an inaccurate representation of the network topology in their routing tables. This would theoretically result in a decrease of throughput.

The experimental results in Figure 6 show that with low node mobility there is an overall decrease of throughput in the network. This result was to be expected because if a node has an old and inaccurate view of the network topology, it may be retaining that old information for a very long time. This is because with low node mobility, there are very little triggered updates happening and the network is relying on periodic updates in order for information to be updated.

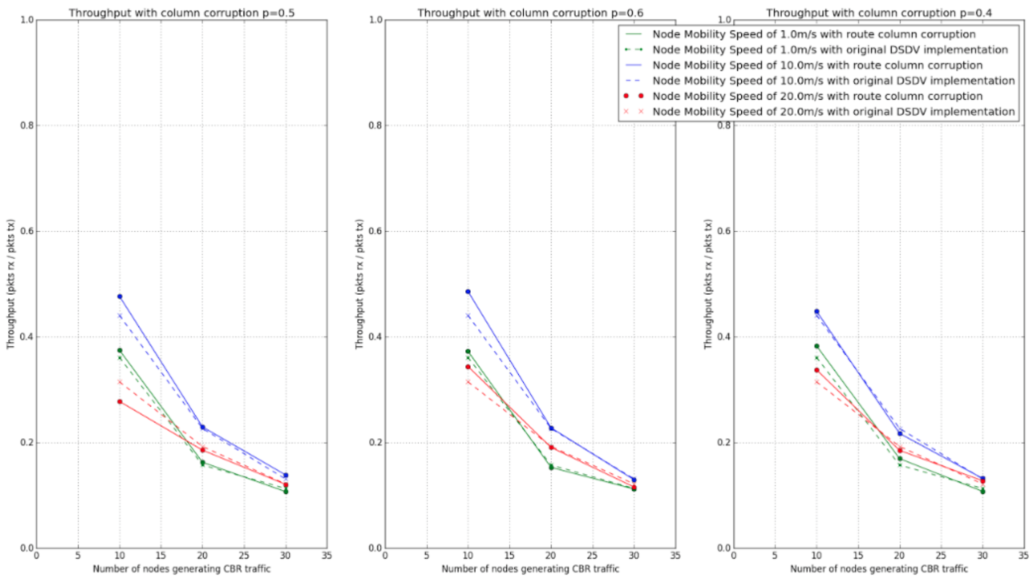


Fig. 5: Experiment 3 results - Effect of incomplete routing information

Furthermore, when a periodic update occurs, every node has to publish its routing table. If, at this time, a node is retaining old information. They will publish that old information to the other nodes in the network. Meaning that they all will have old information.

When node mobility is higher, however, the experimental results actually show an increase in throughput when sequence numbers are ignored. This is because old update packets will have less of an impact because new, triggered, updates are likely to be received very soon.

E. *Experiment 5 - Finding the Optimal Update Interval Period*

This experiment involved modifying the update interval period to find an optimal setting. The default setting used in ns3s implementation is 15 seconds. The results in Figure 7 show that when the periodic update interval is short (at 10 seconds), there is a slight decrease in throughput. This is because periodic updates can take up lots of bandwidth because each node will send out its entire routing table. This means that packets are more likely to collide and be dropped before reaching their destinations.

However, when the periodic update interval is at 20 seconds (higher than the default setting), there is a clear throughput increase for low numbers of CBR traffic and high node mobility. This is because the network topology is changing very rapidly and there are already many partial updates happening. This means that nodes rely less on periodic updates to keep their routing tables up to date. With less periodic updates flooding the network in this situation, packets are less likely to collide with one another and are more likely to reach their destination. The experimental results also show that with a long periodic update interval (20 seconds) and low node mobility, throughput degradation is likely to be experienced. This is because very few triggered updates are occurring and the nodes heavily rely on periodic updates.

It was found that with fast node mobility, it is better to have a longer periodic update interval. However, with a low node mobility, an intermediate update interval (approximately 15 seconds) is appropriate. From the learnings taken from this experiment, those who are implementing an ad-hoc network may be able to take some assumption about the approximate mobility of the nodes that will be in the network and make an informed decision on the periodic update interval that they set. This way, greater throughput can be achieved from the DSDV protocol.

F. *Experiment 6 - A Meta-Experiment In Simulation Scaling*

The plots shown in Figures 8 to 12 are experiments 1 to 5 shown with extra data points. This method of scaling has shown to be very beneficial when conducting studies.

By scaling the number of simulations, we can now make some new observations and gain some new insights. For example, only using full updates (as per Experiment 1) provides superior performance in situations of high mobility and low traffic channels. High mobility means missing partial updates are more likely to occur; so when updates do arrive, receiving the full routing table is much more useful.

V. Conclusion

In this paper, the DSDV protocol has been run under many different scenarios with many different settings. Very interesting and sometimes counter-intuitive results have been produced. This was a valuable exercise because an in depth understanding of the nature of the DSDV protocol has been observed. Taking learnings from this paper, those wishing to implement an ad-hoc network using the DSDV protocol can make informed decisions about the optimal settings in their own context. This way, they could potentially obtain some much greater throughput and performance from the DSDV protocol.

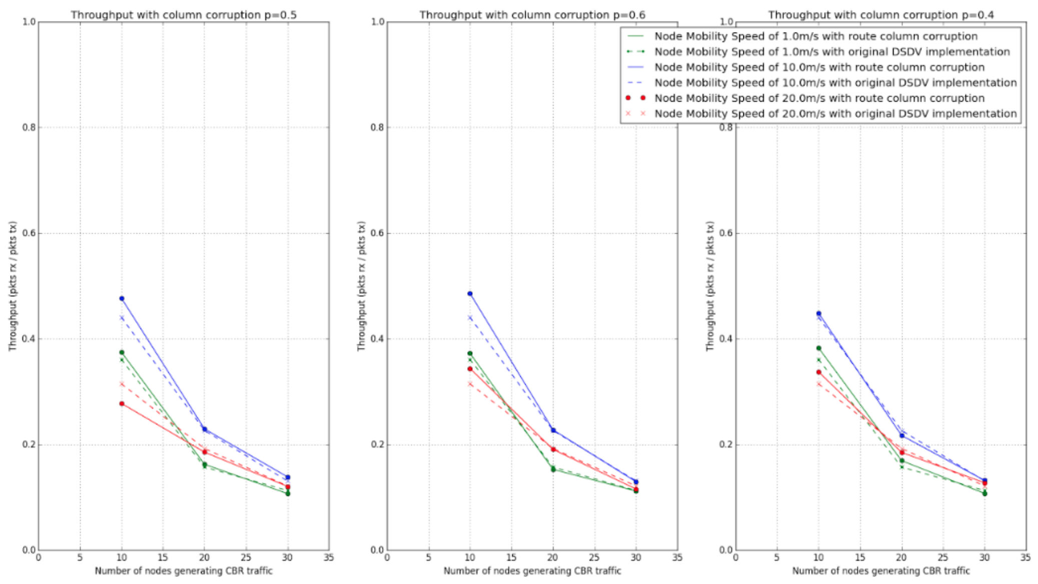


Fig. 6: Experiment 4 results - Effect of sequence numbers

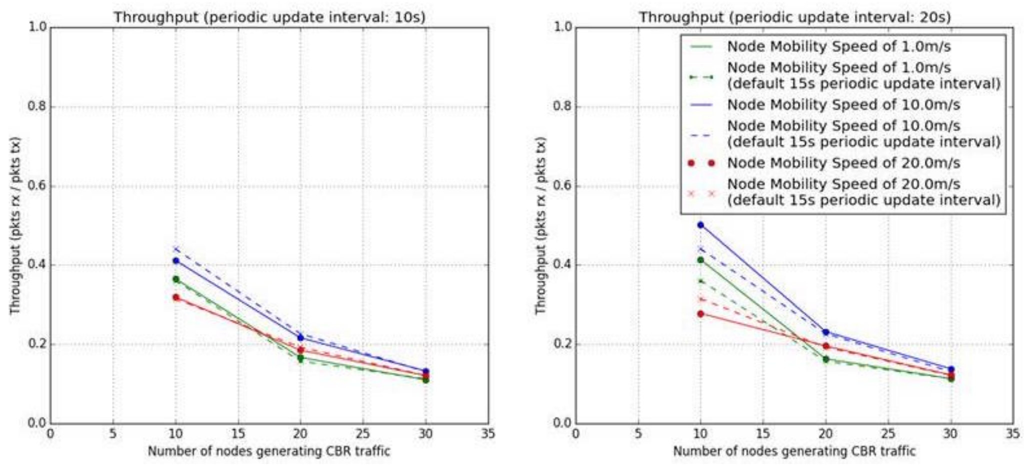


Fig. 7: Experiment 5 results - Finding the Optimal Update Interval Period

REFERENCES

[1] Guoyou He, "Destination-sequenced distance vector (dsdv) protocol.networking laboratory, helsinki university of technology," pp. 1–9, 2002, Accessed: 2016-11-04.

[2] CE Perkins, *Ad Hoc Networking*, Addison-Wesley, 2001.

[3] ns 3 project, "Dsdv routing," https://www.nsnam.org/docs/models/html/dsdv.htmldsdv-routing-overview, 2011, Accessed: 2016-11-04.

[4] ns 3 project, "Dsdv routing protocol source code," https://www.nsnam.org/doxygen/dsdv-routing-protocol_8cc_source.html, 2011, Accessed: 2016-11-10.

[5] P. Bhagwat C. Perkins, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers.," *ACM SIGCOMM Computer Communication Review*, vol. 4, pp. 234–244, 1994, Accessed: 2016-11-10.

[6] Nirbhay Chaubey Akshai Aggarwal, Savita Gandhi, "Performance analysis of aodv, dsdv and dsr in manets," vol. 2, pp. 176–177, 2011, Accessed: 2016-11-07.

[7] ns 3 project, "Dsdv routing model library," https://www.nsnam.org/docs/models/html/dsdv.html, 2011, Accessed: 2016-11-07.

[8] B. Jagdale and D.Javale P. Patil, P.Lahane, "Analysis and comparison of distance vector,dsdv and aodv protocol of manet," *International Journal of Distributed and Parallel Systems*, vol. 2, pp. 121–131, 2012, Accessed: 2016-11-10.

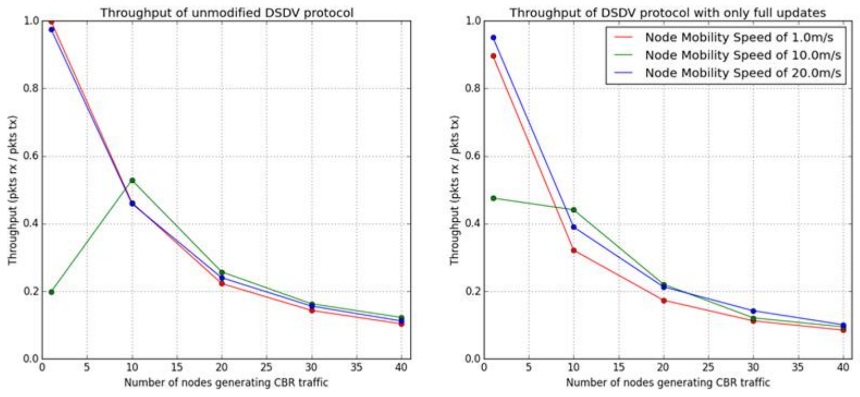


Fig. 8: Experiment 1: Effect of using partial updates versus full updates with extra data points

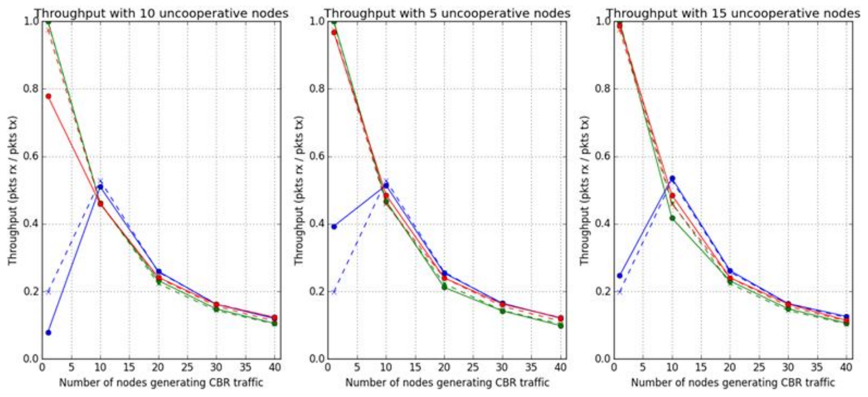


Fig. 9: Experiment 2: Effect of some updates not being forwarded - with extra data points

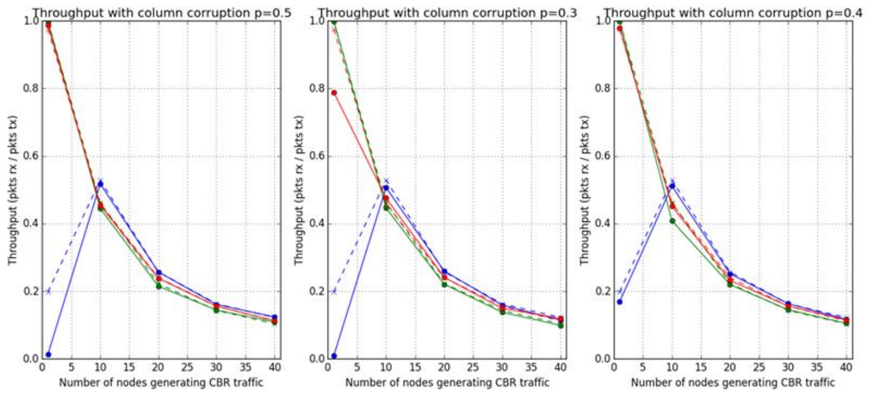


Fig. 10: Experiment 3: Effect of incomplete routing information - with extra data points

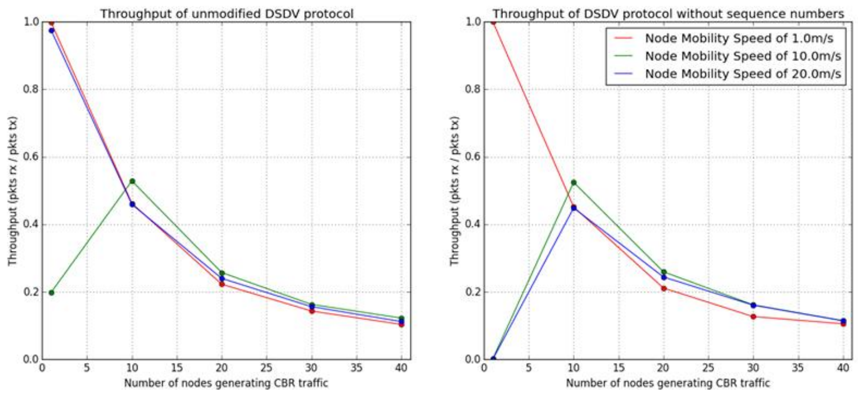


Fig. 11: Experiment 4: Effect of sequence numbers - with extra data points

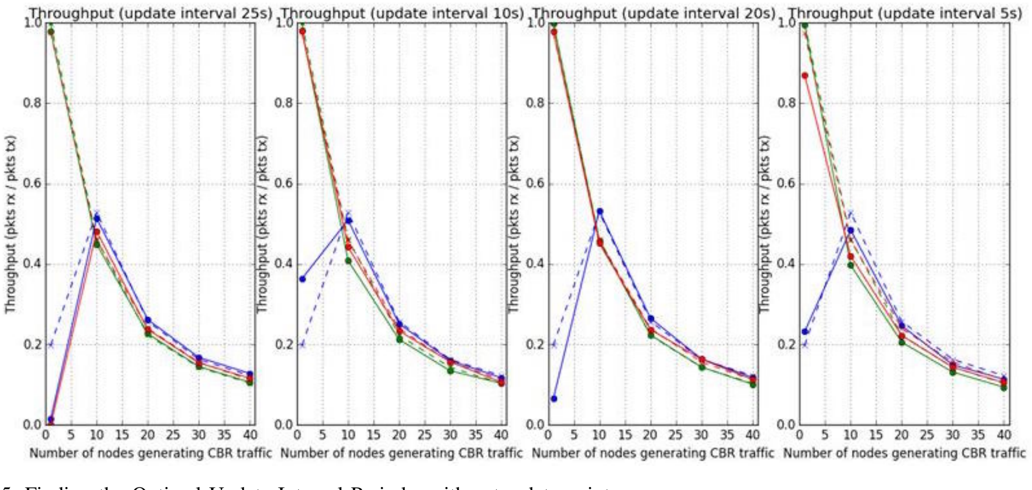


Fig. 12: Experiment 5: Finding the Optimal Update Interval Period - with extra data points