Implementation of Epidemic Routing with IP Convergence Layer in ns-3

Justin P. Rohrer and Andrew N. Mauldin
Naval Postgraduate School
Monterey, California
https://tancad.net/
jprohrer@nps.edu

ABSTRACT

We present the Epidemic routing protocol implementation in ns-3. It is a full-featured DTN protocol in that it supports the message abstraction and store-and-haul behavior. We compare the performance of our Epidemic routing ns-3 implementation with the existing implementation of Epidemic in the ONE simulator, and discuss the differences.

CCS CONCEPTS

• **Networks → Routing protocols**; • **Computing methodologies → Model development and analysis**;

KEYWORDS

Epidemic routing, Epidemic model implementation, DTN, disruption-tolerant networking, delay-tolerant networking, MANET routing, ad-hoc networks, ns-3, network simulation

ACM Reference Format:
Justin P. Rohrer and Andrew N. Mauldin. 2018. Implementation of Epidemic Routing with IP Convergence Layer in ns-3. In *Proceedings of the 2018 Workshop on ns-3 (WNS3 2018), June 2018, Surathkal, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3199902.3199907

1 INTRODUCTION

In this work we present the implementation of the Epidemic routing protocol [11] in ns-3 [1]. While several Mobile Ad-hoc Network (MANET) routing protocols are available in ns-3, it does not currently include any Disruption Tolerant Network (DTN) protocols, and development of such protocols typically proceeds with other simulators, such as The ONE simulator [4]. In various routing simulation efforts [7–9] we have encountered the need to simulate both MANETs and DTNs in the same simulator, and so have developed an architecture for integrating the two. Since there are no other DTN routing protocols in ns-3, we evaluate our results by comparing against the same simulation run in the Opportunistic Network Environment (ONE) simulator.

DTN routing protocols typically deal with "messages" as opposed to "packets". Messages may range in size from a few kB to hundreds of MB, which means they need a mechanism for fragmenting and reassembling these messages on a hop-by-hop basis to meet lower layer requirements. DTN routers also buffer messages if a next-hop is not available, and may replicate messages to multiple next hops. These features differentiate DTN routing protocols from MANET or traditional Internet Protocol (IP) routing protocols. This also leads to various design architectures. DTNs are often implemented as an application-overlay network with Convergence Layer Adapters (CLAs) for particular network types, instead of a native layer-3 routing protocol. The application-layer approach (e.g. the Bundling Protocol [10]) has advantages in ease of development and flexibility, but prevents the overlay from interoperating with traditional IP applications. Due to these limitations, we implement DTN routing protocols as native ns-3 routing protocols so they can be used with the existing ns-3 application classes. In addition we have designed an IP convergence architecture that supports DTN messages made up of multiple packets. Our implementation includes application classes to generate such messages. All of this code is available from our website[1], and we are in the process of contributing it to the ns-3 distribution.

2 PRIOR WORK

This is not the first attempt at implementing the Epidemic routing protocol in ns-3. In 2015 Alenazi *et al.* published their ns-3 implementation of Epidemic [2], the code for which began the review process for inclusion in ns-3 in 2013, however that process appeared to stall in 2015, and the authors did not respond to our inquiries about the status. The existing code however was available online and we attempted to use this version for our DTN simulations, however limitations in this implementation rapidly became apparent and necessitated developing our own implementation. Since Alenazi's implementation routes only atomic packets, not complete messages, we will refer to his implementation as PacketEpidemic in the remainder of the paper, for ease of reference.

The PacketEpidemic implementation for Network Simulator 3 (ns-3) implements the Epidemic logic discussed in [11], however it possesses many limitations. As mentioned, PacketEpidemic handles individual packets instead of messages. PacketEpidemic's node discovery mechanism results in incorrect operation for nodes with large buffers, where the nodes repeatedly resend messages that have already been transferred. Also, PacketEpidemic does not support control message fragmentation, so control messages are limited to the size of one User Datagram Protocol (UDP) packet. Large message buffers can generate summary vectors that exceed

WNS3 2018, June 2018, Surathkal, India
2018. ACM ISBN 978-1-4503-6413-3/18/06…$15.00
https://doi.org/10.1145/3199902.3199907

[1] https://tancad.net/projects/dtn-simulation/

the size of a UDP packet and when this happens the ns-3 simulation crashes. The lack of control packet fragmentation restricts the number of messages that a node can handle. Our ns-3 Epidemic implementation addresses the need for control packet fragmentation and as a result, the control packet headers, node discovery, and data handling, are significantly different from PacketEpidemic.

3 CODE STRUCTURE

We note that implementing Epidemic routing was not our end goal, but a stepping stone to the implementation of many other DTN routing protocols in ns-3, so some design decisions take into account not just epidemic, but other protocols as well. We borrowed the top-level code structure from Alenazi's Epidemic [2], so all ns-3 DTN routing protocols are broken into four main classes. Our Epidemic implementation forwards groups of data packets called messages instead of atomic packets. Section 4 describes how the protocols generate messages and segment them into individual packets. This section provides an overview of the code structure. The DTN routing protocol contains a group of packet classes, packet queue class, queue entry class, and routing protocol class. The Unified Modeling Language (UML) diagram in Figure 1 shows the relationships functions and attributes used by all ns-3 Epidemic protocols to implement DTN logic. We also borrow from our prior experience implementing DSDV for ns-3 [6].

3.1 Packet Classes

The packet classes are the packet header declarations used by the DTN protocols. Each packet header is its own class because the packet header is a data structure that defines a packet. The packet classes inherit ns-3's `Header` class. The `Header` class defines a packet and provides the interface for other classes to interact with the packets. The routing class and packet queue class interact with the packet class to read and write packet headers. Since packet headers have various types of information, they have their own accessors and mutators. However, all packet header declarations require a `Serialize` and `Deserialize` function because of the `Header` class. The `Serialize` function writes the header information to a packet buffer, and the `Deserialize` function reads from a packet buffer. These functions are required because ns-3 passes packets between nodes as byte arrays. When a node receives a packet, it uses the `Deserialize` function to obtain the header information. The `Print` function permits another class to print the header to the screen or log file.

3.2 Packet Queue and Queue Entry Classes

The Packet Queue Class manages a node's message buffer. The Routing Protocol class interacts with the Packet Queue class to manage messages and generate control packets. The Packet Queue class implements a protocol's buffer management scheme, something which is much more significant in other DTN protocols. The `m_BufferSize` attribute defines the maximum size of the message buffer in bytes. `Enqueue` adds messages to the buffer, and `Dequeue` removes messages from the buffer. After a node adds a message to the message buffer, `DropExpiredMessages` removes expired messages. If the message buffer is full, `Purge` removes messages according to the protocol's queue management scheme. `FindDisjointMessages` generates the list of messages to replicate according to a protocol's message priority. The `Drop` function removes a selected message from the buffer. The `GetSize` function returns the number of messages in the buffer.

Since messages are groups of packets, the message buffer requires a data structure to group packets. The `m_queue` is a map matching a message ID to a queue entry defined by the Queue Entry class. A queue entry is a data structure that stores the packets belonging to a message, the IP header, expiration time, and message ID. The `GetMessageByteSize` returns the number of packets contained in message. The `GetMessagePacketTotal` returns the total number of packets belonging to a message. The `GetCurrentPktCnt` returns the number of packets currently contained in the queue entry. The `GetPacketSize` returns the size a data packet in a message. `AddPacket` adds a packet to the queue entry. `GetPackets` returns all of the packets contained in a message. The Packet Queue uses the Queue Entry functions to generate control packets, manage the message buffer, and retrieve messages for the Routing Protocol class.

3.3 Routing Protocol Class

The Routing Protocol class inherits from ns-3's `Ipv4RoutingProtocol`. The Routing Protocol class implements the control logic of the DTN protocols. The `Recv<Protocol>` function executes the control packet exchange based on the packet headers defined in the packet class. The Routing Class initializes the packet queue. The routing class interacts with the packet queue class to generate control packets, and the routing protocol class interacts with the packet queue class to store and retrieve messages. The `SendBeacons` function transmits beacon packets at the specific interval. The `SendDisjointMessages` calls the `FindDisjointMessages` from the Packet Queue class to generate the list of messages to transmit to another node. Then `SendDisjointMessages` calls `SendMessageFromQueue` to transmit the messages from the generated message list. When a node receives a packet, `RouteInput` determines the interface, buffer, or function to execute. `RouteInput` handles the logic for buffering incomplete messages and acknowledging messages described in Section 4. `RouteOutput` handles packets leaving a node.

4 MESSAGE GENERATION AND HANDLING

The ONE and ns-3 handle node data differently. This section discusses how ns-3 and the ONE implement message handling and generation. The discussion includes the differences between the ns-3 PacketEpidemic implementation and our ns-3 Epidemic implementation to handle messages, and ns-3 application-layer message-traffic generation.

4.1 ONE Behavior

In the ONE, traffic generation uses messages. The ONE does not have packets like IP networks, so the ONE does not have packet header definitions. Messages are similar to bundles from the Bundling Protocol [10] because messages are the base unit of DTN data. Unlike IP packets, messages can be any size. The ONE handles messages as a single object. The ONE does not fragment messages, and the ONE does not permit partial messages to propagate throughout the network. If a node does not completely receive a message, then

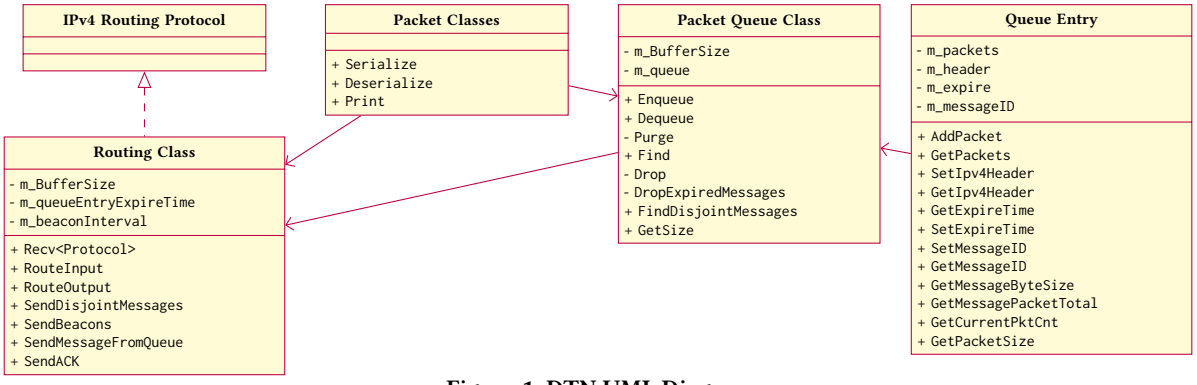


Figure 1: DTN UML Diagram

the node drops the partially received message. In the ONE, messages do not carry control instructions, and nodes share control information by directly accessing another node's data structures in memory. The ONE does not include the overhead of exchanging control instructions in its simulation results.

4.2 ns-3 Behavior

Since ns-3 implements the entire network stack, our ns-3 protocols define groups of packets generated by one source node destined to another node as a message. A message is equivalent to RFC-5050's bundles as the base unit of DTN data. The UDP packets used to share routing information between nodes are control packets. Control packets are not messages because they are routing protocol specific, so they are not the base unit of DTN data. Our protocols assume an IP-based convergence layer. Unlike the ONE, ns-3 cannot generate a message as a single object of any size, so large messages are segmented into groups of individual packets. Each packet is assigned a header with a custom identifier that associates each packet with the rest of the DTN message. This architecture integrates with the existing code base, and does not require the modification of protocols below the routing layer. The following subsections discuss how ns-3 defines and handles messages. The discussion includes the differences from PacketEpidemic message handling, and our ns-3 message-traffic generation.

4.3 Message Definition

Our ns-3 DTN data packet header shown in Figure 3 is the custom header that identifies a packet belonging to a message. The Bundling Protocol (Request For Comments (RFC) 5050) [10] influenced our ns-3 DTN data packet header design, but our ns-3 DTN protocols do not implement the Bundling Protocol. The PacketEpidemic implementation does not have a header for DTN message layer, instead, it uses only UDP packets as the base unit of data with the IP address identifying source and destination nodes.

Each DTN data packet header contains a Message Identification Number, last hop, packet count, and packet index. The Message Identification Number in Figure 2 is a 64-bit unsigned integer derived from the source Node Identification Number and timestamp. The first 16-bits of the Message Identification Number are the source Node Identification Number. The last 48-bits are the message's generation timestamp in microseconds. The source Node Identification

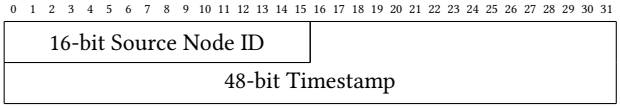


Figure 2: Message Identification Number

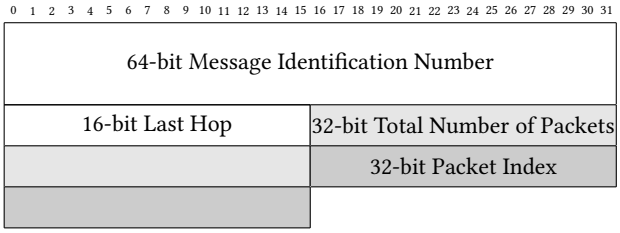


Figure 3: DTN Data Packet Header

Number used by the DTN packet header is the node number from ns-3, which automatically assigns a unique integer to every node in the simulation.

The range of scenarios we expect to simulate require a node to generate several messages in a millisecond at most, but will not require generation of more than one message in a microsecond. The timestamp is in microseconds to ensure that every Message Identification Number is unique. While a per-node sequential message counter would also create a unique identifier, our ns-3 protocols require message generation time and source node identification for routing decisions, so this information is dual-purpose. Some scenarios studied would exceed the largest time in microseconds represented by a 32-bit timestamp, therefore we use 48-bits for the timestamp. The last hop field is the 16-bit Node Identification Number of the last node that forwarded the message. Nodes use the last hop to buffer incomplete messages for message reconstruction. The 32-bit total number of packets and the 32-bit packet index support reordering during message reassembly.

4.4 Message Handling

In PacketEpidemic, nodes pushed all packets selected for transmission to the link layer immediately after completing the control packet exchange. ns-3's link layer contains a packet queue that has a limited size. This link-layer packet queue is First In First

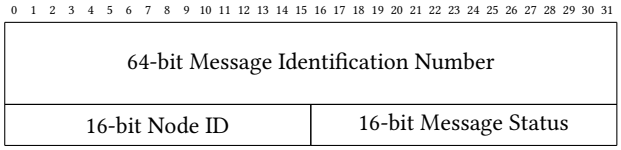


Figure 4: DTN Acknowledgement Header

Out (FIFO) and cannot be manipulated by higher layers in the network stack, they can only add packets [3]. When a node's message buffer equals or exceeds the size of the packet queue, then the node will fill the packet queue and any new packets sent to the full queue are dropped. The link layer will attempt to transmit the packets in FIFO order regardless if the destination node is connected. If the first node moves out of range, then the link layer will still transmit the packets. As a result, the node wastes available bandwidth and meeting opportunities. This behavior is not a requirement of the Epidemic routing protocol, but is a limitation specific to the PacketEpidemic implementation.

In order to improve link utilization, `ACK` packets in Figure 4 control the message exchange sequence. `ACK` packets do not acknowledge messages reaching their final destination, rather, they are a hop-by-hop acknowledgement of messages transmitted between two connected nodes. When a node receives a complete message from another node, it sends an acknowledgement packet consisting of the 64-bit Message Identification Number, 16-bit Node Identification Number, and 16-bit Message Status. The Message Identification Number is the received message's message ID. The Node Identification Number is the node that received the message. The Message Status block permits adding reliability in future work. The Message Status indicates if a node received a message successfully. If the message transfer is unsuccessful, then Message Status is zero. We do not retransmit such messages, but such reliability may be a desirable future enhancement and is supported by the header.

After the transmitting node determines message priority, the node fetches the first message. A node transmits the complete message and waits for an `ACK` before sending the next message. The receiving node queues the message's packets in a message reception buffer. The receiving node maintains a message reception buffer for each node that is connected. Each connection's receiving buffer can buffer only one message. When a message is complete, the receiving node transmits an `ACK`. `RouteInput` handles message reconstruction and `ACK` generation. Upon receiving the `ACK`, the transmitting node transmits the next message. The `Recv<Protocol>` function handles `ACK` reception. If the connection breaks, then the node resets the message reception buffer for that connection. A node considers a connection broken when the node does not receive any packets from that neighbor for two beacon intervals. When a node does not receive a complete message, it deletes the partial message. As a result, nodes do not forward partial messages.

Deleting partial messages may seem wasteful, but dropped messages due to a lost packet occurs infrequently (*e.g.* less than 0.1% of the time in the Helsinki scenario).

While `ACK`s improve link efficiency, they also have limitations. First, a node requires a large link layer buffer because the buffer must store all of the packets contained in a message. When the link layer's packet buffer exceeds the packet limit, the node removes the newest packets from the queue. As a result, those packets fail to transfer causing messages to drop. ns-3's link layer limits the number of packets that can be stored in the buffer, so the scenario configuration file must increase the link layer's buffer to accommodate the expected message sizes. Second, the link layer buffer deletes packets based on a time-limit. When a packet exceeds the time-limit, the link layer buffer deletes the packet. For the purposes of this work, the example scenarios set the link layer buffer to the size of the node's message buffer. The scenarios set the packet queue time-limit to two beacon intervals because two beacon intervals correspond to a dropped connection.

4.5 Message Generation

The DTN Application generates message traffic using the DTN Packet Header, and is adapted from the On-Off Application. The DTN Application generates UDP packets with the DTN Packet Header. The message size parameter in bytes and packet size determines the number of generated packets. The DTN Application generates one message according to the entered parameters. The Message Identification Number uses the source Node Identification Number and message generation time. If a scenario requires more than one message, then the scenario must include multiple DTN Application message generators.

Since future work may use packets instead of messages for data traffic, our ns-3 DTN protocol implementation is backwards compatible with raw UDP packets. When a node generates a UDP packet that does not have the DTN Packet Header, the DTN router on that node creates a DTN Packet Header for that data packet. `RouteInput` handles DTN Packet Header generation for standard UDP packets. The Message Identification Number uses the packet's generation time in microseconds and the packet's source Node Identification Number to generate the Message Identification Number. A node then treats each packet as an individual message, permiting ns-3's default UDP packet generators to work with the DTN protocols.

5 NODE DISCOVERY

DTNs do not have constant connectivity, so nodes must discover other nodes to initiate a connection. Unlike ns-3, the ONE simulator handles node discovery independently of any routing protocol. As a result, the ns-3 DTN protocols must include a node discovery mechanism.

In PacketEpidemic, nodes transmit `BEACON` control packets at a specific time interval for node discovery. The `BeaconInterval` defines the frequency of beacon transmission. To ensure that all nearby nodes can receive the beacon, the node broadcasts a beacon using the network's broadcast address. Nodes are likely to be synchronized in sending broadcasts, resulting in lost beacons due to collisions. In order to prevent synchronization among nodes, a uniform random variable staggers the beacons. The `BeaconRandomness` variable defines the upper bound of the uniform random distribution to add to the base beacon interval. Since each beacon starts the exchange process between nodes, nodes with many packets buffered may already be exchanging packets when the next beacon interval occurs. The `HostRecentPeriod` prevents hosts from re-exchanging redundant control packets [2], however nodes with large buffers will exceed the `HostRecentPeriod` used PacketEpidemic resulting

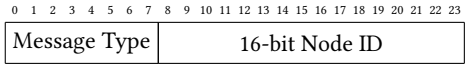


Figure 5: DTN `MessageType` Header

in the nodes re-exchanging control packets and messages while already transferring messages. This causes messages to be sent and re-sent multiple times wasting bandwidth.

Our Epidemic implementation also uses beacons for node discovery. A node broadcasts beacons at a set interval plus a random delay to minimize beacon collisions. However, we do not use a `HostRecentPeriod` to prevent hosts from re-exchanging redundant control packets, instead each node remembers the neighbors with which it is currently in contact. Every time a node receives a control packet or data packet from another node; it updates its record of connected nodes. A node considers a connection broken when the node does not receive anything from the neighbor for two beacon intervals.

A node can have more than one radio interface and IP address, so IP address is not a sufficient identifier to prevent two nodes from restarting a connection when they detect the second radio interface.

The `BEACON` control packet is a `MessageType` header with the field set to `BEACON`. The `MessageType` header field indicates the type of control packet. Since a node does not know whether another node has more than one IP address, the beacon must include the Node Identification Number. Therefore, the `MessageType` header includes the 16-bit Node Identification Number. Figure 5 illustrates the `MessageType` Header. ns-3 provides a unique number to every node in a simulation. A node uses the Node Identification Number to determine whether a node is considered connected. If a node is not connected, then the node adds the Node Identification Number with a timestamp to its list of connected nodes. Then, the node continues the control packet exchange. If the Node Identification Number is in the list of connected nodes and the connected timestamp does not exceed two beacon intervals, then the node ignores the `BEACON`. However, every data packet and control packet other than `BEACON`s update the connection timestamp. A `BEACON` changes the timestamp only when the timestamp exceeds two beacon intervals because it is considered as a new connection.

6 EPIDEMIC LOGIC

The ns-3 Epidemic control logic performs four main steps as shown in Figure 6. After receiving a `BEACON`, nodes exchange `REPLY` and `REPLY_BACK` control packets. `REPLY` and `REPLY_BACK` control packets are the summary vectors discussed in [11]. The goal of sending message summaries is to avoid sending messages that the other node already contains in its buffer. Since both nodes are likely to send a response to a beacon at the same time, an anti-entropy session prevents node responses from colliding. The node with the lower IP address sends its message summary first as a `REPLY` packet. When the node with the higher IP address receives the `REPLY` packet, it sends a response using the `REPLY_BACK` packet. After a node receives the list of messages that the other node contains, the node sends messages that the other node does not have in its buffer.

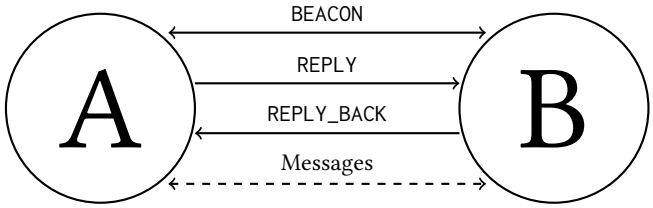


Figure 6: Epidemic Control Packet Exchange Sequence

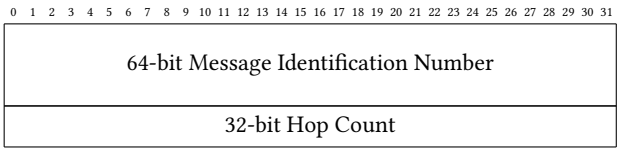


Figure 7: `EpidemicHeader`

6.1 Message Identification and Limits

Since our implementation manages messages instead of packets, we use a different `EpidemicHeader` than PacketEpidemic [2]. Figure 7 illustrates the `EpidemicHeader`. First, the 64-bit Message Identification Number from Section 4 replaced the 32-bit packet identification number. Since the Message Identification Number includes the source node and timestamp of message generation, we do not use a separate timestamp field.

The ONE uses hop limits and message Time to Live (TTL) to reduce network resource consumption. The lower 48-bit portion of the Message Identification Number permits the routing protocol to remove expired messages from a node's buffer. When a node receives a message, the node checks the timestamp against the maximum age of a message. If the timestamp exceeds the amount of time a message can live, then the node discards the message. The `EpidemicHeader` 32-bit hop count field is used limit the number of hops that a message can traverse. When nodes generate a message, they initialize the hop count to the maximum number of hops allowed. At each hop, the node decrements the hop count and when the hop count reaches zero, the message is discarded.

6.2 Control Packet Identification

The `MessageType` Header in Figure 5 identifies control packets. Since the ONE simulator uses shared data structures to exchange routing information it did not implement control packets. In our implementation, control packets are not messages (the do not have a DTN message header). Control packets share routing specific information between two neighboring nodes. The 8-bit Message Type field indicates the type of control packet. For Epidemic, control packets are `BEACON`, `ACK`, `REPLY`, and `REPLY_BACK`. The `MessageType` Header encapsulates the control packets in order to identify the control packet. When a node receives a packet, it checks the Message Type field for the packet type to call the appropriate packet header class.

6.3 Summary Vector

While Epidemic uses the same control packet sequence as PacketEpidemic, the `SummaryVectorHeader` in Figure 8 differs. The

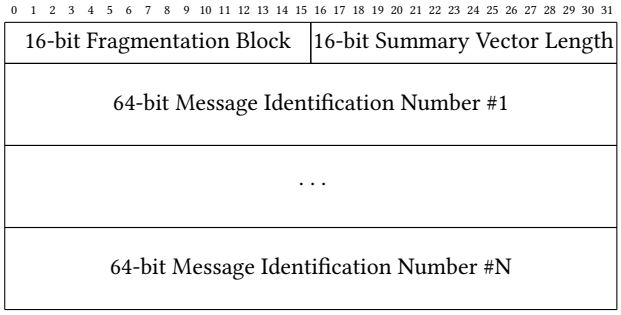


Figure 8: Epidemic `SummaryVectorHeader`

`SummaryVectorHeader` defines `REPLY` and `REPLY_BACK` control packets. We use a 64-bit Message Identification Number in place of packetEpidemic's 32-bit packet identification number. PacketEpidemic's 32-bit Summary Vector Length counts all of the packets held by a node's buffer. In our Epidemic implementation, the 16-bit Summary Vector Length counts the number of Message Identification Numbers in the `SummaryVectorHeader`. A 16-bit unsigned integer is large enough cover a `SummaryVectorHeader`. Since the `MessageType` Header encapsulates the `SummaryVectorHeader`, a node identifies the other node using the Node Identification Number instead of the IP address because nodes can have more than one IP address.

PacketEpidemic did not support control packet fragmentation. In order to support fragmentation, the Fragmentation Block identifies whether there are more `SummaryVectorHeader` packets. When the Fragmentation Block is one, more `SummaryVectorHeader` packets remain. When the Fragmentation Block is zero, that packet is the last `SummaryVectorHeader`. Once a node receives one with the Fragmentation Block set to zero, the node continues the message exchange sequence. The control packet fragmentation can tolerate packet loss if the lost packet had the fragmentation block set to one, but the fragmentation protocol does not support retransmission of control packets. If the lost control packet had the fragmentation block set to zero, then the control packet sequence would stop until a beacon restarts the exchange sequence.

7 EVALUATION

Our primary interest in implementing DTN protocols in ns-3 is to gain fidelity and account for the cost associated with control message exchanges and the overhead of layer-2 protocols, which can not be simulated in the ONE and other popular DTN simulators. With this in mind, our reference point is the performance of the same protocol in the ONE simulator. We note that this comparison would be impossible without support for segmenting messages into multiple packets, due to the use of relatively large messages as the base unit of transmission in the ONE and other DTN simulators. We compare the protocols using three different mobility scenarios, all of which are run in the ONE, which outputs and ns-2 mobility trace which in turn is input to ns-3. In this way, while there is still randomness in the mobility, there is no difference in the mobility of the nodes between the two simulators. Each parameter set is run 10 times with different seeds for the random number generator. The Helsinki mobility scenario is an urban environment, which is the default scenario for the ONE simulator and hence appears commonly in DTN simulation literature. Bold Alligator is our interpretation of a US Marine Corps exercise, and Omaha is our interpretation of the historic amphibious assault at Omaha Beach during World War II. While we do not have room here to fully specify each scenario, they are presented in detail in LT Mauldin's master's thesis [5].

7.1 Helsinki

Table 1 shows that ns-3's Epidemic returns lower Message Delivery Ratio (MDR), higher average latency, and lower message replication overhead. Both simulators show that Epidemic's MDR increases as buffer size increases. Link layer overhead and control packets reduce available bandwidth to share messages. The ONE does not include the time or bandwidth consumed by control packets, so the ONE version increases message replication because nodes have more time to share messages. ns-3's increased sensitivity to transmission speed and higher latency highlights the impact of the added overhead.

7.2 Bold Alligator

In ns-3 and the ONE, Epidemic's MDR and average latency increase asymptotically with buffer size. Epidemic's average hop count increases from the 5 Megabyte (MB) to 10 MB buffer, but Epidemic shows minimal change in average hop count for larger message buffers in both simulators. Message replication overhead in the ONE and ns-3 follow the same trend. Larger message buffers return minimal change in message replication overhead.

While both versions of Epidemic share trends, ns-3 consistently returns lower MDR, higher average latency, and lower message replication overhead. To illustrate this observation, Table 2 contains the percent difference between ns-3 and the ONE. ns-3's MDR is 66% to 74% percent lower than the ONE. Average latency is 52% to 108% higher. As buffer size increases, the difference in average latency between ns-3 and the ONE increases. Message replication overhead is 19% to 31% lower in ns-3. Larger message buffers and higher speed radios increase the message replication overhead performance gap between simulators.

7.3 Omaha

Epidemic's MDR and average latency increase asymptotically with buffer size. Message replication overhead decreases with larger message buffers, the ONE shows a steep decrease compared to ns-3's shallow decrease in message replication overhead. Both versions of Epidemic show that average latency increases asymptotically with larger message buffers. Epidemic's performance in ns-3 is close to the ONE in Table 3. ns-3's MDR is within 35% of the ONE, and the simulators match at several data points. At small buffer sizes, ns-3 has lower latency than the ONE, but large message buffers show ns-3 has higher latency. With respect of message replication overhead, ns-3's message replication overhead is 60% to 83% lower than the ONE. Higher transmission speeds increase the performance gap in message replication overhead between ns-3 and the ONE.

Table 1: Helsinki Simulator Performance Difference

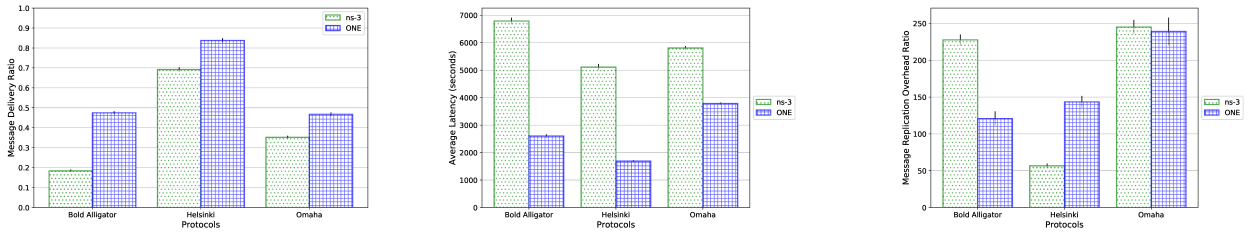
	MDR					Average Latency					Message Replication Overhead Ratio				
	5 MB	10 MB	25 MB	50 MB	100 MB	5 MB	10 MB	25 MB	50 MB	100 MB	5 MB	10 MB	25 MB	50 MB	100 MB
6 Mbps	-62%	-70%	-78%	-78%	-77%	-2.8%	27%	72%	82%	73%	-27%	-11%	50%	35%	43%
12 Mbps	-32%	-45%	-56%	-62%	-61%	-27%	-5.3%	48%	82%	99%	-56%	-50%	-27%	-15%	-14%
24 Mbps	11%	-10%	-25%	-31%	-36%	-38%	-23%	23%	66%	105%	-69%	-68%	-59%	-55%	-49%
36 Mbps	34%	12%	-2.7%	-12%	-20%	-44%	-29%	12%	56%	95%	-73%	-73%	-69%	-67%	-62%
54 Mbps	50%	38%	17%	8.9%	-4.2%	-47%	-32%	6.8%	42%	80%	-76%	-77%	-76%	-76%	-72%

Table 2: Bold Alligator Simulator Performance Difference

	MDR				Average Latency				Message Replication Overhead Ratio			
	5 MB	10 MB	25 MB	50 MB	5 MB	10 MB	25 MB	50 MB	5 MB	10 MB	25 MB	50 MB
12 Mbps	-77%	-74%	-73%	-74%	60%	66%	86%	90%	-19%	-25%	-26%	-27%
24 Mbps	-74%	-71%	-69%	-70%	58%	56%	96%	99%	-21%	-28%	-29%	-29%
36 Mbps	-72%	-70%	-67%	-68%	52%	60%	92%	108%	-21%	-32%	-29%	-31%
54 Mbps	-71%	-69%	-66%	-67%	53%	60%	89%	100%	-30%	-32%	-31%	-31%

Table 3: Omaha Simulator Performance Difference

	MDR					Average Latency					Message Replication Overhead Ratio				
	5 MB	10 MB	25 MB	50 MB	100 MB	5 MB	10 MB	25 MB	50 MB	100 MB	5 MB	10 MB	25 MB	50 MB	100 MB
6 Mbps	0%	-16%	-25%	-31%	-35%	-27%	-17%	8.8%	26%	43%	-74%	-73%	-60%	-61%	-65%
12 Mbps	10%	-8.1%	-15%	-21%	-26%	-35%	-22%	0%	13%	33%	-77%	-75%	-65%	-68%	-71%
24 Mbps	19%	0%	-15%	-16%	-20%	-39%	-30%	-10%	3.7%	29%	-79%	-79%	-69%	-74%	-75%
36 Mbps	23%	0%	-9.1%	-11%	-16%	-42%	-29%	12%	5.7%	29%	-80%	-79%	-72%	-75%	-78%
54 Mbps	29%	1.9%	-8.8%	-13%	-17%	-41%	-30%	-9.1%	3.2%	26%	-83%	-81%	-75%	-78%	-79%



(a) Aggregate Message Delivery Ratio

(b) Aggregate Average Latency

(c) Aggregate Message Replication Overhead

Figure 9: Scenario aggregate performance

7.4 Aggregate Scenario Performance

This section compares the overall performance of ns-3 and the ONE using MDR, average latency, average hop count, and message replication overhead. The bar graphs are an average of all buffer sizes, transmission speeds, and protocol data points for each scenario. Each bar graph uses a 95% confidence interval.

Figure 9(a) compares aggregate protocol and scenario MDR between the ONE and ns-3. Within Bold Alligator, ns-3 delivers 61% fewer messages than the ONE, and 24% fewer messages in Omaha. ns-3 delivers 17% fewer messages in Helsinki. Across all scenarios and protocols, ns-3 delivers 31% fewer messages than the ONE.

Figure 9(b) compares aggregate average latency for protocols and scenarios between simulators. Epidemic's average latency is 21% higher in ns-3 than the ONE. In relation to network overhead in Figure 9(c), protocols with higher message replication overhead tend to return a smaller difference between simulators. Helsinki's average latency in ns-3 is 202% higher than the ONE. Bold Alligator's average latency is 160% higher in ns-3 and Omaha's average latency is 53% higher in ns-3. Across all scenarios and protocols, ns-3's average latency is 119% higher than the ONE.

Unlike MDR and average latency, message replication overhead in Figure 9(c) displays different trends between military and urban scenarios. ns-3 returns lower message replication overhead than the ONE by 61%. However, ns-3's message replication overhead is 88% higher in Bold Alligator and 2.5% higher in Omaha. We note that the ns-3 protocols increase message replication overhead due to messages circling within clusters of nodes. The ONE protocols, except for Epidemic, do not have this behavior because nodes check

whether a message traversed the other node. If the message already traversed the other nodes, then the node does not transmit the message. Helsinki does not form node clusters, so ns-3's Helsinki does not return the higher message replication overhead.

Messages circling within clusters does occur with the ONE's Epidemic, so the ns-3's message replication overhead is 65% lower than the ONE. Across all scenarios and protocols, ns-3's message replication overhead is 5% higher than the ONE.

In summary, ns-3 delivers fewer messages and experiences higher average latency than the ONE. Protocols that share more data to make routing decisions tend to deliver fewer messages in ns-3 than the ONE. Message replication overhead depends on node mobility due to implementation differences. Scenarios that form clusters of nodes return higher message replication overhead in ns-3. Scenarios that do not form clusters of nodes return lower message replication overhead in ns-3.

8 CONCLUSIONS

ns-3 and the ONE employ different levels of abstraction to simulate network protocols. The ONE focuses on simulating the behavior of opportunistic routing protocols, so the ONE abstracts everything below the routing layer. In contrast, ns-3 simulates the entire network stack. The ns-3 routing protocols require packets for messages, node discovery, and sharing routing information between nodes. The ONE does not include link layer overhead, packet header overhead, or control packet overhead.

The ONE sends DTN data as a single object called a message, and the ONE shares routing information by directly accessing communicating nodes' memory data structures. Our ns-3 DTN protocols assume an IP convergence layer adapter by defining groups of packets that compose a DTN message. Control packets share routing information between nodes in ns-3. When comparing the effective throughput of messages transmitted between two connected nodes, the addition of packet headers and link layer overhead reduces effective throughput by 40% to 70% relative to the ONE's radio bandwidth. Depending on the scenario and protocol, packet headers added during message segmentation in ns-3 make up 2% to 5.5% of all transmitted data. Depending on the routing protocol and scenario, ns-3's control packets can consume a significant portion of transmitted data. Control packets make up 0.1% to 33% of all transmitted data in ns-3. Protocols that share less information for routing decisions transmit fewer/smaller control packets. While control packets contribute to network overhead, message replication represents a larger fraction of network overhead. For protocols that use the same message limit algorithm, ns-3 protocols with more control packets consume more power.

When comparing the ONE and ns-3 DTN protocols, the ns-3 protocols returned 31% lower MDR and 119% higher average latency aggregated across all protocols and scenarios. Message replication overhead varies between scenarios. In Helsinki, the ns-3 protocols return lower message replication overhead than the ONE.

Both simulators demonstrate sensitivity to buffer size. Larger message buffers return higher MDR. In Helsinki, larger message buffers reduce average latency. Bold Alligator and Omaha show that larger message buffers increase average latency. ns-3 shows greater sensitivity to transmission speed than the ONE. In ns-3, higher transmission speeds return higher MDR, lower average latency, and increase message replication overhead. Higher transmission speeds in the ONE return small changes in MDR, slightly reduces average latency, and increases message replication overhead. ns-3's increased sensitivity to transmission speed is due to control packets, packet headers, and link layer overhead.

Based on our findings, our future DTN protocol development will continue in ns-3 instead of the ONE. While the ONE permits faster development of new protocols, and ns-3's simulations take 25 to 50 times longer than the ONE, and ns-3 requires a separate program to analyze the large trace files; the ONE's abstraction may not reflect actual protocol performance. The sharing of routing information via shared data structures subsidizes performance of protocols with very large control packets. Considering control packets transmission impacts protocol performance analysis and link layer overhead significantly, and also significantly reduces message replication, as shown in our ns-3 results.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Robert Beverly for his feedback and contribution to improving the rigor of this work. We also thank the WNS3 reviewers for their constructive analysis that strengthened this contribution.

The work was funded in part by the United States Marine Corps and United States Navy. Views and conclusions are those of the authors and should not be interpreted as representing the official policies or position of the U.S. government.

REFERENCES

[1] 2009. The ns-3 Network Simulator. http://www.nsnam.org. (July 2009).
[2] M. J. F. Alenazi, Y. Cheng, D. Zhang, and J. P. G. Sterbenz. 2015. Epidemic Routing Protocol Implementation in ns-3. In *Proceedings of the 2015 Workshop on ns-3 (WNS3 '15)*. ACM, New York, NY, USA, 83–90.
[3] ns-3 Consortium. 2015. *ns-3 Manual*.
[4] A. Keränen, J. Ott, and T. Kärkkäinen. 2009. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST, New York, NY, USA.
[5] A. N. Mauldin. 2017. *Comparative Analysis of Disruption Tolerant Network Routing Simulations in the ONE and ns-3*. Master's thesis. Naval Postgraduate School, Monterey, CA.
[6] H. Narra, Y. Cheng, E. K. Çetinkaya, J. P. Rohrer, and J. P.G. Sterbenz. 2011. Destination-Sequenced Distance Vector (DSDV) Routing Protocol Implementation in ns-3. In *Proceedings of the Workshop on ns-3 (WNS3)*. Barcelona, Spain.
[7] A. Pospischil and J. P. Rohrer. 2017. Multihop Routing of Telemetry Data in Drone Swarms. In *Proceedings of the International Telemetering Conference (ITC)*. Las Vegas, NV.
[8] J. P. Rohrer. 2017. Effects of GPS Error on Geographic Routing. In *Proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, Vancouver, Canada.
[9] J. P. Rohrer and K. M. Killeen. 2016. Geolocation Assisted Routing Protocols for Vehicular Networks. In *Proceedings of the 5th IEEE International Conference on Connected Vehicles (ICCVE)*. Seattle, WA.
[10] K. Scott and S. Burleigh. 2007. Bundle Protocol Specification. RFC 5050 (Experimental). (Nov. 2007). http://www.ietf.org/rfc/rfc5050.txt
[11] A. Vahdat and D. Becker. 2000. *Epidemic Routing for Partially-connected Ad Hoc Networks*. Technical Report CS-200006. Duke University.